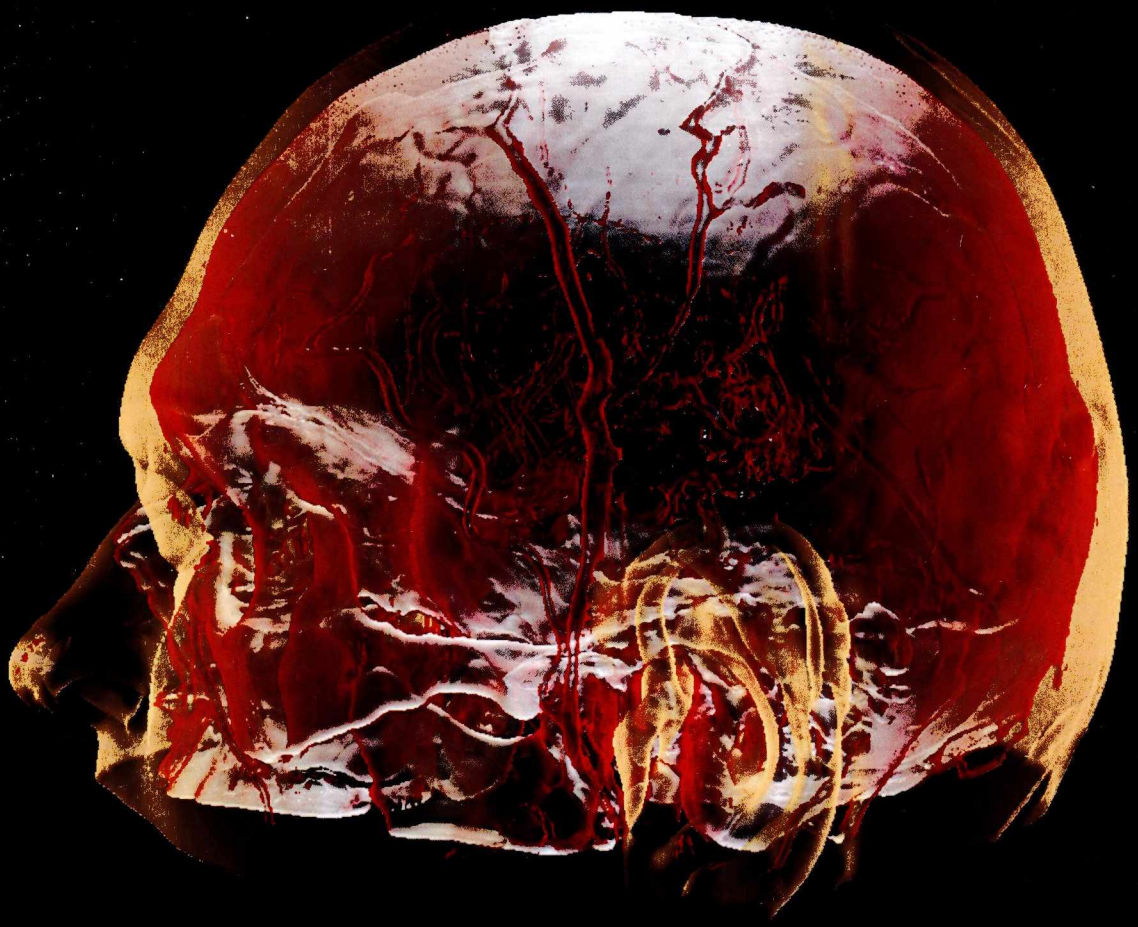
VII. MAGYAR SZÁMÍTÓGÉPES GRAFIKA ÉS GEOMETRIA KONFERENCIA

Budapest
2014. FEBRUÁR 19-20.

Szerkesztette:
Szirmay-Kalos László
Renner Gábor



Neumann János Számítógép-tudományi Társaság

Előszó

Köszöntjük a Hetedik Magyar Számítógépes Grafika és Geometria Konferencia (GRAFGEO 2014) kiadványát kezében tartó kedves olvasót. Ez a konferencia a számítógépes képfeldolgozással, képekkel és geometriával kapcsolatos tudományok seregszemléje, amelyet a Számítógép-tudományi Társaság Számítógépes Grafika és Geometria Szakosztály (GRAFGEO) szervez kétévente. A konferenciára beérkező cikkek száma és minősége mutatja a témakörnek a hazai kutatás-fejlesztésben és egyetemi oktatásban játszott szerepét, hangsúlyait és fejlődési irányait is.

A benyújtott cikkeket a szerkesztők ellenőrizték. A kiadványt Umenhoffer Tamás készítette elő nyomtatásra. A címlapon látható képek Csébfalvi Balázs és Tóth Balázs munkái. A konferencia legfőbb szponzora a Neumann János Számítógép-tudományi Társaság volt. A Számítástechnikai és Automatizálási Kutató Intézet (SZTAKI) a konferenciaterem és az infrastruktúra ingyenes rendelkezésre bocsátásával, a BME Irányítástechnika és Informatikai Tanszék a nyomdai előkészítés átvállalásával járult hozzá a konferencia sikeres megrendezéséhez. A szerkesztők hálásan köszönik a támogatást minden szervezetnek és az önzetlen segítséget a szervezésben közreműködőknek.

Budapest, 2014. február 19.

Renner Gábor és Szirmay-Kalos László

szerkesztők

Tartalomjegyzék

Felületrekonstrukció

Képszintézis

Kép és videófeldolgozás

Orvosi vizualizáció és tomográfia

Seventh Hungarian Conference on Computer Graphics and Geometry, Budapest, 2014

General Matrix Representation of B-Splines and Approximation of B-Spline Curves and Surfaces with Third Order Continuity

Szilvia Béla[1] and Márta Szilvási-Nagy[1]

[1] Department of Geometry, Mathematical Institute, Budapest University of Technology and Economics, Budapest, Hungary

Abstract

In this paper first an algorithm is presented to compute the matrix representation of non-uniform B-Spline functions defined on arbitrary knot sequence. The algorithm is based on the reformulation of the de Boor-Cox recursion. With the help of this reformulation the transformation matrix is obtained between non-uniform B-Splines and Bézier representations. Then a method is presented to approximate separately created B-Spline curves. The input curves and the resulting curve are represented by fourth degree B-splines. The approximation technique minimizes a target function expressed by squared differences in positions and first derivatives of the input and resulting curves at their corresponding points. The solution is the set of control points of the required B-spline curve approximating two input arcs or curves. This method can be applied to merge B-spline curves or B-spline surfaces filling possible gaps between them.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computational Geometry and Object Modeling]: Curve, surface, solid, and object representations I.3.5 [Computational Geometry and Object Modeling]: Splines

1. Introduction

The B-Splines – introduced by Schoenberg [1] – are applied in various fields of computer graphics and geometric modeling. Several algorithms have been developed and implemented in modeling systems to visualize and to manipulate B-spline curves and surfaces. In these computations the form of the spline representation has a special importance.

From the 1970s – when B-spline representation generally spread in curve and surface design – several algorithms were developed to derive the matrix representation of B-spline functions. The matrix representation provides fast and efficient evaluation of points on spline curves and surfaces, computation of derivatives, and speed up several further spline algorithms. The first matrix representation form of splines was derived for Bézier curves by Chang[2]. Then Cohen and Riesenfeld[3] published a formula to represent uniform B-splines in matrix form. In 1990 Choi et al.[4] presented a recursive procedure to generate the matrix representation of NURBS curves and surfaces of arbitrary degree. In his procedure the matrix representation was obtained by using Boehm's knot insertion algorithm. Later Grabowski et al.[5] and Liu et al.[6] presented analogous approaches based on different concepts. The algorithm presented by Grabowski and Li is a recursive algorithm derived from the de Boor-Cox algorithm, while Wang et al. presented two different algorithms, both calculating explicitly the matrix representation of NURBS. One is derived from the computation of divided differences and the other from the Marsden identity.

In the literature several further algorithms were presented, which convert the non-uniform B-spline functions to Bézier form and vice versa. Until 2004 these algorithms were calculating directly with the basis functions, none of them utilized the advantages of matrix representation form. Romani and Sabin[7] proposed the first algorithm, which generates direct matrix transformation between uniform B-Spline and Bézier representation. Later Casciola and Romani[8] generalized the computations also for non-uniform B-splines.

Stitching or merging separately created B-Spline curves and surface patches is frequently used in geometric modeling, and it is an important procedure in CAD-systems. These

algorithms are basically numerical interpolations using the least squares method. Well functioning numerical solutions have been developed, therefore, relatively few papers have been published about the symbolic solution of the merging problem. Tai et al.[12] and Chen et al.[9] presented methods for approximate merging of B-spline curves and surfaces. Hu et al.[10] described an extension algorithm for B-spline curves by adding more interpolation points one by one at the end of the curve. In the paper of Pungotra et al.[11] the construction of a covering surface is shown for unifying more B-spline surfaces.

In this paper first we present a recursive algorithm which generates the matrix representation of non-uniform B-spline functions. The algorithm based on the reformulation of the B-spline recursion. From the new recursion formula we can generate the representation matrix of B-splines in the not normalized Bernstein basis. Then we derive the transformation matrix from the not normalized Bernstein basis to the polynomial space spanned by the power basis. Thus with the algebraic reformulation of the B-spline recursion we gain a transformation matrix to the Bernstein basis and also to the power basis.

Then we approach the problem of merging two separate B-spline curves. In the examples the input curves and the resulting curve are represented by fourth degree B-splines. The input curves may join at their end points precisely or with gaps. The applied approximation technique is minimization of a target function expressed by squared differences in positions and first derivatives of the input and the resulting curves at their corresponding points. The variables in this function are the unknown control points of the approximating curve. In the case of uniform B-splines the coefficients of the basis functions are constant, therefore the minimization problem can be solved symbolically. In the non-uniform case the basis functions are depending on the parameter (knot) values, and they have to be determined for each curve segment separately. The matrix representation provides an efficient method for the solution of the approximation problem avoiding non-linear optimizations. The required control points of the new approximating B-spline curve are the solutions of a system of linear equations, and they are expressed as linear combinations of the input control points. This method is basically different from the stitching method shown in Szilvasi et al.[17] using interpolation and fairing. Matrix representation and fairing conditions for cubic B-spline functions have been applied in papers by Szilvasi[13, 14, 15]. Finally, merging of B-spline surface patches are shown applying the developed curve approximation method for their parameter curves.

2. Preliminaries: B-splines, de Boor-Cox algorithm and reformulation

B-spline functions of order k over the knot vector $\{t_1,\dots t_n\}$ are defined by the de Boor-Cox formula as:

$$N_i^1(t) = \begin{cases} 1, & t\in[t_i,t_{i+1}) \\ 0, & \text{Otherwise,} \end{cases}$$

$$N_i^k(t) = \alpha(t_i,t_{i+k-1};t)\,N_i^{k-1}(t)+\alpha(t_{i+k},t_{i+1};t)\,N_{i+1}^{k-1}(t),$$

where the α function is defined as

$$\alpha(A,B;t)=\frac{t-A}{B-A} \tag{1}$$

for arbitrary A,B parameters, where $A\neq B$, and for all $t\in\mathbb{R}$. This function has several properties, which are very advantageous for B-spline computations. The main properties are

$$\alpha(A,B;A) = 0, \tag{2}$$

$$\alpha(A,B;B) = 1, \tag{3}$$

$$1-\alpha(A,B;t) = \alpha(B,A;t), \tag{4}$$

$$\frac{1}{\alpha(A,B;t)} = \alpha(A,t;B), \qquad t\neq A \tag{5}$$

$$\alpha(A,B;t) = \alpha(A,C;t)\cdot\alpha(A,B;C). \tag{6}$$

These properties can be easily derived from the definition of the α function (1). Using the properties (4) and (6) a further identity can be derived

$$\alpha(A,B;t)=\alpha(C,D;t)\alpha(A,B;D)+\alpha(D,C;t)\alpha(A,B;C) \tag{7}$$

for any $C\neq D$ real numbers. A special case of (7) when we choose $C=0$ and $D=1$ then

$$\alpha(0,1;t) = t$$
$$\alpha(1,0;t) = 1-t,$$

thus

$$\alpha(A,B;t)=t\cdot\alpha(A,B;1)+(1-t)\cdot\alpha(A,B;0) \tag{8}$$

is fulfilled.

As the consequence of the special identity (8) we can rewrite the de Boor-Cox recursion in general:

$$N_i^1(t) = \begin{cases} 1, & t\in[t_i,t_{i+1}) \\ 0, & \text{Otherwise,} \end{cases}$$

$$\begin{aligned} N_i^k(t) &= t\left[\alpha(t_i,t_{i+k-1};1)\,N_i^{k-1}(t)+\alpha(t_{i+k},t_{i+1};1)\,N_{i+1}^{k-1}(t)\right]+ \\ &+ (1-t)\left[\alpha(t_i,t_{i+k-1};0)\,N_i^{k-1}(t)+\alpha(t_{i+k},t_{i+1};0)\,N_{i+1}^{k-1}(t)\right]= \\ &= t\left[\frac{1-t_i}{t_{i+k-1}-t_i}N_i^{k-1}(t)+\frac{1-t_{i+k}}{t_{i+1}-t_{i+k}}N_{i+1}^{k-1}(t)\right]+ \\ &+ (1-t)\left[\frac{t_i}{t_i-t_{i+k-1}}N_i^{k-1}(t)+\frac{t_{i+k}}{t_{i+k}-t_{i+1}}N_{i+1}^{k-1}(t)\right], \end{aligned}$$

where the piecewise polynomial B-spline $N_i^k(t)$ has the support $t\in[t_i,t_{i+k})$.

In order to generate the pieces of the B-spline functions

restricted to one knot interval $[t_j, t_{j+1})$, we can also rewrite the recursion as

$$\begin{aligned} N_i^1(t) &= \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & \text{Otherwise,} \end{cases} \\ N_i^k(t) &= \alpha(t_j, t_{j+1}; t) \left[\alpha(t_i, t_{i+k-1}; t_{j+1}) N_i^{k-1}(t) + \right. \\ &+ \left. \alpha(t_{i+k}, t_{i+1}; t_{j+1}) N_{i+1}^{k-1}(t) \right] + \\ &+ \alpha(t_{j+1}, t_j; t) \left[\alpha(t_i, t_{i+k-1}; t_j) N_i^{k-1}(t) + \right. \\ &+ \left. \alpha(t_{i+k}, t_{i+1}; t_j) N_{i+1}^{k-1}(t) \right]. \end{aligned} \tag{9}$$

According to this form we transform all B-spline segments from the knot span $[t_j, t_{j+1})$ and represent over the unit interval as follows:

$$\begin{aligned} N_i^1(u) &= \begin{cases} 1, & i = j \\ 0, & \text{Otherwise,} \end{cases} \\ N_i^k(u) &= u \left[\alpha(t_i, t_{i+k-1}; t_{j+1}) N_i^{k-1}(t) + \right. \\ &+ \left. \alpha(t_{i+k}, t_{i+1}; t_{j+1}) N_{i+1}^{k-1}(t) \right] + \\ &+ (1-u) \left[\alpha(t_i, t_{i+k-1}; t_j) N_i^{k-1}(t) + \right. \\ &+ \left. \alpha(t_{i+k}, t_{i+1}; t_j) N_{i+1}^{k-1}(t) \right], \end{aligned} \tag{10}$$

where $u \in [0, 1)$.

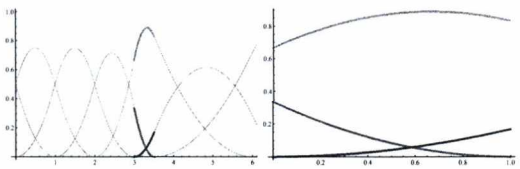


Figure 1: *Quadratic B-spline functions defined on the knot vector $\{-2,-1,0,1,2,3,3.5,6,7,8,9\}$ over the knot span [0,6] (in the left) and transformed to the unit interval from the knot span [3,3.5] using the rewritten recursion formula (10) (in the right).*

3. Transformation matrices

In this section we present how to generate the matrix representation of the B-spline functions of order k defined on an arbitrary knot vector $\{t_1, \ldots t_n\}$. The basis functions are piecewise polynomials on each knot span $[t_j, t_{j+1})$. Over the knot spans, where $j = k, k+1, \ldots, n-k$ the B-splines have k different, non-zero polynomial segments. These segments can be represented by a matrix equation in the not normalized Bernstein basis $\{u^{k-1}, u^{k-2}(1-u), \ldots, u(1-u)^{k-2}, (1-u)^{k-1}\}$ over the unit interval:

$$\begin{pmatrix} N_1^k(t) \\ N_2^k(t) \\ \vdots \\ N_{n-k}^k(t) \end{pmatrix} = \mathbf{C}^k \cdot \begin{pmatrix} u^{k-1} \\ u^{k-2}(1-u) \\ \vdots \\ (1-u)^{k-1} \end{pmatrix}, \quad \begin{matrix} t \in [t_j, t_{j+1}), \\ u \in [0, 1). \end{matrix} \tag{11}$$

Algorithm 1 `BSCoeffMatrix`$(k, \mathbf{v} = \{t_1, \ldots, t_n\}, j)$

```
1:  i := 1
2:  α(A,B;t) := (t−A)/(B−A)        A ≠ B.
3:  C^1[p,1] ← { 1, p = j          {initialization}
                { 0, Otherwise
4:  for i = 2 to k do
5:    Ĉ[p,q] := { C^{i−1}[p,q], 1 ≤ p ≤ n−k; 1 ≤ q < i
                { 0,            Otherwise
6:    for q = 1 to i do
7:      for p = 1 to n−k do
8:        C^i[p,q] := α(t_q, t_{i−1+q}; t_{j+1}) · Ĉ[p,q] +
                      α(t_{i+q}, t_{q+1}; t_{j+1}) · Ĉ[p+1,q] +
                      α(t_q, t_{i−1+q}; t_j) · Ĉ[p,q−1] +
                      α(t_{i+q}, t_{q+1}; t_j) · Ĉ[p+1,q−1]
9:      end for
10:   end for
11: end for
12: return C^k
```

Algorithm 1 shows how to generate the coefficient matrix $\mathbf{C}^k$ of the equation system (11). The procedure is based on the reformulated de Boor-Cox recursion (10). It collects the coefficients of the functions $u^{k-q}(1-u)^q$ for all $q = 1, \ldots k-1$ of the B-spline segment $N_p^k(u)$ to the matrix $\mathbf{C}^k$.

If we represent the B-spline segments from a given knot span $[t_j, t_{j+1})$ in the matrix equation form (11), then it is easy to transform the representation to a matrix representation into the power basis $\{u^{k-1}, u^{k-2}, \ldots, u, 1\}$ and also into the Bernstein basis $\left\{\binom{k-1}{0}u^{k-1}, \binom{k-1}{1}u^{k-2}(1-u), \ldots, \binom{k-1}{k-1}(1-u)^{k-1}\right\}$.

In order to find the transformation matrix of the B-spline segments to the power basis, it is sufficient to find the transformation matrix $\mathbf{P}^k$ from the not normalized Bernstein basis to the power basis for polynomials of degree $k-1$. Then together with the transformation matrix $\mathbf{C}^k$ computed by **Algorithm 1** the transformation matrix can be derived as the multiplication of the two matrices:

$$\begin{pmatrix} N_1^k(t) \\ N_2^k(t) \\ \vdots \\ N_{n-k}^k(t) \end{pmatrix} = \mathbf{C}^k \cdot \mathbf{P}^k \cdot \begin{pmatrix} u^{k-1} \\ u^{k-2} \\ \vdots \\ 1 \end{pmatrix}, \quad \begin{matrix} t \in [t_j, t_{j+1}), \\ u \in [0, 1). \end{matrix} \tag{12}$$

The $\mathbf{P}^k$ matrix is a lower triangular matrix:

$$\mathbf{P}^k[p,q] = \begin{cases} (-1)^{p-q+1} \cdot \binom{p-1}{q-1}, & q \le p, \\ 0, & \text{Otherwise.} \end{cases}$$

This matrix can be easily derived from the following equation according to the Binomial-theorem:

$$u^{k-p}(1-u)^{p-1} = u^{k-p} \cdot \sum_{l=0}^{p-1} \binom{p-1}{l} (-u)^{p-1-l}.$$

The B-spline segments can be transformed to the Bernstein basis with the help of a diagonal matrix $\mathbf{B}^k$, which contains the normalization constants of the basis functions $u^{k-p}(1-u)^p$:

$$\mathbf{B}^k[p,q] = \begin{cases} \dfrac{1}{\binom{k-1}{p-1}}, & p = q, \\ 0, & \text{Otherwise.} \end{cases}$$

Thus the B-spline segments from a given knot span $[t_j, t_{j+1})$ can be represented in the matrix equation form:

$$\begin{pmatrix} N_1^k(t) \\ N_2^k(t) \\ \vdots \\ N_{n-k}^k(t) \end{pmatrix} = \mathbf{C}^k \cdot \mathbf{B}^k \cdot \begin{pmatrix} \binom{k-1}{0} u^{k-1} \\ \binom{k-1}{1} u^{k-2}(1-u) \\ \vdots \\ \binom{k-1}{k-1} (1-u)^{k-1} \end{pmatrix}, \tag{13}$$

where $t \in [t_j, t_{j+1})$ and $u \in [0,1)$.

4. Approximation of B-spline curves using matrix representation

In our solution for approximating two given curves we assume that they are represented by B-spline segments of degree 4. The vector function of the ith segment of such a curve is

$$\mathbf{r}_i(t) = (\mathbf{p}_{i-1}\ \mathbf{p}_i\ \mathbf{p}_{i+1}\ \mathbf{p}_{i+2}\ \mathbf{p}_{i+3}) \cdot \mathbf{M} \cdot \begin{pmatrix} t^4 \\ t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}, 0 \le t \le 1,$$

where $\mathbf{M}$ is the coefficient matrix of the power basis functions and $\mathbf{p}_{i+k}$, $k = -1, \ldots 3$ are the position vectors of the 5 control points determining the curve segment. If the knot vector is uniform periodic, then

$$\mathbf{M} = \frac{1}{24} \begin{pmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -6 & -6 & 6 & 0 \\ -4 & -12 & 12 & 4 & 0 \\ 1 & 11 & 11 & 1 & 0 \end{pmatrix}$$

is constant for each curve segment.

Fourth degree B-splines without multiple knot values and control points are of third degree continuous.

In the first example we assume that two input segments are given by B-spline functions with uniform periodic knot vectors, one by $\mathbf{r}_1(t)$ with control points $\mathbf{p}_{1j}$ and the other by $\mathbf{r}_2(t)$ with control points $\mathbf{p}_{2j}$, $(j = 0, \ldots, 4)$.

The given curve segments can be join continuously or with a possible gap between them.

We generate the resulting B-spline curve with 6 segments $\mathbf{q}_i(t)$, $0 \le t \le 1$, $(i = 1, \ldots, 6)$ determined by 10 control points $\mathbf{b}_j$, $(j = 0, \ldots, 9)$.

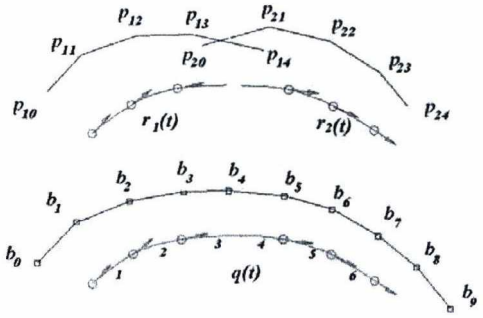


Figure 2: *Control points of two input curves and of the required approximating curve*

In Figure 2 two input curves with a gap and the required approximating curve with their control points are shown. The segmentation of the curves are shown by circles. In the computation of the approximating curve $\mathbf{q}(t)$ the input data are the control points of the given curves $\mathbf{r}_1(t)$ and $\mathbf{r}_2(t)$, and the output data are the 10 control points of the required curve $\mathbf{q}(t)$. We are going to minimize the distances between the corresponding points of the given and the approximating curves. For this purpose each input curve will be decomposed in 3 segments, and all segments will be represented on the parameter interval $t \in [0,1]$. The target function to be minimized is the following

$$\mathrm{TFunc}(b_j) = \sum_{allsegments} \left(\int (\mathbf{r_i} - \mathbf{q})^2\right) +$$

$$+0.3 \cdot \sum_{segments1,2,5,6} \left(\int (\dot{\mathbf{r}}_\mathbf{i} - \dot{\mathbf{q}})^2\right) + 0.1 \cdot \sum_{segments3,4} \left(\int \ddot{\mathbf{q}}^2\right),$$

$j = 0 \ldots 9$, $i = 1, 2$. We have included into the target function also the difference between the first derivatives with the weight factor 0.3 and a smoothing term expressed by the second derivative of the approximating curve with the weight factor 0.1. This third term smoothes the filling part between

the input curves. It can be omitted, if there is no gap. We have computed the integrals numerically, choosing 3 inner points and the end points of each curve segment omitting the end points at the gap. The variables are the unknown control points. The target function is quadratic in the 10 variables, therefore, the minimization leads to a system of linear equations. With the resulting control points the equation of the approximating B-spline curve is determined.

In Figure 3 the control polygons of the two input curves (blue) and the control polygon of the new curve (red) are shown. Figure 4 presents the two input curves (blue) and the resulting curve (red). The approximation error is computed by the integral

$$\sum_{allsegments} (\int (\mathbf{r_i} - \mathbf{q})^2).$$

Its value is 0.0019 in this example computed with uniform knot vectors. As the arc lengths of the two input curves are very different (the first three and the second three arcs, respectively, in Figure 4), we have repeated the approximation with a non-uniform knot vector of "chord length" parametrization. This computation resulted in an error of 6.10^{-6}, which is a significant improvement compared to the first approximation using uniform B-splines. Thus the difference between the input and output curves is not even visible in this case.

This approximating algorithm can be used for merging B-spline surfaces by applying the presented algorithm to the control nets row by row. Figure 5 and 6 show an example, where the given surface patches on a cone surface have been generated by the algorithm presented by Szilvasi et al.[16].

5. Conclusions

We presented a new technique to compute the matrix representation of non-uniform B-splines. Although the recursive generation of the representation matrix was already published, our algorithm is based on a different concept, a reformulation of the de Boor-Cox formula. The structure of this computation method shows the connection between the well-known polynomial basis functions clearly. In the applications the matrix representation of B-spline functions provides effective solutions to interpolation and approximation problems also symbolically, i.e. independently on the numerical values of the input data.

The computations and figures have been made by the symbolic algebraic program package Wolfram Mathematica.

Acknowledgements

The research of the second author was supported in a cooperation with the Technical University Berlin.

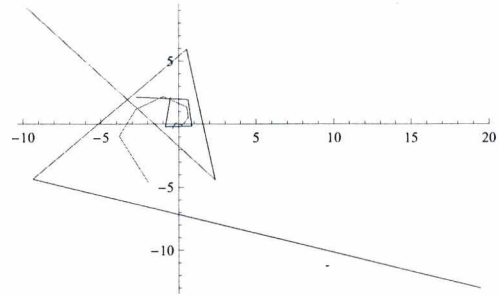


Figure 3: *The control polygons of two input curves (blue) and the control polygon of the new approximating curve (red).*

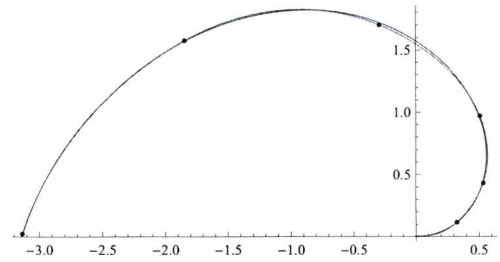


Figure 4: *The two input curves are shown in blue.The resulting curve (red) is presented with the connection points of its B-spline segments.*

References

1. I. J. Schoenberg, Contribution to the Problem of Approximation of Equidistant Data by Analytic Functions *Quart. Appl. Math.* **4**, pp. 45–99, 112–141 (1946).
2. G. Chang, Matrix foundations of Bézier technique *Computer Aided Design* **14**, pp. 345–350 (1982).
3. E. Cohen, R. F. Riesenfeld, General matrix representations for Bezier and B-spline curves *Computers in Industry* **3**, pp. 9-15 (1982).
4. B. K. Choi, W. S. Yoo, C. S. Lee Matrix representation for NURBS curves and surfaces *Computer Aided Design* **22**, pp. 235-240 (1990).
5. H. Grabowski, X. Li, Coefficient Formula and Matrix Representation of nonuniform B-spline functions *Computers in Industry* **3**, pp. 9-15 (1982).
6. L. Liu, G. Wang, Explicit Matrix Representation for NURBS curves and surfaces *Computer Aided Geometric Design* **19**, pp. 409-419 (2002).
7. L. Romani, M. A. Sabin, The Conversion Marix between Uniform B-Spline and Bézier representation *Computer Aided Geometric Design* **21**, pp. 549-560 (2004).

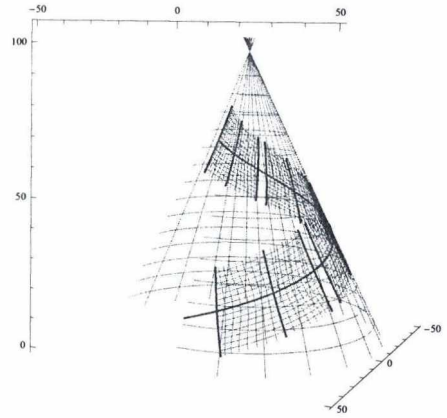


Figure 5: *Four surface patches on the cone with gaps in between.*

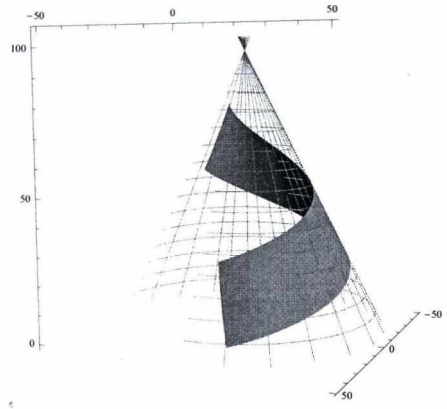


Figure 6: *A surface band computed with merging four separate surface patches (see Fig. 5).*

8. G. Casciola, L. Romani, A Generalized Conversion Matrix between Non-uniform B-Spline and Bézier representations with Applications in CAGD *Multivariate Approximation: Theory and Applications, 24-29 aprile 2003, Cancun, Mexico.* pp. (2004).

9. J. Chen, G.-J. Wang, Approximate merging of B-spline curves and surfaces *Appl. Math. J. Chinese Univ.* **25/4** 429-436 (2010).

10. S.-M. Hu, C.-L. Tai, S.-H. Zhang, An extension algorithm for B-splines by curve unclamping. *Computer-Aided Design* **34** 415-419 2002.

11. H. Pungotra, G. K. Knopf, R. Canas, Merging multiple B-Spline surface patches in a virtual reality Environment. *Computer-Aided Design* **42** 847-859 (2010).

12. C.-L. Tai, S.-M. Hu, Q.-X. Huang, Qi-Xing, Approximate merging of B-Spline curves via knot adjustment and constrained optimization. *Computer-Aided Design* **35** 893-899 (2003).

13. M. Szilvási-Nagy, Shaping and fairing of tubular B-spline surfaces *Computer Aided Geometric Design* **14** 699-706 (1997).

14. M. Szilvási-Nagy, Almost curvature continuous fitting of B-spline surfaces *Journal for Geometry and Graphics* **2** 33-43 (1998).

15. M. Szilvási-Nagy, Closing pipes by extension of B-spline surfaces *KoG* **2** 13-19 (1998).

16. M. Szilvási-Nagy, Sz. Béla, B-spline patches constructed from inner data In *Sixth Hungarian Conference on Computer Graphics and Geometry*, Budapest **2** 30-33 (2012)

17. M. Szilvási-Nagy, Sz. Béla, Stitching B-spline curves symbolically *KoG*, **17** 3-8 (2013).

C^2 Geometric Hermite Spline Surfaces

Gábor Valasek[1] and János Vida[1]

[1] Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary

Abstract

Geometric Hermite interpolation of curves is a generalization of Hermite interpolation, which reconstructs parametrization independent quantities, such as position, tangent directions, curvatures. This paper discusses an extension of geometric Hermite interpolation to surfaces, where quadrilateral surface patches are created such that the parametric corners reconstruct prescribed position, surface normal, principal curvature and principal direction data. An algorithm for creating bi-quintic integral Bézier solutions is presented, the continuous connection of these bi-quintic patches is investigated, and implementation details are presented.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

Geometric Hermite (GH) interpolation is a generalization of Hermite interpolation. Its first application in context of curve construction is due to de Boor et. al. [1], who proposed the construction of planar G^2 spline curves by interpolating prescribed position, tangent direction, and curvature values at each knot. Each segment of the spline is a cubic Bézier curve, which reconstructs the above parametrization independent quantities at its endpoints. This piecewise cubic interpolant has an approximation order of six, however, its existence is not always guaranteed. A throughout geometric characterization of the existence of planar and spatial geometric Hermite curves was published by Schaback in [6].

Geometric Hermite interpolation for second order data, i.e. position, tangent direction, and curvature values, has been extensively studied for a wide range of curves, including integral and rational Bézier curves [6], Pythagorean Hodograph curves [5] and spirals [9].

Interpolating position and normal (PN) data with surfaces can be considered as a first order geometric Hermite interpolation problem. PN interpolation has been studied extensively [8], [2].

This paper summarizes a second order generalization of geometric Hermite interpolation to quadrilateral surfaces in section 2, where prescribed position, surface normal, principal curvature values, and principal directions are to be reconstructed at each parametric corner.

The main contribution of this paper is Section 3, where bi-quintic integral Bézier patches are used to solve the four-corner GH interpolation problem, the degrees of freedom directly not constrained by data are set, and a C^2 spline surface is constructed that consists of four-corner GH interpolant patches.

In section 4, a GPU implementation of these surfaces is discussed. Section 5 summarizes our results.

2. Geometric Hermite surface interpolation

Let us briefly review the conditions of second order geometric Hermite surface interpolation. Proof of theorems and characterization of the existence of interpolants both in the quadrilateral and triangular cases is found in our paper [7].

2.1. One-point reconstruction problem

Let there be given a G^2 *base point data* tuple

$$\mathbf{D} = (\mathbf{p}, \mathbf{n}, \mathbf{t}_1, \mathbf{t}_2, \kappa_1, \kappa_2),$$

where $\mathbf{p} \in \mathbb{E}^3$ denotes a point in the Euclidean space, $\mathbf{n} \in \mathbb{R}^3$ is a surface normal, $\kappa_1, \kappa_2 \in \mathbb{R}$ are principal curvature values and $\mathbf{t}_1, \mathbf{t}_2 \in \mathbb{R}^3$ are corresponding principal directions, $|\mathbf{n}| = |\mathbf{t}_1| = |\mathbf{t}_2| = 1$, and κ_1 is the minimum and κ_2 is the maximum normal curvature. Without loss of generality, the

orthonormal basis $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{n})$ is assumed to be right-handed. We also assume that the normals are oriented consistently.

Let $\mathbf{s} : \mathbb{R}^2 \to \mathbb{E}^3$ be a regular parametric surface, and consider a point in its domain, where $\mathbf{D}$ is to be reconstructed. At this point, let the coordinates of the $\mathbf{s}_u, \mathbf{s}_v$ partial derivatives, with respect to the $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{n})$ orthonormal basis, be $(x_u, y_u, z_u) \in \mathbb{R}^3$, $(x_v, y_v, z_v) \in \mathbb{R}^3$, and let $(x_{uu}, y_{uu}, z_{uu}) \in \mathbb{R}^3$, $(x_{uv}, y_{uv}, z_{uv}) \in \mathbb{R}^3$, $(x_{vv}, y_{vv}, z_v) \in \mathbb{R}^3$ denote the coordinates of the $\mathbf{s}_{uu}, \mathbf{s}_{uv}, \mathbf{s}_{vv}$ partial derivatives in the same basis. Using the above notation, the following can be stated:

Proposition 1 The regular parametric surface $\mathbf{s}(u,v)$ reconstructs the prescribed G^2 base point data of $\mathbf{D}$ at a point $(u_0, v_0) \in dom(\mathbf{s})$ iff the following all hold:

$$\mathbf{s}(u_0, v_0) = \mathbf{p} \tag{1}$$
$$z_u = 0 \tag{2}$$
$$z_v = 0 \tag{3}$$
$$0 < x_u y_v - x_v y_u \tag{4}$$
$$z_{uu} = \kappa_1 x_u^2 + \kappa_2 y_u^2 \tag{5}$$
$$z_{uv} = \kappa_1 x_u x_v + \kappa_2 y_u y_v \tag{6}$$
$$z_{vv} = \kappa_1 x_v^2 + \kappa_2 y_v^2 \tag{7}$$

Please note that we can always find partial derivative vectors which satisfy (2)-(7). The first partial derivatives should lie in the tangent plane of $\mathbf{D}$, and the angle between them should be less than 180 degrees, while the second order partial derivatives can be freely translated parallel to the tangent plane $(\mathbf{t}_1, \mathbf{t}_2)$, because curvature reconstruction only imposes restrictions on their coordinates along the surface normal.

2.2. Four-corner reconstruction problem

Let there be given four

$$\mathbf{D}^{(ij)} = (\mathbf{p}^{(ij)}, \mathbf{n}^{(ij)}, \mathbf{t}_1^{(ij)}, \mathbf{t}_2^{(ij)}, \kappa_1^{(ij)}, \kappa_2^{(ij)}),\ i,j = 0,1$$

G^2 base-point data tuples and let us find a

$$\mathbf{b}(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n} \mathbf{b}_{ij} B_i^n(u) B_j^m(v),$$

degree (n,m) quadrilateral Bézier surface, $n,m \geq 2$, $(u,v) \in [0,1]^2, \mathbf{b}_{ij} \in \mathbb{E}^3, i = 0,..,n,\ j = 0,..,m,$, that reconstructs the $\mathbf{D}^{(ij)}$ base point data at its parametric corners $(u,v) = (i,j), i,j = 0,1$.

First, let us consider the $(u,v) = (0,0)$ corner and examine how control points are constrained by the reconstruction of second order geometric base point quantities!

Interpolation of position requires

$$\mathbf{b}_{00} = \mathbf{p}^{(00)}.$$

The normal at the corner of the Bézier patch is computed as

$$\mathbf{n}(0,0) = \frac{\Delta^{10}\mathbf{b}_{00} \times \Delta^{01}\mathbf{b}_{00}}{|\Delta^{10}\mathbf{b}_{00} \times \Delta^{01}\mathbf{b}_{00}|}.$$

The reconstruction of surface normal is the equation $\mathbf{n}(0,0) = \mathbf{n}$, which imposes restrictions on $\mathbf{b}_{00}, \mathbf{b}_{10}$ and $\mathbf{b}_{01}$.

Principal curvature value and direction reconstruction requires the first and second fundamental forms, since for the normal section curvature along the tangent plane vector $du \cdot \mathbf{s}_u + dv \cdot \mathbf{s}_v$ we have [3]

$$\kappa(du, dv) = \frac{II}{I} = \frac{L du^2 + 2M du dv + N dv^2}{E du^2 + 2F du dv + G dv^2} \tag{8}$$

and at $(u,v) = (0,0)$ we have

$$E(u,v) = \langle \mathbf{b}_u(0,0), \mathbf{b}_u(0,0) \rangle = n^2 \langle \Delta^{10}\mathbf{b}_{00}, \Delta^{10}\mathbf{b}_{00} \rangle$$
$$F(u,v) = \langle \mathbf{b}_u(0,0), \mathbf{b}_v(0,0) \rangle = nm \langle \Delta^{10}\mathbf{b}_{00}, \Delta^{01}\mathbf{b}_{00} \rangle$$
$$G(u,v) = \langle \mathbf{b}_v(0,0), \mathbf{b}_v(0,0) \rangle = m^2 \langle \Delta^{01}\mathbf{b}_{00}, \Delta^{01}\mathbf{b}_{00} \rangle$$
$$L(u,v) = \langle \mathbf{b}_{uu}(0,0), \mathbf{n}(0,0) \rangle = n(n-1) \langle \Delta^{20}\mathbf{b}_{00}, \mathbf{n}(0,0) \rangle$$
$$M(u,v) = \langle \mathbf{b}_{uv}(0,0), \mathbf{n}(0,0) \rangle = nm \langle \Delta^{11}\mathbf{b}_{00}, \mathbf{n}(0,0) \rangle$$
$$N(u,v) = \langle \mathbf{b}_{vv}(0,0), \mathbf{n}(0,0) \rangle = m(m-1) \langle \Delta^{02}\mathbf{b}_{00}, \mathbf{n}(0,0) \rangle$$

Thus, the control points, required for a second order geometric Hermite reconstruction at $(u,v) = (0,0)$, are $\mathbf{b}_{00}, \mathbf{b}_{10}, \mathbf{b}_{01}, \mathbf{b}_{20}, \mathbf{b}_{11}, \mathbf{b}_{02}$, shown in figure 1. The reconstruction poses 8 scalar constraints on these six control points.

To make these constraints more explicit, let (x_{ij}, y_{ij}, z_{ij}) denote the coordinates of $\mathbf{b}_{ij}$ in the $(\mathbf{p}; \mathbf{t}_1, \mathbf{t}_2, \mathbf{n})$ basis, i.e. in the right-handed orthonormal basis with origin $\mathbf{p}$ and axes coinciding with the principal directions and surface normal.

If $\mathbf{b}_{10}$ and $\mathbf{b}_{01}$ are chosen such that

$$z_{10} = z_{01} = 0$$
$$x_{10} y_{01} - x_{01} y_{10} > 0$$

then the only restrictions on the $\mathbf{b}_{11}, \mathbf{b}_{20}, \mathbf{b}_{02}$ control points are

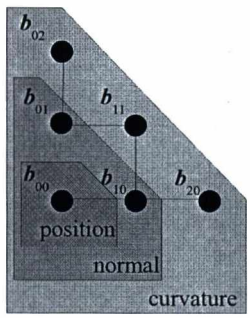


Figure 1: *The control points of a Bézier surface that determine the position, tangent plane, and principal curvature relations at* $(u,v) = (0,0)$

$$z_{20} = \frac{n}{n-1}\left(\kappa_1^{(00)} x_{10}^2 + \kappa_2^{(00)} y_{10}^2\right) \tag{9}$$

$$z_{11} = \kappa_1^{(00)} x_{10} \cdot x_{01} + \kappa_1^{(00)} y_{10} \cdot y_{01} \tag{10}$$

$$z_{02} = \frac{m}{m-1}\left(\kappa_1^{(00)} x_{01}^2 + \kappa_2^{(00)} y_{01}^2\right) \tag{11}$$

that is, they should lie on certain planes. Let

$$\mathbf{M}^{(ij)}(m) = \{\mathbf{q} \in \mathbb{E}^3 | \langle \mathbf{q} - \mathbf{p}^{(ij)}, \mathbf{n}^{(ij)} \rangle = m\}$$

denote the lifted tangent planes at $\mathbf{D}^{(ij)}$ along the surface normal. Using (9)-(11), the constraints on these control points are written as

$$\mathbf{b}_{20} \in \mathbf{M}(z_{20}) \tag{12}$$

$$\mathbf{b}_{11} \in \mathbf{M}(z_{11}) \tag{13}$$

$$\mathbf{b}_{02} \in \mathbf{M}(z_{02}) \tag{14}$$

Now, let us consider all four parametric corners. The control points required for the reconstruction of G^2 base point data are

$$\mathbf{b}_{i\cdot n+(-1)^i k, j\cdot m+(-1)^j l} \in \mathbb{E}^3 \tag{15}$$

$i, j = 0, 1$, and $k + l \leq 2$.

Their coordinates are formulated similarly as in the case of the $(u,v) = (0,0)$ corner, but care must be taken, since the computation of the mixed partial derivatives' $z_{11}^{(ij)}$ coordinate differs in the corners: in the case of $\mathbf{D}^{(10)}$ and $\mathbf{D}^{(01)}$ equation (10) must be multiplied by -1.

Considering the control net of a bi-quintic Bézier patch, it can be easily seen that the following holds:

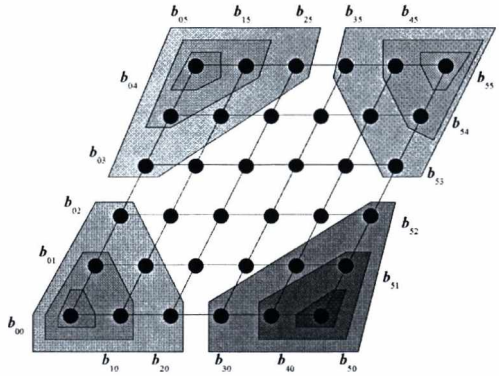


Figure 2: *Control net of the bi-quintic Bézier patch. The red, blue, green, and azure regions correspond to the control points that are necessary for the reconstruction of* G^2 *base point data* $\mathbf{D}^{(00)}$, $\mathbf{D}^{(10)}$, $\mathbf{D}^{(11)}$, *and* $\mathbf{D}^{(01)}$.

Proposition 2 There is always a quadrilateral bi-quintic integral Bézier surface solution to the four corner second order geometric Hermite interpolation problem.

3. C^2 Geometric Hermite spline surfaces

Now, let us consider the problem of constructing C^2 spline surfaces using bi-quintic four-corner second order GH interpolants, provided the base points constitute of a regular rectangular grid.

3.1. Control net based on paraboloids

First, the remaining degrees of freedom of the control net need to be set. In order to do that, let us assign the *base paraboloids*

$$\mathbf{p}^{(ij)}(u,v) = \mathbf{p}^{(ij)} + [\mathbf{t}_1^{(ij)}, \mathbf{t}_2^{(ij)}, \mathbf{n}^{(ij)}] \begin{bmatrix} u \\ v \\ \frac{\kappa_1^{(ij)}}{2} u^2 + \frac{\kappa_2^{(ij)}}{2} v^2 \end{bmatrix} \tag{16}$$

to each $\mathbf{D}^{(ij)}, i, j = 0, 1$.

It is easily seen that the unit surface normal of a paraboloid in (16) at $(u,v) = (0,0)$ is $\mathbf{n}^{(ij)}$ and its principal curvatures and principal directions are $\kappa_1^{(ij)}, \kappa_2^{(ij)}, \mathbf{t}_1^{(ij)}, \mathbf{t}_2^{(ij)}$.

Let us consider the corner $(i,j) = (0,0)$ and let us drop the upper indices, i.e. let $\mathbf{D} = \mathbf{D}^{(00)}, \mathbf{n}^{(00)} = \mathbf{n}, \mathbf{p}(u,v) = \mathbf{p}^{(00)}(u,v)$ and so on.

Let $\mathbf{u} = (a_x, a_y), \mathbf{b} = (b_x, b_y)$ be two points in the domain of the paraboloid $\mathbf{p}(u,v)$, and let $\mathbf{g}_{ij}, i, j = 0, 1, 2$ denote the control points of the bi-quadratic Bézier patch that covers the part of the paraboloid that is above $(0,0) - (\mathbf{a}, \mathbf{b})$, i.e. let

$$\mathbf{g}(u,v) = \mathbf{p}(u \cdot a_x + v \cdot b_x, u \cdot a_y + v \cdot b_y),$$

where $u,v \in [0,1]$ and $\mathbf{g}(u,v) = \sum_{j=0}^{2} \sum_{i=0}^{2} \mathbf{g}_{ij} B_i^2(u) B_j^2(v)$.

To specify the position of the $\mathbf{b}_{ij}, i,j = 0,1,2$ control points, let us elevate the degree of $\mathbf{g}(u,v)$ to five in both the u and v directions and assign the 3×3 control points of the resulting control net around $(u,v) = (0,0)$ to $\mathbf{b}_{ij}, i,j = 0,1,2$.

These 9 control points are expressed explicitly in terms of the power basis coefficients of the given paraboloid portion as follows: let the power basis coefficients be

$$\mathbf{a}_{00} = \mathbf{a}_{21} = \mathbf{a}_{12} = \mathbf{a}_{22} = \mathbf{0},$$

$$\mathbf{a}_{10} = \begin{bmatrix} a_x \\ a_y \\ 0 \end{bmatrix}, \ \mathbf{a}_{01} = \begin{bmatrix} b_x \\ b_y \\ 0 \end{bmatrix},$$

$$\mathbf{a}_{20} = \begin{bmatrix} 0 \\ 0 \\ \frac{\kappa_1}{2} a_x^2 + \frac{\kappa_2}{2} a_y^2 \end{bmatrix}, \ \mathbf{a}_{02} = \begin{bmatrix} 0 \\ 0 \\ \frac{\kappa_1}{2} b_x^2 + \frac{\kappa_2}{2} b_y^2 \end{bmatrix},$$

$$\mathbf{a}_{11} = \begin{bmatrix} 0 \\ 0 \\ \kappa_1 a_x b_x + \kappa_2 a_y b_y \end{bmatrix}$$

Then the coordinates of the control points in the $(\mathbf{p}; \mathbf{t}_1, \mathbf{t}_2, \mathbf{n})$ basis are computed as

$$\mathbf{b}_{00} = \mathbf{a}_{00} \tag{17}$$

$$\mathbf{b}_{10} = \frac{\mathbf{a}_{10}}{5} + \mathbf{a}_{00} \tag{18}$$

$$\mathbf{b}_{20} = \frac{\mathbf{a}_{20} + 4\mathbf{a}_{10} + 10\mathbf{a}_{00}}{10} \tag{19}$$

$$\mathbf{b}_{01} = \frac{\mathbf{a}_{01} + 5\mathbf{a}_{00}}{5} \tag{20}$$

$$\mathbf{b}_{11} = \frac{\mathbf{a}_{11} + 5\mathbf{a}_{10} + 5\mathbf{a}_{01} + 25\mathbf{a}_{00}}{25} \tag{21}$$

$$\mathbf{b}_{21} = \frac{5\mathbf{a}_{20} + 4\mathbf{a}_{11} + 20\mathbf{a}_{10} + 10\mathbf{a}_{01} + 50\mathbf{a}_{00}}{50} \tag{22}$$

$$\mathbf{b}_{02} = \frac{\mathbf{a}_{02} + 4\mathbf{a}_{01} + 10\mathbf{a}_{00}}{10} \tag{23}$$

$$\mathbf{b}_{12} = \frac{4\mathbf{a}_{11} + 10\mathbf{a}_{10} + 5\mathbf{a}_{02} + 20\mathbf{a}_{01} + 50\mathbf{a}_{00}}{50} \tag{24}$$

$$\mathbf{b}_{22} = \frac{5\mathbf{a}_{20} + 8\mathbf{a}_{11} + 20\mathbf{a}_{10} + 5\mathbf{a}_{02} + 20\mathbf{a}_{01} + 50\mathbf{a}_{00}}{50} \tag{25}$$

If the angle between $a_x\mathbf{t}_1 + a_y\mathbf{t}_2$ and $b_x\mathbf{t}_1 + b_y\mathbf{t}_2$ is smaller than 180 degrees, direct evaluation of the differences prove that the choice of control points satisfies the conditions of proposition 1, i.e at $(u,v) = (0,0)$ base point quantities of $\mathbf{D}^{(00)}$ are reconstructed.

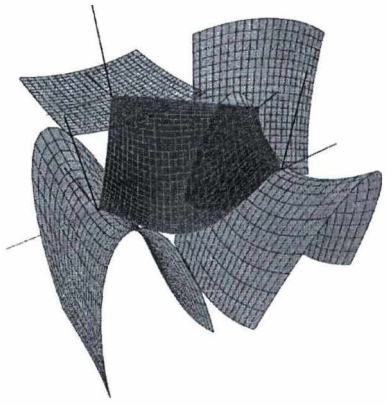


Figure 3: *The bi-quintic Bézier patch (opaque with green isolines) interpolates the prescribed four-corner base point data, illustrated by purple transparent paraboloids, the red and green lines at the corners are principal directions, the blue line segments represent surface normals.*

After handling the remaining three corners as above, 4×9 control points are generated, which together form the control net of a bi-quintic Bézier patch that satisfies the conditions of the four-corner geometric Hermite reconstruction of $\mathbf{D}^{(ij)}, i,j = 0,1$. Figure 3 shows such a bi-quintic patch.

The vectors $a_x^{(ij)}\mathbf{t}_1^{(ij)} + a_y^{(ij)}\mathbf{t}_2^{(ij)}$ and $b_x^{(ij)}\mathbf{t}_1^{(ij)} + b_y^{(ij)}\mathbf{t}_2^{(ij)}$ will be the tangent vectors of the boundary curves in the u and v parametric directions at $(u,v) = (i,j)$, $i,j = 0,1$. Let us refer to these vectors as *base tangents*.

In order to handle all the four corners correctly, we use the following power basis coefficients:

$$\mathbf{a}_{00}^{(ij)} = \mathbf{a}_{21}^{(ij)} = \mathbf{a}_{12}^{(ij)} = \mathbf{a}_{22}^{(ij)} = \mathbf{0}, \tag{26}$$

$$\mathbf{a}_{10}^{(ij)} = \begin{bmatrix} (-1)^i a_x \\ (-1)^i a_y \\ 0 \end{bmatrix}, \ \mathbf{a}_{01}^{(ij)} = \begin{bmatrix} (-1)^j b_x \\ (-1)^j b_y \\ 0 \end{bmatrix}, \tag{27}$$

$$\mathbf{a}_{20}^{(ij)} = \begin{bmatrix} 0 \\ 0 \\ \frac{\kappa_1}{2} a_x^2 + \frac{\kappa_2}{2} a_y^2 \end{bmatrix}, \ \mathbf{a}_{02}^{(ij)} = \begin{bmatrix} 0 \\ 0 \\ \frac{\kappa_1}{2} b_x^2 + \frac{\kappa_2}{2} b_y^2 \end{bmatrix}, \tag{28}$$

$$\mathbf{a}_{11}^{(ij)} = \begin{bmatrix} 0 \\ 0 \\ (-1)^{i+j}(\kappa_1 a_x b_x + \kappa_2 a_y b_y) \end{bmatrix} \tag{29}$$

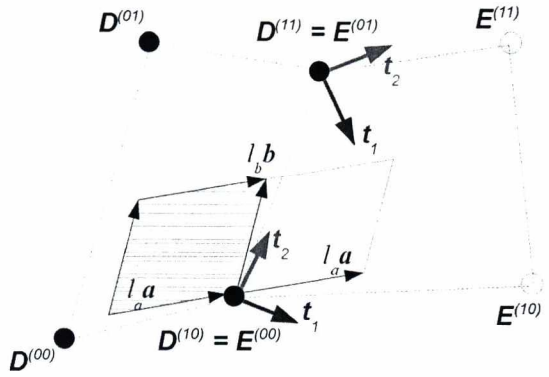


Figure 4: *Connecting second order GH interpolants*

3.2. Continuous connection along boundaries

Let us suppose our data are given in a uniform rectangular grid. C^2 continuity at the corners of Bézier patches follows from the construction. Now, let us consider $\mathbf{b}(u,v)$ and $\mathbf{c}(u,v)$, two bi-quadratic four-corner GH interpolants of base-point data tuples $\mathbf{D}^{(ij)}$ and $\mathbf{E}^{(ij)}, i,j=0,1$, with control points $\mathbf{b}_{ij}, \mathbf{c}_{ij}, i,j=0,..,5$ respectively. See figure 4.

Let $\mathbf{D}^{(1j)} = \mathbf{E}^{(0j)}, j=0,1$ and let us examine how $\mathbf{b}(u,v)$ and $\mathbf{c}(u,v)$ can be connected with second order parametric continuity along the u parametric direction.

C^2 connection along the $\mathbf{b}(1,v) = \mathbf{c}(0,v), v \in [0,1]$ boundary requires that [4]

$$\Delta^{10}\mathbf{b}_{4j} = \Delta^{10}\mathbf{c}_{0j} \tag{30}$$

$$\Delta^{20}\mathbf{b}_{3j} = \Delta^{20}\mathbf{c}_{0j} \tag{31}$$

hold for $j=0,..,5$. This allows us to investigate the continuity condition independently on the two control net portions around the two base point data tuples.

Let us consider the case of $\mathbf{D}^{(10)} = \mathbf{E}^{(00)}$. From (17)-(25) it follows that $\mathbf{c}_{i,j}, i,j=0,1,2$ depend on the selection of the base tangent vectors $\mathbf{a}^{(00)}, \mathbf{b}^{(00)}$.

Similarly, $\mathbf{b}_{3+i,j}, i,j=0,1,2$ depend on the base tangent vectors $\mathbf{c}^{(10)}, \mathbf{d}^{(10)}$, that is, the tangents vectors used for the construction of the $\mathbf{b}_{ij}$ control net.

Since both control net portions share the same base paraboloid, assigned to $\mathbf{D}^{(10)} = \mathbf{E}^{(00)}$, and taking into account the appropriate handling of base tangent directions around this corner, conditions (30)-(31) can be satisfied by using the same base tangent vectors for the u and v parametric directions, i.e. if $\mathbf{a}^{(00)} = \mathbf{c}^{(10)} = \mathbf{a}, \mathbf{b}^{(00)} = \mathbf{d}^{(10)} = \mathbf{b}$. See figure 4.

The control points around the corner $\mathbf{D}^{(11)} = \mathbf{E}^{(01)}$ are handled analogously. Continuity along the v parametric directions is derived similarly.

4. Implementation

We used GLSL in the GPU implementation of the C^2 four-corner GH splines, utilizing the programmable tessellation stages introduced in OpenGL 4.

The GH surfaces are drawn as indexed patch primitives, the vertices of the patches correspond to $\mathbf{D}$ base point data tuples. Each vertex stores 3 vertex attributes:

Attribute index	Type	Semantics
0	`vec3`	$\mathbf{p} \in \mathbf{D}$
1	`vec4`	$\begin{bmatrix}\mathbf{t}_1 \\ \kappa_1\end{bmatrix}$
2	`vec4`	$\begin{bmatrix}\mathbf{t}_2 \\ \kappa_2\end{bmatrix}$

The 4 dimensional vectors of attribute indices 1 and 2 are composit values, the first three coordinates correspond to one of the principal directions, the fourth to the principal curvature value in that direction.

Because continuity between the GH patches depend on the base tangent vectors, they have to be available for the bi-quintic patch construction algorithm.

One way to do that, is to let the user set these vectors for each vertex of the GH patch. This means that an additional vertex attribute location needs to be attached to the vertices:

Attribute index	Type	Semantics
3	`vec4`	$\begin{bmatrix}\mathbf{a} \\ \mathbf{b}\end{bmatrix}$

The base tangent vectors have an intuitive geometric interpretation: while the direction of $\mathbf{a}$ determines the tangent direction of the boundary curve along $v=0$, the length of $\mathbf{a}$ can be considered as the weight of the given control point, i.e. how much the base paraboloid of the base point affects the interpolant GH patch. Figure 5 illustrates the effect of the base vector lengths.

Another way to compute proper base tangents vectors is to provide the neighborhood of each GH patch. If we impose a regularity restriction on the base-point control net, this can be done the way geometry shaders store adjacency information for primitives, i.e. by attaching the indices of the 1-edge neighborhood of the patch to the primitive, increasing the number of indices to 12 per GH patch. See figure 6 on the layout of indices.

If the neighborhood is known, we need to compute base tangent vectors $\mathbf{a}^{(i)}, \mathbf{b}^{(i)}$, $i=0,..,3$. The following simple Catmull-Rom-like formula is used:

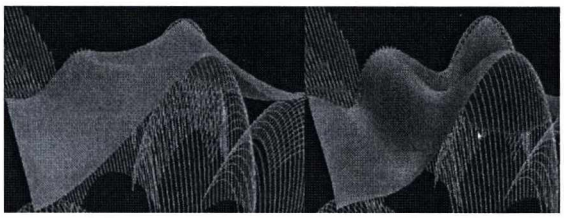


Figure 5: *Connection of two GH bi-quintic interpolants along their common boundary. The length of base tangent vectors are twice as long on the image on the right compared to the ones on the left.*

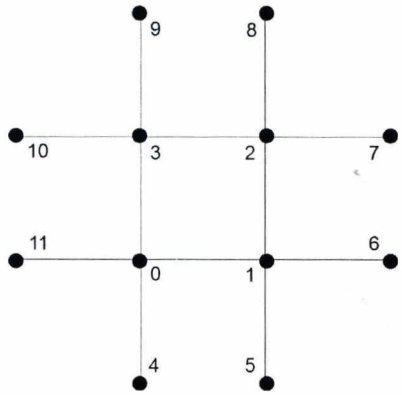


Figure 6: *The patch vertex index layout. The bi-quintic Bézier surface will be constructed from base point data stored in vertices 0,1,2,3. Base tangent vectors are computed utilizing the neighboring vertices, indices 4-11.*

$$\mathbf{a}^{(0)} = \frac{c}{2}\begin{bmatrix}\langle \mathbf{p}^{(1)} - \mathbf{p}^{(11)}, \mathbf{t}_1^{(0)} \rangle \\ \langle \mathbf{p}^{(1)} - \mathbf{p}^{(11)}, \mathbf{t}_2^{(0)} \rangle\end{bmatrix}$$

$$\mathbf{b}^{(0)} = \frac{c}{2}\begin{bmatrix}\langle \mathbf{p}^{(3)} - \mathbf{p}^{(4)}, \mathbf{t}_1^{(0)} \rangle \\ \langle \mathbf{p}^{(3)} - \mathbf{p}^{(4)}, \mathbf{t}_2^{(0)} \rangle\end{bmatrix}$$

where $c \in \mathbb{R}$ is a global constant. The case of base tangent angles greater than 180 degrees has to be handled too, for example by a fallback to principal directions.

Based on these data, the Tessellation Control Shader (TCS) computes the control points of the bi-quintic interpolant. The four base points at the corners of the patch are transformed to world space and passed through to the Tessellation Evaluation Shader (TES). Each invocation of the TCS computes the 3×3 control points associated with the output base point.

The TES evaluates the bi-quintic Bézier patch at the parameter values created by the Tesselator Unit. The outer and inner tessellation levels are set in the TCS.

5. Conclusions

This paper presented a second order geometric Hermite patch construction algorithm, and its application in the creation of C^2 spline surfaces. The conditions of parametrically continuous connections were derived.

A GPU implementation of the algorithm was also presented, with user-defined, and automatically computed base tangent vectors.

Future work includes the connection of non-regular base point control net patches with parametric continuity, as well as the construction of continuous quintic triangular spline surfaces.

Acknowledgements

Our reserch was partially supported by grant number EITKIC_12-1-2012-0001.

References

1. C. de Boor, K. Höllig, M. Sabin: *High accuracy geometric Hermite interpolation*, Computer Aided Geometric Design 1987;4(4):269-78.
2. M. Boschiroli, C. Fünfzig, L. Romani, G. Albrecht, *G^1 rational blend interpolatory schemes: A comparative study*, Graphical Models, Volume 74 Issue 1, January, 2012, Pages 29-49
3. do Carmo: *Differential Geometry of Curves and Surfaces*, Pearson, 1st edition, 1976
4. G. Farin: *Curves and surfaces for CAGD: a practical guide*, 5th edition, ISBN 1-55860-737-4, Morgan Kaufmann Publishers Inc. 2002
5. G. Albrecht, R. T. Farouki, *Construction of C^2 Pythagorean-hodograph interpolating splines by the homotopy method*, Adv. Comput. Math. 5 (1996), no. 4, 417-442. MR MR1414289 (97k:65033)
6. R. Schaback: *Optimal Geometric Hermite Interpolation of Curves*, Mathematical Methods for Curves and Surfaces II, 1998;1-12
7. G. Valasek, J. Vida *Second Order Geometric Hermite Surface Interpolation*, The Mathematics of Surface XIV, ISBN 978-0-905-091-30-3, Pages 277-308, IMA, Edited by Robert J. Cripps, G. Mullineux and M. A. Sabin
8. A. Vlachos, J. Peters, C. Boyd, J. L. Mitchell, *Curved PN Triangles*, I3D '01 Proceedings of the 2001 symposium on Interactive 3D graphics Pages 159-166, ISBN:1-58113-292-1
9. D.J. Walton, D.S. Meek: *A generalisation of the Pythagorean hodograph quintic spiral*, Journal of Computational and Applied Mathematics, 2004;172(2):271-287

Seventh Hungarian Conference on Computer Graphics and Geometry, Budapest, 2014

Multi-sided Surfaces with Curvature Continuity

Péter Salvi, Tamás Várady

Budapest University of Technology and Economics

Abstract

The basic idea of curve network-based design is to construct a collection of smoothly connected surface patches that interpolate boundary constraints extracted solely from the curve network. While the majority of applications demands only tangent plane (G^1) continuity between the adjacent patches, there are applications where curvature continuous connections (G^2) are required. Examples include handling special curve network configurations with supplemented internal edges, "master-slave" curvature constraints and general topology surface approximations over meshes. The first step of the surface generation process is the construction of interpolant surfaces that enforce suitable cross-derivatives for transfinite surface patches; these interpolants are often called ribbons. *For G^2 interpolation we extend Gregory's multi-sided surface scheme, and focus on creating and combining special* parabolic *ribbons. We discuss the basic patch construction including the blending functions and a special sweepline parameterization. A proof of G^2 continuity is given in the Appendix. The application of curvature continuous multi-sided patches is demonstrated by a few simple examples.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling — curvenet-based design, transfinite surfaces, Gregory patches, G^2 continuity

1. Introduction

In curve network-based design, surface models are directly defined by a collection of freeform curves, arranged into a single 3D network with general topology. Curves may come from (i) sketch input, (ii) feature curves extracted from orthogonal views, (iii) curves traced on triangular meshes or (iv) direct 3D editing. Once the curves are defined, all the surfaces are generated automatically. This calls for a representation based on geometric information extracted solely from the boundaries. Transfinite surface interpolation is a natural choice, as it does not require a grid of control points to define the interior shape, and all n boundaries are handled uniformly, unlike in the case of trimmed quadrilateral surfaces. The ability to interactively edit prescribed boundaries and cross-derivatives is also an advantage in contrast to recursive subdivision schemes.

The first step of surface generation is to compute cross-directional data, such as common tangent planes and, when needed, curvatures that will be shared by the adjacent patches. Then interpolant surfaces or *ribbons* are generated, that carry first or second-degree cross-derivative constraints to be eventually interpolated by the transfinite surfaces.

The majority of multi-sided transfinite surfaces are defined over convex domains, combining only linear ribbon surfaces and enabling G^1 continuity between the adjacent patches. At the same time, there are several practical design situations, where this approach is not sufficient, and higher degree continuity is required.

(i) It often occurs that additional curves need to be inserted into the curve network to make it suitable for applying convex methods. The supplemented curves must be compatible with the already defined ribbons, and it is particularly important to produce seamless transitions along these curves. A typical example is when two curves span a concave angle at a common vertex and then a composite patch — with convex domains — is created. Another example is to generate surfaces by interpolating two disjoint loops with prescribed slopes (see Figure 7 later in Section 5).

(ii) Another interesting situation is when a designer wants to retain one of the surfaces (the *master*), but also wants to prescribe a curvature continuous connection for the adjacent patch (the *slave*), see Figure 8.

(iii) A third example is when we have a general topol-

ogy curve network defined over a mesh and want to obtain a good approximation of the interior data points (see Figure 9). Clearly, if we extract not only the normal vectors, but also curvature information from the underlying mesh, we can obtain a more accurate approximation.

In the above cases, it is possible to ensure G^2 continuity by using parabolic ribbons. In Section 2 we give an overview of related publications. In Section 3 various methods to compute linear and parabolic interpolants will be discussed, while in Section 4 we introduce an extended Gregory patch formulation with a new sweepline parameterization that makes it possible to combine parabolic ribbons and yield curvature continuous surfaces. A few examples will illustrate the surface scheme in Section 5.

2. Previous Work

Most papers concerning transfinite surface interpolation assume that either the ribbons, or at least the cross-derivatives are given, and very little is written about how this information can be extracted from a curve network. The reason for this apparent lack of interest may be that at the time when transfinite surface interpolation appeared with the Coons[2] and Gregory[1] patches, its main application was hole filling and vertex blending, where additional cross-directional data were readily available. This trend lived on, even after the advent of curve network-based ideas, such as the Minimum Variation Surfaces[9].

There is a constant interest in multi-sided patches with G^2 continuity, the first efforts dating back 25 years[3]. In particular, an extension of the Gregory patch similar to the one described here was published by Hall and Mullineux[4].

Special treatment of difficult configurations is also a recurring theme in the literature; most approaches use concave domains[7, 8, 6] to handle holes and extreme spatial structures, though we have very limited information concerning the practical design potential and the quality of these shapes.

3. Computing Ribbons

In the course of curve network-based design, users create and/or edit the boundary curves, while surfaces are generated in an automatic manner. This means that cross-derivatives and curvatures are also derived solely from the curve network. Here we will deal only with generating smooth connections, though in a real system the user may want to create sharp edges, as well.

All continuity constraints are accomplished through defining "proper" ribbons, i.e., once we set the adjacent ribbons G^1 or G^2 continuous, the respective transfinite surfaces will inherit this property.

Two adjacent ribbons are G^1 continuous if they share the same sweep of normal vectors (called the *normal fence*) along the common boundary. Figure 1 shows an example, where the fence is rendered as a series of yellow lines. The necessary condition for G^1 continuity is that the first cross-derivatives on both sides are perpendicular to the fence.

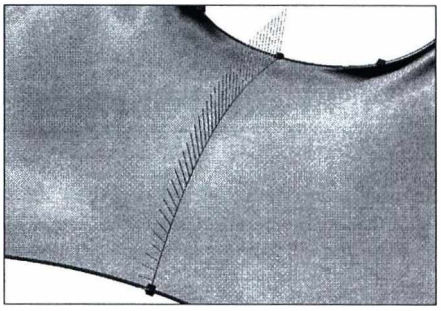


Figure 1: Normal fence and mean curvature map

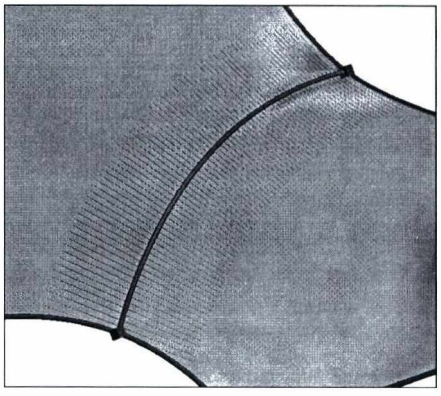


Figure 2: Normal curvature arcs and Gauss curvature map

When we require G^2 continuity between two surfaces, we can make use of the Linkage Curve Theorem[10, 5]:

> Two surfaces tangent along a C^1-smooth linkage curve are curvature continuous, if and only if at every point of the linkage curve, their normal curvature agrees for an arbitrary direction other than the tangent of the linkage curve.

This practically means that if we have a particular directional sweep along the common boundary, and the normal curvatures of the two ribbon surfaces in this direction are always the same, then we have G^2 continuity. Figure 2 shows an example with the common normal curvatures shown as circular arcs.

In the rest of this section, we will investigate how to determine normals and curvatures from the curve network.

3.1. Preliminaries

A ribbon is a four-sided side interpolant surface $R(s,d)$, where s is the *side parameter*, and d is the *distance parameter*. The side parameter is in the interval $[0,1]$ and runs along the boundary curve. The distance parameter is defined in the cross direction; it is zero on the boundary and increases as we move away from it.

For a given n-sided patch, there is a loop of curves, $P_i(s_i)$, and we need to create the corresponding ribbons $R_i(s_i,d_i)$, $i \in [1 \ldots n]$. The parameters (s_i,d_i) can also be regarded as functions that map values from a common domain (see Section 4.2). Note also that the indexing is circular, with 1 coming after n and vice versa, and we have $P_{i-1}(1) = P_i(0)$ for all i.

3.2. Cross-derivatives

We assume that for each vertex of the network the crossing curves define a local tangent plane. For each boundary $P_i(s_i)$ of a given patch, there exists a normal fence $N_i(s_i)$ that interpolates the normals at the related corners and minimizes its rotation along the boundary. This is called a rotation-minimizing frame or RMF. An exact (closed form) solution of the underlying differential equation cannot be determined, but approximations can be computed via a sequence of discrete points[14].

The cross-derivatives of the ribbons are defined as

$$\begin{aligned} T_i(s_i) &:= \frac{\partial}{\partial d_i} R_i(s_i,0) \\ &= \alpha(s_i) D_i(s_i) + \beta(s_i) \frac{\partial}{\partial s_i} R_i(s_i,0), \end{aligned}$$

where $\alpha(s_i)$ and $\beta(s_i)$ are scalar functions, and $D_i(s_i)$ represents a direction vector function, that is perpendicular to $N_i(s_i)$ everywhere. One trivial choice is the binormal

$$D_i(s_i) = N_i(s_i) \times \frac{\partial}{\partial s_i} R_i(s_i,0),$$

but other definitions are also possible.

The scalar functions satisfy end conditions at the corner points ($s_i = 0$ and $s_i = 1$), but there are further degrees of freedom to define these in order to optimize the shape of the patch. For example, cross-derivatives at the middle of the boundary ($s_i = 0.5$) can be prescribed by users for enhanced control of the surface shape. Alternatively, these can be optimized by fairing algorithms, which is the subject of ongoing research.

3.3. Creating Ribbons

We will look at two types of side interpolants: linear ribbons, that are ruled surfaces suitable for creating a G^1-continuous model, and parabolic ribbons, that are quadratic in the cross direction, and rely on the computed common curvature values to ensure G^2 continuity.

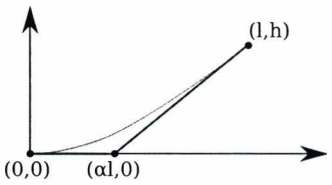


Figure 3: Setting the control points of a parabolic ribbon.

3.3.1. Linear Ribbons

Given a boundary curve $P_i(s_i)$ and the corresponding cross-derivative $T_i(s_i)$, ribbon construction is straightforward:

$$R_i(s_i,d_i) = P_i(s_i) + d_i T_i(s_i).$$

3.3.2. Parabolic Ribbons

Parabolic ribbons are also simple, having the form

$$R_i(s_i,d_i) = P_i(s_i) + d_i T_i(s_i) + \frac{1}{2} d_i C_i(s_i),$$

where $C_i(s_i)$ is the second cross-derivative of the ribbon.

It is a natural choice to calculate the normal curvature in the sweeping direction of the first cross-derivative, i.e., in the plane spanned by $T_i(s_i)$ and $N_i(s_i)$. Let us transform, for ease of computation, the parabolic arc of R_i at a fixed $\hat{s}_i$ into a local coordinate system, where the first point is the origin, and the tangent of the arc is the local x-axis. Then at a given boundary point the equation of the parabola can be written as a quadratic Bézier curve

$$R_i(\hat{s}_i,d_i) = B_0^2(d_i) \cdot (0,0) + B_1^2(d_i) \cdot (\alpha l,0) + B_2^2(d_i) \cdot (l,h),$$

yielding $\kappa = \frac{h}{2\alpha^2 l^2}$ (see Figure 3). Assuming that the width of the parabolic ribbon is the same as the corresponding linear one, l is already defined. Then the prescribed curvature κ can be set by means of α and h, which define the second and third control points of the parabolic arc.

The choice of α gives us a degree of freedom. It may be chosen as a constant. A better choice is to optimize α so that the parabolic ribbon should minimally deviate from the corresponding linear one. This can be formalized as

$$\left(\alpha l - \frac{l}{2}\right)^2 + h^2 \to \min,$$

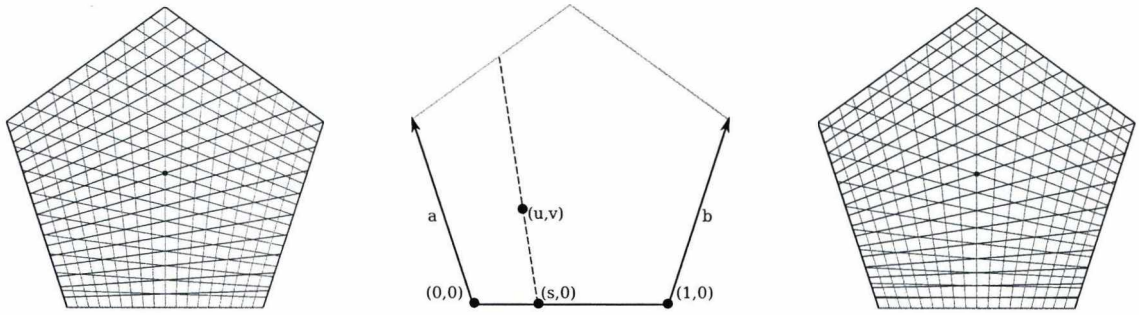
which leads to the depressed cubic equation

$$2\alpha + 16\kappa^2 \alpha^3 l^2 = 1,$$

that can be solved by Cardano's method.

4. Surface Creation

There are various transfinite surface schemes that can be applied to interpolate given ribbons. For a comparison, see a review of the authors[13]. Here we will use Gregory patches, as it is one of the simplest and most well-known methods.



(a) Radial (b) Sweep construction (c) Constrained sweep

Figure 4: Parameterizations

4.1. Ribbons

Gregory patches combine corner interpolants, so our first task is to convert our side-based (linear or parabolic) ribbons to corner-based surfaces. It is well-known from the classical theory of Boolean-sum surfaces[2] that a correction patch $Q_{i,i-1}$ is needed to cancel out the unwanted terms coming from the combination of the two side interpolants (see also Section 4.3):

$$\begin{aligned} R_{i,i-1}(s_i, s_{i-1}) &= R_{i-1}(s_{i-1}, s_i) + R_i(s_i, 1-s_{i-1}) \\ &- Q_{i,i-1}(s_i, s_{i-1}), \end{aligned}$$

where $s_j = s_j(u,v)$ ($j \in [1 \ldots n]$) denote the side parameters defined over the polygonal domain. This brings us to the next topic: how to map the four-sided ribbon surfaces onto the n-sided domain polygon, or inversely, how to determine the local side parameters from a given (u,v) point.

4.2. Parameterization

The domain of a Gregory patch is an n-sided polygon in the 2D plane. Previous research[13] shows that the use of irregular polygons reflecting the spatial distribution of the boundary curves generally improves the quality of transfinite surfaces. In this paper, however, we will use regular domains for simplicity's sake. Experience shows that these behave well unless we have extreme boundary configurations with very uneven side lengths or sudden curvature changes.

Let us determine the parameter s_i for the i-th side. Traditionally, Gregory patches are parameterized by radial sweeping lines[1] (Figure 4a), connecting the domain point in question to the intersection of the extended polygon sides $i-1$ and $i+1$. This is suitable for G^1 continuous surfaces, but further differential properties are required for G^2 continuity. For each point on the i-th side, it is a necessary condition that the parametric speed of the adjacent side parameters s_{i-1} and s_{i+1} is identical, i.e.,

$$\frac{\partial s_{i-1}}{\partial w} = \frac{\partial s_{i+1}}{\partial w},$$

where w is an arbitrary sweeping direction. (See also the proof of continuity given in the Appendix.)

The above property can be satisfied, if we create the sweeping lines using Hermite polynomials. Without loss of generality, let the base edge be a segment from $(0,0)$ to $(1,0)$, a and b edge vectors associated with sides $i-1$ and $i+1$, respectively, and (u,v) the point to be mapped, see Figure 4b. Then we can construct the equation

$$(u,v) = (s,0) + d\,[a \cdot H(s) + b \cdot H(1-s)],$$

where $H(s) = 2s^3 - 3s^2 + 1$, and s and d are unknown. This leads to a fourth-degree equation in s:

$$c_4 s^4 + c_3 s^3 + c_2 s^2 + c_1 s + c_0 = 0,$$

where the coefficients are

$$\begin{aligned} c_4 &= \frac{2}{v}(a^v - b^v), \\ c_3 &= 2(a^u - b^u) + \frac{1}{v}(2u+3)(b^v - a^v), \\ c_2 &= 3(b^u - a^u) - \frac{u}{v}3(b^v - a^v), \\ c_1 &= \frac{1}{v}a^v, \\ c_0 &= a^u - \frac{u}{v}a^v, \end{aligned}$$

with $a = (a^u, a^v)$ and $b = (b^u, b^v)$.

This does not pose any difficulty for real-time computation, as efficient algorithms exist for solving fourth-degree polynomial equations[11], and the values for a given resolution can be cached. For the result, see Figure 4c. Note, that now the blue sweeplines of side $i-1$ and the red sweeplines of side $i+1$ at the bottom are identical in a differential sense.

4.3. Correction Terms

Let us investigate the partial derivatives of two ribbons meeting at a common corner point. We need a single corner inter-

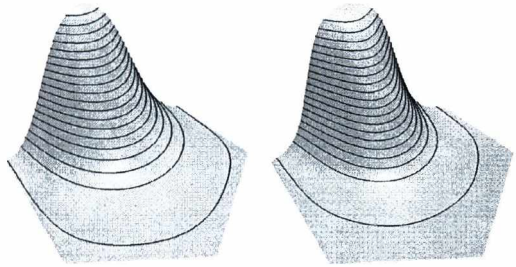


Figure 5: Blend function with $m = 2$ (left) and $m = 3$ (right)

polant, but the partial derivatives are not necessarily identical, which may pose a problem called *twist incompatibility*. Let us introduce the notation $t_i = 1 - s_{i-1}$. We would need

$$\frac{\partial^{p+q}}{\partial s_i^p \partial t_i^q} R_i(0,0) = \frac{\partial^{p+q}}{\partial t_i^q \partial s_i^p} R_{i-1}(1,0) =: W_{p,q} \quad p,q \in \{0,1,2\}.$$

Every curve network satisfies this equation for $p = q = 0$, as the boundaries meet at a fixed corner point. It is also natural to require that the equation holds for $p = 0$, $q = 1$ and $p = 1$, $q = 0$, constraining the first cross-derivative at the boundary to the tangent of the neighboring curve. Most networks do not go beyond this point — and even if the curves match a common surface curvature[5], it only handles the additional cases $p = 2$, $q = 0$ and $p = 0$, $q = 2$, as well as $p = q = 1$.

When some partial derivatives of the ribbons are not compatible, we need to apply Gregory's rational twists. These replace the constant vectors $W_{p,q}$ by rational expressions combining the two parametric variables (see below). In our current research, we assume that the boundary curves match only in position, and for all other terms rational expressions are used. This may create another layer of flexibility for shape optimization.

The correction patch is defined as

$$\begin{aligned} Q_{i,i-1}(s_i,t_i) &= P_i(0) + s_i W_{1,0} + t_i W_{0,1} + s_i t_i W_{1,1} \\ &+ \frac{1}{2} s_i^2 W_{2,0} + \frac{1}{2} t_i^2 W_{0,2} + \frac{1}{2} s_i^2 t_i W_{2,1} \\ &+ \frac{1}{2} s_i t_i^2 W_{1,2} + \frac{1}{4} s_i^2 t_i^2 W_{2,2}, \end{aligned}$$

where each W is a rational function of s_i and t_i. The computation of these is a fairly straightforward generalization of the classical Gregory twists[15]. As an illustration, we show two such terms, the remaining ones are similar:

$$W_{1,0}(s_i,t_i) = \frac{s_i^2 T_{i-1}(1) + t_i^3 \frac{\partial}{\partial s_i} P_i(0)}{s_i^2 + t_i^3},$$

$$W_{1,2}(s_i,t_i) = \frac{s_i^2 \frac{\partial^2}{\partial t_i^2} T_{i-1}(1) + t_i \frac{\partial}{\partial s_i} C_i(0)}{s_i^2 + t_i}.$$

Substituting 0 for either s_i or t_i eliminates one of the conflicting partial derivatives.

4.4. Blending Functions

Every transfinite surface scheme combines individual interpolants by special blending functions that ensure the required interpolation properties and gradually vanish as we move towards the center of the domain. Gregory patches consist of corner interpolants, so we need *corner blends* that interpolate the corner points, gradually fade on the adjacent sides, and vanish on all other sides.

For each (u,v) point in the polygonal domain, we determine an n-tuple of distance values Δ_i, computed as the perpendicular distances from the i-th side. Let $D_{i_1 \ldots i_k} = \prod_{j \notin \{i_1 \ldots i_k\}} \Delta_j^m$, then the corner blend is defined as

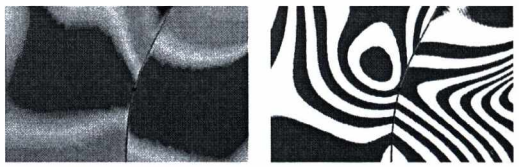
$$B_{i,i-1}(u,v) = \frac{D_{i,i-1}}{\sum_j D_{j,j-1}} \left(= \frac{1/(\Delta_i \Delta_{i-1})^m}{\sum_j 1/(\Delta_j \Delta_{j-1})^m} \right).$$

This function satisfies all requirements — it yields 1 at the $(i-1,i)$ corner, ensures a "gradual" $1 \to 0$ transition on sides $i-1$ and i as we move away from the corner, and vanishes on all the remaining sides.

The exponent m controls how rapidly the contribution of a single ribbon changes, compare the two images in Figure 5. It also ensures that the resulting surface retains the $(m-1)$-th derivatives of the ribbons, so we can use $m = 2$ for linear ribbons, and $m = 3$ for parabolic ribbons to achieve G^1 and G^2 continuity, respectively.

4.5. Surface Equation

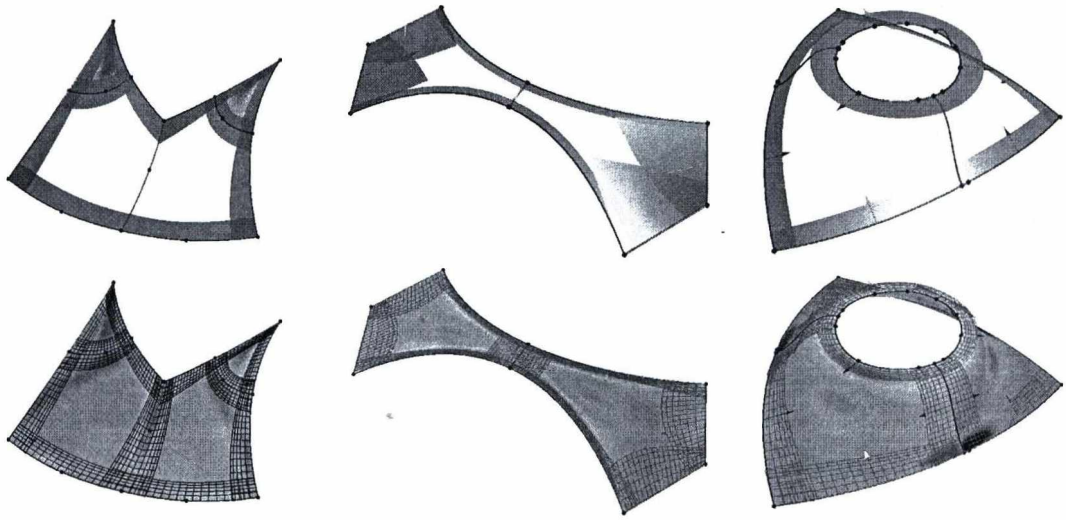
Given a curve network with side interpolants, now we can create all constituents — the corner interpolants, the domain



(a) Linear ribbons (G^1)

(b) Parabolic ribbons (G^2)

Figure 6: Connectivity between Gregory patches



(a) Concave angle (b) Highly curved boundary (c) Internal loop

Figure 7: Improved surfacing using connection curves.

polygon, the parameterization and the blending functions. Putting these together, we arrive at

$$S(u,v) = \sum_{i=1}^{n} R_{i,i-1}(s_i(u,v), s_{i-1}(u,v))B_{i,i-1}(u,v).$$

5. Examples

Figure 6 shows two adjacent patches with linear vs. parabolic ribbons. In this example, the target curvatures were computed by averaging those on the left and right sides. It can be seen, that with parabolic ribbons the curvatures nicely match, and the isophote strips smoothly change across the shared boundary, showing G^2 continuity.

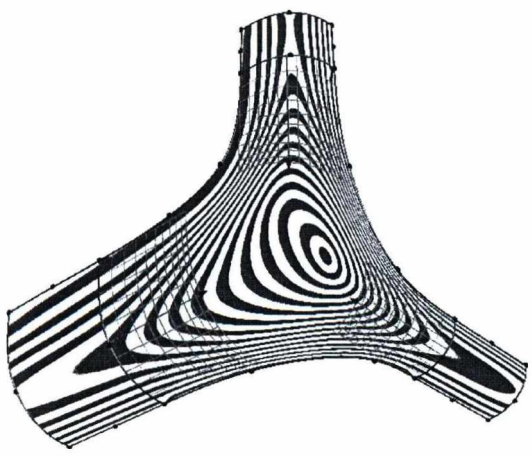


Figure 8: Vertex blend with three master surfaces

In the rest of this section, we present some applications of parabolic ribbons.

5.1. Difficult Curve Network Configurations

While most of the time G^1 transfinite surface interpolation produces a collection of smoothly connected convex patches, there are certain cases, where in order to handle complex configurations or avoid shape artifacts, we need to insert "artificial" *connection curves* into the network:

- curve loops with a concave angle,
- avoiding distortions due to highly curved boundaries,
- connect internal loops (holes).

Examples are shown in Figure 7. In these cases, first we insert connection curves, then create G^1 patches. After taking the average of the curvatures, we compute parabolic ribbons and regenerate the patches now with curvature continuity. These composite patches remain smooth internally and the seamlines are invisible along the connection curves. (Note, that generally connection curves are automatically generated, and remain hidden from the users.)

5.2. Master-Slave constructions

There are various methods to compute a target curvature function along a given curve. The most straightforward solution is averaging the curvatures of two adjacent G^1 patches,

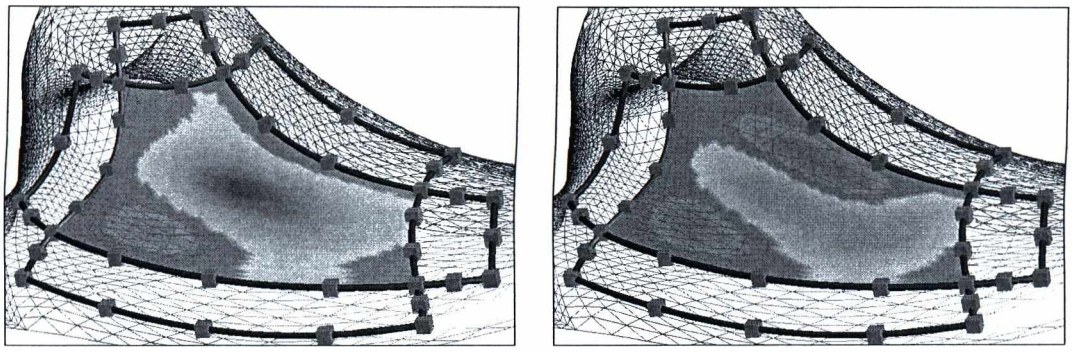


Figure 9: Deviation from the mesh using linear (left) and parabolic (right) ribbons

as before. Another typical situation is, when the curvature of a *master* patch needs to be retained. Then these curvatures are propagated to the surrounding *slave* patches using parabolic ribbons. Such an example is shown in Figure 8, where a setback vertex blend was created, satisfying curvature continuity joining three edge blends, i.e., we have three master surfaces, and one slave in the middle.

5.3. Mesh Approximation

Creating a concise representation of a mesh is an important task in general topology surface modeling. First, a network of curves is drawn on the mesh, which also defines a multi-sided patch structure. By deducing boundary curves and cross-derivatives from the mesh, transfinite surfaces can nicely approximate the data points in the interior. Using locally estimated normal vectors we can create normal fences and linear ribbons for G^1 patches. If we estimate local curvatures along the boundaries, as well, this makes it possible to create parabolic ribbons and G^2 patches, which will produce smoother and more accurate surface models (see Figure 9).

Conclusion

We have discussed an approach for G^2 transfinite surface interpolation combining parabolic ribbons. These ribbons match curvatures that are associated with the edges of a general topology curve network. The formulation is based on corner interpolants and a special sweepline parameterization. Continuity issues and the computation of linear and parabolic ribbons were explained in details. A few applications where G^2 ribbons are needed have also been presented.

There are several open issues in transfinite surface interpolation for future research. Ribbon creation is one of the fundamental ones, as they are constrained, but not uniquely defined. We are currently investigating ribbon optimization approaches for surface fairing and obtaining the best possible transfinite approximations over triangular meshes.

Acknowledgements

This work was supported by the Hungarian Scientific Research Fund (OTKA, No. 101845). The pictures in this paper were generated by the Sketches system developed by ShapEx Ltd., Budapest. The contribution of György Karikó to develop this prototype system is highly appreciated.

References

1. P. Charrot, J. A. Gregory, A pentagonal surface patch for computer aided geometric design, Computer Aided Geometric Design 1 (1) (1984) 87–94.
2. S. A. Coons, Surfaces for computer-aided design of space forms, Tech. Rep. MIT/LCS/TR-41, Massachusetts Institute of Technology (1967).
3. J. A. Gregory, J. M. Hahn, A C^2 polygonal surface patch, Computer Aided Geometric Design 6 (1) (1989) 69–75.
4. R. Hall, G. Mullineux, Continuity between gregory-like patches, Computer aided geometric design 16 (3) (1999) 197–216.
5. T. Hermann, G. Lukács, F.-E. Wolter, Geometrical criteria on the higher order smoothness of composite surfaces, Computer Aided Geometric Design 16 (9) (1999) 907–911.
6. K. Hormann, M. S. Floater, Mean value coordinates for arbitrary planar polygons, Transactions on Graphics 25 (4) (2006) 1424–1441.
7. K. Kato, Generation of n-sided surface patches with holes, Computer-Aided Design 23 (10) (1991) 676–683.
8. K. Kato, N-sided surface generation from arbitrary boundary edges, in: Curve and Surface Design, Innovations in Applied Mathematics, Vanderbilt University Press, 2000, pp. 173–181.

9. H. P. Moreton, C. H. Sequin, Minimum variation curves and surfaces for computer-aided geometric design, in: N. S. Sapidis (ed.), Designing Fair Curves and Surfaces, SIAM, 1994, pp. 123–159.
10. J. Pegna, F.-E. Wolter, Geometrical criteria to guarantee curvature continuity of blend surfaces, Journal of Mechanical Design 114 (1) (1992) 201–210.
11. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical recipes in C (2nd ed.): the art of scientific computing, Cambridge University Press, 1992.
12. P. Salvi, T. Várady, A. Rockwood, Ribbon-based transfinite surfaces, Computer Aided Geometric Design (submitted).
13. T. Várady, A. Rockwood, P. Salvi, Transfinite surface interpolation over irregular n-sided domains, Computer Aided Design 43 (11) (2011) 1330–1340.
14. W. Wang, B. Jüttler, D. Zheng, Y. Liu, Computation of rotation minimizing frames, Transactions on Graphics 27 (1) (2008) 2.
15. A. Worsey, A modified C2 Coons' patch, Computer Aided Geometric Design 1 (4) (1984) 357–360.

Appendix A: Proof of Continuity

We will provide here a short proof that Gregory patches with matching parabolic ribbons are indeed G^2 continuous. We will prove a stronger statement: the modified Gregory patches described in the paper interpolate their ribbons with C^2 continuity.

To save space, arguments of functions are omitted, when this cannot cause any misunderstanding. From here onwards, we look at a point on the ith side, i.e., $s_{i-1} = 1$ and $s_{i+1} = 0$. In this situation, the blending function has several important properties[12], which we list here without proof:

$$B_{i,i-1} + B_{i+1,i} = 1, \tag{1}$$

$$\frac{\partial}{\partial w} B_{j,j-1} = 0, \quad j \notin \{i, i+1\}, \tag{2}$$

$$\frac{\partial^2}{\partial w^2} B_{j,j-1} = 0, \quad j \notin \{i, i+1\}, \tag{3}$$

w being an arbitrary direction. From the above it also follows that

$$\frac{\partial}{\partial w}(B_{j,j-1} + B_{j+1,j}) = 0, \quad j \notin \{i-1, i+1\}, \tag{4}$$

$$\frac{\partial^2}{\partial w^2}(B_{j,j-1} + B_{j+1,j}) = 0, \quad j \notin \{i-1, i+1\}. \tag{5}$$

C^0 Continuity

This is very straightforward, using property (1) of the blend functions:

$$\begin{aligned} S &= R_{i,i-1} B_{i,i-1} + R_{i+1,i} B_{i+1,i} \\ &= R_i(s_i, 0)(B_{i,i-1} + B_{i+1,i}) = R_i(s_i, 0). \end{aligned}$$

C^1 Continuity

As before, most of the equation vanishes, leaving

$$\begin{aligned} \frac{\partial}{\partial w} S &= \left[\frac{\partial}{\partial s_{i+1}} R_{i+1,i} \frac{\partial s_{i+1}}{\partial w} + \frac{\partial}{\partial s_i} R_{i+1,i} \frac{\partial s_i}{\partial w} \right] B_{i+1,i} \\ &+ \left[\frac{\partial}{\partial s_i} R_{i,i-1} \frac{\partial s_i}{\partial w} + \frac{\partial}{\partial s_{i-1}} R_{i,i-1} \frac{\partial s_{i-1}}{\partial w} \right] B_{i,i-1} \\ &+ \left[R_{i+1,i} \frac{\partial}{\partial w} B_{i+1,i} + R_{i,i-1} \frac{\partial}{\partial w} B_{i,i-1} \right]. \end{aligned}$$

Since $R_{i+1,i} = R_{i,i-1}$, we can use property (4) to eliminate the last term. After some calculation, we arrive at

$$\frac{\partial}{\partial w} S = P_i'(s_i) \frac{\partial s_i}{\partial w} + T_i(s_i) \left[B_{i+1,i} \frac{\partial s_{i+1}}{\partial w} + B_{i,i-1} \frac{\partial s_{i-1}}{\partial w} \right],$$

which is a combination of the ribbon's side- and cross-derivatives, proving G^1 continuity. Using the parameterization constraint $\frac{\partial s_{i+1}}{\partial w} = \frac{\partial s_{i-1}}{\partial w}$, we get back the ribbon's first derivative, so we have proved C^1 continuity.

C^2 Continuity

In a similar vein, we can eliminate a large part of the equation, after which the following remains:

$$\begin{aligned} \frac{\partial^2}{\partial w} S &= 2 \left[\frac{\partial}{\partial s_{i+1}} R_{i+1,i} \frac{\partial s_{i+1}}{\partial w} + \frac{\partial}{\partial s_i} R_{i+1,i} \frac{\partial s_i}{\partial w} \right] \frac{\partial}{\partial w} B_{i+1,i} \\ &+ 2 \left[\frac{\partial}{\partial s_i} R_{i,i-1} \frac{\partial s_i}{\partial w} + \frac{\partial}{\partial s_{i-1}} R_{i,i-1} \frac{\partial s_{i-1}}{\partial w} \right] \frac{\partial}{\partial w} B_{i,i-1} \\ &+ \left[\frac{\partial^2}{\partial s_{i+1}^2} R_{i+1,i} \frac{\partial s_{i+1}^2}{\partial w^2} + 2 \frac{\partial^2}{\partial s_{i+1} s_i} R_{i+1,i} \frac{\partial s_i}{\partial w} \frac{\partial s_{i+1}}{\partial w} \right. \\ &+ \left. \frac{\partial^2}{\partial s_i^2} R_{i+1,i} \frac{\partial s_i^2}{\partial w^2} \right] B_{i+1,i} + \left[\frac{\partial^2}{\partial s_i^2} R_{i,i-1} \frac{\partial s_i^2}{\partial w^2} \right. \\ &+ \left. 2 \frac{\partial^2}{\partial s_i s_{i-1}} R_{i,i-1} \frac{\partial s_{i-1}}{\partial w} \frac{\partial s_i}{\partial w} + \frac{\partial^2}{\partial s_{i-1}^2} R_{i,i-1} \frac{\partial s_{i-1}^2}{\partial w^2} \right] B_{i,i-1} \\ &+ \left[R_{i+1,i} \frac{\partial^2}{\partial w^2} B_{i+1,i} + R_{i,i-1} \frac{\partial^2}{\partial w^2} B_{i,i-1} \right]. \end{aligned}$$

The last term vanishes once again, due to property (5). This leads to

$$\begin{aligned} \frac{\partial^2}{\partial w^2} S &= 2 T_i(s_i) \left[\frac{\partial}{\partial w} B_{i+1,i} \frac{\partial s_{i+1}}{\partial w} + \frac{\partial}{\partial w} B_{i,i-1} \frac{\partial s_{i-1}}{\partial w} \right] \\ &+ P_i''(s_i) \frac{\partial s_i^2}{\partial w^2} + 2 T_i'(s_i) \frac{\partial s_i}{\partial w} \left[B_{i+1,i} \frac{\partial s_{i+1}}{\partial w} + B_{i,i-1} \frac{\partial s_{i-1}}{\partial w} \right] \\ &+ C_i(s_i) \left[B_{i+1,i} \frac{\partial s_{i+1}^2}{\partial w^2} + B_{i,i-1} \frac{\partial s_{i-1}^2}{\partial w^2} \right]. \end{aligned}$$

Applying the parameterization constraint, we get

$$\frac{\partial^2}{\partial w^2} S = P_i''(s_i) \frac{\partial s_i^2}{\partial w^2} + 2 T_i'(s_i) \frac{\partial s_i}{\partial w} \frac{\partial d_i}{\partial w} + C_i(s_i) \frac{\partial d_i^2}{\partial w^2},$$

which is the second derivative of the ribbon, so we have proved C^2 continuity.

A Geometry Model for Logarithmic-time Rendering

László Szécsi

Budapest University of Technology and Economics, Budapest, Hungary

Abstract

Complex geometries, like those of plants, rocks, terrain or even clouds are challenging to model in a way that allows for real-time rendering but does not make concessions in terms of visible detail. In this paper we propose a modeling approach called KRS, or kernel-reflection sequences, *inspired by iterated function systems. KRS visualization avoids expanding the procedural definition into polygons all along the rendering pipelines. The model is composed of kernel geometries and reflection transformations that multiply them. We show that a distance function can be evaluated over this structure extremely effectively, allowing for the implementation of real-time sphere tracing in pixel shaders. We also show how the algorithm easily delivers continuous level-of-detail and minification filtering. We discuss how exploiting screen-space coherence can enhance ray-casting performance. We propose several techniques to hide symmetry that could be disturbingly obvious when viewing the models from certain angles. In order to prove that the seemingly limited model can be used to realize various natural phenomena in uncompromising detail without obvious clues of symmetry, we render trees, grass, terrain and rock in real-time.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism Fractals

1. Introduction

Geometries occurring in nature typically feature intricate detail and great extents at the same time. They pose challenges throughout both the modeling and the rendering pipeline, from content creation to final visualization. The usual approach of modeling triangle meshes and rendering them through incremental rasterization fails at both stages. First, every leaf in a forest cannot be modeled manually. It has to be scripted, thus already requiring a procedural description of the geometry, and a way of expanding that description to a visualizable representation. Second, the resulting geometry can be too complex to be rendered in real-time even with the immense triangle throughput of the latest graphics hardware. That gives emergence to convoluted level of detail techniques, which all need to address the issue of transition between different triangle mesh representations.

The later in the pipeline the procedural description is expanded, the less stages can prove to be a bottleneck. CPU expansion would require large amounts of geometry to be communicated to the graphics hardware. With geometry shaders, this is no longer necessary, which has propelled the use of procedural geometries in real-time applications to considerable momentum[6]. However, a large number of triangles would still need to be rasterized, and level-of-detail schemes are still necessary.

In this paper, we propose a formally limited, but practically versatile procedural modeling and visualization approach that defers the expansion of the procedural geometry to the final, pixel shader stage. In other words, we show that ray tracing of the procedural geometry can be done in real-time, at a computational complexity superior to rasterization, and thus vastly outperforming incremental rasterization for sufficiently complex geometries.

2. Previous work

IFS, or *iterated function systems* were conceived by Hutchinson[5]. An IFS describes self-similar geometry by specifying a set of functions mapping the self-similar components to itself[10]. The geometry is the attractor of the iteration of the function system, meaning the geometry can be approximated with arbitrarily small error by iterating any initial bound point set. IFSs are capable of generating geometries of fractal dimensions, occasionally resembling nat-

ural phenomena, but usually with an easily discernable pattern.

CSG, or *constructive solid geometry* defines geometries as results of regular set operations on point sets, which are either specified in the same manner, or are primitives. These primitives are usually given as implicit surfaces. Self-similar, natural geometries can be generated by introducing circles in the construction graph, leading to cyclic object-instantiation graphs, or CSG-PL-Systems[2]. Efficient ray tracing can be performed by generating bounding objects for self-similar components[12].

F-rep[7] defines objects as sets of points for which a function is non-negative. It supports set operations and recursion.

Sphere tracing was proposed by Hart[3]. It is an iterative technique for ray intersection against a geometry for which a distance function is known. This distance function must return a tight underestimation for the distance of a point and the ray traced geometry, or, in other words, the radius of an *unbounding* sphere [4] centered at the point. The algorithm progresses by advancing a point along the ray with this distance, to the surface of the unbounding sphere.

Procedural models can be expanded to triangle meshes in geometry shaders. Algorithms for L-systems[8] and split grammars have been discussed by Sowers[9].

3. Kernel-reflection sequences

Let us start with self-similar geometry defined by an IFS. For the attractor point set A it is true that:

$$A = \bigcup_{j=0}^{m-1} \mathfrak{F}_j A,$$

where $\mathfrak{F}_j$ is a contractive, invertible linear transformation for any j, and the number of component transformations is m. This attractor can be obtained as a limit of the sequence:

$$A_{i+1} = \bigcup_{j=0}^{m-1} \mathfrak{F}_j A_i,$$

$$A_0 = K_0,$$

where K_0 is an arbitrary point set, which we will call a *kernel set*. At this point, because of the contractiveness of the operators, the limit is independent of the choice of K_0. Later, we will abandon contractiveness and this will not be true. In practice, A is approximated as A_n with a suitably large n. For all equations below, $i \in \{0,\dots,n-1\}$ must be true unless otherwise noted. As we wish to use sphere tracing, the kernel set will be defined by distance function $k_0(\mathbf{x})$, which gives the geometric distance between point $\mathbf{x}$ and K_0.

First, let us generalize this construction by allowing different linear transformations on different iterations. $\mathfrak{F}_{j,i}$ denotes the jth transformation operator for iteration i. Then, without the loss of generality, we can assume that $m = 2$, as the polyadic union can always be expressed as a composition of dyadic unions. Thus, the recursive formula for A_i becomes

$$A_{i+1} = \mathfrak{F}_{0,i} A_i \cup \mathfrak{F}_{1,i} A_i.$$

For sphere tracing, an underestimation $a_i(\mathbf{x})$ for the distance between point $\mathbf{x}$ and A_i is required. As described by Hart[3], this can be obtained if the Lipschitz constants of the transformations (denoted by $\mathrm{Lip}\mathfrak{F}$) are known.

$$a_{i+1}(\mathbf{x}) \le \min\left(a_i(\mathfrak{F}_{0,i}^{-1}\mathbf{x})\mathrm{Lip}\mathfrak{F}_{0,i}^{-1}, a_i(\mathfrak{F}_{1,i}^{-1}\mathbf{x})\mathrm{Lip}\mathfrak{F}_{1,i}^{-1}\right),$$

$$a_0(\mathbf{x}) = k_0(\mathbf{x}).$$

The recursive computation of this formula for $a_n(\mathbf{x})$ requires 2^n evaluations of $k_0(\mathbf{x})$, which means that the performance would be exponential in n, and linear in the number of generated kernel instances. Thus, the algorithmic complexity of sphere tracing could be at best identical to incremental rendering of the same geometry, but with a much worse constant factor. It is also notable that ray tracing in general has logarithmic average complexity[11], but it is linear in worst case, and recursively traversing subdivision hierarchies makes it ill-fit for GPU processing.

If bounding hulls are known, then bounds for $a_i(\mathfrak{F}_{0,i}^{-1}\mathbf{x})$ and $a_i(\mathfrak{F}_{1,i}^{-1}\mathbf{x})$ can be obtained, and the performance can be significantly improved by prioritization and lazy evaluation of recursion branches. However, we aim for complete, unconditional elimination of the recursion in order to get an algorithm that has linear complexity in n, and thus logarithmic complexity in the number of kernel instances. Such an algorithm can be implemented as an iteration, effectively executable on graphics processors. To those ends, we drastically limit the transformations we can use. Let $\mathfrak{F}_{0,i}$ always be identity and $\mathfrak{F}_{1,i} \equiv \mathfrak{R}_i$, a reflection on the plane of equation $\mathbf{m}_i \cdot \mathbf{x} - c_i = 0$. We always choose $\mathbf{m}_i$ to have unit length, thus the value of $\mathbf{m}_i \cdot \mathbf{x} - c_i$ also gives the signed distance of point $\mathbf{x}$ to the plane. The operator $\mathfrak{R}_i$ means reflection on this plane.

$$\mathfrak{R}_i \mathbf{x} = \mathbf{x} - 2(\mathbf{m}_i \cdot \mathbf{x} - c_i)\mathbf{m}_i.$$

The $\mathfrak{M}_i$ operator denotes the reflection of a direction vector:

$$\mathfrak{M}_i \omega = \omega - 2(\mathbf{m}_i \cdot \omega)\mathbf{m}_i.$$

Both the indentity and reflection transformations are isometries, thus their Lipschitz constants are unity. Identity and reflection are not contractions, and the sequence is divergent. The kernel geometries are preserved at their original size and detail. Increasing n will increase the extents of the set. Now, the sequence of attractor iterations is as simple as:

$$A_{i+1} = A_i \cup \mathfrak{R}_i A_i,$$

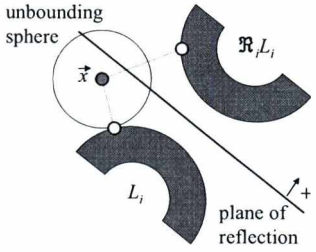


Figure 1: *The distance of the closest point and its reflected image.*

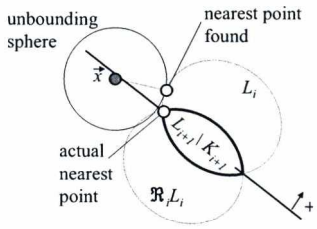


Figure 2: *Clipped kernel part might influence the distance estimate.*

$$A_0 = K_0.$$

We further assume that

$$\mathbf{m}_i \cdot \mathbf{x} - c_i > 0 \longrightarrow \mathbf{x} \notin A_i,$$

$$\mathbf{m}_i \cdot \mathbf{x} - c_i \leq 0 \longrightarrow \mathbf{x} \notin \mathfrak{R}A_i.$$

This means that the geometry is composed of two disjoint parts on the two sides of the mirror plane, which are reflected images of each other (see Figure 1). We call this the assumption of *kernel separation*. A simple test can tell which part is at less distance to point $\mathbf{x}$. We just need to determine if the point is behind or in front of the mirror plane, that is, whether $\mathbf{m}_i \cdot \mathbf{x} - c_i \leq 0$. The closest point of the geometry must be on the same side. (If it were not, the reflected image of the closest point would be even closer, resulting in contradiction.)

This construction allows only for reflection-multiplied instances of the same, unscaled kernel geometry. While this might be enough to model some natural phenomena, hierarchies (like branches of a tree) and non-symmetric parts are not covered. In order to remedy this, let us add a new kernel set K_i at each iteration. These are defined by a sequence of possibly all different $k_i(\mathbf{x})$ distance functions, where $i \in \{0,\ldots,n\}$. To emphasize the difference in construction, we replace the notation A_i with L_i, and call L_i an *expansion level*. These also form a finite sequence, where L_{i+1} is recursively defined as:

$$L_{i+1} = K_{i+1} \cup L_i \cup \mathfrak{R}_i L_i,$$

$$L_0 = K_0.$$

Thus, an expansion level consists of two symmetric instances of the previous level, and an additional kernel set.

We call such a construct a KRS, or *kernel-reflection sequence*. Formally, it is an ordered pair of two finite sequences, one consisting of kernel sets, and another of reflection operators.

$$\text{KRS} = (K_0,\ldots,K_n;\mathfrak{R}_0,\ldots,\mathfrak{R}_{n-1})$$

The distance function for a KRS is:

$$l_{i+1}(\mathbf{x}) = \min(k_{i+1}(\mathbf{x}), l_i(\mathbf{x}), l_i(\mathfrak{R}_i\mathbf{x})),$$

$$l_0(\mathbf{x}) = k_0(\mathbf{x}).$$

In order to evaluate the formula, we do not have to compute all the terms. As L_i and $\mathfrak{R}_i L_i$ are known to be mirrored images, we can decide which distance is going to be smaller by finding on which side of the mirror plane $\mathbf{x}$ is.

$$l_{i+1}(\mathbf{x}) = \begin{cases} \min(k_{i+1}(\mathbf{x}), l_i(\mathbf{x})) & \text{if } \mathbf{m}_i \cdot \mathbf{x} - c_i \leq 0 \\ \min(k_{i+1}(\mathbf{x}), l_i(\mathfrak{R}_i\mathbf{x})) & \text{if } \mathbf{m}_i \cdot \mathbf{x} - c_i > 0 \end{cases}$$

Note that if the assumption of kernel separation does not hold, with this decision we implicitly enforce it by clipping the distance functions to the mirror plane. Therefore, we do not need to take care of this assumption when picking kernels or reflection planes. The returned value might be smaller than the actual distance. In Figure 2, $\mathbf{x}$ is on the negative side, so L_i is closer. The clipped part of the kernel is also considered, making the underestimation somewhat less tight, but still a conservative choice for sphere tracing. The iterative algorithm to evaluate the distance is given in Algorithm 1.

Algorithm 1 Returns distance between $\mathbf{x}$ and the KRS.

```
1:  function DISTANCE(x)
2:      p ← x
3:      d ← k_n(p)
4:      for i = n − 1 downto 0 do
5:          if p · m_i − c_i > 0 then
6:              p ← ℜ_i p
7:          end if
8:          d ← min(d, k_i(p))
9:      end for
10:     return d
11: end function
```

4. Ray-casting

There are multiple options for the GPU visualization of a KRS, practically any IFS visualization method could be generalized. Most prominently, kernel instance transformations

can be computed in geometry shaders, and then kernel geometries rendered with geometry instancing. However, the main motivation behind the construction of KRSs is that they can be ray-traced with an iterative algorithm. Thus, in this paper, we focus on visualization with ray-casting. A full-viewport quadrilateral is rendered, and pixel shaders find the intersection points using sphere tracing. The search is terminated when the ray has passed through the scene or when the computed distance falls below an error treshold level. The value is inversely proportional to the camera depth, and is set to assure that the final unbounding sphere, projected onto the viewport, is smaller than a pixel. A higher treshold level will, in practice, make the kernel geometries appear thicker, as points in close proximity are considered to be members. Therefore, geometries at a large distance will merge into smoother formations, loosing sub-pixel details. This, combined with the smooth shading and texturing techniques we describe in Sections 6.1 and 6.2, eliminates aliasing and achieves automatic, continuous level-of-detail.

4.1. Acceleration with unbounding spheres

The sphere tracing process can be accelerated if we exploit the screen-space coherence of the ray-casting problem. Shaders processing neighboring pixels will execute very similar steps, at least in the beginning. These can be avoided, if, in a cheap preprocessing step, we can find tighter *free distances* to start sphere tracing from. Various algorithms could be based on the idea that when an unbounding sphere is found, the information might be useful for more than one pixel. Unbounding spheres may be rasterized with depth buffering suitably set up, or stored in a searchable data structure and queried from final ray-casting pixel shaders. We conjecture that any such method will produce a starting distance field of practically similar quality, when the cost compared to the full ray-casting itself has to be negligible. We base this claim on the intuitive recognition that a *free distance* map coarser than the viewport resolution can only be useful in front of the first layer of depth. For those expensive, and not uncommon rays that have passed by the geometry closely, the map can give no more clues.

We implemented a scheme that divides the viewport into tiles, and traces beams of primary rays that pass through a tile. Sphere tracing progresses along the ray through the center of the tile up to the last unbounding sphere that covers the complete solid angle of the beam (Figure 3). Then, this sphere and a few more are stored for the tile, and used when ray-casting in pixels of the tile. Care is taken to ensure that the union of all stored spheres, intersected by the beam is convex as seen from the eye. Figure 4 depicts this process. At the ray exit point on the last stored unbounding sphere E, a new, tentative unbounding sphere T is generated. The next stored sphere F must touch the intersection of the two unbounding sphere shells, and its tangents there must go through the eye. This acceleration scheme resulted in about 10-30% less rendering time.

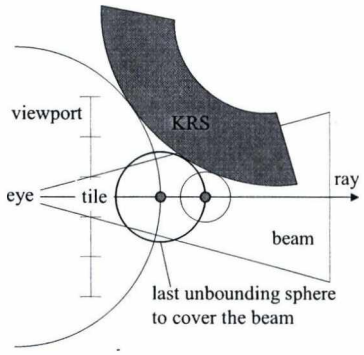


Figure 3: *Traversing a ray for a tile, up until the last sphere that covers the beam.*

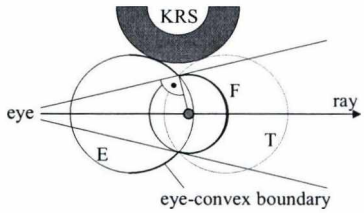


Figure 4: *Finding an unbounding sphere that makes a convex profile. (Note that the center of T has been placed further away than the free distance for a more readable figure.)*

4.2. Acceleration with procedural bounding geometry

Free distances for sphere tracing can also be found by rendering bounding objects. Here, we can easily avoid the exponential explosion, as a choice of level-of-detail is not critical. Abrupt changes in the bounding geometry will not influence the correctness of sphere tracing, and there are no popping artifacts.

5. Techniques to hide symmetry

There are two features of a KRS that strain its credibility as a natural occurrence. Symmetry might be visible and identical motifs are repeating. One countermeasure is that non-symmetric kernel elements are added on every iteration. More importantly, planes of reflection should be selected so that the eye can only be near to a few of them at the same time. If the eye is not near to the mirror plane, the symmetry of the 3D object will not be perceived on a 2D image. Thus, it cannot happen that our geometry appears like an obviously artificial fractal pattern from a viewpont. The undesirable symmetry effects on the global scale are eliminated. Figure 5 offers a comparison.

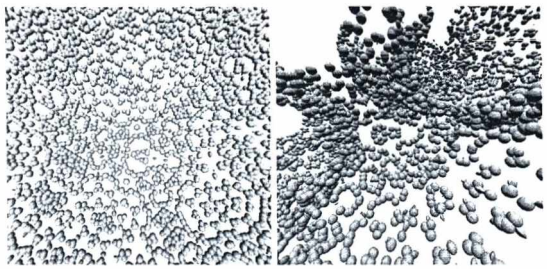


Figure 5: *Spheres reflected by aligned mirrors and unaligned mirrors.*

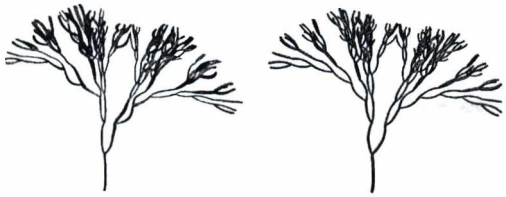


Figure 6: *A tree with isometric transformations and with Lipschitz distortion.*

Fighting symmetry on the local scale is more challenging. There will always be a viewpoint from where two subsets are visibly the reflected images of each other. This can only be handled if the symmetry is indeed broken.

5.1. Combination of multiple KRSs

Where a single KRS cannot produce the desired effect of natural disorder, the union of multiple KRSs can. When, in a forest, there are three completely different trees between the two that are mirror images of each other, symmetry is undetected. Sphere tracing multiple KRSs can be effectively implemented by maintaining the free distance along the ray for all components, and always advancing the ray to the minimum. Compared to the single KRS case, the performance is only decreased where different silhouettes overlap.

Procedural or projective texturing can also be considered to be a way of combining features that exhibit periodicity at different frequencies. We will detail techniques in Section 6.2.

5.2. Distance distortion

The Lipschitz constant of KRS transformation functions is unity, resulting in exact values for the distance (save for non-separated kernels). By sacrificing some performance, we can handle any Lipschitz transformation of the KRS geometry as described by Hart [3]. The resulting geometry does not have to be symmetric any more (Figure 6).

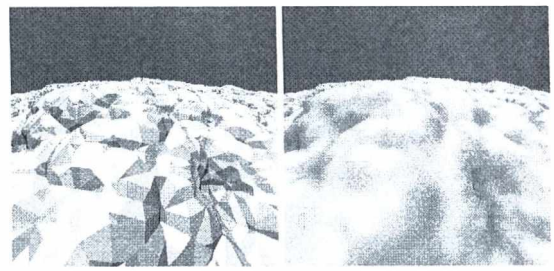


Figure 7: *Rock composed of planar kernels with flat and smooth shading.*

6. Combination with other real-time techniques

KRS ray-casting can only be a viable alternative to geometry-shader produced geometries if it supports all the incremental image synthesis techniques contributing to realism.

6.1. Local shading

KRS kernels are solids with well defined surface normals. The only difficulty is their transformation from kernel space to world space. If b_i is one if $\mathbf{p} \cdot \mathbf{m}_i - c_i > 0$ and zero otherwise, then the complete transformation of the kernel is:

$$\mathbf{p}_{\text{world}} = \mathfrak{R}_{n-1}^{b_{n-1}} \circ \mathfrak{R}_{n-2}^{b_{n-2}} \circ \cdots \circ \mathfrak{R}_{0}^{b_0} \cdot \mathbf{p}_{\text{kernel}}.$$

The normal $\mathbf{v}$ must be transformed with the inverse transpose of the transformation matrix. The inverse of a reflection is itself, transposition and inversion both turn the order of matrix multiplication.

$$\mathbf{v}_{\text{world}} = \mathfrak{M}_{n-1}^{b_{n-1}} \circ \mathfrak{M}_{n-2}^{b_{n-2}} \circ \cdots \circ \mathfrak{M}_{0}^{b_0} \cdot \mathbf{v}_{\text{kernel}}.$$

Unfortunately, the evaluation of this formula requires us either to record decision variables b_i during Algorithm 1, or to maintain a product transformation as the iteration proceeds in decreasing order of i. The latter solution is desirable, as it scales better with increasing n. However, it is even more efficient to transform the world space light vector (and view vector, if necessary) into kernel space, and evaluate the shading there.

When kernels and mirrors are not selected so that kernel separation is upheld, there is likely to be a visible discontinuity of surface normals where the plane of reflection intersects the mirrored geometry. These can be smoothed by interpolating between normals or light directions near these planes. This technique allows the generation of smooth surfaces from a kernel as simple as an infinite plane. Figure 7 compares flat and smooth shading of a surface composed of planar kernels. The shading algorithm complete with light direction interpolation is listed as Algorithm 2. The *lerp* function performs linear interpolation between its first two arguments weighted by the third argument clamped to unit range.

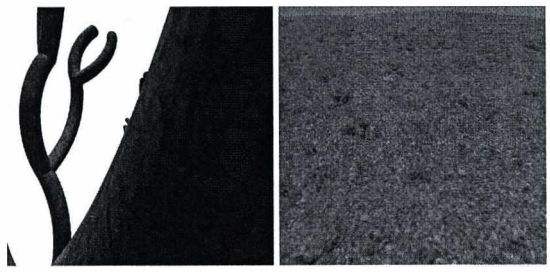


Figure 8: *Texturing with kernel parametrization on tree branches and triplanar projection on terrain.*

The e_{treshold} distance influences the amount of smoothing. To eliminate aliasing artifacts, it should be inversely proportional to the camera depth. The shading algorithm runs only once, after sphere tracing has found the intersection point.

Algorithm 2 Returns the diffuse shaded color of KRS at point $\mathbf{x}$ with surface normal ν illuminated by light from direction τ. $\mathbf{r}$ is the incoming radiance and $\mathbf{d}$ is the diffuse BRDF coefficient.

```
1:  function SHADE(x, ν, τ, r, d)
2:      p ← x
3:      ω ← τ
4:      d ← k_n(p)
5:      for i = n − 1 downto 0 do
6:          e ← p · m_i − c_i          ▷ signed distance to plane
7:          if e > 0 then
8:              p ← ℜ_i p
9:          end if
10:         ρ ← 𝔐_i ω
11:         ω ← lerp(ω, ρ, e/e_treshold + 0.5)
12:         d ← min(d, k_i(p))
13:     end for
14:     return d
15: end function
```

6.2. Texturing

The kernel solids are usually simple objects (e.g. torus segments) that lend themselves to easy u,v parametrization. There are two cases when this solution is not feasible. First, if we use kernels where such a parametrization is not trivial. Second, if the kernels are simplistic, like infinite planes, that even the smallest details of geometry are determined by the reflection transformations. In this second case, texturing all kernel instances with the same coordinates would produce an extremely repetitive pattern, emphasizing symmetries undesirably. Procedural 3D or triplanar[1] texturing is applicable with convincing results (Figure 8). Procedural geometry and procedural or wrapped textures combine to eliminate the observable repetitiveness of each other.

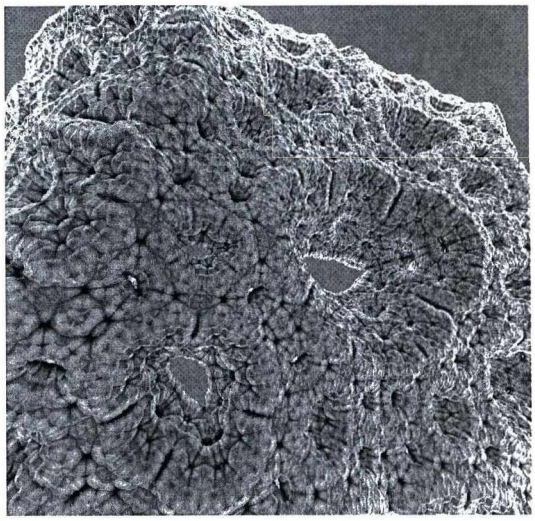


Figure 9: *Sponge with ambient occlusion.*

6.3. Depth composition

A ray-cast KRS can easily be integrated into scenes rendered incrementally. The depth buffer can be used for early termination of rays. The ray casting shader can also output depth if it is necessary, e.g. if transparent geometry is to be rendered afterwards, or if shading is deferred.

6.4. Collision and destructibility

Based on the distance function and surface normal computation, a KRS can be integrated into any collision detection and response scheme.

A KRS is not locally controllable. However, it is possible to create *empty zones* by specifying unbounding spheres over subsets. The regions of these spheres should be skipped during sphere tracing. The subset of the KRS within these empty zones can be substituted with instances of kernel geometry rendered incrementally. These solid instances can then be subjects of any kind of physical simulation. E.g. when a branch of a tree in a forest should break, the complete tree is covered with an empty sphere, and an identical tree of rigid body branches and leaves is built. This can then be manipulated independently.

6.5. Global shading

Beyond sheer triangle throughput, KRS geometry has another key advantage over incrementally rendered geometry. Sphere tracing can not only be performed for eye rays, but also for secondary rays. Shadows, reflections, ambient occlusion (Figure 9), or any global illumination techniques based on ray tracing can be implemented.

7. Modeling

KRSs appear to be limited at what can be modeled with them. At an iteration count and scale large enough for the individual kernels not to be distinguishable, the geometry tends to resemble the 3D equivalent of a Lévy C-curve. However, a forest from the air, a cloud in the sky, a hilly landscape, or a battered rock look exactly like that, from far enough. In this section we are going to present a few examples of application.

7.1. Coral, terrain and rock

Those natural features that are traditionally well modeled by IFS are good candidates for KRS. The variation in scale that is lost because we only use isometries is compensated by additional kernels and any statistical self-similarity induced by the choice of our transformations. When modeling these geometries, the choice of kernel sets is also less important: spheres or infinite planes are sufficient. The separation of kernels is neither desirable nor always possible. Even the smallest details will be defined by the transformations. Smooth shading and procedural or triplanar texturing greatly enhance the visual quality.

7.2. Tree and forest

A tree is a classic hierarchical structure, where the kernel sets added at each iteration become crucial. The first few kernels and reflections define the leaf geometry and the thinnest twig. Then, every new expansion level doubles this geometry, with a reflection placed so that the two main branches start from the same point. A new, thicker branch that ends at the junction is added as the next kernel. For kernels acting as branches, we used *toroidal capsules*, composed of a torus segment and two capping spheres. The coefficients of the distance functions are computed from intuitive modeling parameters. First, forking positions along one route from the trunk to a twig end must be given. Branches along this route will be the kernels. Then, a control point on every such branch can be moved to set curvature. Two forking positions and a control point define the generating circle of the torus. Branch width gives the section radius. Planes of reflection must be placed at forking positions, with editable normals. A forest can be generated by adding more reflections (identical to those of the terrain, if it is also present) with empty extra kernel sets.

8. Results

There were two distinct types of test scenes: ones with complex kernels (trees, grass) and ones with simples kernels (terrain, rock). In the following table we give how much time it took (in microseconds) to trace a single ray on average, versus the triangle count equivalent of the scene complexity (how many triangles would be necessary to achieve the same result.)

Figure 10: *A groove of trees.*

Triangle count	Complex kernel (μs)	Simple case (μs)
2^6	21	
2^{12}	37	
2^{18}	50	47
2^{20}	54	93
2^{22}	58	146
2^{24}	61	191
2^{26}	64	250
2^{28}	65	300
2^{30}	68	355
2^{32}	71	397

Increasing the expansion level by one resulted in an average increase of rendering time of 25 microseconds for complex kernels and 3 microseconds for simple ones. We can conclude that it is possible to achieve interactive speeds on scenes equivalent to billions of triangles.

9. Conclusion

KRS rendering can be used together with incremental image synthesis in real-time applications. It is capable of modeling a wide range of natural geometries, without the possibility of local control, but with fine details and large extent at the same time. Features that could only be represented by billions of triangles, like grasslands or forests, can be rendered in real time. Thus, KRS can add the desired natural richness of detail to virtual worlds without the need for customized level-of-detail techniques.

Animation of KRS geometries is left for future work. While animation of the model parameters is unlikely to pro-

duce credible motion, subtle changes to low-index transformations might be acceptable. Time-dependent Lipschitz transformations appear more promising.

10. Acknowledgements

This work has been supported by OTKA PD-104710 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

1. Ryan Geiss. *Generating complex terrains using the GPU*, chapter 1, pages 7–37. Addison-Wesley Professional, 2007.
2. M. Gervautz and C. Traxler. Representation and realistic rendering of natural phenomena with cyclic CSG graphs. *The Visual Computer*, 12(2):62–74, 1996.
3. J.C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.
4. J.C. Hart and D.J. Sandin. Louis H Kauffman t. Ray Tracing Deterministic 3D Fractals. *Computer Graphics*, 23(3), 1989.
5. J.E. Hutchinson, Dept. of Mathematics, and University of Melbourne. *Fractals and Self Similarity*. University of Melbourne.[Dept. of Mathematics], 1979.
6. H. Nguyen. Gpu gems 3. Part I - Geometry. 2007.
7. A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.
8. P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc. New York, NY, USA, 1990.
9. B. Sowers, T. Menzies, T. McGraw, A. Ross, and W.V. Morgantown. Increasing the Performance and Realism of Procedurally Generated Buildings. 2008.
10. L. Szirmay-Kalos. *Számítógépes grafika*. Computer-Books, Budapest, 1999.
11. L. Szirmay-Kalos and G. Márton. Worst-case versus average-case complexity of ray-shooting. *Journal of Computing*, 61(2):103–131, 1998.
12. C. Traxler and M. Gervautz. Efficient ray tracing of complex natural scenes. *Proceedings Fractals*, 97, 1979.

Rotational-minimizing surfaces in sphere-based surface design

Miklós Hoffmann,[1] Juan Monterde[2]

[1] Institute of Mathematics and Computer Science, Károly Eszterházy College, Leányka str. 4, H-3300 Eger, Hungary
[2] Dep. de Geometria i Topologia, Universitat de València, Avd. Vicent Andrés Estellés, 1, E-46100-Burjassot (València), Spain

Abstract

This work is aiming at computing the blending surface of two given spheres, where the touching circles (and therefore the tangent cones) are predefined on the spheres. The blending surface (or skin) has to touch the given spheres at predefined circles. This is a subtask of a larger project, where - instead of the well-known control point-based design - sphere-based design of surfaces is introduced. The main advantage of the presented method over the existing ones is the minimization of unwanted distortions of the blending surface. This is achieved by the application of rotational-minimizing frames for the transportation of a vector along a given curve, which technique, beyond its theoretical interest, helps us to determine the corresponding points along the touching circles of the two spheres. The final blending surface is also defined by the help of the rotational minimizing transportation. The discussion of the method follows our recently submitted paper [10].

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computational Geometry and Object Modeling]: Curve, surface, solid, and object representations

1. Introduction and related work

The problem of joining spheres by a surface being tangent to them is considered by several authors using various approaches. This surface, also called blending surface or skin, touches the spheres along circles, where the tangent cone of the sphere contains the end tangents of the surface. This is a specific problem of a larger project KSpheres[11, 12] (see Fig.1), where sphere-based design of characters and other surfaces is introduced, a set of spheres has been blended by surface patches. In this approach the touching circles are computed with the help of Apollonian circles and the blending surface is formed by cubic Hermite interpolants.

Our initial problem is uniquely defined by the touching cones of two spheres (c.f. Fig. 2). This problem is of essential importance in newly emerged animation design software tools, such as ZSpheres® or Spore™, where the point-based design is altered by sphere-based construction, but it also appears in medical applications[15]. Dupin cyclides and their generalizations have been applied to this problem (see[14, 13] and references therein) with a deep geometric view but also with restrictions to the positions of the initial data in case of a single cyclide surface, while in case of two joining cyclides the geometric form of the blending surface has not always been satisfactory.

An iterative method of skinning has been introduced in[16, 15], where, starting from an initial skin, an energy function is minimized during the iterations in order to reach the optimal surface. This method has certain advantages from the viewpoint of optimization, but it is also quite sensitive for the initial position of the skin and due to the iterative nature it is quite slow with no proven convergence.

Although our earlier method in KSpheres[11] works for almost any positions of the initial data, the final surface, just as in case of all the above mentioned methods, may suffer from distortions due to the unwanted rotation along the central (spine) curve. The source of this distortion is the fact that the pairing of corresponding points on the two touching circles is not solved in a sufficient way. To avoid the unnecessary rotation of the blending surface along the spine curve a straightforward idea is to apply the recent results of rotational-minimizing frames (RMF). These frames, originally developed as an alternative of the well-known Frenet frames of parametric curves, were also known as relatively

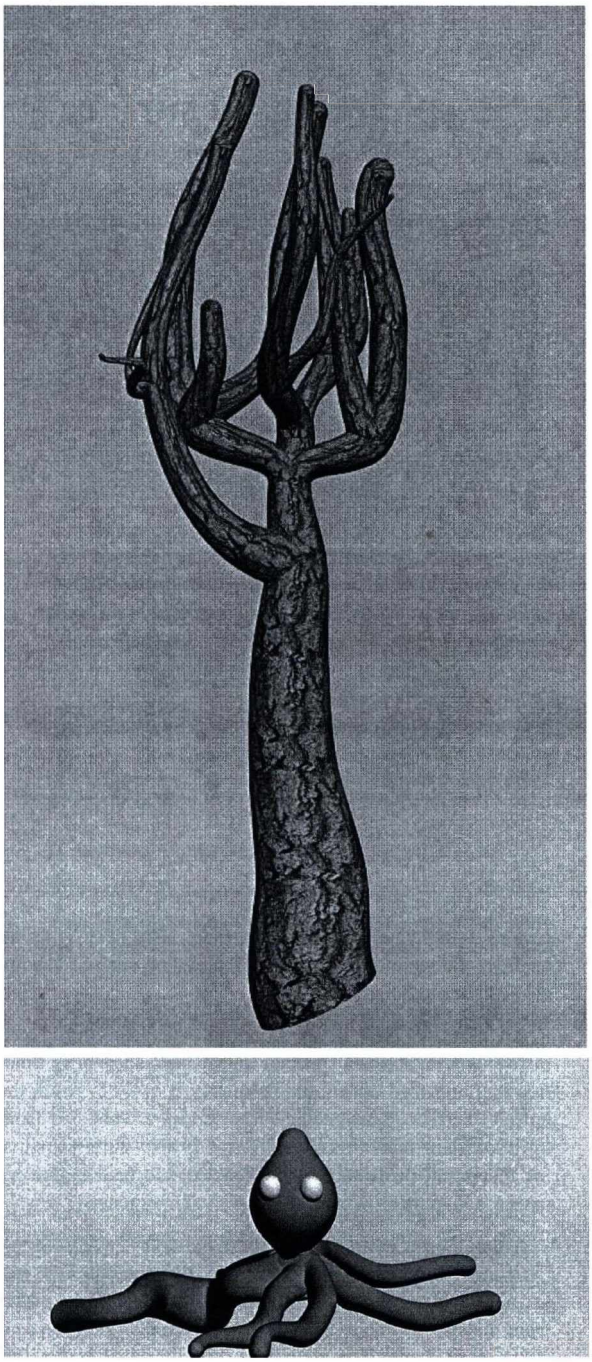


Figure 1: *Sphere-based design of surfaces in KSpheres.*

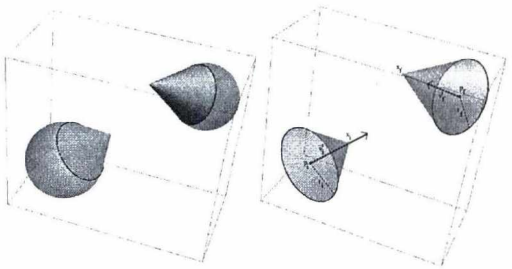


Figure 2: *Visualization of the statement of the problem. The initial data contain two spheres and their touching cones.*

parallel adapted frames or Bishop frames[1]. They can provide a sufficient solution of minimizing this kind of distortion along a curve[8, 17]. The rotational-minimizing frames are excellent tools in curve design[4, 7, 5]. They have been also applied in constructing parametric surfaces in[2] but that approach provides only an approximate solution, moreover it requires a continuous one-parameter set of spheres instead of discrete input data.

The aim of this paper is at providing a method to blend two spheres by a rotational-minimizing blending surface. Contrary to the previous approaches, the RMF is applied for a spine curve of the future surface, by the help of which the surface is computed in a way that the parameter curves follow the RMF. Moreover, in earlier approaches the spine curve is a planar curve, while in our method it can be a spatial curve as well. The surface is rotational-minimizing in a sense that these parametric curves of the surface have no distortion comparing to the corresponding rotational-minimizing frame along the spine curve. The method works for a wide range of initial data and can be computed in seconds, although not necessarily in real time. In Section 2 of this paper the problem is precisely formulated, together with the overview of the method provided in[11]. The basic notations and the necessary computational methods are summarized in terms of rotational-minimizing frames in Section 3, where we also define the notion of rotational-minimizing transportation along a curve. The solution and the detailed computation are provided in Section 4.

2. The problem

Suppose that two spheres are given in $\mathbb{R}^3$, with two circles (p_i, r_i) and (p_f, r_f) on them, having centers p_i and p_f and radii r_i and r_f, respectively. Let x_i be the apex of the tangent cone touching the first sphere along circle (p_i, r_i), while x_f be the apex of the tangent cone touching the second sphere along circle (p_f, r_f) (see Fig. 2 and 6).

Further on, let $\overrightarrow{v_i}$ and $\overrightarrow{v_f}$ be vectors from the center of the given circles, parallel to the axes of the touching cones, that is $c_i\overrightarrow{v_i} = \overrightarrow{p_i x_i}$ and $c_f\overrightarrow{v_f} = \overrightarrow{p_f x_f}$ for some $c_i, c_f \in \mathbb{R} \setminus \{0\}$.

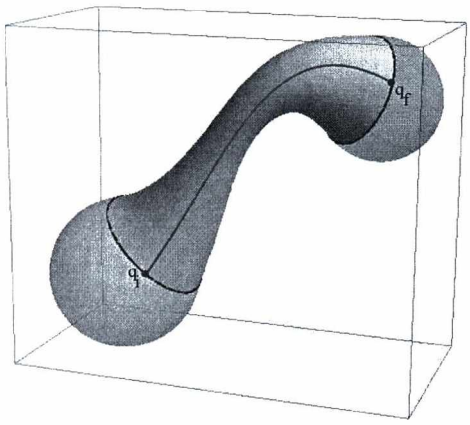


Figure 3: *A surface blending the given spheres touching them at the given circles.*

Our aim is to define a surface $S(\theta, t)$ touching the given spheres at the given circles with the least possible distortion. It is constructed in a way that the parametric curve $S(\theta_0, t)$ which belongs to a fixed value θ_0 will connect a point q_i of the first circle and the corresponding point q_f of the second circle, while tangent vectors of the curve in these points will be parallel to the corresponding generatrix of the tangent cone, that is

$$S(\theta_0, 0) = q_i, \qquad S(\theta_0, 1) = q_f,$$

$$\frac{\partial S}{\partial t}(\theta_0, 0) = \overrightarrow{w_i}, \qquad \frac{\partial S}{\partial t}(\theta_0, 1) = \overrightarrow{w_f}$$

where

$$\overrightarrow{w_i} = a_i\left(\overrightarrow{p_i x_i} - \overrightarrow{p_i q_i}\right) = a_i\left(c_i\overrightarrow{v_i} - \overrightarrow{p_i q_i}\right) \tag{1}$$

and

$$\overrightarrow{w_f} = a_f\left(\overrightarrow{p_f x_f} - \overrightarrow{p_f q_f}\right) = a_f\left(c_f\overrightarrow{v_f} - \overrightarrow{p_f q_f}\right) \tag{2}$$

with $a_i, a_f \in \mathbb{R}^+$.

Also notice that we can write

$$\overrightarrow{w_i} = a_i\overrightarrow{q_i x_i}, \qquad \overrightarrow{w_f} = a_f\overrightarrow{q_f x_f}.$$

The key question is that which point q_f of the second circle should belong to the initial point q_i. In[11] a fast and simple algorithm is provided to compute the corresponding points. By defining an arbitrary direction $\overrightarrow{e}$ not parallel to $\overrightarrow{v_i}$ and $\overrightarrow{v_f}$, the two corresponding points are defined by the vectors $\overrightarrow{v_i} \times \overrightarrow{e}$ and $\overrightarrow{e} \times \overrightarrow{v_f}$, more precisely

$$q_i = p_i \pm r_i \frac{\overrightarrow{e} \times \overrightarrow{v_i}}{\|\overrightarrow{e} \times \overrightarrow{v_i}\|}$$

$$q_f = p_f \pm r_f \frac{\overrightarrow{e} \times \overrightarrow{v_f}}{\|\overrightarrow{e} \times \overrightarrow{v_f}\|},$$

where the sign of the latter tags depends on the measure of the angles between the vectors $\overrightarrow{v_i}$ and $\overrightarrow{p_i p_f}$, and $\overrightarrow{v_f}$ and $\overrightarrow{p_i p_f}$, respectively. If the angle is greater than $\pi/2$, then the sign is negative, otherwise it is positive. Further pairs of points are obtained by the rotation of the original pairs along the circles.

Note, that the choice of $\overrightarrow{e}$ has not uniquely been defined in this algorithm. A similar, even more evident choice of the two corresponding points can be defined simply by the vector $\overrightarrow{v_i} \times \overrightarrow{v_f}$ as

$$q_i = p_i + r_i \frac{\overrightarrow{v_i} \times \overrightarrow{v_f}}{\|\overrightarrow{v_i} \times \overrightarrow{v_f}\|}$$

$$q_f = p_f + r_f \frac{\overrightarrow{v_i} \times \overrightarrow{v_f}}{\|\overrightarrow{v_i} \times \overrightarrow{v_f}\|}.$$

This straightforward but somehow ad hoc selection of corresponding points of the circles works in a satisfactory way for several cases, for example when the vectors $\overrightarrow{v_i}, \overrightarrow{v_f}$ and $\overrightarrow{p_i p_f}$ are coplanar, that is the axes of the touching cones are parallel or intersecting lines. In most figures of [11] this is the case. However, if the positions of the touching circles differ from this case, that is the axes of the touching cones are skew lines, the above mentioned method of mapping of points cannot be justified. In fact, the blending surface is heavily affected by the choice of corresponding points and can have a kind of distortion in some cases, as it can be seen in Figures 4. Our aim is to provide a theoretically funded method of finding the most suitably corresponding pairs of points.

3. Rotation Minimizing Frames

In this section we briefly overview the basic concept of Rotational Minimizing Frames (RMF in short), and we define the rotational minimizing transportation of a vector along the curve.

Definition 1 (See [8], [17]) Given a regular curve $\alpha : [0,1] \to \mathbb{R}^3$ a positive oriented orthonormal frame $\{\overrightarrow{w_0}, \overrightarrow{w_1}, \overrightarrow{w_2}\}$ along the curve is called rotation-minimizing frame (RMF) if

1. $\overrightarrow{w_0} = \overrightarrow{\mathbf{t}}$,
2. $\overrightarrow{w_1}'$ is parallel to $\overrightarrow{\mathbf{t}}$.

Remark 1 The second condition is equivalent to ask for $\dot{\overrightarrow{w_1}}$ being parallel to $\overrightarrow{\mathbf{t}}$, where the dot denotes arc-length derivative.

3.1. Transportation along a curve using a RMF

Given a unit vector $\overrightarrow{v_0}$, orthogonal to $\alpha'(0)$, there is a unique rotation-minimizing frame along the curve α, denoted by $\{\overrightarrow{\mathbf{t}}, \overrightarrow{w_1}, \overrightarrow{w_2}\}$, such that

$$\overrightarrow{w_1}(0) = \overrightarrow{v_0}.$$

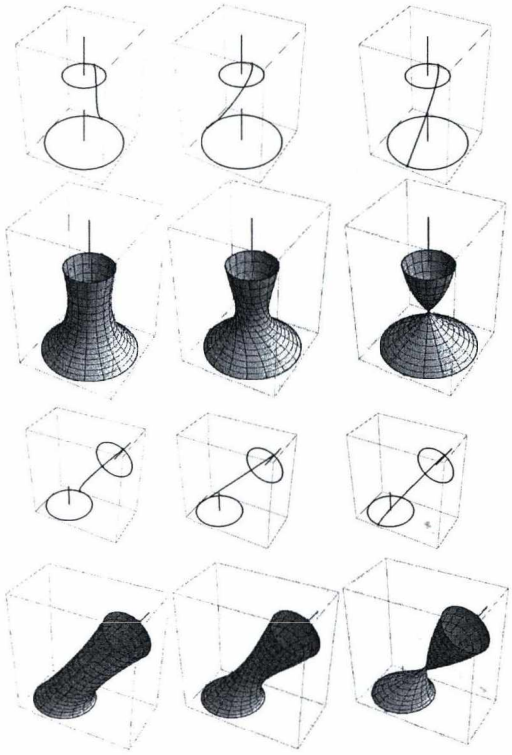


Figure 4: *The effect of the different choices of corresponding points and possible distortions of the surfaces in two different positions of initial data*

Let us recall how to compute it. If we denote the Frenet frame of the curve α by

$$\{\vec{\mathbf{t}}, \vec{\mathbf{n}}, \vec{\mathbf{b}}\}$$

then

$$\vec{w_1}(t) = \cos(\theta(t))\,\vec{\mathbf{n}}(t) - \sin(\theta(t))\,\vec{\mathbf{b}}(t), \qquad (3)$$

where

$$\theta(t) = \theta_0 - \int_0^t \tau(s)\,ds, \qquad (4)$$

being θ_0 the oriented angle defined by the normal vector $\vec{\mathbf{n}}(0)$ and $\vec{v_0}$, and being τ the torsion of the curve α. Arc-length parameter is denoted by s. If another parameter is used for the curve α, then

$$\int_0^t \tau(s)\,ds = \int_0^t \tau(u)||\alpha'(u)||\,du.$$

Definition 2 Let us define the *rotation-minimizing transportation* (RM-transportation for short) to $\alpha(1)$ of the vector $\vec{v_0}$ along the curve α using a rotation-minimizing frame as the vector $\vec{w_1}(1)$.

Notice that $||\vec{w_1}(1)|| = 1$ and $\vec{w_1}(1) \perp \alpha'(1)$.

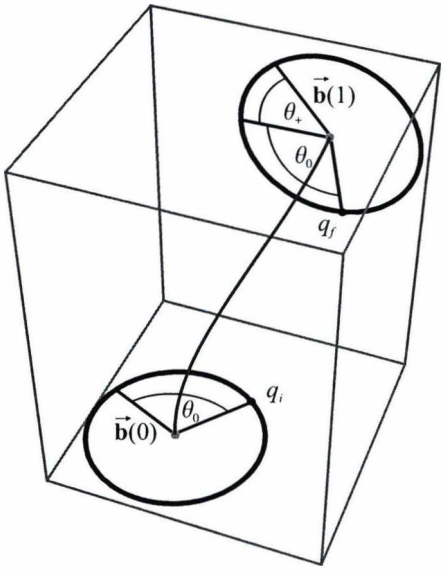


Figure 5: *The rotation-minimizing map $q_i \to q_f$ between the perpendicular unit circles at the end points of the curve $\alpha(t)$.*

Therefore, we can define a map between the perpendicular unit circles at the end points of the curve α. Given a point q_i in the unit circle on the normal plane to the curve at $\alpha(0)$, let us consider the vector $\vec{v_0} = \overrightarrow{\alpha(0)q_i}$, then $q_f = \alpha(1) + \vec{w_1}(1)$. The initial angle θ_0 is the angle between $\vec{v_0} = \overrightarrow{\alpha(0)q_i}$ and the binormal vector to the curve $\alpha(t)$ at $\alpha(0)$, $\vec{\mathbf{b}}(0)$. The angle θ_+ (see Fig. 5) is defined as

$$\theta_+ = -\int_0^1 \tau(u)||\alpha'(u)||\,du.$$

Notice that in this way we can control which point q_f in the second circle is associated with a given point q_i in the first circle, while the tangent vectors of the parameter curve of the surface joining q_i and q_f are inherited from the touching cones.

4. The solution

In order to apply the rotational-minimizing frame method, at first we define a spine curve $\alpha(t)$ of the future surface. For the notations see Figure 6.

Let $\alpha : [0,1] \to \mathbb{R}^3$ be a regular curve satisfying the C^1-Hermite conditions:

$$\alpha(0) = p_i, \qquad \alpha(1) = p_f,$$
$$\alpha'(0) = \vec{v_i}, \qquad \alpha'(1) = \vec{v_f}.$$

Let $\vec{v_0} = \frac{\vec{p_i q_i}}{r_i}$ and let $\vec{v_1}$ be the RM-transportation to $\alpha(1)$ of the vector $\vec{v_0}$.

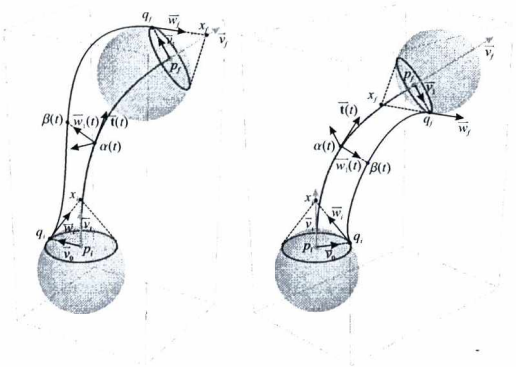


Figure 6: *Visualization of various notations in two different positions of initial data (with unit spheres). Spine curve $\alpha(t)$ (purple) and one of the parameter curves $\beta(t)$ (black) can also be seen.*

Being r_f the radius of the second circle, let us define the corresponding point q_f as

$$q_f = p_f + r_f \overrightarrow{v_1}.$$

We construct the parametric curve $S(\theta_0, t) = \beta(t) : [0,1] \to \mathbb{R}^3$ such that

$$\begin{aligned} \beta(0) &= q_i, \qquad \beta(1) = q_f, \\ \beta'(0) &= \overrightarrow{w_i}, \qquad \beta'(1) = \overrightarrow{w_f}, \end{aligned} \tag{5}$$

hold, and further on the curve follows the RM-transportation, that is for any $t \in [0,1]$, the vector $\overrightarrow{\alpha(t)\beta(t)}$ is parallel to $\overrightarrow{w_1}(t)$, where $\overrightarrow{w_1}(t)$ is the actual position of the RM-transported vector.

The last condition implies that there exists a function $\lambda : [0,1] \to \mathbb{R}$ such that

$$\beta(t) = \alpha(t) + \lambda(t)\overrightarrow{w_1}(t).$$

Therefore

$$\begin{aligned} \beta'(t) &= \alpha'(t) + \lambda'(t)\overrightarrow{w_1}(t) + \lambda(t)\overrightarrow{w_1}'(t) \\ &= ||\alpha'(t)||\,(1 + \mu(t)\lambda(t))\,\overrightarrow{\mathbf{t}}(t) + \lambda'(t)\overrightarrow{w_1}(t) \end{aligned}$$

where we have used that, since $\{\overrightarrow{\mathbf{t}}, \overrightarrow{w_1}, \overrightarrow{w_2}\}$ is a rotation-minimizing frame, then

$$\dot{\overrightarrow{w_1}}(s) = \mu(s)\,\overrightarrow{\mathbf{t}}(s)$$

or, equivalently

$$\overrightarrow{w_1}'(t) = ||\alpha'(t)||\mu(t)\,\overrightarrow{\mathbf{t}}(t).$$

Let us compute explicitly the function $\mu(s) = \langle \dot{\overrightarrow{w_1}}(s), \overrightarrow{\mathbf{t}}(s) \rangle$.

From Eq. (3) we have

$$\begin{aligned} \dot{\overrightarrow{w_1}} &= -\dot{\theta}\sin(\theta)\,\overrightarrow{\mathbf{n}} + \cos(\theta)\,\dot{\overrightarrow{\mathbf{n}}} - \dot{\theta}\cos(\theta)\,\overrightarrow{\mathbf{b}} - \sin(\theta)\,\dot{\overrightarrow{\mathbf{b}}} \\ &= -\dot{\theta}\sin(\theta)\,\overrightarrow{\mathbf{n}} + \cos(\theta)(-\kappa\,\overrightarrow{\mathbf{t}} - \tau\,\overrightarrow{\mathbf{b}}) - \dot{\theta}\cos(\theta)\,\overrightarrow{\mathbf{b}} - \sin(\theta)\tau\,\overrightarrow{\mathbf{n}} \\ &= -\cos(\theta)\kappa\,\overrightarrow{\mathbf{t}}, \end{aligned}$$

where we have applied that $\dot{\theta} = -\tau$ (see Eq. (4)).

Therefore,

$$\mu(s) = -\cos(\theta(s))\kappa(s). \tag{6}$$

Let us write conditions (5) in terms of the function $\lambda(t)$.

$$\begin{aligned} \beta(0) &= \alpha(0) + \lambda(0)\overrightarrow{w_1}(0) \\ &= p_i + \lambda(0)\overrightarrow{v_0} \\ &= p_i + \lambda(0)\frac{\overrightarrow{p_i q_i}}{r_i}. \end{aligned}$$

Since we asked for $\beta(0) = q_i = p_i + \overrightarrow{p_i q_i}$, then

$$\lambda(0) = r_i. \tag{7}$$

Analogously,

$$\lambda(1) = r_f. \tag{8}$$

More conditions on λ can be obtained through the derivative of the curve β:

$$\begin{aligned} \beta'(0) &= ||\alpha'(0)||\,(1 + \mu(0)\lambda(0))\,\overrightarrow{\mathbf{t}}(0) + \lambda'(0)\overrightarrow{w_1}(0) \\ &= (1 + \mu(0)r_i)\,\overrightarrow{v_i} + \lambda'(0)\frac{\overrightarrow{p_i q_i}}{r_i}. \end{aligned}$$

Since $\beta'(0) = \overrightarrow{w_i} = a_i\left(c_i\overrightarrow{v_i} - \overrightarrow{p_i q_i}\right)$ (see Eqs. 5 and 1), then

$$\lambda'(0) = -a_i r_i \tag{9}$$

and

$$1 + \mu(0)r_i = a_i c_i.$$

According to Eq. (6), $\mu(0) = -\cos(\theta_0)\kappa(0)$. Therefore,

$$\boxed{a_i = \frac{1}{c_i}\left(1 - \cos(\theta_0)\kappa(0)r_i\right).} \tag{10}$$

Notice that this means that the length of the initial tangent vector to the curve β depends on $\theta_0, \kappa(0)$ and r_i.

Analogously,

$$\lambda'(1) = -a_f r_f \tag{11}$$

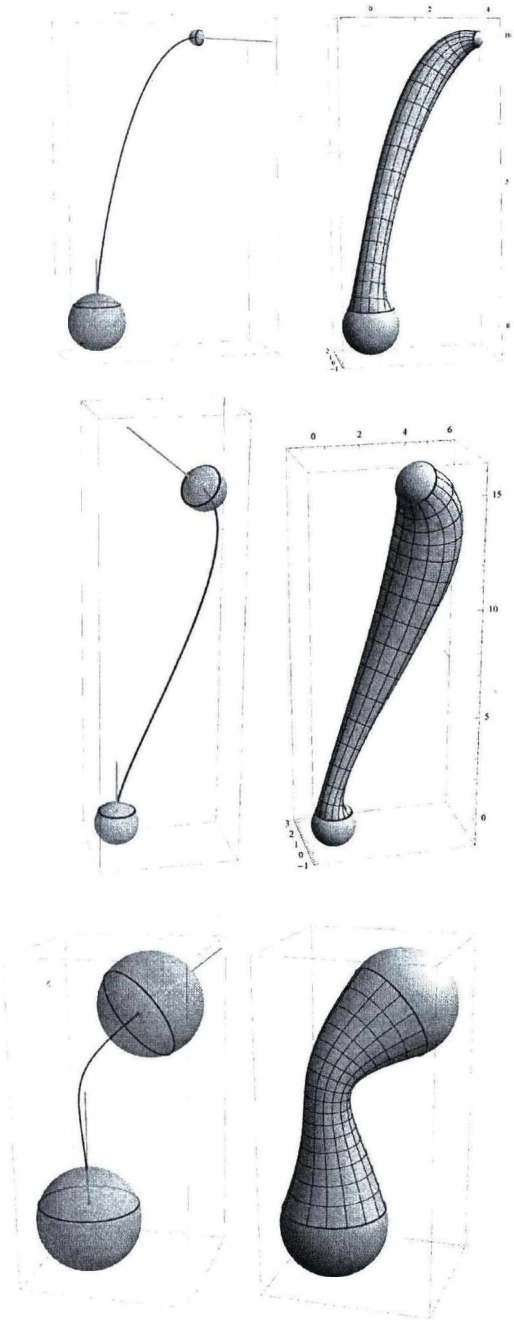


Figure 7: *Various positions of initial data with the spine curve (left) and the computed surface (right).*

and

$$\boxed{a_f = \frac{1}{c_f}\left(1-\cos(\theta(1))\kappa(1)r_f\right).} \tag{12}$$

Remark 2 A restriction must have to be considered. Since both a_i and a_f must be positive, then the spine curve has to be such that

$$\kappa(0)r_i \leq 1, \qquad \textit{and} \qquad \kappa(1)r_f \leq 1.$$

Such conditions can be satisfied through a convenient choice of the initial and final vectors, $\alpha'(0) = \overrightarrow{v_i}$ and $\alpha'(1) = \overrightarrow{v_f}$ of the spine curve. A simple computation shows that the curvature of the spine curve at the initial point is

$$\kappa(0) = 2\frac{||\overrightarrow{v_i} \wedge (3\overrightarrow{p_i p_f} - \overrightarrow{v_f})||}{||\overrightarrow{v_i}||^3}.$$

Short length initial and final vectors could produce spine curves with large initial and final curvatures. Curvatures not satisfying the restrictions at the endpoints of the spine curve may yield self-intersections of the resulting surface, as it can be seen in Figure 8. In the special case of very close (deeply intersected) spheres and large angle of planes of touching circles the necessarily restricted spine curve cannot be computed and thus our method cannot provide the rotational minimizing blending surface.

The easiest solution for function λ is the Hermite solution of Eqs. (7,8,9,11):

$$\lambda(0) = r_i, \quad \lambda(1) = r_f, \quad \lambda'(0) = -a_i r_i, \quad \lambda'(1) = -a_f r_f,$$

where a_i and a_f are given by Eqs. (10,12). This is

$$\lambda(t) = r_f t^2(3-2t+a_f(1-t)) - r_i(-1+t)^2(-1+(-2+a_i)t).$$

While the results of our method for various initial data can be seen in Figure 7, one can also observe the advantages of this technique in the comparison of the results of the method of [11] and the provided solution (Figure 9). The earlier method may yield unnecessary distortion of the surface, while our rotation minimizing surface has no such distortion. In the sense of rotational minimization, our blending surface is optimal.

5. Conclusion

A novel method of skinning of spheres has been presented where the key step is the rotational minimizing transportation of a vector along the spine curve. By the help of this technique the corresponding points of the touching circles are determined and thus the resulted surface has no unwanted distortions. Due to the time consuming integration of the torsion during the computation, a couple of seconds are required to compute the skin, that is no real time computation is possible, in which sense further improvement

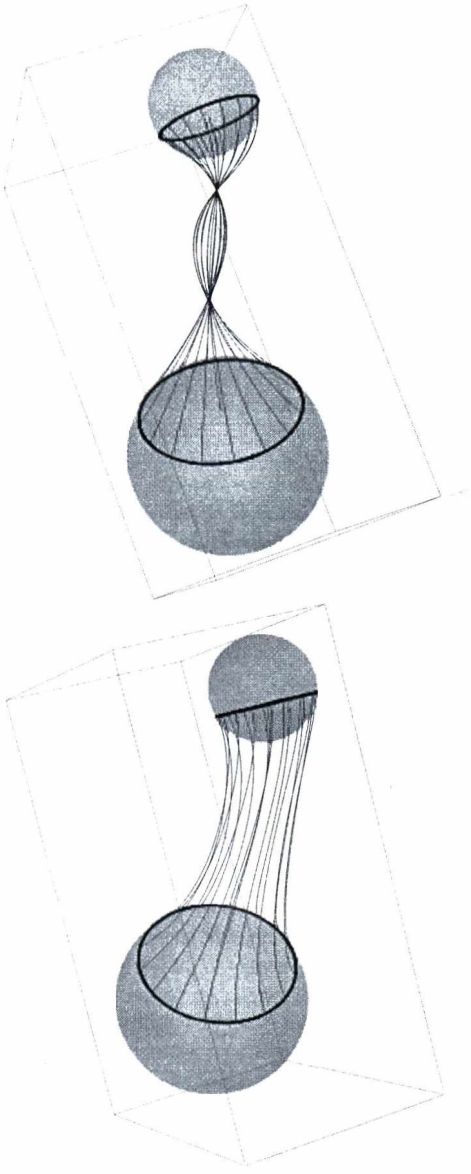


Figure 8: *Curvatures not satisfying the restrictions at the endpoints of the spine curve may yield self-intersections of the resulting surface (left). Smooth blending surface can be obtained for the same initial data with an appropriate choice of spine curve (right).*

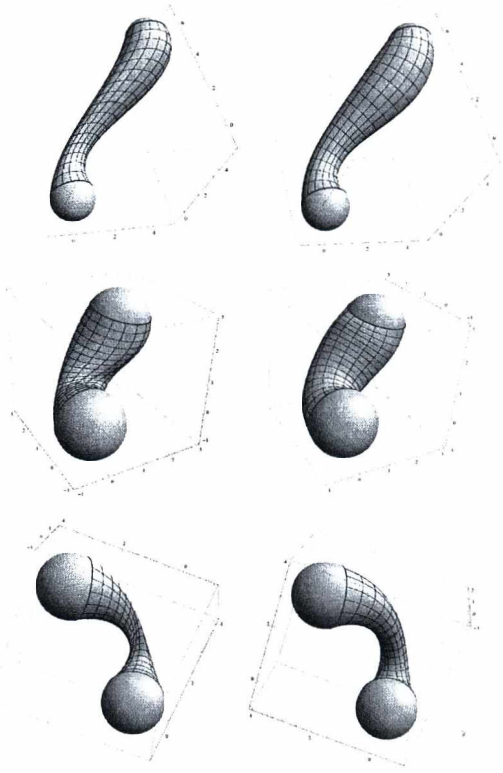


Figure 9: *Comparison of results of the earlier method (left) and our method (right). Surfaces at the right side have visibly less distorted shape.*

may be necessary, perhaps by the help of rational rotation-minimizing frames [6].

6. Acknowledgement

The first author would like to thank the Universitat de València for the warm hospitality during his stay. The second author has been partially supported by grant MTM2009-08933 from the Spanish Ministry of Science and Innovation.

References

1. Bishop, L. R.: There is more than one way to frame a curve. Amer. Math. Monthly 83, 246–251 (1975)
2. Choi, H. I., Kwon, S.-H., Wee, N.-S.: Almost rotation-minimizing rational parametrization of canal surfaces. Comput. Aided Geom. Des. 21, 859–881 (2004)
3. Farouki, R. T.: Pythagorean-Hodograph Curves. Algebra and Geometry inseparable. Springer, Heidelberg (2008)

4. Farouki, R. T.: Quaternion and Hopf map characterizations for the existence of rational rotation-minimizing frames on quintic space curves. Adv. Comput. Math. 33, 331–348 (2010)

5. Farouki, R. T., al-Kandari, M., Sakkalis, T.: Hermite Interpolation by Rotation-Invariant Spatial Pythagorean-Hodograph Curves. Adv. Comput. Math. 17, 369–383 (2002)

6. Farouki, R. T., Giannelli, C., Sestini, A.: Geometric Design Using Space Curves with Rational Rotation-Minimizing Frames. Lecture Notes in Computer Science 5862, 194–208 (2010)

7. Farouki, R. T., Giannelli, C., Sestini, A.: An interpolation scheme for designing rational rotation-minimizing camera motions. Adv. Comput. Math. 38, 63–82 (2013)

8. Farouki, R. T., Han, C. Y.: Rational approximation schemes for rotation-minimizing frames on Pythagorean-hodograph curves. Comput. Aided Geom. Des. 20, 435-Ű454 (2003)

9. Farouki, R. T., Sakkalis, T.: Pythagorean hodographs. IBM Journal of Research and Development 34, 736–752 (1990)

10. Hoffmann, M., Monterde, J.: Blending of spheres by rotational-minimizing surfaces, Advances in Computational Mathematics (submitted)

11. Kunkli, R., Hoffmann, M.: Skinning of circles and spheres. Comput. Aided Geom. Des. 27, 611–621 (2010)

12. Kunkli, R., Kruppa, K., Bana, K., Hoffmann, M.: KSpheres - An Efficient Algorithm for Joining Skinning Surfaces, Computer Aided Geometric design (submitted)

13. Paluszny, M., Boehm, W.: General cyclides. Comput. Aided Geom. Des. 15, 699–710 (1998)

14. Shene, C. K.: Blending two cones with Dupin cyclides. Comput. Aided Geom. Des. 15, 643–673 (1998)

15. Slabaugh, G., Whited, B., Rossignac, J., Fang, T., Unal, G. B.: 3D ball skinning using PDEs for generation of smooth tubular surfaces. Computer-Aided Design 42, 18–26 (2010)

16. Slabaugh, G., Unal, G. B., Fang, T., Rossignac, J., Whited, B., 2008. Variational Skinning of an Ordered Set of Discrete 2D Balls. In: Chen, F., Jüttler, B. (eds.) Proc. of GMP 2008, 450–461 (2008)

17. Wang, W., Jüttler, B., Zheng, D., Liu, Y.: Computation of Rotation Minimizing Frames. ACM Transactions on Graphics 27, article 2 (2008)

Mesh Parameterization with Geometric Constraints

Márton Vaitkus,[1] Tamás Várady[1]

[1] Budapest University of Technology and Economics

Abstract

Mesh parametrization is an important topic in computer graphics and 3D geometric modelling. Methods differ in how the distortion of the parameterized mesh is defined and what sort of optimization methods are applied to map the mesh into the domain in a computationally efficient manner. There exist several emerging applications where in addition to minimizing the distortion, we wish to prescribe the mapping of a few selected feature curves and feature regions, and their relationship. We will evaluate and compare existing methods focusing on their capability of enforcing various geometric constraints. Then we introduce two new algorithms that are suitable to satisfy these demands. The first is an extension of the As-Rigid-As-Possible approach, where we perform optimization and simultaneously maintain constraints. The second method is based on an iterative deformation of an existing mapping until the prescribed constraints get satisfied. Several examples illustrate our approaches comparing mappings without and with constraints.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling — triangular meshes, parameterization, constraints

1. Introduction

Parameterization, or flattening of 3D triangle meshes is a fundamental task in computer graphics and geometry with applications including, but not limited to: texturing, remeshing, surface fitting and mesh repair. The problem, due to its importance and inherent difficulty has been a topic of intense research, which continues to this day, as well.

In mesh parameterization our goal is to map a triangle mesh embedded in 3D to the plane, i.e. compute a pair of coordinates (usually denoted by u and v) for each point on the mesh. For surfaces with disc topology, this mapping can be assumed continuous and piecewise-affine, which means that in practice it is sufficient to determine the coordinates of the mesh vertices. Restricted to a triangle, a parameterization is simply an affine mapping, which can be represented in local coordinates by a 2×2 matrix corresponding to the Jacobian of the function, see Figure 2. The distortion of the parameterization is quantified by considering these Jacobian matrices, which depend linearly on the vertex coordinates.[24]

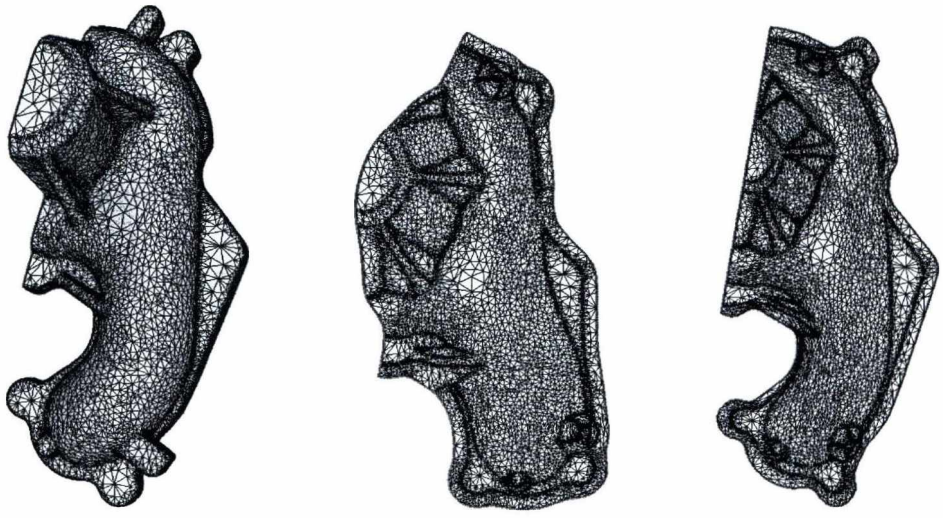
Up until now, the research community has been mainly concerned with minimizing the distortion of the parameterization. A huge variety of distortion measures have been proposed along with efficient algorithms for their minimization. Nevertheless, there exist practical applications, where the demand to preserve and enforce certain geometric constraints is considered even more important than low geometric distortion. A large amount of work has concentrated on constraints involving only discrete vertex positions[11,14] and constant coordinate lines,[5,20] mostly in the context of texture mapping and quadrilateral remeshing, respectively.

In this paper we consider a more general class of high-level parameterization constraints that define how certain entities on the mesh are mapped to the domain. Some important examples of such *geometric constraints* are:

(C1) A sequence of edges is to be mapped to a line.
(C2) Vertices of a closed sequence of edges are to lie on a circle.
(C3) Angles between certain edges are prescribed, including orthogonality or parallelism.
(C4) A feature curve is to preserve its shape.
(C5) A planar or developable region is to preserve its shape.

To our knowledge - despite the vast amount of published research on parameterization - these constraints have not yet considered in such generality; and we are aware of only a few isolated attempts to enforce certain subsets of them.

Hereinafter we assume that the feature curves and the



(a) (b) (c)

Figure 1: Results for an automotive part - (a): Model with constraints (green curve is to be mapped to a line, cyan curve is a planar curve and must preserve its shape, (b): Unconstrained ARAP parameterization, (c): Constrained ARAP parameterization.

boundaries of the features regions are bounded by polylines lying on the mesh. These polylines may not necessarily run on the edges of the triangulation; in these cases a preprocessing phase is performed that splits the triangles involved and embed related edges into the mesh structure.

The paper is structured as follows. First we give an overview of the relevant literature (section 2), then analyze the aforementioned geometric constraints, with the aim of reducing them to elementary relations between mesh elements (section 3). Next, we evaluate known parameterization methods according to their potential, regarding the enforcement of geometric constraints (section 4). Our main contribution is presented in section 5, where we propose an extension to the As-Rigid-As-Possible approach, along with a novel initialization scheme. In section 6, we present another method to enforce geometric constraints on an already computed parameterization, by deforming the planar mesh iteratively.

2. Previous Work

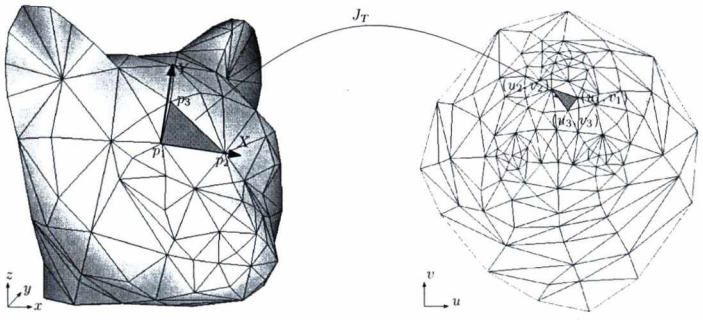
Mesh parameterization has a vast and diverse literature, here we only refer to the comprehensive surveys[24,25] and the references therein.

The majority of commonly used parameterization methods aims at angle preservation, i.e. they compute an optimized *conformal* mapping. This can be done by minimizing (typically quadratic) energies involving vertex positions,[10,15] angles[23,30] or edge lengths.[2,9,19,27]

More general distortion measures typically necessitate computationally expensive non-linear optimization. A notable exception is the application of the As-Rigid-As-Possible mesh deformation algorithm[26] to mesh parameterization,[17] which computes an optimized isometric mapping using an efficient iterative procedure.

We know about only a few, isolated works in the parameterization literature that consider some kinds of geometric constraints. Bennis et al.[4] and Azariadis et al.[1] investigate the problem of preserving the shape (geodesic curvature) of feature curves. Wang et al.[29] and Igarashi et al.[12] give methods that preserve the length of curves in the context of cloth and garment design. Chen et al.[16] preserve of shape of feature curves in a finite element simulation of elastic flattening. Valet and Levy[28] extends the ABF algorithm to handle geometric constraints such as (C1).

Although geometric constraints are relatively new to the subfield of mesh parameterization, they have been thoroughly investigated and applied in geometric modeling. The enforcement of geometric constraints, known as *constrained modeling*, is a quintessential part of the majority of computer-aided design systems. Constrained modeling has a very voluminous literature, and is a topic of ongoing research; we refer to the survey[13] and the references therein. An important difference between our approach and traditional constrained modeling techniques is that we work with unorganized triangle meshes, not the organized models made up of larger scale geometric primitives common in CAD-systems. The constrained fitting problem,[3] arising in the reverse engineering of CAD-models, is another related research field.



(a)

Figure 2: Illustration of mesh parameterization. The parameterization is defined by the vertex coordinates and is a collection of affine functions (J_T), mapping each triangle from a reference position (in local coordinate system $X-Y$) to its final position in the $u-v$ plane.

More closely related is the actively developing field of constrained mesh deformation, see the recent survey[18] and the references therein. Our approach is novel in the sense that some of the features we are considering are not required to be present in the original model in the desired form. One recent work, close to ours in spirit, is that of Bouaziz et al.,[6] where certain kinds of geometric constraints are enforced during the deformation process by a generalization of the local-global algorithm for the minimization of the ARAP energy:[26] first, they fit each constrained subset independently with the required shape, then merge these mesh elements together in a global optimization step.

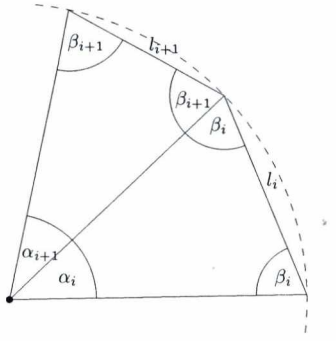


Figure 3: Notations for the circle constraints.

3. Analysis of geometric constraints

Our aim is to reduce the semantic high-level constraints (C1)-(C5) mentioned in section 1, to low-level constraints involving typical optimization variables such as positions or angles. Constraints such as (C1), and (C3) can be decomposed immediately into a set of requirements involving angles between certain edges of the mesh. To do the same for (C2) and (C4) we will need some non-trivial arguments. For (C2) we require that a closed sequence of edges shall map to a closed polyline with a circumscribed circle, i.e. a *cyclic polygon*.

Assume that we have a closed loop made out of N edges on the mesh, with edge lengths $l_0, l_1, \ldots, l_{N-1}$. Now imagine that in the parameterization domain, said vertices lie on a circle, and each of the edges keeps its length or gets scaled by the same factor. For a *cyclic polygon* the side lengths divide the length of the whole polygon in the same way as the respective sector angles divide 2π. Also observe that the triangle corresponding to each sector is equilateral. Then, given two neighboring edges l_i and l_{i+1} (see Figure 3):

$$\frac{l_i + l_{i+1}}{\sum_{i=0}^{N-1} l_i} = \frac{\alpha_i + \alpha_{i+1}}{2\pi}$$

$$\beta_i + \beta_{i+1} = \pi - \frac{\alpha_i + \alpha_{i+1}}{2}$$

Substituting the former equation into the latter, we get the following for the interior angle between the edges:

$$\beta_i + \beta_{i+1} = \pi \left(1 - \frac{l_i + l_{i+1}}{\sum_{i=0}^{N-1} l_i} \right).$$

If the loop is actually a hole loop (interior boundary) of a multiply connected surface, we take the conjugate of these angles. We have made no assumption about the coplanarity of the edges, as for any set of edge lengths (that is possible on meshes), there exists a unique cyclic polygon.[21]

For (C4), i.e. for planar feature curves, the situation is trivial if the edges are exactly coplanar. In other cases, we fit a

plane to the vertices, then project the vertices to the plane spanned by them for the angle calculations.

Constraint (C5) usually requires special care, in a way that, as we will see later, depends on the employed parameterization method.

In summary, all of the constraints we have been considering can be reduced to some combination of the following two kinds of low-level constraints:

- Two edges shall span a prescribed angle and - excluding (C1) and (C3) - their length ratio shall be unchanged.
- A triangle shall keep its shape, i.e. its angles and/or edge lengths.

4. Evaluation of known parameterization methods

Based on the results of the previous section, we can easily add geometric constraints to any parameterization method optimizing directly for vertex positions in the parameter plane, such as DNCP[10] or LSCM.[15] When we require that two edges (whether they share vertex in the mesh or not) shall span an angle in the parameterization, this is equivalent to demanding that one edge (e_{kl}) is the scaled and rotated version of the other (e_{ij}). The scaling can be arbitrary, but the most natural choice is the ratio of the edge lengths, which results in a curve similar to their original counterparts on the 3D mesh. Quantitatively this can be expressed as:

$$u_l - u_k = \frac{|e_{kl}|}{|e_{ij}|}(\cos(\varphi)(u_j - u_i) - \sin(\varphi)(v_j - v_i))$$

$$v_l - v_k = \frac{|e_{kl}|}{|e_{ij}|}(\sin(\varphi)(u_j - u_i) + \cos(\varphi)(v_j - v_i))$$

where u_i, v_i are the coordinates of the vertex i, and φ is prescribed angle between the edges.

The shape of triangles can be preserved exactly by requiring the Cauchy-Riemann equations to be satisfied, but, for conformal methods this has a negligible effect usually, as the mappings have negligible angular distortion in developable regions by default.

We omit results for these algorithms as our implementation for the DNCP method gave degenerate results in cases involving constraints on the boundary. We conjecture that this is a result of a conflict between the Neumann boundary conditions and the constraints, but we have found no straightforward way to remedy the problem.

Angle-based methods, such as ABF[23,30] might appear capable to enforce geometric constraints, as was already noted by Vallet and Lévy,[28] but they have a fundamental limitation, namely that with the exception of (C1) and (C3), geometric constraints cannot be expressed purely in terms of angles. For example: an N-gon has $2N-4$ degrees of freedom, of which we can control only N via the interior angles.

Most of the known position and angle-based methods have a common, serious drawback: they are typically conformal methods, and thus, they only strive to preserve the mesh up to its angles, i.e. up to (local) similarity. As these methods are agnostic to scale, they allow us no direct, tractable control (e.g. expressible via linear equations in the optimization variables) over the metric properties of the parameterization.

We also note that almost all of the published distortion measures can be expressed as functions of the Jacobian singular values, which might appear to give us control over the local scale of the mapping and thus the scale of the features, but as the singular values are in a complicated nonlinear relationship with the actual optimization variables, this is not tractable for practical problem sizes.

There exist a wide class of differential geometric methods optimizing for edge length scale factors,[2,9,19,27] which might appear promising from the viewpoint of geometric constraints. It has turned out that although these methods seem to employ a variational principle, what they optimize is not a distortion measure, but an integrability condition, and their result is fundamentally unique up to the boundary conditions. Thus adding constraints involving the mesh interior would violate the mathematical assumptions underlying these methods and render the problem infeasible.

In summary, we conclude that the only algorithm that is feasible of incorporating all of the constraints under consideration in a computationally tractable way is the As-Rigid-As-Possible parameterization method.

5. As-Rigid-As-Possible Parameterization with Geometric Constraints

5.1. Baseline algorithm

The algorithm based on the iterative minimization As-Rigid-As-Possible (ARAP) energy was first proposed by Liu et al.[17] It builds on earlier work on mesh deformation by Sorkine and Alexa.[26] The proposed distortion measure, the *ARAP energy* measures, per triangle, the distance (in Frobenius norm) of the parameterization (represented by a 2-by-2 Jacobian matrices in local coordinate frames) from a local isometry, i.e. a rotation matrix:

$$\begin{array}{ll} \underset{\text{for } J_T, T \in F}{\text{minimize}} & \Sigma_{T \in F} A_T \, \|J_T - R_T\|_{\mathrm{F}} \\ \text{subject to} & R_T \in SO(2) \end{array}$$

where J_T is the Jacobian matrix for the triangle T.

This is a highly nonlinear, non-convex problem, a locally optimal solution of which can be nonetheless computed quite efficiently using the so-called *local-global algorithm*. Notice that if we keep one term of the energy fixed, while optimizing for the other, the problem simplifies:

- Given a fixed set of Jacobians, i.e. a parameterization, one can compute the closest set of rotations on a per triangle

basis using a simple closed-form formula. As this can be done independently, even parallel for each triangle, this is called the *Local phase*.

- Given a set of rotation matrices, one can fit a set of Jacobians to it in a least squares sense by solving a pair of Poisson equations for the vertex positions:

$$Lu = b_u$$
$$Lv = b_v$$

where L is the usual cotangent Laplacian (i.e. a weighted vertex-vertex adjacency matrix, depending only on the original $3D$ geometry), and b_u (resp. b_v) is the discrete divergence for the vector field given by applying the rotation matrices to the u (resp. v) coordinates (see Liu et al. for details). This is called the *Global phase*.

By iterating between these two phases, one can find a local minima of the ARAP energy, for the effective cost of a single matrix factorization (as the coefficient matrix in the global phase stays constant through the iterations).

5.2. Initialization of rotation matrices

The iterative algorithm described above is usually initialized by computing a set of Jacobians, i.e. a parameterization using some other method, then proceeding with the first local phase. This is obviously difficult if geometric constraints are imposed, and furthermore — as it was observed by Myles and Zorin[19] — initializing the *rotation matrices* and proceeding with the first global step leads to significantly faster convergence in practice. We reformulate the basic idea of Myles and Zorin in a form more suitable to our goals, by appealing to the analogy with vector field design.

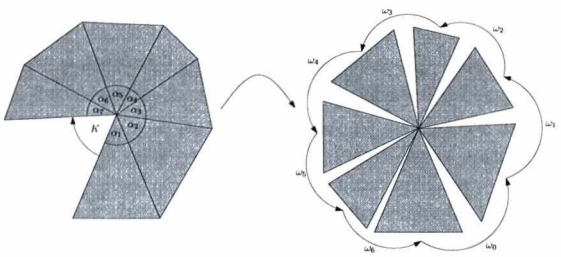


Figure 4: Interpretation of the ARAP initialization algorithm.

For the motivation of our initialization scheme, consider the naive way to compute a set of rotation matrices by isometrically flattening the 3D mesh on a per triangle basis, traversing a spanning tree of the faces starting from a arbitrary root. Obviously, such an approach would result in a highly non-optimal, even degenerate mapping after the subsequent global iteration, as the Gaussian curvature at a vertex is equal to the angular defect for the corresponding triangle fan, i.e. the amount by which the fan fails to close up after isometric flattening. This can be remedied by applying an appropriate amount of extra rotation on each triangle during the traversal of the spanning tree, with the aim of distributing the Gaussian curvature evenly in each triangle fan, see Figure 4. The criterion that in each inner triangle fan the net effect of the rotations shall counteract the Gaussian curvature can be expressed as an underdetermined set of linear equations for the rotation angles, of which we want to compute the solution with the smallest ℓ^2-norm†, a standard optimization problem:

$$\begin{array}{ll} \underset{\text{for } \omega_{ij},\ ij \in E}{\text{minimize}} & \|\omega\|_2^2 \\ \text{subject to} & d_0^{\mathrm{T}}\omega = K \end{array}$$

where d_0 is the vertex-edge adjacency matrix (the exterior derivative for 0-forms[7]), ω is the vector of unknown rotation values for each (dual) edge, and K is the vector of Gaussian curvatures (for the interior vertices).

5.3. Geometric constraints

Geometric constraints must be enforced in each of the iterations. Observe that the actual shape and orientation of the constrained features are determined during initialization, they are just preserved or scaled in the subsequent local and global phases (which is the main reason why one must initialize the rotation matrices instead of the Jacobians).

5.3.1. Initialization phase

As we have an underdetermined linear system we can add any linear equations or split one of the existing ones.

- If two adjacent edges are required to make a prescribed angle, for interior vertices we split the corresponding equation, requiring that the rotations for the dual edges between the two constrained ones result in the given alignment, while assigning constant zero value to the rotations over the constrained edges, see Figure 5. For boundary vertices we simply add an additional constraint. We have observed that for multiply connected meshes, constraints involving inner boundary curves might conflict with the ones prescribing 'zero-curvature'. Thus, we omit the default constraint for hole loops in the presence of a user-defined one.
- When two separate edges are required to span a prescribed angle, we build a dual path (by simple breadth-first search) between them and constrain the sum of rotations accordingly. More precisely, refer to Figure 6: as-

† We note that this is practically equivalent to the method of Crane et al.[8] for computing a *trivial connection*, represented as a discrete differential dual 1-form, i.e. a scalar function on the dual edges, that is the closest to the canonical *Levy-Civita connection* of the surface, for the purposes of vector field generation.

sume that we have two (oriented) edges e_1 and e_2 constrained to span an angle φ, belonging to faces f_1 and f_2, which have, as their basis vectors the (oriented) edges b_1 and b_2. If α and β are the angles between e_1 and b_1 and e_2 and b_2 respectively, our constraint can be expressed as $\beta - \alpha' = \varphi$, where α' is the angle between b_2 and e_1. Along the chosen dual path b_2 is rotated with respect to b_1 by the angles dictated by the isometric flattening, denoted by ω and by the additional unknown rotations we apply along the traversed dual edges; thus, as vectors transform with the inverse of coordinate transforms, these rotations have to be *subtracted* from α, so in summary: $\alpha' = \alpha - \sum\omega$ and after separating the constants and the unknowns, our constraint becomes the following:

$$\sum_{\text{(dual path)}} \omega_{add.} = \varphi - \beta + \alpha - \sum_{\text{(dual path)}} \omega_{isom.}.$$

- For the preservation of planar regions, when an edge belongs to two constrained faces, we consider the corresponding rotation as constant zero and simply remove them from the optimization problem. It might appear natural to do the same for the edges on the boundary of the constrained region, but we have run into numerical stability problems while doing so.

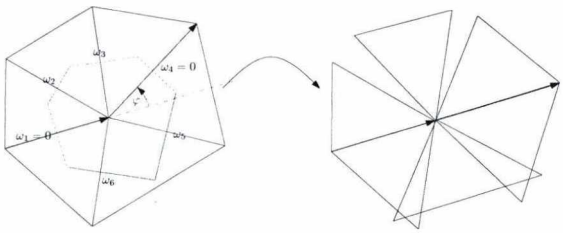


Figure 5: Notations for constraints involving two adjacent edges in ARAP. On the right we illustrate the effect of the constraint, if the prescribed angle is $\varphi = 0$.

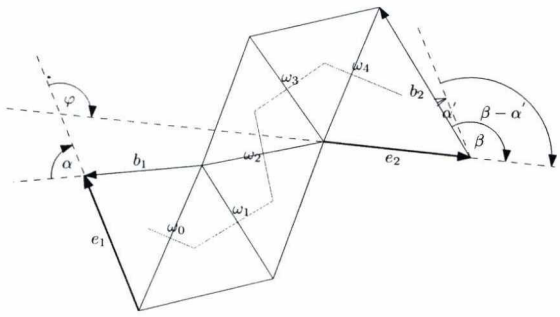


Figure 6: Notations for constraints involving two separate edges in ARAP.

5.3.2. Global phase

First, we recast the problem of solving the linear systems approximating the Poisson equations as minimizations of quadratic forms, then we enforce linear constraints by the use of *Lagrange multipliers* so the optimization problem for the u (resp. v) coordinates

$$\begin{array}{ll} \underset{\text{for } u \in \mathbb{R}^{N_V}}{\text{minimize}} & u^T L u - b_u^T u \\ \text{subject to} & Cu = d_u \end{array}$$

is solved via the symmetric, indefinite linear system

$$\begin{bmatrix} L & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_u \\ d_u \end{bmatrix}.$$

For edge-based constraints, we simply require that the given edges are mapped to the plane exactly with the corresponding rotation matrices. The same constraints could, in principle be used on the edges of preserved regions, but we have found that it is numerically more stable to constrain the Jacobian of each triangle to be equal to the corresponding rotation.

Note that these constraints enforce isometry for said edges or triangles, but the user has the freedom to prescribe arbitrary scaling factors for each feature independently. This approach is not without its drawbacks, however, examine the task shown in Figure 1, where a planar curve constraint and straight line mapping is prescribed. It is obvious, that it is impossible to satisfy both constraints by mapping the boundary edges isometrically. Although one could find appropriate scaling factors heuristically, the general solution is to allow constraints that enforce geometric properties only up to similarity as done in position-based methods, which, however creates a coupling between the u and v coordinates, resulting in a much larger linear system.

As a cotangent Laplacian is used to fit a valid parameterization to the rotations, the resulting map might not be one-to-one. This could happen for a single triangle, or what is more common in the presence of geometric constraints, an entire region could 'spill over' a highly curved, concave boundary. Although there are ways to prevent such artifacts explicitly,[22] we have implemented a less expensive ad-hoc procedure instead: if a triangle reverses its orientation, we modify the Laplacian matrix by adding a positive weight to the entries that correspond to said triangle, and repeat the global phase with the updated matrix, iterating until there are no flipped triangles. With this method we have managed to avoid overlaps completely in all of our examples.

We have also experimented with the more refined weighting scheme in,[5] called *local stiffening*. We have found that while this procedure might be well-suited for the prevention of local, isolated overlaps, it fails to converge in a reasonable time for the more expansive 'spills' we have encountered.

5.3.3. Local phase

To preserve constraints enforced in a preceding phase, we simply ignore those triangles that are part of a preserved region or contain a bounded edge.

6. Enforcement of Geometric Constraints via Iterative Deformation

Previously, we have considered extensions to various parameterization methods. Our next aim is to enforce geometric constraints by the deformation of a previously created, already exisiting parameterization, i.e. a planar mesh. This can be achieved using a two-step iterative process: first isolate the constrained mesh elements and tweak them to the desired shape, then deform the rest of the mesh accordingly.

6.1. Deforming the constrained elements

If the shape is known beforehand, such as in the case of constraints (C4) and (C5), we simply fit said shape to the corresponding parts of the parameterization via standard procrustes analysis, i.e. computing the SVD of the covariance matrix, after removing the mean of the point-sets. In theory the same method could be applied to the cases (C1) and (C2), but instead we have implemented a more robust scheme based on the curvature flow of Crane et al.,[9] which is isometric by construction and is stable for extraordinarily large timesteps. This allows us to perform the deformation gradually in a natural way, enabling more refined user interactions.

6.2. Deforming the parameterization

Knowing the new positions of the constrained vertices, we can easily deform the entire 2D mesh accordingly, by applying the usual local-global ARAP iterations with the positions of the constrained vertices treated as constants in the global phase.

7. Results

Results with the ARAP algorithm for a variety of geometric constraints can be seen in Figure 1, Figure 7, Figure 8 and Figure 9. All results were obtained after initialization and a single global step, with the exception of those of Figure 1 where several global iterations were necessary to resolve overlaps at concave regions of the boundary.

8. Conclusions and Future Work

The enforcement of geometric constraints is an emerging new research area in the context of mesh parameterization. We have analyzed a wide class of geometric constraints and evaluated existing parameterization methods according to their capability to enforce them. We have found that a straightforward extension of the As-Rigid-As-Possible method is capable handling all of the constraints under consideration, and it can also be used to deform an existing parameterization to satisfy geometric constraints.

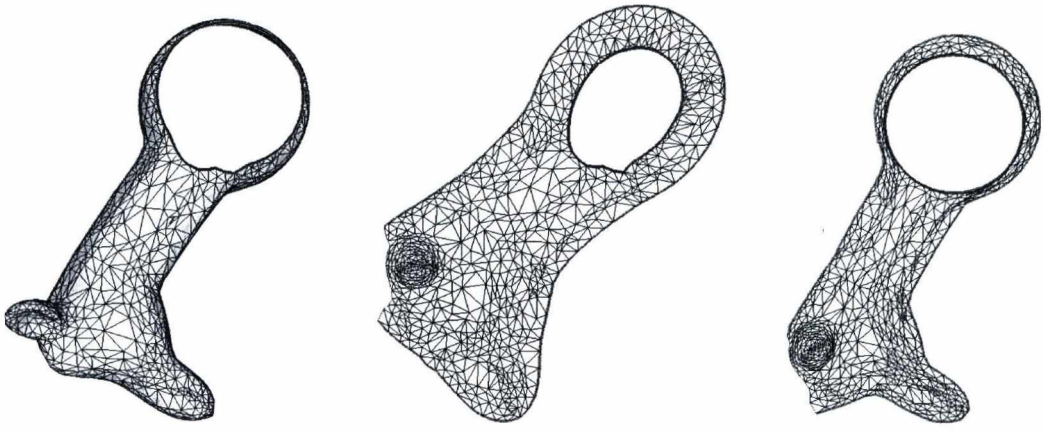
In the future we would like investigate further variations of position-based methods including algorithms based on eigenvalue computations, find ways to optimize edge scaling factors in an explicit way in the context of ARAP, and consider more general constraints e.g. ones represented by inequalities or nonlinear equations. We also plan to apply our algorithms to optimize the parameterization of data points for fitting free-form surfaces.

Acknowledgements

We are grateful to Péter Salvi and Kai Hormann for useful technical discussions. This project was supported by the Hungarian Scientific Research Fund (OTKA No. 101845).

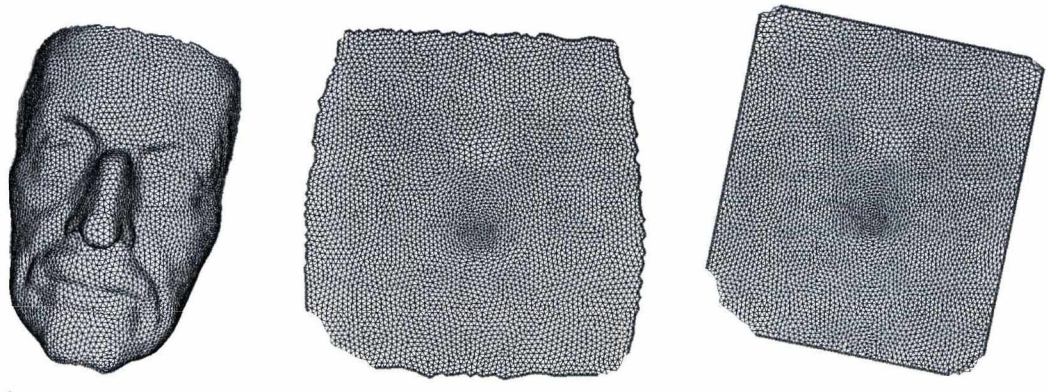
References

1. P. N. Azariadis and Nikos A. Aspragathos. Geodesic curvature preservation in surface flattening through constrained global optimization. *Computer-Aided Design*, 33(8):581–591, 2001.
2. M. Ben-Chen, C. Gotsman, and G. Bunin. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum*, 27(2):449–458, 2008.
3. P. Benkö, G. Kós, T. Várady, L. Andor, and R. Martin. Constrained fitting in reverse engineering. *Computer Aided Geometric Design*, 19(3):173–205, 2002.
4. C. Bennis, J. Vézien, and G. Iglésias. Piecewise surface flattening for non-distorted texture mapping. *ACM SIGGRAPH Computer Graphics*, 25(4):237–246, 1991.
5. D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. *ACM Transactions on Graphics (TOG)*, 28(3):77, 2009.
6. S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, and M. Pauly. Shape-up: Shaping discrete geometry with projections. *Computer Graphics Forum*, 31(5):1657–1667, 2012.
7. K. Crane, F. de Goes, M. Desbrun, and P. Schröder. Digital geometry processing with discrete exterior calculus. *ACM SIGGRAPH 2013 courses*, 2013.
8. K. Crane, M. Desbrun, and P. Schröder. Trivial connections on discrete surfaces. *Computer Graphics Forum*, 29(5):1525–1533, 2010.
9. K. Crane, U. Pinkall, and P. Schröder. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG)*, 2013.



(a) (b) (c)

Figure 7: Results for Giraffe model - (a): Model with constraints (magenta curve is to be mapped to a circle, cyan curve is planar and must preserve its shape), (b): Unconstrained ARAP parameterization, (c): Constrained ARAP parameterization.



(a) (b) (c)

Figure 8: Results for Maxface model - (a): Model with constraints (green curves are to be mapped to a line, and be perpendicular), (b): Unconstrained ARAP parameterization, (c): Constrained ARAP parameterization.

(a) (b) (c)

Figure 9: Results for Darmstadt model - (a): Model with constraints (green planar regions are to preserve their shape), (b): Unconstrained ARAP parameterization, (c): Constrained ARAP parameterization.

10. M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21(3):209–218, 2003.
11. I. Eckstein, V. Surazhsky, and C. Gotsman. Texture mapping with hard constraints. *Computer Graphics Forum*, 20(3):95–104, 2001.
12. Y. Igarashi, T. Igarashi, and H. Suzuki. Interactive cover design considering physical constraints. *Computer Graphics Forum*, 28(7):1965–1973, 2009.
13. C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: a survey. *International Journal of Computational Geometry & Applications*, 16(05–06):379–414, 2006.
14. V. Kraevoy, A. Sheffer, and C. Gotsman. Matchmaker: constructing constrained texture maps. *ACM Transactions on Graphics (TOG)*, 22(3):326–333, 2003.
15. B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics (TOG)*, 21(3):362–371, 2002.
16. Wen liang Ch., Peng W., and Yidong B. Surface flattening based on linear-elastic finite element method. 5(7):728 – 734, 2011.
17. L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler. A local/global approach to mesh parameterization. *Computer Graphics Forum*, 27(5):1495–1504, 2008.
18. N. Mitra, M. Wand, H. Zhang, D. Cohen-Or, and M. Bokeloh. Structure-aware shape processing. *Eurographics 2013-State of the Art Reports*, pages 175–197, 2012.
19. A. Myles and D. Zorin. Global parametrization by incremental flattening. *ACM Transactions on Graphics (TOG)*, 31(4):109, 2012.
20. A. Myles and D. Zorin. Controlled-distortion constrained global parametrization. *ACM Transactions on Graphics (TOG)*, 32(4):105, 2013.
21. I. Pinelis. Cyclic polygons with given edge lengths: existence and uniqueness. *Journal of Geometry*, 82(1-2):156–171, 2005.
22. C. Schüller, L. Kavan, D. Panozzo, and O. Sorkine-Hornung. Locally injective mappings. *Computer Graphics Forum*, 32(5):125–135, 2013.
23. A. Sheffer and E. de Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers*, 17(3):326–337, 2001.
24. A. Sheffer, K. Hormann, and B. Lévy. Mesh parameterization: Theory and practice. *ACM SIGGRAPPH, course notes*, 2007.
25. A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision*, 2(2):105–171, 2006.
26. O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *ACM International Conference Proceeding Series*, volume 257, pages 109–116. ACM, 2007.
27. B. Springborn, P. Schröder, and U. Pinkall. Conformal equivalence of triangle meshes. *ACM Transactions on Graphics (TOG)*, 27(3):77, 2008.
28. B. Vallet and B. Lévy. What you seam is what you get. Technical report, INRIA - ALICE Project Team, 2009.
29. C. Wang. Wirewarping: A fast surface flattening approach with length-preserved feature curves. *Computer-Aided Design*, 40(3):381–395, 2008.
30. R. Zayer, B. Lévy, and H. P. Seidel. Linear angle based parameterization. In *Fifth Eurographics Symposium on Geometry Processing-SGP 2007*, pages 135–141, 2007.

Creating connection between skinning surfaces

Kinga Kruppa[1], Kornél Bana[1], Roland Kunkli[1] and Miklós Hoffmann[2]

[1] Department of Computer Graphics and Image Processing, Faculty of Informatics, University of Debrecen, Debrecen, Hungary
[2] Institute of Mathematics and Informatics, Eszterházy Károly University College, Eger, Hungary

Abstract

In this paper we present a new method and a software with which we can easily create smooth connection between skinning surfaces based on spheres. This type of techniques becomes more and more popular nowadays, well-known CAD software use methods which are based on this idea. But the used algorithms has several drawbacks. In the paper we clearly identify these problems and show that our method can handle them.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

Interpolation of geometric data sets is of central importance in computer aided geometric design. Nowadays there is a growing demand to extend the known methods based on points to new methods which rely on geometric objects of different types such as circles and spheres.

Surface modeling based on such principles has already appeared in computer graphics and in the last few years several scientific papers have been published. Popular CAD softwares (e.g. ZBrush®[15], Spore™[14]) seem to apply similar methods and these implementations are available for everyday users. The methods used in these applications, however, do not provide results good enough especially in those circumstances when the joining of skinned spheres is aimed.

Joining of given geometric objects is an important part of surface modeling since in this way more complicated objects can be generated than the starting forms. Therefore there is a persistent demand from both designers and users to develop methods to give refined and better results.

Our aim in this research was to give better algorithms than the presently available ones. More precisely, we aimed to join complicated topological surfaces, sharp branches and to handle random set of spheres. We concentrated on overcoming the problem of joining of surfaces. At the first phase of our research we have managed to give an efficient method for the joining and branching problem. Our results are also implemented in our newly developed, user friendly modeling software based on projective geometry.

2. Related Work

Blending is a well-known technique to create connection between two separated surfaces. We can read about this problem (blending of curved objects) in several papers[12, 13, 1], but neither can solve the problem generally. Blending surfaces are often constructed only between boundary curves of the two given surfaces[3, 6].

If we consider a set of spheres instead of two surfaces, the problem becomes more difficult, because we should handle also the relative positions of the spheres and we also need to determine the possible touching curves if we haven't got any additional information, like in our case.

G. Slabaugh has solved the problem by an iterative way[9]. For this method the user needs to determine an initial position of the blending surface, then the final output comes from an energy minimization process based on partial differential equations. It is very important to note that the convergency is not proved in this case, and miserably chosen inputs can result unacceptable outputs. Solving the necessary differential equations could take seconds for a surface.

R. Kunkli and M. Hoffmann has published a method[5] which uses direct computation to get the touching circles and the final surface, avoiding iterative ways. This algorithm can handle extreme situations too, and it can be used for interactive graphical softwares, because it works in real time owing to the fast computation which is based on classical geometric results. R. Kunkli, M. Hoffmann and R. Tornai analyzed the results[5, 11] and they has concluded that this mentioned

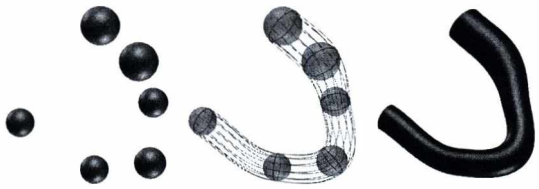


Figure 1: An example output of the original method[5].

method can provide smoother outputs than the algorithms which are based on iterative techniques.

The method presented in this paper is based on the idea of this mentioned algorithm from 2010, but it is extended to complicated shapes with several branches, which can help the creation of more structured tubular surfaces or animation characters. In the following section we briefly describe the steps of this algorithm[5].

3. The basic technique for skinning

Given a sequence of spheres $s_1, s_2, s_3, \ldots, s_n$ with centers $\mathbf{o}_i$, $(i = 1, \ldots, n)$, we are looking for a G^1 continuous parametric surface $\mathbf{s}(\phi, t)$ (called *skin*) of the given spheres satisfying the following requirements:

- There is a circle of contact (touching circle) c_i on the sphere s_i for all $i = 1, \ldots, n$ such that the skin $\mathbf{s}(\phi, t)$ and sphere s_i have common tangent planes at each point of c_i. Circle c_i is an isoparametric curve of $\mathbf{s}(\phi, t)$
- c_i is not contained or intersected by any other sphere than s_i for all $i = 1, \ldots, n$.

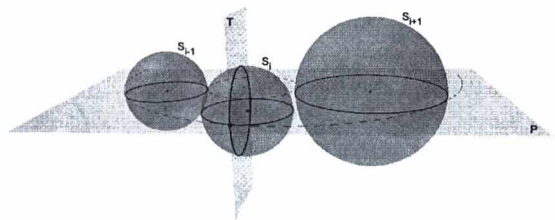


Figure 2: This picture shows how we can determine the touching circle (blue) on the middle sphere. The solutions of the considered Apollonian problem are noted by red. The plane of the blue circle is orthogonal to the plane of the centers of the basic circles.

At first we have to determine the touching circle c_i on the sphere s_i with center $\tilde{\mathbf{o}}_i$ and radius $\tilde{r}_i$ $(i = 1, \ldots, n)$ (c.f. Figure 2). For the moment, let us consider the spheres s_i, where $i = 2, \ldots, n-1$, that is we exclude the first and the last spheres. Consider the plane P_i determined by the centers $\mathbf{o}_{i-1}, \mathbf{o}_i, \mathbf{o}_{i+1}$ of the considered sphere and its direct neighbours. Intersecting the spheres by this plane we obtain three circles. With the help of two suitable Apollonius circles, that is circles touching all the three given circles from outside or inside, respectively (see the dashed red circles in Figure 2), we can find two points in the middle circle. There exists exactly one plane T_i $(i = 2, \ldots, n-1)$ passing through these two points and being orthogonal to plane P_i of the centers of the spheres. The intersection of sphere s_i and this orthogonal plane T_i (blue circle in Figure 2) will be the touching circle of the future skinning surface and the sphere s_i. Note, that the touching circle we constructed now is identical to the one in which the Dupin cyclide defined by three given spheres s_{i-1}, s_i, s_{i+1} touches the sphere s_i.

We can localize a circle by this method on every sphere except the first and the last one, s_1 and s_n, where one of the neighbours is evidently missing. In these two cases regular touching cones of spheres s_1 and s_2, and s_{n-1} and s_n are considered, respectively. One can easily determine the touching circles on s_1 and s_n of these cones, which circles will be considered as the touching circles of the blending surface on these spheres. Note, that the touching circles are not necessarily great circles of the given spheres.

Once the touching circles with center $\tilde{\mathbf{o}}_i$ and radii $\tilde{r}_i$ on each sphere are obtained, one can start to create the skin by patches defined successively to each pair of spheres using Hermite interpolants through corresponding points of the touching circles.

The future blending patch $\mathbf{s}_i(\phi, t)$ of the skin between touching circle c_i on sphere s_i and touching circle c_{i+1} on sphere s_{i+1} is computed as follows. Circle c_i is the isoparametric curve $\mathbf{s}_i(\phi, 0)$, while c_{i+1} is the isoparametric curve $\mathbf{s}_i(\phi, 1)$ of this patch. At first we define the starting point on c_i as $\mathbf{z}_i = \mathbf{s}_i(0,0)$ and on c_{i+1} as $\mathbf{z}_{i+1} = \mathbf{s}_i(0,1)$. Then rotating them by the same angle ϕ along the circles, corresponding pairs of points $\mathbf{z}_i(\phi) = \mathbf{s}_i(\phi, 0)$, $\mathbf{z}_{i+1}(\phi) = \mathbf{s}_i(\phi, 1)$ are defined.

Tangent directions $\mathbf{w}_i - \mathbf{z}_i(\phi)$ at the endpoints are inherited from the touching cones of the spheres, where $\mathbf{w}_i$ is the apex of the regular cone touching the sphere s_i at the circle c_i. These cones can be degenerated to regular cylinders if the touching circle c_i is a great circle of the sphere s_i, in this case the tangent directions are all parallel to the reguli of the cylinder. Suitable lengths l_i of tangent vectors are computed by the help of the radical plane M_i of the two spheres, more precisely by the help of the distance $\mathrm{d}(M_i, \mathbf{z}_i(\phi))$ of the actual point and the radical plane. To avoid unnecessary torsion of the future surface patch, corresponding points are associated to each other by the help of a fixed spatial direction $\mathbf{e}$, which can be e.g. the direction of the z axis, not parallel to any of the vectors $\mathbf{w}_i - \mathbf{o}_i$, or

$$s(x) = \begin{cases} -1, & \text{if } x < 0,; \\ 1, & \text{else} \end{cases}$$

$$p(s_i) = \begin{cases} s\left(\left\langle \frac{\mathbf{w}_i - \mathbf{o}_i}{\|\mathbf{w}_i - \mathbf{o}_i\|}, \frac{\mathbf{o}_{i+1} - \mathbf{o}_i}{\|\mathbf{o}_{i+1} - \mathbf{o}_i\|} \right\rangle\right), & \text{if } i \neq n; \\ s\left(\left\langle \frac{\mathbf{w}_i - \mathbf{o}_i}{\|\mathbf{w}_i - \mathbf{o}_i\|}, \frac{\mathbf{o}_i - \mathbf{o}_{i-1}}{\|\mathbf{o}_i - \mathbf{o}_{i-1}\|} \right\rangle\right), & \text{else} \end{cases}$$

where $\langle , \rangle$ is the standard inner product.

Now let $\mathbf{z}_i$ be defined as

$$\mathbf{z}_i = \tilde{\mathbf{o}}_i + \tilde{r}_i \cdot \frac{\mathbf{e} \times (p(s_i) \cdot (\mathbf{w}_i - \tilde{\mathbf{o}}_i))}{\|\mathbf{e} \times (p(s_i) \cdot (\mathbf{w}_i - \tilde{\mathbf{o}}_i))\|} \qquad (i = 1, 2, \ldots, n).$$

where $\tilde{r}_i$ is the radius of the touching circle c_i. Further corresponding points $\mathbf{z}_i(\phi)$ of the touching circles are defined by rotating $\mathbf{z}_i$ by angle ϕ around the line passing through $\tilde{\mathbf{o}}_i$ and having direction $p(s_i) \cdot (\mathbf{w}_i - \tilde{\mathbf{o}}_i)$.

The tangent vectors at the endpoints are defined as follows:

$$\mathbf{v}_i(\phi) = 2 \cdot p(s_i) \cdot d(M_i, \mathbf{z}_i(\phi)) \cdot \frac{\mathbf{w}_i - \mathbf{z}_i(\phi)}{\|\mathbf{w}_i - \mathbf{z}_i(\phi)\|}$$

and

$$\mathbf{v}_{i+1}(\phi) = 2 \cdot p(s_{i+1}) \cdot d(M_i, \mathbf{z}_{i+1}(\phi)) \cdot \frac{\mathbf{w}_{i+1} - \mathbf{z}_{i+1}(\phi)}{\|\mathbf{w}_{i+1} - \mathbf{z}_{i+1}(\phi)\|}.$$

Finally the blending surface patch $\mathbf{s}_i(\phi, t)$ between two neighboring spheres s_i and s_{i+1} is as follows

$$\begin{aligned} \mathbf{s}_i(\phi, t) = & H_0^3(t)\mathbf{z}_i(\phi) + H_1^3(t)\mathbf{z}_{i+1}(\phi) \\ & + H_2^3(t) \cdot \mathbf{v}_i(\phi) + H_3^3(t) \cdot \mathbf{v}_{i+1}(\phi) \end{aligned} \qquad (1)$$

where $H_j^3(t)$, $(j = 0, 1, 2, 3)$ are the cubic Hermite functions. The algorithm assures G^1 continuity between the blending patches. For further details of the algorithm see the mentioned paper[5].

4. New branches

4.1. The construction of the new touching circle

Let us consider three neighbouring spheres s_{i-1}, s_i, s_{i+1} from the original sequence and assume that we would like to connect a new branch starting at s_i. Let p_{ij} denote the jth sphere in the new branch, that is $p_{i1} = s_i$ $(i \in [1, n], j \in [1, m], n, m \in \mathbb{N})$.

At first we determine a new touching circle on s_i from where the new branch can start. For this purpose we apply the basic algorithm for sphere triplets s_{i-1}, s_i, p_{i2} and s_{i+1}, s_i, p_{i2}, respectively, to obtain two circles on s_i, c_{i1} and c_{i2} (Figure 3).

The new touching circle (let us denote it by c_i') is fitting on the common points of c_{i1} and c_{i2}. We can determine its normal vector $\mathbf{n}'_i$ as the sum of the normalized normal vectors

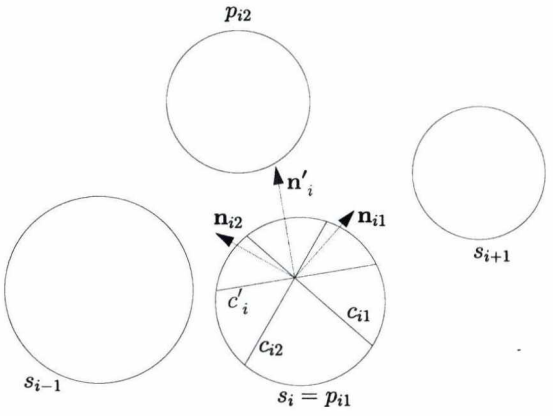


Figure 3: Constructing new touching circle (red) for the joining branch.

of c_{i1} and c_{i2}, $\mathbf{n}_{i1}$, $\mathbf{n}_{i2}$, respectively. In most cases the common points of c_{i1} and c_{i2} exist, if not, then we can consider the plane passing through the center of c_{i1} and having normal vector $\mathbf{n}'_i$. The intersection of this plane and the sphere p_{i1} will be the circle c_i' in question.

So we have constructed a new touching circle on s_i with a method which is sensitive to its neighbours. After this step we can easily construct a new branch starting from $s_i = p_{i1}$ with the help of the basic algorithm, blending the spheres p_{ij}, $(j = 1, \ldots, m)$. This way the branch surfaces will not be connected smoothly, but they will have an intersection curve, as one can observe in Figure 7, where we did not applied the smoothing algorithm described in the next section.

In the following part we describe how we can achieve a G^1 continuous connection of the branches. For this purpose we create a boundary curve on the basic branch and the new branch will start from this curve.

4.2. The construction of the boundary curve

At first we determine a point $\mathbf{m}_i$ on circle c_i which is the original touching circle for the basic branch. This point will be the so-called "midpoint" of the boundary curve.

To define point $\mathbf{m}_i$, let $\mathbf{n}_i$ denote the normal vector of c_i, $||\mathbf{n}_i|| = 1$. We would like to find vector $\mathbf{h}_i$ such that

$$\mathbf{n}'_i = \mathbf{h}_i + \lambda \cdot \mathbf{n}_i \quad \text{and} \quad \langle \mathbf{h}_i, \mathbf{n}_i \rangle = 0,$$

where $\lambda \in \mathbb{R}$.

From equation $\langle \mathbf{n}'_i - \lambda \cdot \mathbf{n}_i, \mathbf{n}_i \rangle = 0$ we can easily calculate the value of λ, so $\mathbf{h}_i$ can be determined as $\mathbf{h}_i = \mathbf{n}'_i - \lambda \cdot \mathbf{n}_i$. After this step $\mathbf{h}_i$ can be used to describe vector $\mathbf{m}_i$:

$$\mathbf{m}_i \doteq \tilde{\mathbf{o}}_i + \tilde{r}_i \cdot \frac{\mathbf{h}_i}{||\mathbf{h}_i||}.$$

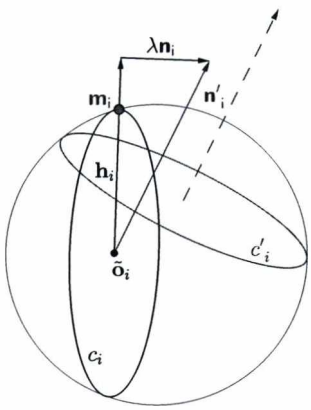


Figure 4: Constructing the center of the boundary curve.

Practically this is an orthogonal projection of a special representant of $\mathbf{n}_i'$ to the plane of c_i (for the notations see Figure 4).

Now we can define a continuous boundary curve on the blending surface of the basic branch. It is clear that $\mathbf{m}_i$ fits on c_i such as $\mathbf{z}_i$. Based on the notes of the paper[5] which contains the basic algorithm and Section 3 we can determine an angle $\alpha_i \in [0, 2\pi]$ for $\mathbf{m}_i$ that $\mathbf{z}_i(\alpha_i) = \mathbf{m}_i$.

Let us consider the following curve:

$$\mathbf{L}_{i1}(\theta) = H_0^3(t_0)\,\mathbf{z}_i(\alpha_i+\theta) + H_1^3(t_0)\,\mathbf{z}_{i+1}(\alpha_i+\theta) + H_2^3(t_0)\cdot \mathrm{p}(s_i)\cdot 2\cdot \mathrm{d}(M_i, \mathbf{z}_i(\alpha_i+\theta))\cdot \frac{\mathbf{w}_i - \mathbf{z}_i(\alpha_i+\theta)}{\|\mathbf{w}_i - \mathbf{z}_i(\alpha_i+\theta)\|} + H_3^3(t_0)\cdot \mathrm{p}(s_{i+1})\cdot 2\cdot \mathrm{d}(M_i, \mathbf{z}_{i+1}(\alpha_i+\theta))\cdot \frac{\mathbf{w}_{i+1} - \mathbf{z}_{i+1}(\alpha_i+\theta)}{\|\mathbf{w}_{i+1} - \mathbf{z}_{i+1}(\alpha_i+\theta)\|},$$

where $t_0 = \frac{1}{\pi}\sqrt{(\frac{\pi}{4})^2 - \theta^2}$ (based on the equation $y = \sqrt{r^2 - x^2}$ of a semicircle with radius r and centered at the origin), $\theta \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$. This curve will be one arc of the boundary curve between s_i and s_{i+1}. The second arc is defined from s_i to s_{i-1} by

$$\mathbf{L}_{i2}(\theta) = H_0^3(1-t_0)\,\mathbf{z}_{i-1}(\alpha_i+\theta) + H_1^3(1-t_0)\,\mathbf{z}_i \cdot (\alpha_i+\theta) + H_2^3(1-t_0)\cdot \mathrm{p}(s_{i-1})\cdot 2\cdot \mathrm{d}(M_{i-1}, \mathbf{z}_{i-1}\cdot (\alpha_i+\theta))\cdot \frac{\mathbf{w}_{i-1} - \mathbf{z}_{i-1}(\alpha_i+\theta)}{\|\mathbf{w}_{i-1} - \mathbf{z}_{i-1}(\alpha_i+\theta)\|} + H_3^3(1-t_0)\cdot \mathrm{p}(s_i)\cdot 2\cdot \mathrm{d}(M_{i-1}, \mathbf{z}_i(\alpha_i+\theta))\cdot \frac{\mathbf{w}_i - \mathbf{z}_i(\alpha_i+\theta)}{\|\mathbf{w}_i - \mathbf{z}_i(\alpha_i+\theta)\|},$$

where t_0 and θ has the same value as above. The two arcs of the boundary curve can be seen in Figure 5.

As we have mentioned previously, with the help of the basic algorithm and the new touching circle c_i' on sphere s_i we can determine touching circles on the spheres of the new branch. So to create a G^1 continuous connection from the boundary curve we have to consider its points and assign endpoints on the touching circle of p_{i2} to them. Let us denote this circle by c_{i2}.

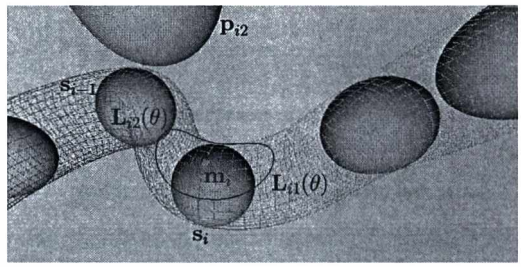


Figure 5: The boundary curve.

With the above mentioned technique based on orthogonal projection we can localize a matching point for $\mathbf{L}_{i1}(0)$ by projecting vector $\mathbf{L}_{i1}(0) - \mathbf{m}_i$ (the starting point is the center of c_i') onto the plane of c_{i2}. This point will be the endpoint of the Hermite arc starting at $\mathbf{L}_{i1}(0)$. Then with rotations by angles between 0 and 2π the first part of the blending surface of the new branch can be constructed from s_i to p_{i2} analogously to the basic algorithm (see Figure 6).

To compute the tangent vectors at the points of the boundary curve, the tangent plane of the original surface $\mathbf{s}_i(\phi, t)$ has to be computed first. The partial derivatives of the surface (1) are as follows

$$\frac{\partial}{\partial\phi}\mathbf{s}_i(\phi,t) = H_0^3(t)\,\dot{\mathbf{z}}_i(\phi) + H_1^3(t)\,\dot{\mathbf{z}}_{i+1}(\phi) + H_2^3(t)\cdot\dot{\mathbf{v}}_i(\phi) + H_3^3(t)\cdot\dot{\mathbf{v}}_{i+1}(\phi)$$

$$\frac{\partial}{\partial t}\mathbf{s}_i(\phi,t) = \frac{d}{dt}H_0^3(t)\,\mathbf{z}_i(\phi) + \frac{d}{dt}H_1^3(t)\,\mathbf{z}_{i+1}(\phi) + \frac{d}{dt}H_2^3(t)\cdot\mathbf{v}_i(\phi) + \frac{d}{dt}H_3^3(t)\cdot\mathbf{v}_{i+1}(\phi).$$

The normal vector of the tangent plane will be the cross product of the partial derivatives.

Now we use the described orthogonal projection again to create tangent vectors at each point of the boundary curve.

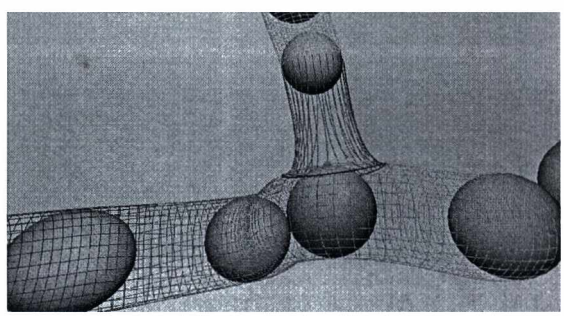


Figure 6: Connection of two branches. Hermite arcs (isoparametric curves of the patch), starting at the boundary curve can be seen.

For this purpose we project orthogonally the vector $\mathbf{L}_{i1}(\theta) - \mathbf{m}_i$ onto the tangent plane at $\mathbf{L}_{i1}(\theta)$ for each $\theta \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$. The length of the tangent vector is the distance of the point from the radical plane of the two spheres.

5. Results and comparison with other methods

Based on the theoretical results described above, an easy-to-use software tool is provided to create surfaces and characters by spheres. We can define spheres, adjust their positions and radii, and choose a sphere from where a new branch will start. The blending surface is computed in real-time, automatically, with several possibilities of modification (colour, rendering etc.). As we have mentioned, this kind of tools have already been introduced in computer graphics, but in several cases our method provides better results, especially in terms of connection of branches. This problem of smooth connection is especially noteworthy when branches meet at small spheres, that is the radius of the sphere $p_{i1} = s_i$ is much smaller than the radii of s_{i-1}, s_{i+1} and p_{i2}. It can cause unwanted and sometimes unacceptable forms in other softwares, while our method is not sensitive to the suddenly changed radii of the given spheres. If the new branch starts at a relatively small sphere, then the obtained branches can be connected in an unpredictable manner (Figure 7). In these cases our software provides a more natural connecting patch. In case of Spore™, the other alternative of sphere based modeling tools, the smooth connection of branches is not everywhere solved in a satisfactory way, the surface can have crisps or sharp edges at this point (see Figure 8), while our method can provide smooth connection of different branches.

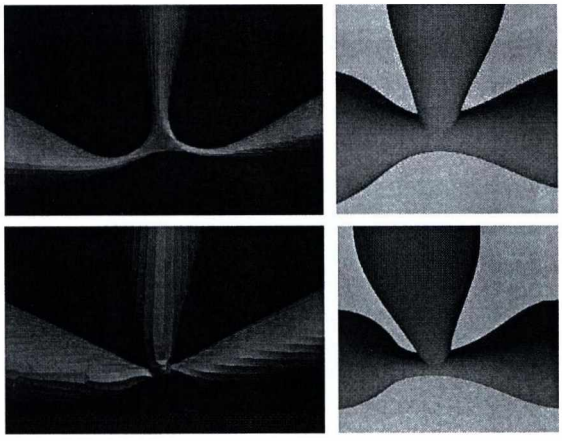


Figure 7: If the radii of spheres are drastically changed around the connection, ZSpheres® surfaces (left) can have unpredictable behaviour at the join. Our method (right) can handle this problem (branch connection without smoothing).

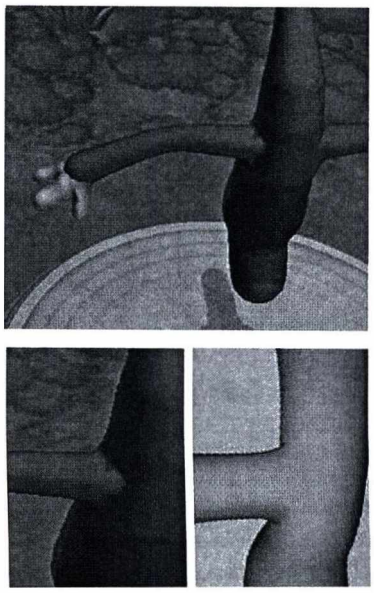


Figure 8: In Spore™, connection of branches are less attractively solved (above and below left). Our software provides much smoother (G^1 continuous) connection (below right).

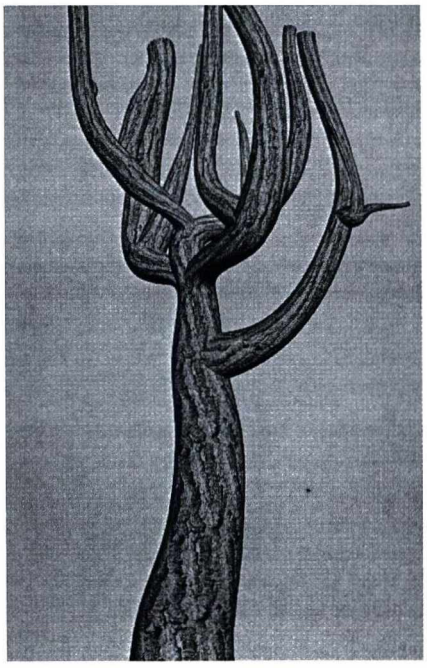


Figure 9: Our method can handle several branches and multiple connections as well.

6. Conclusion

Sphere based modeling is a real alternative of point based techniques in fast, interactive surface design. As one can observe using available methods and softwares, the two crucial steps of this paradigm are the smooth blending of two neighbouring spheres and the correct connection of branches. Both are solved in this paper, with sufficient results in those cases as well where existing methods provide less perfect solutions.

Directions of future improvement of this method may include the optimization of the software, and the incorporation of further user-friendly tools.

Acknowledgement

This research was supported by the **European Union** and the **State of Hungary, co-financed by the European Social Fund** in the framework of TÁMOP 4.2.4.A/2-11-1-2012-0001 'National Excellence Program'.

References

1. K.C. Hui and Y.H. Lai. Smooth blending of subdivision surfaces. *Computer-Aided Design*, **38**:786–799, 2006.
2. K.S. Karan and E. Kokkevis. Skinning Characters using Surface-Oriented Free-Form Deformations. *Proc. of Graphics Interface 2000*, pp. 35–42, 2000.
3. K. Karčiauskas and R. Krasauskas. Rational rolling ball blending of natural quadrics. *Mathematical Modelling and Analysis*, **5**:97–107, 2000.
4. R. Kunkli. Localization of touching points for interpolation of discrete circles. *Annales Mathematicae et Informaticae*, **36**:103–110, 2009.
5. R. Kunkli and M. Hoffmann. Skinning of circles and spheres. *Computer Aided Geometric Design*, **27**:611–621, 2010.
6. M. Paluszny and F. Tovar. Envelopes and tubular splines. *Mathematics and Computers in Simulation*, **79**:1971–1976, 2009.
7. J. Rossignac and J. J. Kim. HelSweeper: Screw-sweeps of canal surfaces. *Computer-Aided Design*, **44**:113–122, 2012.
8. J. Rossignac, B. Whited, G. Slabaugh, T. Fang and G. Unal. Pearling: 3d interactive extraction of tubular structures from volumetric images. *Proc. of MICCAI Workshop: Interaction in Medical Image Analysis and Visualization*, 2007.
9. G. Slabaugh, J. Rossignac, B. Whited, T. Fang and G. Unal. 3D Ball Skinning using PDEs for Generation of Smooth Tubular Surfaces. *Computer-Aided Design*, **42**:18–26, 2010.
10. G. Slabaugh, G. Unal, T. Fang, J. Rossignac and B. Whited. Variational Skinning of an Ordered Set of Discrete 2D Balls. *Lecture Notes on Computer Science*, **4795**:450–461, 2008.
11. Robert Tornai. Measurement of visual smoothness of blending curves. *Annales Mathematicae et Informaticae*, **40**:155–160, 2012.
12. P. Zhou and W.-H. Qian. Blending multiple parametric normal ringed surfaces using implicit functional splines and auxiliary spheres. *Graphical Models*, **73**:87–96, 2011.
13. P. Zhou and W.-H. Qian. Gn-blending of multiple parametric normal ringed surfaces by adding implicit closings Gn-continuous with the surfaces. *Computers and Graphics* , **36**:297–304, 2012.
14. Spore. `www.spore.com` [computer program], Accessed January 10, 2014.
15. ZSpheres. `pixologic.com/zbrush/features/ZSpheres/` [computer program], Accessed January 10, 2014.

A generalization of the Overhauser spline

Imre Juhász[1] and Ágoston Róth[2]

[1] Department of Descriptive Geometry, University of Miskolc, H-3515 Miskolc-Egyetemváros, Hungary
[2] Department of Mathematics and Computer Science, Babeş–Bolyai University, RO-400084 Cluj-Napoca, Romania

Abstract

We propose a method for the interpolation of a given sequence of data points with C^n continuous trigonometric spline curves of order $n+1$ ($n \geq 1$) by blending elliptical arcs. Explicit formulae are provided for the computation of control points of the interpolating arcs that depend on a global parameter $\alpha \in (0,\pi)$ enabling global shape modifications. The proposed scheme is a generalization of the Overhauser spline that blends parabolic arcs, and includes a C^n Bézier spline interpolation method as the limiting case $\alpha \to 0$.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, J6 [Computer-Aided Engineering]: Computer-Aided Design

1. Introduction

The standard description form of curves in nowadays CAD and CAGD is

$$\mathbf{c}(t) = \sum_{i=0}^{n} F_i(t)\,\mathbf{d}_i,\ t \in [a,b] \subset \mathbb{R} \tag{1}$$

where $\mathbf{d}_i \in \mathbb{R}^\delta$ ($\delta \geq 2$) are called control points, and $F_i : [a,b] \to \mathbb{R}$ are sufficiently smooth combining functions. The most widespread examples are Bézier, B-spline and NURBS curves. In general, curve (1) does not pass through the control points just approximates the shape of the control polygon determined by them. Nevertheless, if functions $\{F_i(t) : t \in [a,b]\}_{i=0}^n$ are linearly independent, we can always solve the following interpolation problem. Given the sequence of data points $\left\{\mathbf{p}_i \in \mathbb{R}^\delta\right\}_{i=0}^n$ along with associated strictly monotone parameter values $\{t_i \in [a,b]\}_{i=0}^n$, find those control points of (1) for which conditions

$$\mathbf{c}(t_i) = \mathbf{p}_i,\ i = 0,1,\ldots,n$$

are fulfilled. This problem can be reduced to solve the system of linear equations

$$\begin{bmatrix} F_0(t_0) & F_1(t_0) & \cdots & F_n(t_0) \\ F_0(t_1) & F_1(t_1) & \cdots & F_n(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ F_0(t_n) & F_1(t_n) & \cdots & F_n(t_n) \end{bmatrix} \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{bmatrix}$$

for the unknown control points $\{\mathbf{d}_i\}_{i=0}^n$. A drawback of this method is that the interpolating curve will globally be controlled by data points, i.e., the displacement of any $\mathbf{p}_i$ results in the change of the whole curve, even if the combining functions are splines.

This disadvantage can be eliminated by applying a local interpolating scheme, e.g. by the one that was introduced by Overhauser[5]. He considered a sequence of data points $\mathbf{p}_i$ with associated parameter values t_i and constructed a C^1 cubic interpolating spline curve the arcs of which are linearly blended parabolic arcs. The ith ($i = 1,2,\ldots,n-2$) arc of this spline curve is of the form

$$\mathbf{a}_i(t) = \left(1 - \frac{t-t_i}{t_{i+1}-t_i}\right)\mathbf{c}_i(t) + \frac{t-t_i}{t_{i+1}-t_i}\mathbf{c}_{i+1}(t),\ t \in [t_i,t_{i+1}],$$

where $\mathbf{c}_i(t), t \in [t_{i-1},t_{i+1}]$ is the parabolic arc that passes through the points $\mathbf{p}_{i-1},\mathbf{p}_i,\mathbf{p}_{i+1}$ at t_{i-1},t_i,t_{i+1}, respectively. This construction is illustrated in Fig. 1.

The blending concept of Overhauser has various generalizations. Pobegailo[7] blended line segments and circular arcs with polynomials of degree $2n+1$ (specified in power basis) to produce interpolating curve of variable smoothness at data points. Wiltsche[14] studied blending in a wider context, used polynomials like Pobegailo[7] but specified them in Bernstein basis to blend different types of parametric curves, in addition introduced shape parameters. Wenz[13] linearly blended line segments and circular arcs to produce G^1 interpolating spline. Szilvási-Nagy and Vendel[12] applied trigonomet-

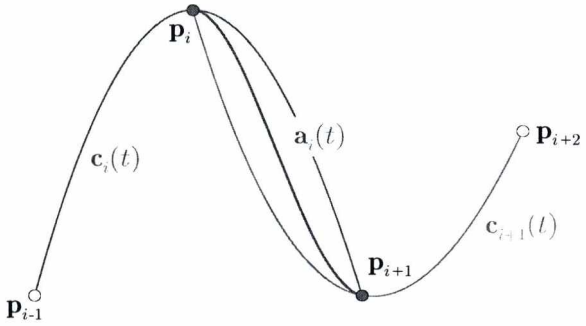


Figure 1: *The arc* $\mathbf{a}_i(t)$ *of the Overhauser spline is a blend of parabolas* $\mathbf{c}_i(t)$ *and* $\mathbf{c}_{i+1}(t)$.

ric blending of line segments and circular arcs to produce G^2 interpolating spline. Röschel[8] blended Lagrange interpolants with B-spline basis functions to obtain interpolating C^{k-1} splines of degree $2k-1$. Gfrerrer and Röschel[1] generalizes the previous idea by replacing the Lagrange interpolants by Hermite ones. Schneider[11] uses shape parameters and special weight functions in order to pull the parabolic arcs of the classical Overhauser spline towards the chords between consecutive data points and to ensure the smoothness of the interpolating curves, respectively.

The advantage of these methods is that there is no need for the solution of linear systems, the interpolating curve can be computed locally, consequently the spline curve is locally modifiable. A disadvantage of these procedures is that the control point representation is lost when the arcs to be blended are specified by control points.

Our proposed method combines the advantages of the two options described above, i.e., it produces the interpolating spline locally and provides directly the control points of the interpolating arcs with arbitrary order of continuity at joints. We demonstrate the method for trigonometric curves, that include polynomial curves as a special case. In the forthcoming sections we summarize our results, details can be found in Juhász and Róth[2].

2. Auxiliary tools

We will produce interpolating trigonometric curves, that are described by means of the combination of control points and basis functions, therefore we need a proper basis in the space of trigonometric polynomials

$$\mathcal{F}_{2n}^{\alpha} = \operatorname{span}\left\{\cos(kt), \sin(kt) : t \in [0,\alpha]\right\}_{k=0}^{n}. \qquad (2)$$

2.1. Trigonometric B-basis and curves

We will use a transformed version of the basis specified in Sánchez-Reyes[10]. Let $\alpha \in (0,\pi)$ be an arbitrarily fixed parameter and consider the function system

$$\left\{A_{2n,i}^{\alpha}(t) : t \in [0,\alpha]\right\}_{i=0}^{2n} = \left\{c_{2n,i}^{\alpha} \sin^{2n-i}\left(\frac{\alpha-t}{2}\right) \sin^{i}\left(\frac{t}{2}\right) : t \in [0,\alpha]\right\}_{i=0}^{2n},$$

where the normalizing positive coefficients

$$c_{2n,i}^{\alpha} = \frac{1}{\sin^{2n}\left(\frac{\alpha}{2}\right)} \sum_{r=0}^{\left\lfloor \frac{i}{2} \right\rfloor} \binom{n}{i-r} \binom{i-r}{r} \left(2\cos\left(\frac{\alpha}{2}\right)\right)^{i-2r}$$

have the symmetry

$$c_{2n,i}^{\alpha} = c_{2n,2n-i}^{\alpha},\ i = 0,1,\ldots,n. \qquad (3)$$

Function system (3) is the B-basis of the vector space (2), thus

$$\sum_{i=0}^{2n} A_{2n,i}^{\alpha}(t) \equiv 1, \qquad (4)$$

$$A_{2n,i}^{\alpha}(0) = \begin{cases} 1, & \text{for } i = 0, \\ 0, & \text{otherwise,} \end{cases} \qquad (5)$$

$$A_{2n,i}^{\alpha}(\alpha) = \begin{cases} 1, & \text{for } i = 2n, \\ 0, & \text{otherwise.} \end{cases} \qquad (6)$$

Basis functions (3) are symmetric in the sense

$$A_{2n,i}^{\alpha}(\alpha - t) = A_{2n,2n-i}^{\alpha}(t),\ \forall t \in [0,\alpha], \qquad (7)$$

for all $i = 0,1,\ldots,n$, and their product can be expressed in the form

$$A_{2n,i}^{\alpha}(t) A_{2m,j}^{\alpha}(t) = \frac{c_{2n,i}^{\alpha} c_{2m,j}^{\alpha}}{c_{2(n+m),i+j}^{\alpha}} A_{2(n+m),i+j}^{\alpha}(t),\ \forall t \in [0,\alpha] \qquad (8)$$

for all $i = 0,1,\ldots,2n$ and $j = 0,1,\ldots,2m$.

By means of the linear reparametrization $t(v) = \alpha v$, $v \in [0,1]$, the system

$$\left\{A_{2n,i}^{\alpha}(\alpha v) : v \in [0,1]\right\}_{i=0}^{2n} = \left\{c_{2n,i}^{\alpha} \sin^{i}\left(\frac{\alpha v}{2}\right) \sin^{2n-i}\left(\frac{\alpha(1-v)}{2}\right) : v \in [0,1]\right\}_{i=0}^{2n}$$

is obtained, using which it can be shown that (3) is a generalization of the Bernstein basis of degree $2n$.

Proposition 2.1 If $\alpha \to 0$ basis functions (3) converge to the Bernstein polynomials of degree $2n$, i.e.,

$$\lim_{\alpha \to 0} A_{2n,i}^{\alpha}(\alpha v) = B_i^{2n}(v),\ \forall v \in [0,1],\ i = 0,1,\ldots,2n. \qquad (9)$$

Definition 2.1 By means of functions (3) and control points $\left\{\mathbf{d}_i \in \mathbb{R}^{\delta}\right\}_{i=0}^{2n}$ $(\delta \geq 2)$ the curve of order $n \geq 1$ (degree $2n$)

$$\mathbf{g}_n^{\alpha}(t) = \sum_{i=0}^{2n} A_{2n,i}^{\alpha}(t)\, \mathbf{d}_i,\ t \in [0,\alpha], \qquad (10)$$

can be defined, where α is a global shape parameter.

Curves (10) form a proper subset of rational curves, i.e., any curve of type (10) has a rational parametrization. However, rational parametrizations have unfavorable properties, e.g., their higher order derivatives are difficult to deal with and the parametrization of the curve is often poor (cf. Piegl[6]), thus the usage of trigonometric parametrization is reasonable.

2.1.1. Subdivision of quadratic trigonometric curves

Basis functions (3) form the normalized B-basis of function space (2), thus there must be a recursive corner cutting algorithm for curves (10), cf. Mainar and Peña[3]. The general formula is unknown as yet, there is a scheme only for a very specific arrangement of control points in Sánchez-Reyes[9]. We provide a solution the problem for the special case $n = 1$ for any placement of control points, that will be used in Subsection 2.3.

Consider the quadratic curve

$$\mathbf{g}_1^{\alpha}(t) = \sum_{i=0}^{2} A_{2,i}^{\alpha}(t)\,\mathbf{d}_i,\ t \in [0,\alpha]. \tag{11}$$

For any arbitrarily chosen value $\beta \in (0,\alpha)$ we introduce subdivision points

$$\mathbf{d}_0^1 = \left(1-\mu_0^1(\beta;\alpha)\right)\mathbf{d}_0 + \mu_0^1(\beta;\alpha)\,\mathbf{d}_1,$$
$$\mathbf{d}_1^1 = \left(1-\mu_1^1(\beta;\alpha)\right)\mathbf{d}_1 + \mu_1^1(\beta;\alpha)\,\mathbf{d}_2$$

and

$$\mathbf{d}_0^2 = \left(1-\mu_0^2(\beta;\alpha)\right)\mathbf{d}_0^1 + \mu_0^2(\beta;\alpha)\,\mathbf{d}_1^1,$$

where

$$\mu_0^1(t;\alpha) = 1 - \frac{\sin\left(\frac{\alpha-t}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{t}{2}\right)},$$
$$\mu_1^1(t;\alpha) = \frac{\sin^2\left(\frac{t}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)\left(\sin\left(\frac{\alpha}{2}\right) - \sin\left(\frac{\alpha-t}{2}\right)\cos\left(\frac{t}{2}\right)\right)},$$
$$\mu_0^2(t;\alpha) = 1 - \frac{\sin\left(\frac{\alpha-t}{2}\right)\cos\left(\frac{t}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)}.$$

Using these subdivision points we define curves

$$\mathbf{g}_{1,\ell}^{\beta}(t) = A_{2,0}^{\beta}(t)\,\mathbf{d}_0 + A_{2,1}^{\beta}(t)\,\mathbf{d}_0^1 + A_{2,2}^{\beta}(t)\,\mathbf{d}_0^2, \tag{12}$$
$$t \in [0,\beta],$$

$$\mathbf{g}_{1,r}^{\alpha-\beta}(t) = A_{2,0}^{\alpha-\beta}(t)\,\mathbf{d}_0^2 + A_{2,1}^{\alpha-\beta}(t)\,\mathbf{d}_0^1 + A_{2,2}^{\alpha-\beta}(t)\,\mathbf{d}_2, \tag{13}$$
$$t \in [0,\alpha-\beta]$$

for which the following statement holds.

Proposition 2.2 Quadratic trigonometric curves (12) and (13) form two consecutive arcs of curve (11) and the continuity at their common point $\mathbf{d}_0^2$ is C^{∞}, i.e.,

$$\mathbf{g}_{1,\ell}^{\beta}(t) = \mathbf{g}_1^{\alpha}(t),\ \forall t \in [0,\beta], \tag{14}$$

$$\mathbf{g}_{1,r}^{\alpha-\beta}(t) = \mathbf{g}_1^{\alpha}(\beta+t),\ \forall t \in [0,\alpha-\beta] \tag{15}$$

and

$$\frac{\mathrm{d}^z}{\mathrm{d}t^z}\mathbf{g}_{1,\ell}^{\beta}(t)\Big|_{t=\beta} = \frac{\mathrm{d}^z}{\mathrm{d}t^z}\mathbf{g}_{1,r}^{\alpha-\beta}(t)\Big|_{t=0},\ \forall z \in \mathbb{N}.$$

We are going to describe an interpolation scheme that is based on curve blending, therefore at first we specify an appropriate blending function, then we show an essential continuity property of the resulted blended curve.

2.2. Trigonometric blending function and blended curve

Definition 2.2 Using basis functions (3), we define the blending function

$$b_{2n}^{\alpha}(t) = \frac{1}{2}A_{2n,n}^{\alpha}(t) + \sum_{i=n+1}^{2n} A_{2n,i}^{\alpha}(t),\ t \in [0,\alpha]. \tag{16}$$

Blending function (16) shares the following properties.

- It is non-negative and strictly increasing, specifically

$$b_{2n}^{\alpha}(0) = 0, \tag{17}$$
$$b_{2n}^{\alpha}(\alpha) = 1. \tag{18}$$

- Symmetric in the sense

$$b_{2n}^{\alpha}(t) = 1 - b_{2n}^{\alpha}(\alpha - t),\ \forall t \in [0,\alpha],$$

  i.e., its graph is symmetric with respect to the point of coordinates $\left(\frac{\alpha}{2},\frac{1}{2}\right)$.
- Its derivatives vanish at the endpoints up to the order $n-1$, i.e.,

$$\frac{\mathrm{d}^z}{\mathrm{d}t^z}b_{2n}^{\alpha}(t)\Big|_{t=0} = 0, \tag{19}$$
$$\frac{\mathrm{d}^z}{\mathrm{d}t^z}b_{2n}^{\alpha}(t)\Big|_{t=\alpha} = 0 \tag{20}$$

for all $z = 1,2,\ldots,n-1$.

Let us consider curves $\mathbf{c}_1(t)$ and $\mathbf{c}_2(t), t \in [0,\alpha]$ that fulfill conditions

$$\mathbf{c}_1(0) = \mathbf{c}_2(0) = \mathbf{p}_1, \tag{21}$$
$$\mathbf{c}_1(\alpha) = \mathbf{c}_2(\alpha) = \mathbf{p}_2.$$

We blend these two curves with function (16) that yields

$$\mathbf{c}(t) = \left(1-b_{2n}^{\alpha}(t)\right)\mathbf{c}_1(t) + b_{2n}^{\alpha}(t)\,\mathbf{c}_2(t),\ t \in [0,\alpha]. \tag{22}$$

It is immediate that $\mathbf{c}(t)$ is a convex combination of curves $\mathbf{c}_1(t)$ and $\mathbf{c}_2(t)$, and the blended curve fulfills the endpoint conditions

$$\begin{cases} \mathbf{c}(0) = \mathbf{p}_1, \\ \mathbf{c}(\alpha) = \mathbf{p}_2. \end{cases}$$

Proposition 2.3 If curves $\mathbf{c}_1(t)$ and $\mathbf{c}_2(t), t \in [0,\alpha]$ are at

least n times continuously differentiable at $t = 0$ and $t = \alpha$, then for the blended curve (22) equalities

$$\left.\frac{\mathrm{d}^z}{\mathrm{d}t^z}\mathbf{c}(t)\right|_{t=0} = \left.\frac{\mathrm{d}^z}{\mathrm{d}t^z}\mathbf{c}_1(t)\right|_{t=0},$$
$$\left.\frac{\mathrm{d}^z}{\mathrm{d}t^z}\mathbf{c}(t)\right|_{t=\alpha} = \left.\frac{\mathrm{d}^z}{\mathrm{d}t^z}\mathbf{c}_2(t)\right|_{t=\alpha}$$

hold for all $z = 1, 2, \ldots, n$.

2.3. Interpolating elliptical arcs

We are going to use elliptical arcs for local interpolation, thus we have to describe the elliptical arc as a quadratic trigonometric curve of type (10) that interpolates three consecutive points at specified parameter values.

Given the sequence of data points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ and associated parameter values $u_0 < u_1 < u_2$, with the constraint $u_2 - u_0 < \pi$, find those control points $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$ for which the curve

$$\mathbf{e}(u) = \sum_{j=0}^{2} A_{2,j}^{\alpha_0+\alpha_1}(u)\,\mathbf{q}_j,\ u \in [0, \alpha_0+\alpha_1] \qquad (23)$$

fulfills the boundary conditions

$$\mathbf{e}(0) = \mathbf{p}_0,$$
$$\mathbf{e}(\alpha_0) = \mathbf{p}_1,$$
$$\mathbf{e}(\alpha_0+\alpha_1) = \mathbf{p}_2,$$

where $\alpha_i = u_{i+1} - u_i$, $(i = 0, 1)$.

The required control points are

$$\mathbf{q}_0 = \mathbf{p}_0,$$
$$\mathbf{q}_1 =: \mathbf{q}_1^{\alpha_0+\alpha_1}$$
$$= \frac{1}{A_{2,1}^{\alpha_0+\alpha_1}(\alpha_0)}\left(\mathbf{p}_1 - A_{2,0}^{\alpha_0+\alpha_1}(\alpha_0)\,\mathbf{p}_0 - A_{2,2}^{\alpha_0+\alpha_1}(\alpha_0)\,\mathbf{p}_2\right),$$
$$\mathbf{q}_2 = \mathbf{p}_2.$$

Using the subdivision formula specified in Subsection 2.1.1, we split the elliptical arc (23) at the parameter value $u = \alpha_0$. The left arc is

$$\mathbf{e}_\ell(u) = \sum_{j=0}^{2} A_{2,j}^{\alpha_0}(u)\,\mathbf{a}_{j,\ell},\ u \in [0, \alpha_0]$$

with control points

$$\mathbf{a}_{0,\ell} = \mathbf{p}_0,$$
$$\mathbf{a}_{1,\ell} =: \mathbf{a}_{1,\ell}^{\alpha_0} = \left(1 - \mu_0^1(\alpha_0; \alpha_0+\alpha_1)\right)\mathbf{p}_0 + \mu_0^1(\alpha_0; \alpha_0+\alpha_1)\,\mathbf{q}_1,$$
$$\mathbf{a}_{2,\ell} = \mathbf{p}_1,$$

while the right arc

$$\mathbf{e}_r(u) = \sum_{j=0}^{2} A_{2,j}^{\alpha_1}(u)\,\mathbf{a}_{j,r},\ u \in [0, \alpha_1]$$

is generated by

$$\mathbf{a}_{0,r} = \mathbf{p}_1,$$
$$\mathbf{a}_{1,r} =: \mathbf{a}_{1,r}^{\alpha_1} = \left(1 - \mu_1^1(\alpha_0; \alpha_0+\alpha_1)\right)\mathbf{q}_1 + \mu_1^1(\alpha_0; \alpha_0+\alpha_1)\,\mathbf{p}_2,$$
$$\mathbf{a}_{2,r} = \mathbf{p}_2.$$

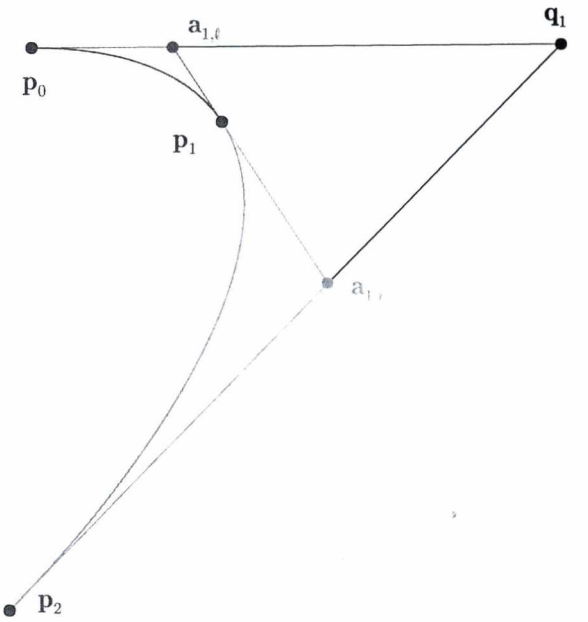


Figure 2: *The elliptical arc that interpolates data points* $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ *that is split into two arcs. Settings are* $\alpha = \pi/2$ *and knots are chosen by centripetal parametrization.*

These arcs are illustrated in Fig. 2. Note, if points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ are collinear the above process results in two straight line segments parametrized as (10).

3. The interpolating scheme

Our objective is to generalize the Overhauser spline by replacing parabolic arcs with elliptical ones, i.e. by solving the following interpolation problem.

3.1. Interpolating trigonometric splines

Given the sequence of data points $\{\mathbf{p}_i\}_{i=0}^m$ and associated parameter values $t_0 < t_1 < \ldots t_{m-1} < t_m$, moreover the shape parameter $\alpha \in (0, \pi)$, find the control points of the spline curve of continuity C^n $(n \geq 1)$ that interpolates the given data points, and that consists of arcs of type (10).

We use elliptical arcs (23) defined by consecutive triplets of data points $\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}$ $(i = 1, 2, \ldots, m-1)$, where the

maximum of domains $t_{i+1}-t_{i-1}$ equals α, thus we have to scale parameter values t_i to obtain the new knots

$$u_0^\alpha = t_0,$$
$$u_i^\alpha = u_{i-1}^\alpha + \frac{\alpha}{d_{\max}}(t_i - t_{i-1}),\ i = 1,2,\ldots,m,$$

where $d_{\max} = \max\{t_{i+1}-t_{i-1} : i = 1,2,\ldots,m-1\}$. Therefore the domain of the interpolating elliptical arc will be $\alpha_{i-1}+\alpha_i$, with $\alpha_i = u_{i+1}^\alpha - u_i^\alpha$, $(i = 0,1,\ldots,m-1)$. Its control points are $\mathbf{p}_{i-1}, \mathbf{q}_i^{\alpha_{i-1}+\alpha_i}, \mathbf{p}_{i+1}$, where

$$\mathbf{q}_i^{\alpha_{i-1}+\alpha_i} = \frac{\mathbf{p}_i - A_{2,0}^{\alpha_{i-1}+\alpha_i}(\alpha_{i-1})\mathbf{p}_{i-1} - A_{2,2}^{\alpha_{i-1}+\alpha_i}(\alpha_{i-1})\mathbf{p}_{i+1}}{A_{2,1}^{\alpha_{i-1}+\alpha_i}(\alpha_{i-1})}.$$

Splitting this arc into two (also quadratic) elliptical arcs, as it is described in Subsection 2.3, we obtain the left and right arcs

$$\mathbf{e}_{i,\ell}^{\alpha_{i-1}}(u) = A_{2,0}^{\alpha_{i-1}}(u)\mathbf{p}_{i-1} + A_{2,1}^{\alpha_{i-1}}(u)\mathbf{a}_{i,\ell}^{\alpha_{i-1}} + A_{2,2}^{\alpha_{i-1}}(u)\mathbf{p}_i,$$
$$u \in [0,\alpha_{i-1}]$$

and

$$\mathbf{e}_{i,r}^{\alpha_i}(u) = A_{2,0}^{\alpha_i}(u)\mathbf{p}_i + A_{2,1}^{\alpha_i}(u)\mathbf{a}_{i,r}^{\alpha_i} + A_{2,2}^{\alpha_i}(u)\mathbf{p}_{i+1},$$
$$u \in [0,\alpha_i],$$

where

$$\mathbf{a}_{i,\ell}^{\alpha_{i-1}} = \left(1-\mu_0^1(\alpha_{i-1};\alpha_{i-1}+\alpha_i)\right)\mathbf{p}_{i-1} + \mu_0^1(\alpha_{i-1};\alpha_{i-1}+\alpha_i)\mathbf{q}_i^{\alpha_{i-1}+\alpha_i},$$

$$\mathbf{a}_{i,r}^{\alpha_i} = \left(1-\mu_1^1(\alpha_{i-1};\alpha_{i-1}+\alpha_i)\right)\mathbf{q}_i^{\alpha_{i-1}+\alpha_i} + \mu_1^1(\alpha_{i-1};\alpha_{i-1}+\alpha_i)\mathbf{p}_{i+1}.$$

Blending of $\mathbf{e}_{i,r}^{\alpha_i}(u)$ and $\mathbf{e}_{i+1,\ell}^{\alpha_i}(u)$ $(i = 1,2,\ldots,m-1)$ results in the sequence of arcs

$$\mathbf{g}_{n+1,i}^{\alpha_i}(u) = \left(1-b_{2n}^{\alpha_i}(u)\right)\mathbf{e}_{i,r}^{\alpha_i}(u) + b_{2n}^{\alpha_i}(u)\mathbf{e}_{i+1,\ell}^{\alpha_i}(u), \quad (24)$$
$$u \in [0,\alpha_i],$$
$$i = 1,2,\ldots,m-1.$$

Proposition 3.1 Arcs $\mathbf{g}_{n+1,i}^{\alpha_i}(u)$ and $\mathbf{g}_{n+1,i+1}^{\alpha_{i+1}}(u)$ $(i = 1,2,\ldots,m-2)$ are C^n continuous at their joint $\mathbf{p}_{i+1}$.

In what follows, we provide an exact formula for the control points of blended arcs $\left\{\mathbf{g}_{n+1,i}^{\alpha_i}(u) : u \in [0,\alpha_i]\right\}_{i=1}^{m-1}$. In the ith blended arc

$$\mathbf{g}_{n+1,i}^{\alpha_i}(u) = \left(\sum_{j=0}^{n-1} A_{2n,j}^{\alpha_i}(u) + \frac{1}{2}A_{2n,n}^{\alpha_i}(u)\right)\cdot\left(A_{2,0}^{\alpha_i}(u)\mathbf{p}_i + A_{2,1}^{\alpha_i}(u)\mathbf{a}_{i,r}^{\alpha_i} + A_{2,2}^{\alpha_i}(u)\mathbf{p}_{i+1}\right)$$
$$+\left(\frac{1}{2}A_{2n,n}^{\alpha_i}(u) + \sum_{j=n+1}^{2n} A_{2n,j}^{\alpha_i}(u)\right)\cdot\left(A_{2,0}^{\alpha_i}(u)\mathbf{p}_i + A_{2,1}^{\alpha_i}(u)\mathbf{a}_{i+1,\ell}^{\alpha_i} + A_{2,2}^{\alpha_i}(u)\mathbf{p}_{i+1}\right)$$

we rearrange the right hand side, using identity (8), and collect the coefficients of basis functions $\left\{A_{2(n+1),j}^{\alpha_i}(u) : u \in [0,\alpha_i]\right\}_{j=0}^{2(n+1)}$, as a result of which we obtain control points

$$\mathbf{d}_j^{\alpha_i} = \begin{cases} \dfrac{c_{2n,j}^{\alpha_i}c_{2,0}^{\alpha_i}\mathbf{p}_i + c_{2n,j-1}^{\alpha_i}c_{2,1}^{\alpha_i}\mathbf{a}_{i,r}^{\alpha_i} + c_{2n,j-2}^{\alpha_i}c_{2,2}^{\alpha_i}\mathbf{p}_{i+1}}{c_{2(n+1),j}^{\alpha_i}}, \\ \text{for } j = 0,1,\ldots,n, \\ \dfrac{c_{2n,n+1}^{\alpha_i}c_{2,0}^{\alpha_i}\mathbf{p}_i + \frac{1}{2}c_{2n,n}^{\alpha_i}c_{2,1}^{\alpha_i}\left(\mathbf{a}_{i+1,\ell}^{\alpha_i}+\mathbf{a}_{i,r}^{\alpha_i}\right) + c_{2n,n-1}^{\alpha_i}c_{2,2}^{\alpha_i}\mathbf{p}_{i+1}}{c_{2(n+1),n+1}^{\alpha_i}}, \\ \text{for } j = n+1, \\ \dfrac{c_{2n,j}^{\alpha_i}c_{2,0}^{\alpha_i}\mathbf{p}_i + c_{2n,j-1}^{\alpha_i}c_{2,1}^{\alpha_i}\mathbf{a}_{i+1,\ell}^{\alpha_i} + c_{2n,j-2}^{\alpha_i}c_{2,2}^{\alpha_i}\mathbf{p}_{i+1}}{c_{2(n+1),j}^{\alpha_i}}, \\ \text{for } j = n+2,n+3,\ldots,2(n+1), \end{cases}$$

where the convention $c_{2n,j}^{\alpha_i} = 0$ if $j < 0$ or $j > 2n$ is adopted. Fig. 3 illustrates the construction of an arc of a C^2 interpolating spline along with its control points.

The above described method does not provide the first and last arcs of the interpolating spline. In order to specify these arcs, we can apply, e.g., pseudo data points $\mathbf{p}_{-1}, \mathbf{p}_{m+1}$, or we can use arcs $\mathbf{e}_{1,\ell}^{\alpha_0}(u)$, $\mathbf{e}_{m-1,r}^{\alpha_{m-1}}(u)$. Both methods guarantee the required order of continuity at $\mathbf{p}_1$ and $\mathbf{p}_{m-1}$, respectively.

Data points have a local influence on the interpolating curve, since the relocation of data point $\mathbf{p}_i$ affects at most four arcs, namely $\left\{\mathbf{g}_{n+1,j}^{\alpha_j}(u) : u \in [0,\alpha_j]\right\}_{j=i-2}^{i+1}$, moreover control points $\left\{\mathbf{d}_j^{\alpha_i}\right\}_{j=0}^{n}$ and $\left\{\mathbf{d}_j^{\alpha_i}\right\}_{j=n+2}^{2(n+1)}$ are in the plane spanned by data points $\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}$ and $\mathbf{p}_i, \mathbf{p}_{i+1}, \mathbf{p}_{i+2}$, respectively.

Fig. 4 illustrates the effect of the parametrization (the choice of knots t_i) of data points, while Fig. 5 shows the influence of the global shape parameter α on the shape of the interpolating spline curve.

3.2. Interpolating Bézier splines as a special case

In this section we study the limiting case $\alpha \to 0$ of the interpolating trigonometric spline curve of order $n+1$. Observe that

$$\lim_{\alpha\to 0}\alpha_i = \lim_{\alpha\to 0}\left(u_{i+1}^\alpha - u_i^\alpha\right) = \lim_{\alpha\to 0}\frac{\alpha}{d_{\max}}(t_i - t_{i-1}) = 0$$

for all $i = 0,1,\ldots,m-1$. By means of parametrization

$$u(t) = (\alpha_{i-1}+\alpha_i)\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}},\ t \in [t_{i-1},t_{i+1}]$$

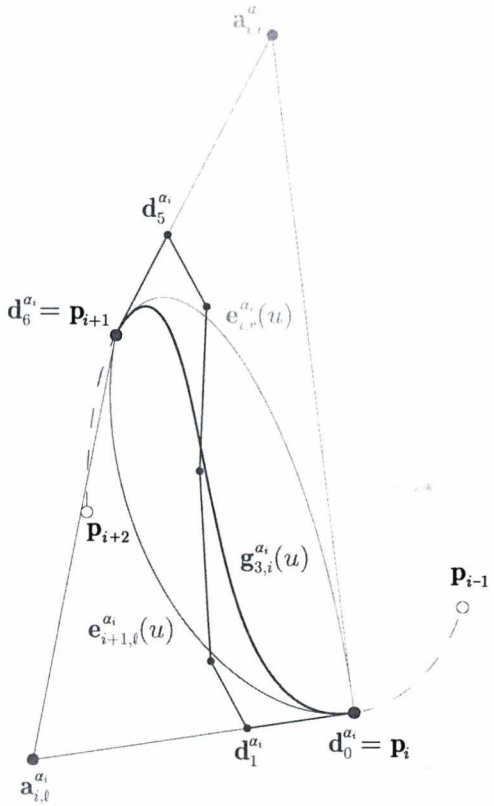


Figure 3: *The arc $\mathbf{g}_{3,i}^{\alpha_i}(u)$ of the C^2 interpolating spline is obtained by blending elliptical arcs $\mathbf{e}_{i,r}^{\alpha_i}(u)$ and $\mathbf{e}_{i+1,\ell}^{\alpha_i}(u)$ with $b_4^{\alpha_i}(u)$; $\alpha = 3$ and knots t_i are chosen by chord length parametrization.*

and Proposition 2.1, the interpolating elliptical arc

$$\mathbf{e}_i^{\alpha_{i-1}+\alpha_i}(u) = \sum_{j=0}^{2} \mathbf{q}_{j,i}^{\alpha_{i-1}+\alpha_i} A_{2,0}^{\alpha_{i-1}+\alpha_i}(u),\ u \in [0, \alpha_{i-1}+\alpha_i]$$

of degree two degenerates to the the parabolic arc (i.e., to the quadratic Bézier curve)

$$\mathbf{e}_i^0(t) := \sum_{j=0}^{2} \mathbf{q}_{j,i}^0 B_j^2\left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\right),\ t \in [t_{i-1}, t_{i+1}], \quad (25)$$

in the limiting case $\alpha \to 0$, where

$$\mathbf{q}_{0,i}^0 = \mathbf{p}_{i-1},$$

$$\mathbf{q}_{1,i}^0 = \frac{\mathbf{p}_i - \mathbf{p}_{i-1} B_0^2\left(\frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\right) - \mathbf{p}_{i+1} B_2^2\left(\frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\right)}{B_1^2\left(\frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\right)},$$

$$\mathbf{q}_{2,i}^0 = \mathbf{p}_{i+1}.$$

Subdivision scheme, described in Section 2.3, becomes

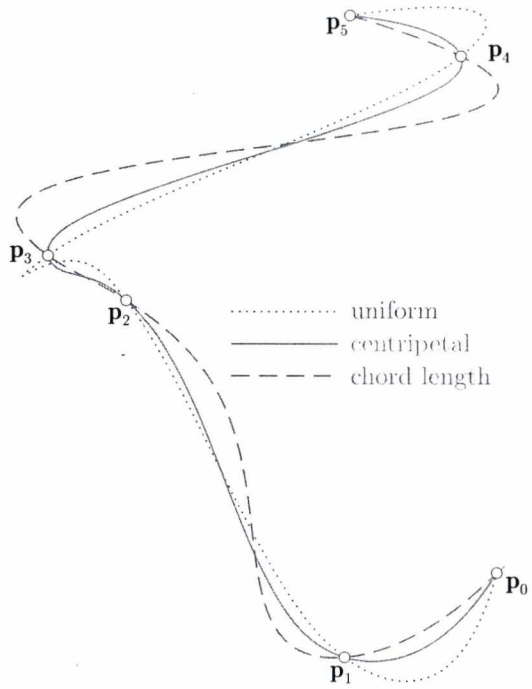


Figure 4: *The effect of the parametrization of data points (the choice of knots t_i) on a C^2 interpolating curve; $\alpha = \pi/2$.*

the de Casteljau algorithm of quadratic Bézier curves when $\alpha \to 0$. Splitting the parabola (25) into two Bézier arcs, one obtains the left and right parabolic arcs

$$\begin{aligned}&\mathbf{e}_{i,\ell}^0(t)\\ &= B_0^2(v_{i-1}(t))\,\mathbf{p}_{i-1} + B_1^2(v_{i-1}(t))\,\mathbf{a}_{i,\ell}^0 + B_2^2(v_{i-1}(t))\,\mathbf{p}_i,\\ &t \in [t_{i-1}, t_i],\end{aligned}$$

$$\begin{aligned}&\mathbf{e}_{i,r}^0(t)\\ &= B_0^2(v_i(t))\,\mathbf{p}_i + B_1^2(v_i(t))\,\mathbf{a}_{i,r}^0 + B_2^2(v_i(t))\,\mathbf{p}_{i+1},\\ &t \in [t_i, t_{i+1}],\end{aligned}$$

where

$$\mathbf{a}_{i,\ell}^0 = \left(1 - \frac{t_i - t_{i-1}}{t_{i+1}-t_{i-1}}\right)\mathbf{p}_{i-1} + \frac{t_i - t_{i-1}}{t_{i+1}-t_{i-1}}\mathbf{q}_{1,i}^0,$$

$$\mathbf{a}_{i,r}^0 = \left(1 - \frac{t_i - t_{i-1}}{t_{i+1}-t_{i-1}}\right)\mathbf{q}_{1,i}^0 + \frac{t_i - t_{i-1}}{t_{i+1}-t_{i-1}}\mathbf{p}_{i+1},$$

and

$$v_i(t) = \frac{t-t_i}{t_{i+1}-t_i},\ t \in [t_i, t_{i+1}],\ i = 1, 2, \ldots, m-1.$$

By blending arcs $\mathbf{e}_{i,r}^0(t)$ and $\mathbf{e}_{i+1,\ell}^0(t)$ $(i = 1, 2, \ldots, m-1,$

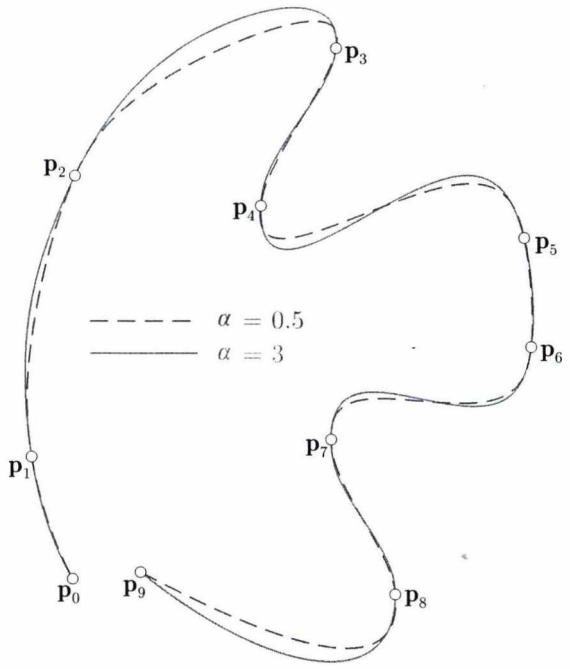


Figure 5: *The impact of the global shape parameter* α *on the shape of a* C^2 *interpolating spline curve; knots* t_i *are specified by chord length parametrization.*

$t \in [t_i, t_{i+1}]$) we obtain the sequence of Bézier arcs

$$\begin{aligned} \mathbf{g}^0_{n+1,i}(t) &= \left(1 - b^0_{2n}(t)\right) \mathbf{e}^0_{i,r}(t) + b^0_{2n}(t)\, \mathbf{e}^0_{i+1,\ell}(t), \\ & t \in [t_i, t_{i+1}], \\ i &= 1, 2, \ldots, m-1 \end{aligned}$$

of degree $2(n+1)$, where

$$\begin{aligned} b^0_{2n}(t) &= \lim_{\alpha \to 0} b^{\alpha_i}_{2n}(t) \\ &= \frac{1}{2} B^{2n}_n(v_i(t)) + \sum_{k=n+1}^{2n} B^{2n}_k(v_i(t)), \ \forall t \in [t_i, t_{i+1}] \end{aligned}$$

denotes the blending function of degree $2n$. Naturally, Bézier arcs $\mathbf{g}^0_{n+1,i}(t)$ and $\mathbf{g}^0_{n+1,i+1}(t)$ $(i = 1, 2, \ldots, m-2)$ are C^n continuous at their joint $\mathbf{p}_i$. It can be shown that the actual degree of the polynomial blending function $b^0_{2n}(t)$ is $2n-1$, i.e. its degree can be reduced, thus the special case $n = 1$ of the above process is the classical cubic Overhauser C^1 interpolation scheme.

Control points of the ith arc are

$$\mathbf{d}^0_j = \begin{cases} \dfrac{\binom{2n-1}{j}\binom{2}{0}}{\binom{2n+1}{j}} \mathbf{p}_i + \dfrac{\binom{2n-1}{j-1}\binom{2}{1}}{\binom{2n+1}{j}} \mathbf{a}^0_{i,r} + \dfrac{\binom{2n-1}{j-2}\binom{2}{2}}{\binom{2n+1}{j}} \mathbf{p}_{i+1}, \\ \text{for } j = 0, 1, \ldots, n, \\ \dfrac{\binom{2n-1}{j}\binom{2}{0}}{\binom{2n+1}{j}} \mathbf{p}_i + \dfrac{\binom{2n-1}{j-1}\binom{2}{1}}{\binom{2n+1}{j}} \mathbf{a}^0_{i+1,\ell} + \dfrac{\binom{2n-1}{j-2}\binom{2}{2}}{\binom{2n+1}{j}} \mathbf{p}_{i+1}, \\ \text{for } j = n+1, n+2, \ldots, 2n+1. \end{cases}$$

Note, that in this special case original knots $\{t_i\}_{i=0}^m$ do not have to be rescaled, since there is no limitation for the length of the domain of Bernstein polynomials.

4. Conclusions

The benefits of the proposed method are: treats both trigonometric and polynomial local interpolating spline curves in a unified way; provides a ready to use control point based description which is favorable from data storage and transmission point of view and is suitable for nowadays CAD/CAM systems; it is intuitive and user friendly; possesses a global shape parameter; offers a more versatile tool for shape design than polynomial methods, since these trigonometric curves form a proper subset of rational curves. A further direction of the research is to extend this method (to apply this kind of blending functions) to smooth surface interpolation.

Acknowledgements

Imre Juhász carried out his research in the framework of the Center of Excellence of Mechatronics and Logistics at the University of Miskolc. The research of Ágoston Róth was supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP-4.2.4.A/2-11/1-2012-0001 'National Excellence Program'.

References

1. A. Gfrerrer and O. Röschel. Blended Hermite interpolants. *Computer Aided Geometric Design*, **18**(9):865–873, 2001.
2. I. Juhász and Á. Róth. A scheme for interpolation with trigonometric spline curves. *Journal of Computational and Applied Mathematics*, **263**(June):246–261, 2014.
3. E. Mainar and J. M. Peña. Corner cutting algorithms associated with optimal shape preserving representations. *Computer Aided Geometric Design*, **16**(9):883–906, 1999.
4. M.-L. Mazure. Chebyshev-Bernstein bases. *Computer Aided Geometric Design*, **16**(7):649–669, 1999.
5. A. W. Overhauser. Analytic definition of curves and surfaces by parabolic blending. Technical report no.

SL68-40. *Ford Motor Company Scientific Laboratory*, 1968.

6. L. Piegl. On NURBS: a Survay. *Computer Graphics and Applications*, **11**(1):55–71, 1991.

7. A. P. Pobegailo. Local interpolation with weight functions for variable smoothness curve design. *Computer-Aided Design*, **23**(8):579–582, 1991.

8. O. Röschel, An interpolation subspline scheme related to B-spline techniques, in: Computer Graphics International, 1997. Proceedings, IEEE, 1997, pp. 131–136.

9. J. Sánchez-Reyes. Single-valued curves in polar coordinates. *Computer-Aided Design*, **22**(1):19–26, 1990.

10. J. Sánchez-Reyes. Harmonic rational Bézier curves, p-Bézier curves and trigonometric polynomials. *Computer Aided Geometric Design*, **15**(9):909–923, 1998.

11. W. Schneider. A simple technique for adding tension to parabolic blending interpolation. *Computers & Mathematics with Applications*, **12**(11):1155–1160, 1986.

12. M. Szilvási-Nagy and T. P. Vendel. Generating curves and swept surfaces by blended circles. *Computer Aided Geometric Design*, **17**(2):197–206, 2000.

13. H.-J. Wenz. Interpolation of curve data by blended generalized circles. *Computer Aided Geometric Design*, **13**(8):673–680, 1996.

14. A. Wiltsche. Blending curves. *Journal for Geometry and Graphics*, **9**(1):67–75, 2005.

Viewpoint-free Video Synthesis with an Integrated 4D System

Csaba Benedek, Zsolt Jankó, Attila Börcs, Iván Eichhardt, Dmitry Chetverikov, and Tamás Szirányi
Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI)
{firstname.lastname}@sztaki.mta.hu
http://web.eee.sztaki.hu/i4d

Abstract

In this paper, we introduce a complex approach on 4D reconstruction of dynamic scenarios containing multiple walking pedestrians. The input of the process is a point cloud sequence recorded by a rotating multi-beam Lidar sensor, which monitors the scene from a fixed position. The output is a geometrically reconstructed and textured scene containing moving 4D people models, which can follow in real time the trajectories of the walking pedestrians observed on the Lidar data flow. Our implemented system consists of four main steps. First, we separate foreground and background regions in each point cloud frame of the sequence by a robust probabilistic approach. Second, we perform moving pedestrian detection and tracking, so that among the point cloud regions classified as foreground, we separate the different objects, and assign the corresponding people positions to each other over the consecutive frames of the Lidar measurement sequence. Third, we geometrically reconstruct the ground, walls and further objects of the background scene, and texture the obtained models with photos taken from the scene. Fourth we insert into the scene textured 4D models of moving pedestrians which were preliminary created in a special 4D reconstruction studio. Finally, we integrate the system elements in a joint dynamic scene model and visualize the 4D scenario.

Categories and Subject Descriptors (according to ACM CCS): I.4.5 [Computer vision]: Reconstruction

1. Introduction

Recently, an internal project called Integrated 4D (or just i4D) has been launched by two units of the Institute for Computer Science and Control of the Hungarian Academy of Sciences (MTA SZTAKI). The name of the project refers to an unconventional attempt to combine two very different sources of spatio-temporal information, namely, a LIDAR and a 4D reconstruction studio. The main motivation for the integration of the two types of data is our desire to measure and represent the visual world at different levels of detail.

A LIDAR sensor provides a global description of a dynamic outdoor scene in the form of a time-varying 3D point cloud. The latter is used to separate moving objects from static environment and obtain a 3D model of the environment. A 4D studio builds a detailed dynamic model of an actor (typically, a person) moving in the studio. By integrating the two sources of data, one can modify the model of the scene and populate it with the avatars created in the studio. In this paper, we report on the current state of the ongoing i4D project and describe all major processing steps of the integrated system, from the acquisition of the row data (point clouds and videos) to the creation and visualisation of an augmented spatio-temporal model of the scene.

LIDAR sensors have been traditionally used in applications such as road extraction in urban areas [9], vehicle safety and environment recognition [15], airborne data processing and digital terrain modelling [5, 14], measurement of trees in a forest [17], and modelling of buildings [19] and other constructions. LIDAR data have been integrated with high resolution imagery [9] and fused with multispectral data [17].

To our best knowledge, there has been no previous attempt to integrate LIDAR data with the output of a 4D reconstruction studio before [1]. A typical 4D studio is a green or blue "box" equipped with multiple synchronised, calibrated video cameras. The video streams are used to create a dynamic model of an actor in real time or offline. The degree of realism in shape and appearance varies depending on the approach and the facilities, but the motion of the model obtained in a 4D studio is usually more realistic than that of an artificially created CAD model. Recently, we have built at MTA SZTAKI a 4D studio operating offline [10] and in real time [8]. Sec. 3 gives a brief description of our 4D studio. The

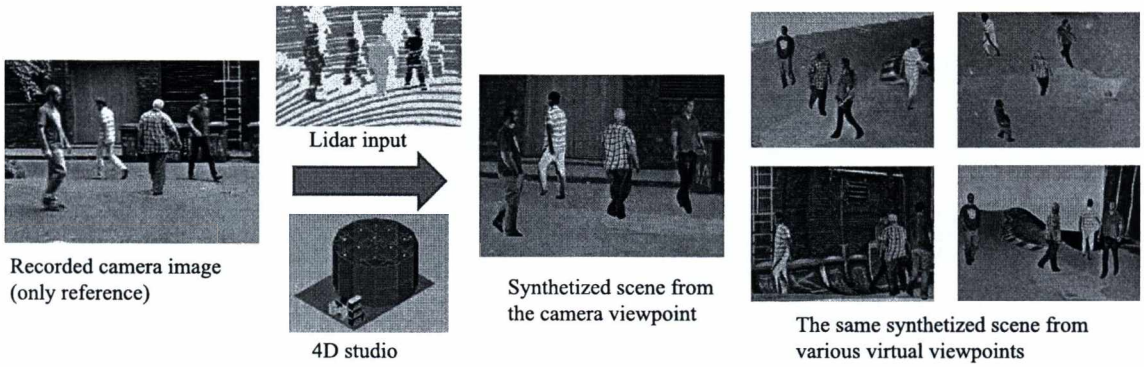


Figure 1: *Demonstration of the integrated 4D reconstruction system.*

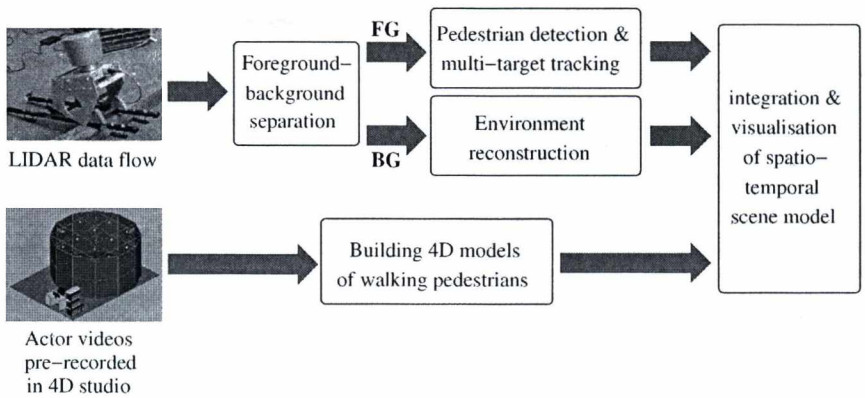


Figure 2: *Flowchart of the proposed system.* ***BG*** *is background,* ***FG*** *foreground.*

reader is referred to paper [10] for a survey of advanced 4D studios in Europe and the USA, and a discussion of their applications in game production, film industry, and other areas.

2. LIDAR data processing

In this section, we present a hybrid method for dense foreground-background point labelling in a point cloud obtained by a Velodyne HDL-64E RMB-LIDAR device that monitors the scene from a fixed position. The method solves the computationally critical spatial filtering tasks applying an MRF model in the 2D range image domain. The ambiguities of the point-to-pixel mapping are handled by joint consideration of the true 3D positions and the 2D labels. Then, we execute detection and tracking of moving pedestrians for the foreground points. Next, we transform the background point cloud into a polygon mesh while maintaining the information about individual objects such as ground, walls, and trees. Finally, the models of the environment objects are manually textured using photos taken in the scene. Below, we describe these steps in more detail.

2.1. Foreground-background separation

The rotating multi-beam LIDAR device records 360°-view-angle range data sequences of irregular point clouds. Examples of measured point clouds will be shown later in this paper. To separate dynamic foreground from static background in a range data sequence, we apply a probabilistic approach [2].

To ensure real-time operation, we project the irregular point cloud to a cylinder surface yielding a depth image on a regular lattice, and perform the segmentation in the 2D range image domain. A part of a range image showing several pedestrians is demonstrated in Fig. 3. Spurious effects are caused by the quantisation error of the discretised view angle, the non-linear position corrections of sensor calibration, and the background flickering, e.g., due to vegetation motion.

Figure 3: *Example of foreground-background segmentation.*

One can model the dynamic range image as a Mixture of Gaussians and update the parameters similarly to the standard approach [18]. This provides a segmentation of the point cloud which is quite noisy because of the spurious effects. These effects are significantly decreased by the dynamic MRF model [2] that describes the background and foreground classes by both spatial and temporal features. The model is defined in the range image space. The 2D image segmentation is followed by a 3D point classification step to resolve the ambiguities of the 3D-2D mapping. Using a spatial foreground model, we remove a large part of the irrelevant background motion which is mainly caused by moving tree crowns. Fig. 3 shows an example of foreground segmentation.

2.2. Pedestrian detection and multi-target tracking

In this section, we present the pedestrian tracking module of the system. The input of the module step is a point cloud sequence, where each point is marked with a segmentation label of foreground or background. The output consists of clusters of foreground regions so that the points corresponding to the same person receive the same label over the sequence. We also generate a 2D foot point trajectory of each pedestrian to be used by the 4D scene reconstruction module.

First, the point cloud regions classified as foreground are clustered to obtain separate blobs for each moving person. We fit a regular lattice to the ground plane and project foreground regions onto this lattice. Morphological filters are applied in the image plane to obtain spatially connected blobs for different persons. Then we extract appropriately sized connected components that satisfy area constraints determined by lower and higher thresholds.

This procedure is illustrated in Fig. 4. The centre of each extracted blob is considered as a candidate for foot position

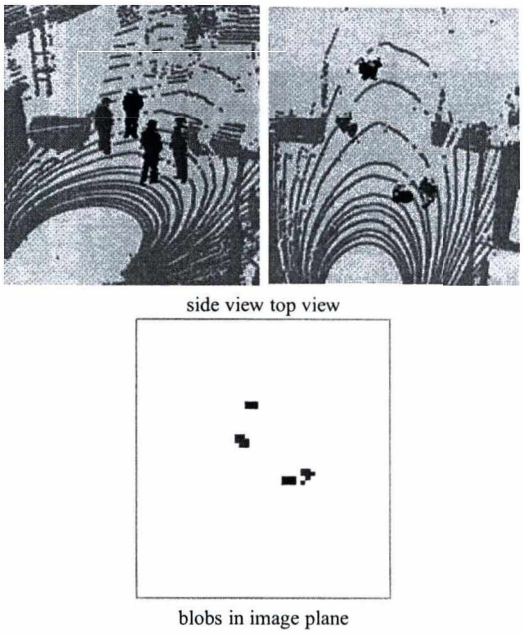


Figure 4: *Illustration of pedestrian separation.*

in the ground plane. Connected pedestrian shapes may be merged into one blob, while blobs of partially occluded persons may be missed or broken into several parts. Instead of proposing various heuristic rules to eliminate these artefacts at the level of the individual time frames, we developed a robust multi-tracking module which efficiently handles the problems at the sequence level.

Our multi-tracking algorithm receives the measured ground plane positions and for each frame iterates three basic operations, namely, data assignment, Kalman filter correction and Kalman filter prediction. The assignment operation assigns the candidate positions to objects, then the object positions are corrected and, finally, predictions for the subsequent positions are made and fed back to the assignment procedure. The algorithm can handle false positives as well as tracks starting and terminating within a sequence. Temporary track discontinuities are bridged in a post-processing step, while short false tracks are removed based on their length.

The tracker module provides a set of pedestrian trajectories, which are 2D foot centre point sequences in the ground plane. To determine the points corresponding to each pedestrian in a selected frame, the connected foot blobs around a given trajectory point should be vertically back-projected to the 3D point cloud. A result of tracking is demonstrated in Fig. 5 that shows two segmented point cloud frames from a measurement sequence in a courtyard. It also shows the video frames taken in parallel as reference. One can observe

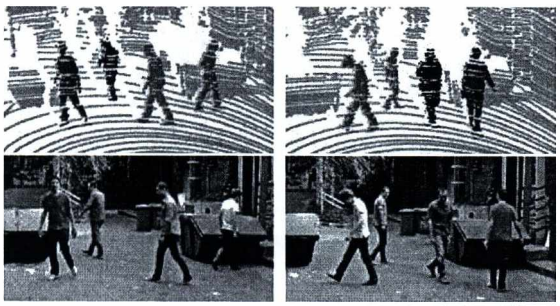


Figure 5: *Example of pedestrian tracking in a LIDAR sequence. Top row: point clusters whose colours identify the tracked persons. Bottom row: corresponding video frames displayed for verification.*

that during the tracking the point cluster of a pedestrian preserves its colour.

2.3. Environment reconstruction

In this section, we describe our method for static environment reconstruction. First we accumulate the background points of the LIDAR sequence collected over several frames, which results in a dense point cloud that represents the ground, walls, trees, and other background objects. Assuming that the ground is reasonably flat and horizontal, we fit an optimal plane to this point cloud using the robust RANSAC [7] algorithm that treats all other objects as outliers. Points close to this plane are considered as ground points in the following. For vegetation detection and removal, we have developed an algorithm, which calculates a statistical feature for each point in the merged point cloud based on the distance and irregularity of its neighbors, and also exploits the intensity channel which is an additional indicator of vegetation, which reflects the laser beam with a lower intensity. The remaining points are then projected vertically to the obtained ground plane, where projections of wall points form straight lines that are extracted by the Hough transform [6]. Applying the Ball-Pivoting algorithm [3] to the 3D points that project to a straight line, we create a polygon mesh of a wall.

In the reconstruction phase, static background objects of the scene, such us trees, containers or parking cars are replaced with 3D models obtained from Google's 3D Warehouse. The recognition of these objects from the point cloud is currently done manually, and we are now working on the automation of this step. For example, one can adopt here the machine learning based approach of [12], which extracts various object level descriptors for point cloud blobs representing the detected objects, while to obtain similar representations of the training models from the 3D Warehouse, they perform ray casting on the models to generate point clouds, finally the classification is performed in the descriptor space.

Sample results of our environment reconstruction are shown in Fig. 9. Model texturing is based on a set of photographs taken in the scene.

3. Creating 4D models of walking pedestrians

Relatively small objects such as pedestrians cannot be reconstructed from the LIDAR range data in sufficient detail since the data is too sparse and, in addition, it only gives 2.5D information. Therefore we create properly detailed, textured dynamic models indoors, in a 4D reconstruction studio. The hardware and software components of such a studio can be found in [4, 8]. For completeness, we give below a brief description of the reconstruction process.

Fig. 7 shows a sketch and a panorama of the studio where green curtains and carpet form homogeneous background to facilitate segmentation of the actor. The frame carries 12 calibrated and synchronised video cameras placed uniformly around the scene, and one additional camera on the top in the middle. The cameras are surrounded by programmable LEDs that provide direct illumination. The studio has ambient illumination, as well. Seven PC-s provide the computing power and control the cameras and the lighting.

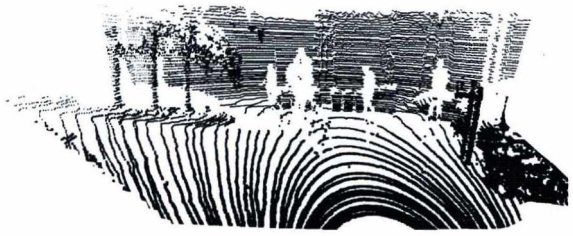
Currently, each set of 13 simultaneous video frames captured by the cameras is processed independently from the previous one. For a set of 13 images, the system creates a textured 3D model showing a phase of actor's motion. The main steps of the completely automatic 3D reconstruction process are as follows:

1. Colour images are extracted from the captured raw data.
2. Each colour image is segmented to foreground and background. The foreground is post-processed to remove shadows [4].
3. A volumetric model is created using the Visual Hull algorithm [13].
4. A triangulated mesh is obtained from the volumetric model using the Marching Cubes algorithm [16].
5. Texture is added to the triangulated mesh based on triangle visibility [8].

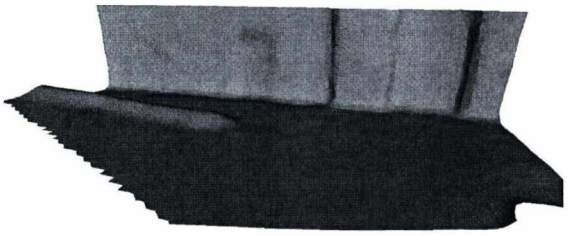
Fig. 8 shows an example of augmented reality created with the help of the 4D reconstruction studio. Several consecutive phases of an avatar walking in a virtual environment are displayed.

4. Integrating and visualising the spatio-temporal scene model

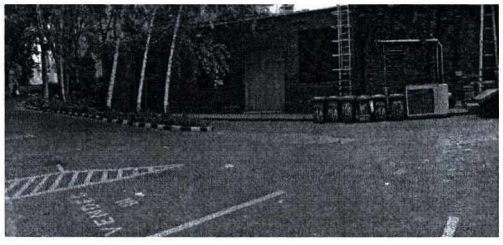
The last step of the workflow is the integration of the system components and visualisation of the integrated model. The walking pedestrian models are placed into the reconstructed environment so that the center point of the feet follows the trajectory extracted from the LIDAR point cloud sequence. Currently, we use the assumptions that the pedestrians walk



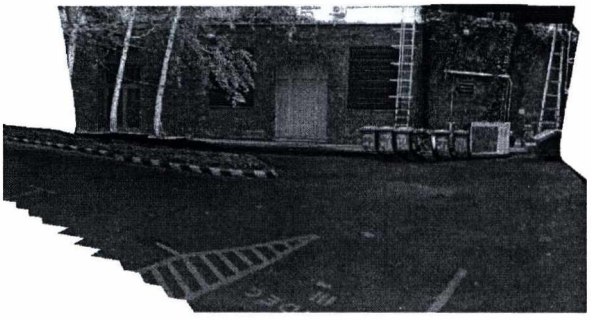
segmented background points



polygon mesh of walls and ground

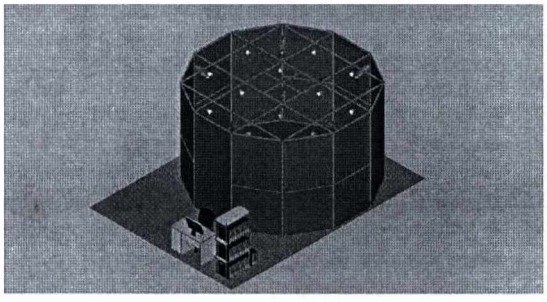


panoramic image for scene texturing



complete 3D model of environment

Figure 6: *Point cloud segmentation and environment reconstruction.*



sketch

panorama

Figure 7: *Sketch and panorama of a 4D reconstruction studio.*

forward along their trajectories. The top view orientation of a person is calculated from the variation of the 2D track.

To combine the 3D-4D data of different types arriving in different formats and visualise them in a unified format, we have developed a customised software system. All models are converted to the general-purpose OBJ format [20] which is supported by most 3D modelling programs and enables user to specify both geometry and texture.

Our visualisation program is based on the VTK Visualisation Kit [11]. Its primary goal is to efficiently support combining static and dynamic models allowing their multiplication and optimising the usage of computational resources. One can easily create mass scenes that can by viewed from arbitrary viewpoint, rotated and edited. Any user interaction with the models, such as shifting and scaling, is allowed and easy to perform.

The dynamic shapes can be multiplied not only in space, but in time, as well. Our 4D studio is relatively small. Typically, only two steps of a walking sequence can be recorded and reconstructed. This short sequence can be multiplied and

Figure 8: *A 4D studio actor walking in virtual environment.*

seamlessly extended in time to create an impression of a walking person. To achieve this, the system helps the user by shifting the phases of motion in space and time while appropriately matching the sequence of the models.

An important requirement was to visualise the 3D motions of the avatars according to the trajectories provided by the LIDAR pedestrian tracking unit. An avatar follows the assigned 3D path, while rotation of the model to the left or right in the proper direction is automatically determined from the trajectory. Sample final results of the complete 4D reconstruction and visualisation process are demonstrated in Fig. 9.

5. Conclusion and outlook

In this paper, we have introduced a complex system on the interpretation and 4D visualisation of dynamic outdoor scenarios containing multiple walking pedestrians. As a key novelty, we have connected two different modalities of perception: a LIDAR point cloud stream from a large outdoor environment, and an indoor 4D reconstruction studio, which is able to provide detailed models of moving people. The proposed approach points towards real-time free-viewpoint and scalable visualisation of large scenes, which will be a crucial point in future augmented reality and multi modal communication applications. As future plans, we aim to extend the investigations to point cloud sequences collected from a moving platform, and also implement automatic field object recognition and surface texturing modules.

References

1. C. Benedek, Z. Jankó, C. Horváth, D. Molnár, D. Chetverikov, and T. Szirányi. An integrated 4D vision and visualisation system. In *International Conference on Computer Vision Systems (ICVS)*, volume 7963 of *Lecture Notes in Computer Science*, pages 21–30. St. Petersburg, Russia, 2013.
2. C. Benedek, D. Molnár, and T Szirányi. A dynamic MRF model for foreground detection on range data sequences of rotating multi-beam lidar. In *International Workshop on Depth Image Analysis, LNCS*, Tsukuba City, Japan, 2012.
3. F. Bernardini and et al. Mittleman, J. The Ball-Pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.
4. C. Blajovici, D. Chetverikov, and Zs. Jankó. 4D studio for future internet: Improving foreground-background segmentation. In *IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, pages 559–564. IEEE, 2012.
5. Q. Chen. Airborne LIDAR data processing and information extraction. *Photogrammetric engineering and remote sensing*, 73(2):109, 2007.
6. R.O. Duda and P.E. Hart. Use of the hough transformation to detect lines and curves in pictures. In *Comm. of the ACM*, volume 15, pages 11–15, 1972.
7. M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Comm. of the ACM*, volume 24, pages 381–395, 1981.
8. J. Hapák, Z. Jankó, and D. Chetverikov. Real-time 4D reconstruction of human motion. In *Proc. 7th International Conference on Articulated Motion and Deformable Objects (AMDO 2012)*, volume 7378 of *Springer LNCS*, pages 250–259, 2012.
9. X. Hu, C.V. Tao, and Y. Hu. Automatic road extraction from dense urban area by integrated processing of high resolution imagery and LIDAR data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35:B3, 2004.
10. Z. Jankó, D. Chetverikov, and J. Hapák. 4D reconstruction studio: Creating dynamic 3D models of moving actors. In *Proc. Sixth Hungarian Conference on Computer Graphics and Geometry*, pages 1–7, 2012.
11. Kitware. VTK Visualization Toolkit. `http://www.vtk.org`, 2013.
12. K. Lai and D. Fox. Object recognition in 3D point clouds using web data and domain adaptation. *International Journal of Robotic Research*, 29(8):1019–1037, 2010.

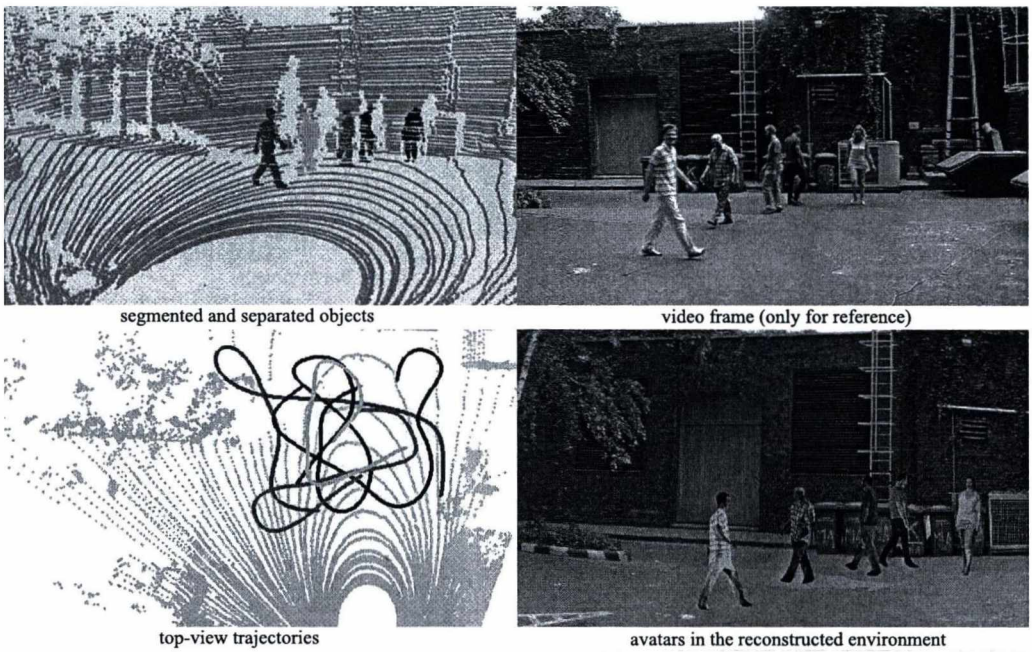


Figure 9: *Results of object tracking and integrated dynamic scene reconstruction.*

13. A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:150–162, 1994.

14. H.S. Lee and N.H. Younan. DTM extraction of LiDAR returns via adaptive processing. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(9):2063–2069, 2003.

15. P. Lindner and G. Wanielik. 3D LIDAR processing for vehicle safety and environment recognition. In *IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, pages 66–71. IEEE, 2009.

16. W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. ACM SIGGRAPH*, volume 21, pages 163–169, 1987.

17. S.C. Popescu and R.H. Wynne. Seeing the trees in the forest: Using LIDAR and multispectral data fusion with local filtering and variable window size for estimating tree height. *Photogrammetric Engineering and Remote Sensing*, 70(5):589–604, 2004.

18. C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:747–757, 2000.

19. F. Tarsha-Kurdi, T. Landes, and et al. Grussenmeyer, P. Hough-transform and extended RANSAC algorithms for automatic detection of 3D building roof planes from LIDAR data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Systems*, 36:407–412, 2007.

20. Wavefront Technologies. OBJ file format. Wikipedia, Wavefront .obj file, 2013.

Learning Shape Matching for Augmented Reality

Marton Szemenyei[1] and Ferenc Vajda[1]

[1] Department of Control Engineering and Informatics, Budapest University of Technology and Economics, Budapest, Hungary

Abstract

Tasks involving 3D shape are relatively common in Virtual and Augmented Reality systems. In this paper we present a learning shape matching method which we will use to replace real-world objects with virtual ones that are similar in shape. We will use this algorithm in an Adaptive Augmented Reality system that will fill a real-world scene with virtual objects so that the objects would fit into the scene. In our method, we first segment the 3D scene into primitive shapes, then we construct a graph encoding the properties and the relations of the shapes. We then use a graph kernel function to make our problem compatible with common learning methods, such as support vector machines. We have also tested the method using our own dataset consisting of stereo images, and have found satisfying results.

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Image Processing And Computer Vision]: Object recognition

1. Introduction

Tangible User Interfaces[1] and Tangible Augmented Reality[2] has been an area of intense research in the last decade. The goal of these systems is to allow users to interact with virtual objects by manipulating real objects, which results in a natural interface. Dynamic Shader Lamps[3] and Dynamic Object Texturing[4] are two noteworthy examples, both applying virtual textures on real objects. Other important examples are Tangible Bits[1], and the Virtual Round Table (VRT)[5], which uses real objects as placeholders for virtual ones.

Our system is somewhat similar to VRT: Using a stereo camera pair, we aim to create an Adaptive Augmented Reality system that finds the most fitting place for virtual objects in an unknown environment.

Our goal with this paper is to present our solution to achieving a logical pairing of virtual and real objects. The main criterion of the pairing method is that the real and virtual objects must have similar physical properties, so that the user could effortlessly manipulate the real object as he would do with the virtual one. However, many important physical properties, like mass, roughness of surface cannot be measured visually, therefore we must restrict our collection of determining physical properties to shape and size.

It logically follows, that the object pairing method needs to be based on shape matching, or shape similarity. Nonetheless, there is a large number of shape matching methods. Our aim for this work was to create a shape similarity method that requires no a priori information on the virtual objects, but can learn from instances of labeled real objects. Also, we wanted to make no assumptions on whether large-scale or small-scale shape attributes are important for pairing, therefore we attempted to encode all information on the shape of an object, and allowed the learning algorithm to make that decision.

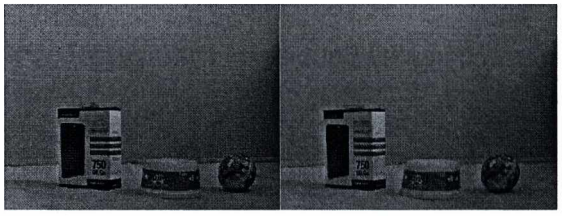


Figure 1: *A stereo image pair of an example scene*

Our method first converts the stereo image pair (Figure 1) into a 3D point cloud for further processing. Then, we use a modified RANSAC algorithm by Schnabel[6] to seg-

ment the point cloud into primitive shapes, like planes, cylinder, spheres, etc. Then we construct a graph from these primitives, where nodes represent the primitives themselves, while edges encode the spatial relationship between the nodes. Our algorithm then uses our modified random walk graph kernel[7] to insert the problem into a support vector machine.

2. Related Work

In this section we will give an overview of the work related to ours. In the first part of this chapter we will discuss shape matching methods. The second part will be devoted to the problem of learning on graphs.

Shape matching is one of the basic problems of computer vision and Augmented Reality. There are a number of methods available, like the RANSAC[11] algorithm, however, for this method a reference model of the object needs to be given. Numerous learning shape matching methods are available as well, that use 3D shape distributions[9], or features[10] to learn, therefore they don't need reference models. However, in order to use these methods a separate segmentation step is needed first.

One other method, created by Schnabel[6] uses a different approach. They created a modified RANSAC algorithm, that is able to find multiple instances of several different primitive shapes in a point cloud efficiently, and they use it to segment the point cloud. From that, they construct a topology graph to encode neighborhood relations of the primitives. Finally, they use brute-force graph matching to compare the graph to a reference model.[8] In our method (see Section 3.) we replaced the brute-force matching part with a learning algorithm in order to allow more flexibility and avoid depending on a priori information.

Learning on graphs is an important task in machine learning. Still, it is a difficult problem, since most machine learning algorithms work well with vectors, but not with structured objects, like graphs. The difficulty is that the simplest ways to assemble a vector from a graph is not invariant to the ordering of the graph nodes.[12] Additionally, graphs of different sizes will result in vectors of different lengths, which adds further technical difficulty.

One of the possible solutions is described in detail by White[12] who examines the different spectral representations of graphs and combines them to create a generative part-based model for learning. However, this representation only makes the feature vector constructed from the graph semi-invariant to the node order, and alignment step is still needed for matching.[12]

Another possible solution is using a kernel function. A kernel function is a positive semi-definite, symmetric function, that returns the similarity of two arguments.[7] One of the basic graph kernels is the random walk kernel, which measures two graphs by performing random walks along its edges. The similarity score of the two graphs is derived from the probability of performing the same walk on the two graphs simultaneously. The score is calculated as the following:[7]

$$K = \sum_{i=1}^{N} w(i)\mathbf{e}^T A^i \mathbf{s} \tag{1}$$

$$w(i) = e^{-\xi i} \tag{2}$$

Where A is the adjacency matrix of the direct product of the two graphs, $\boldsymbol{s}$ and $\boldsymbol{e}$ are the probabilities of starting and ending a walk on a given node, and $w(i)$ is the relative weight of i long walks, ξ is a hyperparameter, and N is the maximum length of the walks considered. It follows from the equation that the random walk kernel is invariant to node ordering. Moreover, since the kernel calculates the direct product of the graphs, it can compare graphs of different sizes.

3. Learning Shape Matching

In this section we will describe the shape matching algorithm in detail. As mentioned earlier, there are two major parts of the algorithm: The first is the process of assembling the graph, while the second is the random walk kernel implementation. In the third part of this section we will describe our method for training and testing the algorithm.

3.1. Assembling the Graph

The first step in constructing the graph was to use the modified RANSAC algorithm of Schnabel to segment the point cloud into primitive shapes, for which we used the implementation available online. One difficulty is that with point clouds that have noisy surfaces, and are reconstructed from a single range image, the RANSAC algorithm frequently miscategorizes the primitives. This means, that the learning algorithm has to be able to learn the most typical mistakes.

The next step is building a graph from the primitive shapes (an example graph is shown in Figure 2). The nodes in this graph will be the primitives that the segmenting algorithm has found. However, each of these nodes has a number of additional features that describe the primitive shape. Moreover, different types of primitives always have a different set of features, which would mean that the nodes of the graph have feature vectors of different sizes, and meaning. To avoid this complication we use just one feature vector, which is a concatenation of the unique vectors of each primitive type. The undefined values of the vector (e.g. if the primitive is a cylinder, then the features of a plane, sphere, etc.) are set to zero. In our implementation, we considered the following features of the primitive shapes:

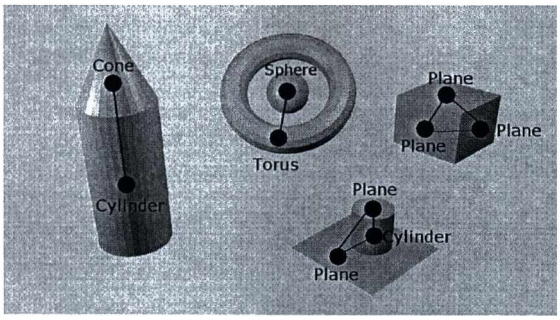


Figure 2: *The graph constructed from primitive shapes (only significant edges are drawn)*

Plane:	Area Diameter Bounding Box Area
Sphere	Radius
Cone	Radius Height
Cylinder	Radius Height Angle
Torus	Inner Radius Body Radius

In addition to that each nodes have a coordinate system, which is defined by a special point of the primitive, and a direction (usually the direction of an axis, or a surface normal). The only exception is the sphere, which has no unambiguous direction. However, this means that we can define a transformation between the primitive shapes i.e. the nodes of the graph as the transformation between their coordinate systems. These transformations will be the edges of the graph, encoded by a quaternion and a translation vector.

This way we will always construct a full graph, since the information whether two nodes are neighbors will be encoded in the features of the edge between them. The main advantage of this is, that the adjacency property between two nodes becomes a real value instead a binary one, which means that we are unlikely to have problems later caused by missing edges, resulting in a more robust solution.

3.2. Random Walk Kernel

The next step is to insert this graph into a random walk graph kernel. However, we need to make a slight modification to the original algorithm, because the random walk kernel originally works with weighted graphs, where the weights are real numbers, not vectors. To achieve this, we need to define a kernel function between nodes, and between edges. Then, we need to rewrite the random walk kernel to use these subkernels wherever the original method requires the product of two node or edge weights.

Thus, the kernel function between the edges will be used to calculate the Adjacency matrix of the direct product graph. However, the edge kernel will only use two features of an edge between two nodes: The $\cos(\alpha/2)$ part of the quaternion, and the squared distance between the nodes. The reason for excluding the rest is that the exact orientation of a node is ambiguous, and we may introduce significant noise into the algorithm. To get the exact form of the edge kernel we are using a variation of the RBF kernel:

$$E_v = e^{(-\gamma \cdot \Delta\omega - \delta \cdot \Delta d - \mu \cdot d)} \tag{3}$$

Where γ, δ and μ are the hyperparameters of the edge kernel, $\Delta\omega$ is the absolute difference of the cosine of the angles, Δd is the absolute difference of the squared distances of the two edges, d is the average of the squared distances. We have introduced the average distance into the kernel because nodes that are close in space are likely to be the part of the same object, whereas distant nodes are not. Therefore, we chose a measure of edge similarity which punishes long edges regardless of their similarity.

We will also construct the vector containing the starting and ending probabilities of a walk using a kernel function defined between the nodes: To create the vector, we will compare every node of graph A to every node of graph B, thus assigning a weight to every node of the direct product graph. This means that we will be more likely to start a walk on nodes that are similar, than on ones that are dissimilar. But first, we need to overcome the problem that we have constructed a feature vector for nodes from features that differ in scale significantly. There are area-type, distance-type and angle-type features, which means that a simple kernel functions, like polynomial of RBF are not sufficient. To tackle this problem, we calculate an inverse goodness value from the features, by dividing the absolute difference of the two feature values by their average. This normalizes all factors into the [0-1] range. Then, we applied an RBF kernel to the inverse goodness value as well.

$$W = \sum_{i=1}^{M} \frac{|a_i - b_i|}{a_i + b_i} \tag{4}$$

$$N_v = e^{-\theta \cdot W} \tag{5}$$

Where θ is another hyperparameter. The random walk kernel has two more hyperparameters: The first one is N, the maximum length of walks considered, while the other is the slope of the exponential weight function $w(i)$. There is one more modification that we have introduced to the kernel. We have observed that the score of the kernel function depended significantly on the length of the graphs, which was undesired. In order to compensate for this, we have modified the final

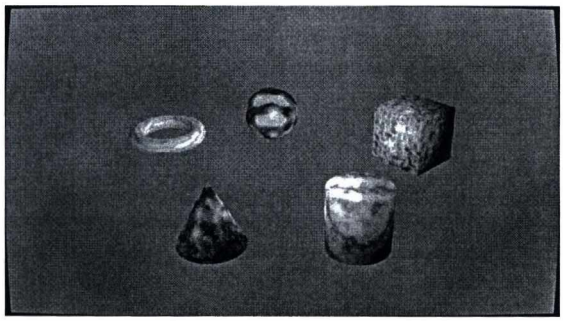


Figure 3: *The example scene created in Blender*

score of the kernel to be:

$$K_f = \frac{K}{(m \cdot n)^{\varepsilon}} \tag{6}$$

Where m and n are the lengths of the two graphs being compared, and ε is the seventh hyperparameter. There is the possibility of using an exponential or logarithmic scaling of the graph sizes, but the polynomial variant gave us the best results.

3.3. Training and Testing

To train and test the algorithm, we have created a virtual scene in Blender (shown in Figure 3) consisting of five objects: A cube, a torus, a cylinder, a sphere and a cone. We have used a virtual scene for testing, because our focus was to test the performance of the learning algorithm, and we didn't want camera-specific problems, etc. to interfere. We have taken 200 stereo images of the scene from different angles, and labeled the set with bounding boxes. For each shape we created 400 graphs: a positive and a negative example for each image. In order to find the optimal values of the seven hyperparameters of the algorithm we have implemented a cross-validation training method using the SVM implementation of MATLAB. We have used 280 training examples 60 validation and 60 test examples. Since optimizing with seven parameters would take a vast amount of time, we made a very rough pre-optimization step using only a fraction of our data, which allowed us to narrow down the range in which the optimum might be.

4. Results

The results of our cross-validation and testing are shown in Table 1. As seen there, we were able to achieve relatively low error rates, which means that the learning has been successful. Moreover, the hyperparameters required to achieve near optimal performance are close for all five shapes, which means that the process of tuning hyperparameters won't have to be redone when the algorithm has to learn to recognize more shapes. The best sets of hyperparameters we have found during cross-validation are shown in Table 2.

Primitive	Training	Validation	Test
Cube	7.5%	11.66%	15%
Sphere	2.14%	3.33%	8.33%
Cone	4.28%	8.33%	10%
Cylinder	4.28%	6.66%	11.66%
Torus	1.78%	6.66%	6.66%

Table 1: *The Error Rate of the Algorithm*

	θ	γ	δ	μ	ε	N	ξ
Cone & Sphere	0.01	0.01	0.01	10^{-4}	2.5	10	13
Others	0.01	0.01	0.01	10^{-4}	3.5	10	16

Table 2: *The Found Parameters*

5. Conclusions

In this paper we have introduced and described a method for shape matching that uses a primitive graph and a kernel function to learn. We have shown that the algorithm is capable of achieving satisfying results, even in a dataset with many occlusions.

In the future we are considering to introduce further features that will hopefully improve the accuracy of the method. We are also planning to modify the algorithm, so that it would be capable of localizing the detected shape in a large scene, since this will be essential if this method is to be used in our Augmented Reality system.

References

1. Hiroshi Ishii, Brygg Ullmer, Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms, *Proceedings of CHI*, 1997.
2. Mark Billinghurst, Hirokazu Kato, Ivan Poupyrev, Tangible Augmented Reality, 2001.
3. Deepak Bandyopadhyay, Ramesh Raskar, Henry Fuchs, Dynamic Shader Lamps : Painting on Movable Objects, *The Second IEEE and ACM International Symposium on Augmented Reality*, 2001.
4. Kresimir Matkovic, Thomas Psik, Ina Wagner, Denis Gracanin, Dynamic Texturing of Real Objects in an

Augmented Reality System *Proceeding of IEEE Virtual Reality 2005*,pp. 245–248, 2005.

5. Wolfgang Broll, Eckhard Meier, Thomas Schardt, The Virtual Round Table - a Collaborative Augmented Multi-User Environment, *Proc. of the ACM Collaborative Virtual Environments*, pp. 39–46, 2000.
6. Ruwen Schnabel, Roland Wahl, Reinhard Klein, Efficient RANSAC for Point-Cloud Shape Detection, *Computer Graphics Forum*, **26**(2):214–226, 2007.
7. S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, Karsten M. Borgwardt, Graph Kernels, *Journal of Machine Learning Research*, pp. 1201–1242, 2010.
8. R. Schnabel, R. Wahl, R. Wessel, R. Klein, Shape Recognition in 3D Point Clouds, *Computer Graphics Technical Report*, 2007.
9. Robert Osada, Thomas Funkhouser, Bernard Chazelle, David Dobkin, Matching 3D Models with Shape Distributions, *International Conference on Shape Modeling and Applications*, pp. 154–166, 2001.
10. Aleksey Golovinskiy, Vladimir G. Kim, Thomas Funkhouser, Shape-based Recognition of 3D Point Clouds in Urban Environments, *International Conference on Computer Vision*, 2009.
11. Martin A. Fischler, Robert C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography, *ACM Graphics and Image Processing*, **24**(6):381–395, 1981.
12. David H. White, Generative Models for Graphs, 2009.

INTERAKTÍV MESEKÖNYV GYEREKEKNEK - A KISKAKAS GYÉMÁNT FÉLKRAJCÁRJA ESETTANULMÁNY

Ruttkay Zsófia, Bényei Judit

Moholy-Nagy Művészeti Egyetem Budapest

Kivonat

Napjainkban egyre fiatalabb korban kerül a gyerekek kezébe a kicsi, érintéssel használható, vonzó számítógépek új családjának egy-egy képviselője, mint a táblagépek vagy okostelefonok. Egyes szülők és szakemberek az olvasás halálát vizionálják az eszközök láttán, míg mások mint az ismeretszerzés forradalmian új, vonzó médiumát üdvözlik. Az éppen csak születőben lévő műfajok sokfélesége és az összehasonlítási alapok hiánya miatt egyelőre nincs megfelelő tudományos alap a kérdés (árnyalt) megválaszolására. A cikkben a MOME TechLabban készült, A kiskakas gyémánt félkrajcárja című mese interaktív változatát vesszük nagyító alá, és osztjuk meg egy tudatos tervezés és tesztelés tapasztalatait.

Először összefoglaljuk az interaktív mesekönyv tervezési dimenzióit és az interakciók lehetséges narratív szerepét, majd bemutatjuk az általunk követett tervezési elvet. Végül egy kisiskolásokkal végzett feltáró empirikus vizsgálatot ismertetünk. E kutatás fő célja annak a feltérképezése, hogy az interakcióval életre keltendő illusztráció milyen módon hat az olvasás folyamatára mind az olvasástechnika, mind a motiváció szintjén, illetve a kisgyerekek hogyan „vesznek részt" az interaktív elemeken keresztül a narratíva megformálásában A tanulmányt a kutatás tartalmi és metodológiai tanulságaival és további vizsgálatok lehetőségének körvonalazásával zárjuk..

Kulcsszavak (according to ACM CCS): 1.3.3 [Computer Graphics]: interaktív mesekönyv, olvasás, illusztráció, multimodális interakció, táblagép, empirikus vizsgálat.

1. Bevezetés

Napjainkban egyre fiatalabb korban kerül a gyerekek kezébe a kicsi, érintéssel használható, vonzó számítógépek új családjának egy-egy képviselője, mint a táblagépek vagy okostelefonok. Egyes szülők és szakemberek az olvasás halálát vizionálják az eszközök láttán, míg mások mint az ismeretszerzés forradalmian új, vonzó médiumát üdvözlik. Az éppen csak születőben lévő műfajok sokfélesége és az összehasonlítási alapok hiánya miatt egyelőre nincs megfelelő tudományos alap a kérdés (árnyalt) megválaszolására (P. Cocozza (2014)). Ezzel összefüggésben, a tervező is bajban van ha "jó tervezési elveket" szeretne követni. Csábító az interakció nyújtotta lehetőségek sokasága. Valóban, jópár alkalmazás az interaktív elemek nagy számával véli elérni – a lebilincselést előtérbe helyező – célját, és meghódítani a piacot. De igazán nem tudjuk, hogy mikor, mennyi interaktív elemet érdemes, jó alkalmazni, hogyan kell élni a multimediális átfedésekkel tartalmi és jel szinten, hogyan időzítsük az interakciókat.

Ebben a cikkben egy saját tervezésű interaktív mesekönyv tapasztalatairól számolunk be. A könyvet amolyan "orvosi ló" mintájára, de nagyon tudatosan terveztük és valósítottuk meg, majd vizsgáltuk hogy miként használják azt 8 éves gyerekek.

A következő fejezetben összefoglaljuk az interaktív könyv tervezésének dimenzióit, a tervezői kérdéseket. Majd bemutatjuk A kiskakas gyémánt félkrajcárja című interaktív mesekönyvünket és az ott követett elveket. Részletesen tárgyaljuk annak tesztelését és az eredményeket. Végül összefoglaljuk a tapasztaltakat, és további szükséges kutatásokat vázolunk.

2 Az interaktív könyv tervezési dimenziói

Már egy lapozós gyerekkönyv esetében is tetten érhető, hogy az illusztráció és az írott szöveg viszonya többféle lehet: a kép törekedhet a szövegben elmondottak szigorúan hű leképezésére (noha a kép mindig többet, mást is megjelenít meg, mint a szöveg), vagy éppen az ott nem kifejtett részletek révén ad hozzá az élményhez és interpretációhoz, avagy nem is törekszik tartalmi, "szemantikus illusztrálásra", hanem inkább hangulatot, vagy érzelmeket jelenít meg jelzésszerűen vagy absztrakt módon. A „megfelelési szintjétől", illetve a képnek a szöveget erősítő vagy éppen annak ellentmondó hatásáról egy elméleti keretben tárgyalja Varga Emőke (Varga Emőke, (2012.)). Sőt, az általunk feldolgozott mese, A kiskakas gyémánt félkrajcárja is már sokféle illusztrációval került forgalomba, hagyományos mesekönyv és dia formájában. Ezek összehasonlítása önmagában igen tanulságos (Varga Emőke, szóbeli közlés).

Amint a kép meg tud mozdulni, sőt, hanghatásokra is képes, a két, illetve három médium jel és jelentés szintű kapcsolatának lehetősége kibővül. A megjelenítésen túl a mozgás és hang időzítése, azok előhívásának módja és ismételhetősége is .szerepet kap, mind az „olvasói" értelmezés mind az interaktív használat szintjén. Ebből következően egy sor további tervezői dimenzió, kérdés merül fel. Egy interaktív könyv oldalán (azaz a kézben tartott képernyőn) tipikusan valamilyen gesztussal (leggyakrabban érintéssel) idézhetünk elő valamilyen változást. Egy-egy interaktív könyv „interakciós elemeit" az alábbi szempontok alapján elemezhetjük és hasonlíthatjuk össze:

Az interakció időzítése és feltétele
Az illusztráció életre kelése történhet *automatikusan*, mintegy filmbejátszásként, vagy *interaktívan*, az olvasó beavatkozása által. Noha az előbbire is találunk példákat interaktív könyvekben, ahol a mese egy-egy részének szöveges elbeszélését akár helyettesíti is a „könyvbe ágyazott film", a további vizsgálódásunkban eltekintünk az automatikus megoldásoktól.

Az életre keltés vagy *feltétel nélkül* bármikor megtörténhet, vagy valamilyen *feltételhez kötött*. A feltétel lehet (fiktív) *időbeli*, csak az (elbeszélt) narratíva egy bizonyos pontján vagy azon túl elevenedhet meg a kép. Ezen túl a feltétel lehet *egy vagy több korábbi interakció* megtörténte. Speciálisan, egyes változások csak *korlátozott számban* – akár csak egyszer – hívhatók életre, míg mások *akárhányszor* ismételhetők.

Az interakció eredménye
Az olvasó által előidézett változások lehetnek *időlegesek* (pl. a madár csicsereg, a gazdasszony ringatózik a hintaszéken) vagy *maradandó hatásúak* (pl. csökken a tűz intenzitása). Az utóbbi esetben tovább árnyalja a lehetőségeket, hogy egy (mentálisan) egyszeri és maradandó esemény újra meg újra előidézhető, ha automatikusan „visszarendeződik" a hatás. Például a Kiskakast üldöző török császárra koppintva az a földre huppan, de egy idő múlva magától talpra áll és változatlan módon folytatja az üldözést.

Az interakciók funkciója
A nemzetközi piacon forgalmazott interaktív mesékben az interaktív módon előidézett változásoknak igen gyakran *nincs narratív szerepe*. Rossz esetben ezek még a történethez sem köthetők, csak magának az *interaktív élménynek* (az eszközhasználat kiélésének) kedvéért szerepelnek, míg jó esetben *atmoszférateremtő és/vagy ismeretközlő funkciót* töltenek be.

Ezzel szemben vannak olyan változások, melyek interaktív előidézése egy *narratív eseményhez kötődik*. Ilyenkor az olvasó mintegy újrajátssza az eseményt. Elképzelhető lenne egy olyan szereposztás is, hogy az olvasó ne is „lapozhasson" mindaddig, amíg nem idézte elő a narratíva kívánta változásokat a képen. Még bonyolultabb a kérdés és a tervezői tér, ha az interaktív könyv nem egy (általában már korábban nyomtatott könyv vagy esetleg film formában megjelent) lineáris történetet mond el, hanem a történet a szöveg és az interakciók együtteseként bontakozik ki.

Az életre keltés modalitásai
A táblagépeket leginkább az érintőképernyőn keresztül, egy vagy több ujjas gesztusokkal – szimpla-dupla érintés (touch), elhúzás (drag), seprés (swipe) – használjuk. De a kéz gesztusok modalitása mellett további – sokak által nem ismert és kevésbé kihasznált – lehetőség rejlik az eszköz rázásban-ringatásában, az eszköz pozíciójának megváltoztatásában, illetve a mikrofonon keresztüli, elsősorban nem szövegfelismerésre, hanem hangintenzitás érzékelésére (pl. fújás) alapuló inputban.

Hanghatások
Ha a képen előidézett változások *természetes hanggal* járnak, kell hogy kísérjék őket a jellemző hangok. Furcsán, amolyan némafilmesen hat, ha nem jár puffanással, dörrenéssel egy könyv leesése vagy egy ágyú elsülése. Kérdés, hogy legyen-e hangja a képen állandóan, vagy huzamosan jelen levő mozgó elemeknek? Például esetünkben a méhek zümmögése a

méhek jellemzője, melynek reprodukálása ismeretet is közvetít. Ugyanakkora hanghatások zavarhatják az olvasásra való koncentrálást, és nehezítik a hangos narráció megértését. Mindenképpen rossznak tartjuk a narratívához nem kapcsolódó állandó háttérzene használatát, még ha az pl. a történet filmes változatából való is.

Segítség

Az interaktív képernyők éppen a használat közvetlensége és a természetessége miatt válnak népszerűvé, nincs szükség a vizuális megjelenéstől idegen input mechanizmusok és eszközök beiktatására. Ugyanakkor, ha egy képen több, esetenként időben is változó interakciós lehetőség is van, és azok nem egyetlen módon aktiválhatók, akkor nem magától értetődő, hogy mikor és miként kelthető életre a képen valami. E téren alapvetően kétféle tervezői gyakorlat figyelhető meg. Ha a tervező a *kép életre keltésének felfedezését magát is kalandnak*, élménynek szánja, akkor akár semmiféle, a használatra vonatkozó segítséget nem épít be, hanem bízik az olvasó eszközismeretében és játékos kedvében. A másik lehetőség az, amikor van valamilyen segítség. Az úgynevezett *előhívható segítség* egy felhasználói menü gombra (mely lehet a képbe illeszkedő grafikus elem is) kattintva, vagy például az eszközt fejünk felé tartva jelenik meg vizuálisan, olvasható el szövegesen. Az *automatikus segítség* állandóan látható, vagy egy idő elteltével automatikusan jelenik meg az illusztráción magán. A vizuális megjelenést úgy kell megtervezni, hogy az ne uralkodjon el magán az illusztráción, de mégis egy szempillantás alatt érthető legyen.

Tipográfia

A betűmérettel, színnel való kiemelésre, valamint a *szöveg és kép (részleges) együttes kezelésére* már egyes kortárs gyerekkönyvekben is látunk fantáziadús példákat. Ugyanakkor a képernyőn a szöveg is változhat. A szöveg vizuális megjelenése az olvasók szintjeinek megfelelően opcionálisan alakítható (szótagolt vagy egybeírt, kis-nagybetűs, írott-nyomtatott), illetve egy-egy szóhoz kapcsolható interakció: érintésre a szó elhangzik (Dr. Seuss. (2010) The Cat in the Hat) (ez elsősorban a kiejtés gyakorlását segíti, vagy megjelenik róla lexikonszerű és/vagy grafikus magyarázat (Chocolapps. (2012) The Three Little Pigs). De fordított szereposztás is előfordul: a kép egyes elemeit megérintve, megjelenik annak szóbeli megnevezése (Dr. Seuss. (2010) The Cat in the Hat). A kép és szöveg együttes mozgatásának játékos lehetőségeit gyakran figyelhetjük meg az abc-t vagy számokat tanító interaktív könyvek esetében.

A szöveges és képi tartalom elhelyezése sokszor – mintegy jobb elv híján – a hagyományos könyvbeli konvenciókat követi. A nyomtatott könyv hagyományos tervezési átvétele szerintünk nemcsak unalmas, de a technológia nyújtotta lehetőségeket sem használja ki az interaktív könyv mint új műfaj megteremtésére. Egy másik, gyakran előforduló hiba, amikor a (sokszor pl. a nyomtatott könyvből, vagy esetleg filmből átvett) *kép uralja a képernyőt*, és arra a grafikához kompozícióban nem illeszkedő szövegdobozt helyeznek, mely gyakran kicsi a belé szuszakolt szöveghez viszonyítva.

3. Egy interaktív mesekönyv tanulságai

3.1 A kiskakas gyémánt félkrajcárja tervezése

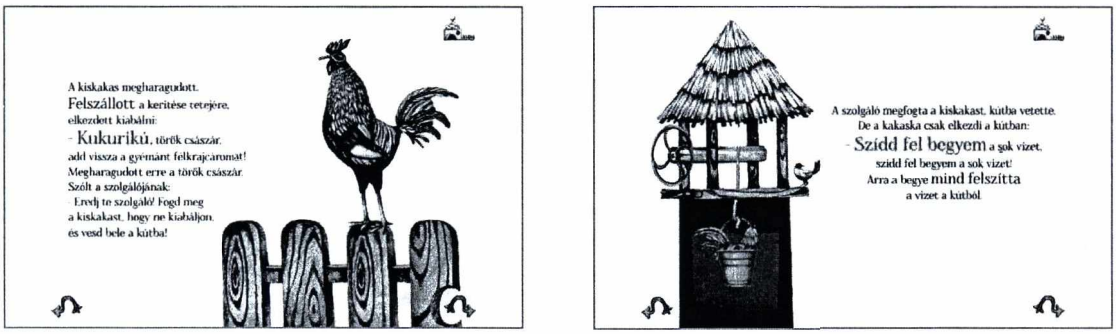
Az interaktív könyv alapja a (nem csak) magyar nyelvterületen népmeseként fennmaradt és mai napig népszerű A kiskakas gyémánt félkrajcárja történet. Pontosabban, a sok szöveges változat közül Arany László átdolgozását választottuk (Arany László (1900-1901)). Ez a feldolgozás jól tagolt, ismétlést és szimmetriát a jó (kiskakas) és rossz (török császár) konfliktusában végigvívő történetet mesél el sok mozgást és hanghatást megjelenítő, ritmikus elemeket is tartalmazó, de archaikus nyelven. A helyzet tipikusan falusi, több olyan vonatkozással (kemencébe vetés, kút, méhkas) amit a mai városi kisgyerek nem ismerhet. E jegyek alapján a történet mintegy "felkínál" lehetőségetek az interakcióra, mozgó képre.

Az alkalmazás 5-8 éves, az olvasással éppen ismerkedő gyerekeknek készült, és Android és iOS operációs rendszerű eszközökre ingyenesen letölthető, angol nyelven is (ld. MOME TechLab web oldal).

A tervezés során az alábbi elveket szögeztük le:

1. Az illusztráció legyen míves, magas színvonalú, eredeti, az appok és interaktív könyvek világában uralkodó vektor-grafikus ábrázolástól eltérő. Minden lapon egyedileg tervezett elrendezésben jelenik meg a szöveg és az (interaktív) illusztráció. Az illusztrációk ceruzával készült részletgazdag, monokróm rajzok, a mese hangulatához illeszkedő archaikus stílusban. A kompozíció levegős, és kellő helyet hagy a szövegnek.
2. Ha a képen szereplő jelenségek, események hanggal járnak, akkor a képet, illetve az interakciót is egészítse ki hanghatás.
3. Az interaktívan előidézhető változások legnagyobb része a narratívához kapcsolódjon – ezeket narratív interakciónak nevezzük a továbbiakban (NI).

4. Szerepeljen néhány, a narratívához nem kapcsolódó atmoszférateremtő interakció (A).
5. Inputként a jól ismert érintés mellett egy bonyolultabb gesztus is szerepeljen, valamint a kevéssé használt döntögetés.
6. Az interakciókra a gyerek maga kell hogy rájöjjön, minden segítség nélkül.
7. A tervezés minden aspektusa az elsősorban olvasás útján megismert történet megértését segítse.



1. ábra: Két jelenet A kiskakas gyémánt félkrajcárja interaktív meséből.

3.2 Az empirikus vizsgálat körülményei

Az interaktív mesekönyv használatára egy első, részletező-feltáró kutatást végeztünk, mely az „olvasás" folyamatának megfigyeléséből valamint a használat utáni interjúból tevődött össze. A kísérletet 2 csoporttal végeztük el. Az egyik (T csoport) az interaktív mesével ismerkedett meg, míg a másik (P csoport, egyben kontroll csoport) először a mese illusztrált, nyomtatott változatát olvashatta el. A kísérlet előtt a tanítónőktől kaptunk képet a gyerek profiljáról, illetve egy, a gyerekkel készített előzetes interjúban, arról, hogy mennyire ismeri a szóban forgó mesét. Kísérletünkhöz a T (tablet) csoportban egy budai általános iskola 7, második osztályba járó tanulója volt alany. A csoportban fiú-lány, jól és rosszul olvasó, illetve az érintőképernyős média (tablet, okostelefon) rutinos és első használója kiegyensúlyozottan szerepeltek.

A gyerek a felmérő beszélgetés után kézhez kapott egy Galaxy Nexus tabletet. Először egy rövid, kifejezetten az eszköz lehetőségeinek a bemutatására készült alkalmazáson próbálhatta ki a mesében majd szereplő interakciós gesztusokat, beleértve a kerék tekerést és a billegetést is. Ezek után „Olvasd el a mesét ahogy tetszik, kedvedre játsszál az illusztrációkkal ameddig akarsz!" felszólítással magára hagytuk. Ténykedését (és az interjúkat is) videóra vette egy korábban felállított kamera. Miután a gyerek jelzett, hogy befejezte az ismerkedést a mesével, a kísérletvezető megkérte, hogy idézze fel a történetet, illetve célzott kérdéseket tett fel a képben-interakcióban megjelenített, a gyerek számára feltehetőleg új információkra (kút és kemence működése). Végül kötött és szabad kérdések formájában az élményre, illetve a tetszésre gyűjtöttünk adatokat. A videóra felvett anyagot a kísérletvezető mellett egy kívülálló személy kódolta.

A fenti csoport mellett hasonló protokoll szerint zajlott a másik (P: print) csoporttal is felmérés. Ez a 8 (szintén másodikos, budai általános iskolás) gyerek először egy hagyományos, nyomtatott könyv formátumát kapta meg a mesének, amelyet az interaktív mesekönyv egy-egy karakterisztikus mozzanatát mutató képpel illusztráltunk. Nekik ezt kellett elolvasniuk és ez alapján felidézni a történetet, illetve válaszolni ugyanazokra az információkra (kerekes kút és kemence működése) vonatkozó kérdésekre, mint az előző csoportnak. Ennek az eredményeit itt külön nem részletezzük, csak mint kontroll csoportra hivatkozunk.

3.3 Eredmények

Az 1. táblázat az egyes *interakciók használatáról* ad képet a T csoportban. Az adatok, valamint esetenként az interjúk alapján az alábbi megállapításokat tehetjük:

Az olvasók az N (narratív) interakciókat sokkal többet használták (relatíve), mint az A (atmoszféarteremtő) interakciókat. Mondhatjuk, hogy az olvasók mintegy „újrajátszották" a történetet.

A gyerekek több olyan interakciót hajtottak végre, mellyel a török császárt (illetve szolgálóját) büntették-ugratták, mint olyat, amellyel a kiskakast segítették. Ez a szimmetrikus párokat tartalmazó jeleneteknél (2 és 5) vált különösen egyértelművé. De az is kiugró, hogy a gyerekek már-már szadisztikus örömmel csipkedtették a török császárt a méhekkel (N8).

A gyerekek kellő számban ismételt interakcióval többnyire „beteljesítették" a történetben szereplő eseményeket azokban az esetekben, ahol a cél egyértelmű volt (N4.1, N6, N7, N9). Érdekes, hogy a legelső, a mese szempontjából kulcs fontosságú N1 eseményt csak kevesebb mint a fele teljesítette. Ennek több oka is lehet: Egyrészt az esemény, a félkrajcárra bukkanás nem konfliktus jellegű, másrészt maga az akció, szemétdombon kapirgálás, nem vonzó, és nem jár látható részeredménnyel. Végül lehet, hogy az 5 ismétlés túl monoton, hosszú időt vesz igénybe éppen a történet legelején.

Egyetlen gyerek sem élt az A4 interakcióval. Ehhez az eszközt billegtetni kellett volna, ami egy szokatlan vezérlő modalitás. De az is lehetséges, hogy sok gyerek nem ismerte a hintaszéket, annak működését. Továbbá ez egy záró „happy end" esemény.

A kútkerék működését imitáló körkörös mozgást (N4.2) nem igazán használták (jól) a gyerekek. Ennek oka részben az lehet, hogy az esemény egyáltalán nem szerepel a történetben, csak a mozgatható ábrán. (Itt az „olvasó" húzza fel a kakast a kerék tekerésével.) Másrészt, az interjúkból kiderült, hogy legtöbb gyerek nem tudta, hogyan működik a kerekes kút. Így a valós életbeli ismeret hiánya okozta azt, hogy a fikcióban, az interaktív felületen nem tudták azt „kipróbálni".

Egyes gyerekeknél igen eltérő interakciós viselkedési mintákat fedezhetünk fel. Például T1 szinte alig „nyúlt" a képekhez, míg T6 igen aktív volt e téren. (Mindketten használtak már érintőképernyős eszközöket, T1 lány volt, T6 fiú.)

Az olvasási stratégiára a videók elemzése valamint az utólagos interjú alapján következtettünk. Az alábbi megfigyeléseket tettük:

1. Egyetlen gyereket kivéve mindenki teljesen végigolvasta a szöveget, időnként még visszalapozva újra is olvasták.

az interakció hatása	kód	hányszor	T1	T2	T3	T4	T5	T6	T7	átlag
K kapirgál, végül megjeleni a félkrajcár	N1	5	4	3	3	5	1	5	2	3,29
K felugrik futás közben	N2.1	korlátlan	1	1	2	2	1	0	0	1,00
T a földre huppan futás közben	N2.2	korlátlan	0	0	3	3	0	6	0	1,71
K kukorékol	N3	korlátlan	2	1	2	5	1	2	2	2,14
K felszívja a vizet (fokozatosan)	N4.1	3	3	3	3	3	2	3	3	2,86
K-t kihúzza a kútból	N4.2	1	0	2	1	2	0	2	2	1,29
madár csicsereg	E1	korlátlan	0	2	2	0	0	2	2	1,14
K elugrik a szolgáló elől	N5.1	korlátlan	2	1	2	6	1	0	0	1,71
szolgáló felugrik a K után	N5.2	korlátlan	0	0	0	5	4	7	3	2,71
szélkakas elfordul	A2	korlátlan	0	1	1	0	0	0	2	0,57
K kioltja a tüzet (fokozatosan)	N6	6	0	6	6	6	3	6	6	4,71
K felszívja a darazsakat (fokozatosan)	N7	4	1	4	4	4	4	4	4	3,57
egy darázs elrepül a virágról (majd visszarepül)	A3	korlátlan	0	1	3	3	5	0	2	2
T szenved a méhektől	N8	korlátlan	0	1	3	7	4	9	1	3,57
K felszívja a pénzt (fokozatosan)	N9	3	3	3	3	3	3	3	3	3
hintaszék billeg	A4	korlátlan	0	0	0	0	0	0	0	0
a táblagépet kézbe vette a gyerek	--	---	I	I	I	I	I	N	I	6/7

1. táblázat: *Az interaktív elemek használata.*

2. A gyerekek többsége először a szöveghez fordult, általában annak elolvasása után kezdett a szöveg melletti képpel játszani. Csak 1 gyerek „vetette rá" magát az interaktív képre elsőként. Néhányan ugrálós stratégiát követtek, a szövegben olvasott akciót mintegy rögtön kipróbálva interaktívan.

Elmondhatjuk, hogy sem az interaktív tartalom, sem a hanghatások nem vonták el a gyerekeket az olvasástól. Ez a kísérlet nem alapozza meg azt a gyakran hangoztatott állítást, hogy a szövegnek, olvasásnak nincs esélye, ha interaktív akciókra is lehetőség van. Viszont az állandóan és eléggé intenzíven hallható, dramaturgiailag izgalmas (üldözés) eseményt kísérő hanghatás zavarta a koncentrálást.

A változatos tipo (oldalanként néhány, dramaturgiai szempontból fontos és/vagy a képen interakcióval életre kelthető szóra eltérő méretű, színű és vastagságú betűkkel hívtuk fel a figyelmet) vonzóbbá tette a szöveget, ami az izgalmas interaktív kép mellett elengedhetetlen. Egyes gyerekek – a médium sugallta konvenciók alapján – a kiemelést hiperlinkként értelmezték az első oldalakon, megpróbáltak „rákoppintani" egy-egy kiemelt szóra. Az interjúkból kiderült, hogy érzékelték a kiemelések szerepét: az a legfontosabb és/vagy interaktívan is megjeleníthető történés. Véleményünk szerint ez is hozzájárul ahhoz, hogy minden segítség és szinte „melléütések" nélkül megtalálták az interaktív illusztráció lehetőségeit, illetve hogy az N interakciókat preferálták.

Végül mind az alkotókat, mind a szakmabelieket leginkább az érdekli, hogy az interaktív könyv mennyiben motiválja az olvasást, és segíti-e a szövegértést. Az előbbi kérdésre az interjú kérdések (tetszett-e, miért, ajánlanád-e másnak, kinek, az interaktív vagy a hagyományos változatát olvasnád-e inkább, akarsz-e egy következő ilyen meseolvasásban részt venni) elemzése alapján kaptuk a következő képet:

1. A gyerekeknek kivétel nélkül tetszett az interaktív könyv. Indokként a mozgó képet, a történetbe való beavatkozás lehetőségét, különösen a császár megbüntetését, valamint vicces jeleneteket említették.

2. Többen, nagyrészt fiúk túl gyerekesnek tartották a történetet, ők több „akciót" szerettek volna. A lányoknak nem voltak ilyen jellegű kifogásaik.

3. Noha a gyerekek mind szívesen vettek volna részt egy újabb hasonló kísérletben, és szóban pozitívan értékelték az élményt, viselkedésük inkább „feladatteljesítő", feszült volt, érzelmeket legfeljebb szóban fejeztek ki, nem arccal vagy gesztussal. (Viselkedésüket feltehetően befolyásolta, hogy a kutatásra iskolájukban, közvetlenül a tanórák után került sor.)

A szövegértésre két forrás alapján következtettünk. Egyrészt a gyerek „szabadon" elmesélte (ismét) a történetet, másrészt kérdéseket tettünk fel egyes, csak a (mozgó-hangzó) képben közölt részletekre, mint a kút vagy kemence „működése". Alább a legfontosabb megfigyelések:

1. Ami az új fogalmak „kipróbálás általi" elsajátítását illeti, nem igazolódott a feltételezésünk. A kerekes kút példájánál maradva, a gyerekek, ha nem ismerték azt, akkor nem is jöttek rá hogy hogyan lehetne „megmozgatni" a képen. Néhányan kifejezetten kifogásolták, hogy „nem ilyen a kút" (mivel az eltért az általuk ismerttől), és a „kemencébe vetés"-nél pizzalapátot emlegettek többen, valaki meg párnának értelmezte a lapátot. Ez azt jelzi, hogy nem lehet elvárni, hogy a gyerek kontextusból (múltbeli falusi élet) kiragadott tárgyakat pusztán interaktív használattal meg tudjon ismerni. Sőt, maga az interakció sem működik, ha nincs meg az „affordance" mentális alapja (kútkereket tekerve emelkedik fel a vödör). Ugyanakkor a T csoportban több (és részletesebb) helyes válasz született a kerekes kút és kemence működésére, mint a P csoportban – tehát van pozitív szerepe a mozgásnak és hangnak a megismerésben.

2. Az olvasás után mindenkinél javult a történet felidézése (említett események, azok sorrendje). Az előzetes elmondáshoz képest nőtt az archaikus szavak illetve nyelvi fordulatok használata. Mindkét tekintetben a T csoport felülmúlta a P csoportot.

3. Érdemes kitérni az „olvasni nem szerető, táblagéppel otthonosan bánó fiú", T6 alanyunkra. Az ő számára ez az olvasásnak a vonzó formája. Amellett hogy használatban és az interjúkban is hosszasan időzött az interakcióknál, az ő esetében is érvényesült az előbbi plusz, sőt, azt is pozitívumkén tekinthetjük, hogy végigolvasta a számára kissé gyerekes történetet, és hasonló interaktív könyveket szívesen olvasna.

4. Többen kritikusan kérdeztek rá azokra a kis képi divergens részletekre (marad víz a kútban, egy méhecske megmenekül), melyek nem voltak összhangban a szövegben hangsúlyos narratív elemmel (nevezetesen a „mind" nem teljesedett be a képeken).

4. Tanulságok és további feladatok

Összességében elmondhatjuk, hogy a felhasználók azokat az interakciós lehetőségeket találták meg, értékelték pozitívan és értelmezték a narratíva „újrajátszásának", ahol a kép, a hang és az interakcióval elindított mozgás megerősítette a szöveg jelentését, betöltötte a szövegben szabadon hagyott értelmezéseket

Empirikus, feltáró jellegű vizsgálatunk eredményeként már levonhatunk néhány olyan konkrét tanulságot, melyek további kutatások kiindulópontjául is szolgálhatnak:

1. Ez a korosztály (még) nyitott a tömegestől eltérő, művészi képi ábrázolásra. Az ilyen, akár archaikus ábrázolás „megél" a legmodernebb eszközökön is.

2. A kép és szöveg, illetve egy fogalom ábrázolása és "saját fogalmi képük" közötti eltérésre a gyerekek kritikusan reagáltak, a szöveget illetve a saját prekoncepciót nem írta felül a (divergens vagy kiegészítő) képi tartalom.

3. Az új interaktív médium ígéretes lehetőségeket rejt az olvasásra, betűre szoktatásra is.

4. Az interakciók tervezésekor az olvasó és a megjelenített szereplők viszonya, valamint az olvasó meglévő mentális képei fontos szempontok kell hogy legyenek, az elsősorban (vagy kizárólag) informatikai terminusokkal operáló gyakorlattal szemben.

Ugyanakkor a benyomásaink érvényességi körét illetőleg további vizsgálatokra van szükség.

Mindenekelőtt nagyobb, szociálisan vegyesebb csoporttal, illetve a történetet egyáltalán nem ismerőkkel lenne érdemes megismételni a kísérletet. Az utóbbira ad lehetőséget az angol nyelvű változat, mely egyben egy, a magyar (vizuális, nyelvi és népi) kultúrától eltérő kultúrájú közeget is jelentene. Egy másik lehetőség a korosztályilag a mesére jobban fogékony óvodások bevonása a kísérletekbe (az önálló olvasást felolvasással helyettesítve).

Köszönetnyilvánítás

Az interaktív mesekönyv illusztrációit Szepesi Szűcs Barbara készítette, programozását Karasz Dániel végezte. A kutatás az EU FP7 „TERENCE", valamint a MOME Szolgáltató Nonprofit Kft. „Interaktív gyerekkönyv kutatás" projektjén belül készült. A projekt megvalósulást a Belügyminisztérium - Országos Főépítészi Iroda támogatta.

Köszönetet mondunk Gyúró Mónikának, Popella Dórának és Sárközi Zsoltnak, valamint a Budenz József Általános Iskola tanulóinak és tanítóinak a kísérletekben való közreműködésért.

Hivatkozások

[1] Arany László (1900-1901) Összes Műve. Közrebocsátja Gyulai Pál. IV. köt.: Magyar népmesegyűjteménye. – Bp.: Franklin Társ., 1900-1901.

[2] Varga Emőke (2012). Az illusztrált könyvek mediális átfedési: definíció és módszer.; Illusztrációtipológia – In.: Az illusztráció a teóriában, a kritikában, az oktatásban. – Bp.: L' Harmattan Kiadó, 2012. 35-41.; 49-63. p.

[3] Goodbeans (2012) Nighty Night! Interaktív könyv app.

[4] Dr. Seuss. (2010) The Cat in the Hat, Oceanhouse Media. Interaktív könyv app.

[5] Chocolapps. (2012) The Three Littlee Pigs, Interaktív könyv app. http://www.chocolapps.com/en/icemagproducts/the-three-little-pigs/ [letöltési idő: 2013. szeptember]

[6] MOME TechLab weboldal http:// techlab.mome.hu/felkrajcar [letöltési idő: 2013. szeptember]

[7] Ruttkay, Zs., Bényei, J., Sárközi, Zs. (2013) Evaluation of Interactive Childrens Book Design, in Proceedings of EbuTel Workshop, 13 September 2013 Trento, Italy, Springer. (megjelenés alatt)

[8] Paula Cocozza (2014) Are iPads and tablets bad for young children? TheGuardian online, 2014.1.8 http://www.theguardian.com/society/2014/jan/08/are-tablet-computers-bad-young-children

Implementation of a GPU Accelerated Image Segmentation Algorithm on Android Platform

Opra István Balázs,[1] Vajda Ferenc[1]

[1] Department of Control Engineering and Information Technology, Budapest University of Technology and Economics

Abstract

Among the image segmentation algorithms, color-based segmentation is becoming a hotspot of research due to the increasing performance of computational hardware, and the more frequent application of color images. Touchscreen mobile devices are already widespread, and they have considerable processing capability and color cameras – enabling them to perform more image processing tasks. Hence it is a key goal to find and implement better image-based detection methods on this new platform. In this work we propose a pixel-based color segmentation algorithm, which categorizes pixels based on their Mahalanobis distance in color space. A significant speed increase is achieved by performing certain steps on the GPU commonly available in Android devices. For testing, we use the problem of decoding the electronic resistor color code.

Categories and Subject Descriptors (according to ACM CCS): I.4.6. [Image processing and computer vision]: Pixel classification

1. Introduction

The aim of image segmentation is to reduce the original image to segments, which have a high correlation to real-world properties. These properties can include brightness, focus, or – perhaps the most straightforward one – color. A color image contains larger amount of information, which allows the usage of more sophisticated algorithms. This usually means a significantly higher load on the hardware performing the operation, and thus traditionally a lower speed. Additionally, in the past color image acquisition also incurred additional hardware cost, since color cameras were more expensive.

These days however, color cameras are available in virtually all touchscreen mobile devices, and the computational capability of top-of-the-line smartphones is on par with desktop PCs 3 years ago. The number of these devices is also constantly rising. This means that finding segmentation methods with effective mobile-based implementations is an important goal of research.

There are multiple popular methods of color image segmentation. A popular one uses region-growing (see [1] for an example). In this, certain kernel pixels are selected in the image, either on a predetermined, or a stochastic basis. Then, surrounding pixels are examined, and according to the specified homogeneity criteria, they are deemed either similar, or dissimilar to the original point (or set of points), and are thus included in or excluded from the region. An example of a successful homogeneity criterion is found in the work of Hernandez et al.[2]. The process is repeated until all pixels of the image have been classified into a region. The run time of such algorithms can be difficult to calculate deterministically, since the number of necessary steps usually depends on hard-to-quantify properties of the image. Also, there are additional concerns of stability - increased care need to be taken to ensure that the algorithm will not run into a situation, where two or more regions are consuming each other indefinitely.

Another method is to classify each pixel with a set criterion (pixel classification). There are techniques in this category, which employ iteration, and sample some qualities of surrounding pixels to adjust the classification of a given pixel. An example for this is the work of Loog et al.[3] in medical imaging. However, such methods also present the issue of non-calculable, or generally lower execution speed.

If the run time of the pixel-classification criterion is deterministic, and does not depend on other pixels, then the run time of the whole algorithm is $O(n)$, n being the number of pixels in the image. Implementation of such pixel-based

algorithms is also more straightforward, and is a great candidate for parallelization, since the classification of each pixel is independent from others. These qualities allow the usage of these methods in cases when stricter execution time limitations are present, such as hard real-time applications. An example for such a pixel classifier is the one proposed by Celik et al. in [4]. They use both a statistical boundary condition in color space, and a fuzzy logic system with rules based on intuitive observations about the pixels to be detected.

In this paper we explore a simpler, less calculation-heavy statistical pixel-based algorithm which uses the Mahalanobis distance in 3-dimensional color space to determine whether each pixel belongs to a set of prespecified colors. The algorithm is effective in finding objects with a homogeneous, artificial color in real-world lighting environments, and can achieve high execution speed on Android mobile devices with using the graphics processing unit.

For testing purposes, we have selected the problem of detecting the color code of through-hole mounted resistors, and thus, parts of the implementation are specific to this problem. For a description of the color code, see the official publication of IEC[5].

First, we give a brief overview of key concepts used in the algorithm. Then, we go over the major steps, also giving information on the actual implementation. Finally, we summarize the achieved results.

2. Background

2.1. Color representations

Objects with an artificial, homogeneous paint vary their color in real-world environments. The variation can be due to the actual incident angle of the illumination with respect to the camera, or the curvature and shape of the objects. It is important to note, however, that in a given lighting environment, the perceived color is largely the same with respect to *shade* (eg. blue, red), and only varies its *brightness* (eg. bright red, darker red). These descriptors are intuitive, and are largely tied to human vision. Thus, it is key to employ a color representation, which shows a high correlation with the human color perception. This means that small perceived differences in the color should correspond to differences of similar scale of location in the color-space. Another useful attribute is that the chromaticity and the brightness information should be explicitly accessible in the representation.

We have found that the RGB color space representation of a homogeneous color band of a resistor generally forms a peculiar, crescent-shaped distribution, see Figure 1. The shape is due to the fact that color shade and brightness information cannot be directly gained from a single axis of the RGB space. The crescent shape is difficult to characterize with mathematical methods, which means that determining whether a pixel is in this shape or not requires complex, potentially slower computational steps. This finding reinforces the commonly accepted position that the RGB color space is not a good candidate for color segmentation purposes.

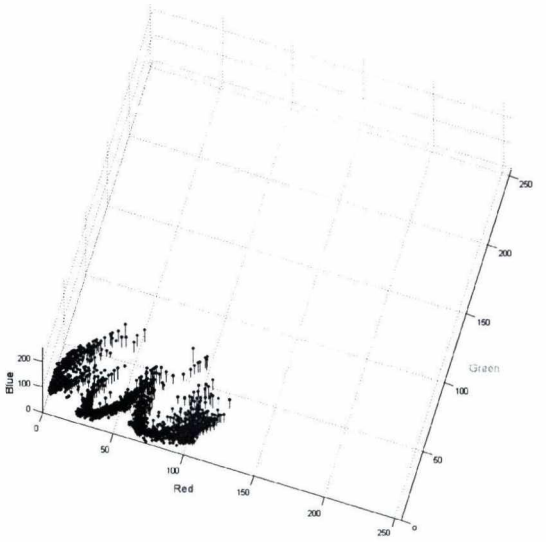


Figure 1: *Color bars in RGB space*

Instead of the red-green-blue representation, we elected the CIE $L^*a^*b^*$ color space. This is the first mathematically defined color space, based on the findings about human vision which suggest that different cells in the eye are sensitive to different wavelengths, and color perception is based on the difference in the generated signal strengths. The L (lightness) channel directly corresponds to the brightness of a color, while the a and b channels code the chromaticity. In this regard, it is similar to the HSV color space, which was efficiently employed by Guo et al. in [6].

The distribution of color band pixels in $L^*a^*b^*$ is shown in Figure 2. It can be observed that the pixels are more densely packed, and the main orientation of the distribution is parallel to the L axis. This is according to expectation - the color mainly only varies in its brightness, explicitly indicated by the L value. Given the shape of the pixel clouds, a simpler characterization is possible.

The traditional Euclidean distance however would not give good results, since it models the data set as a sphere, which is not accurate enough and would cause considerable overlap between the colors. S.L Phung et al. observe multiple pixel-based segmentation methods in their article [9]. In section 2.2.3, they approximate the pixel cloud with normal distributions, and use the Mahalanobis-distance to calculate the distance. We use this idea in our work for an ellipsoid-based modeling of the colors in $L^*a^*b^*$ space.

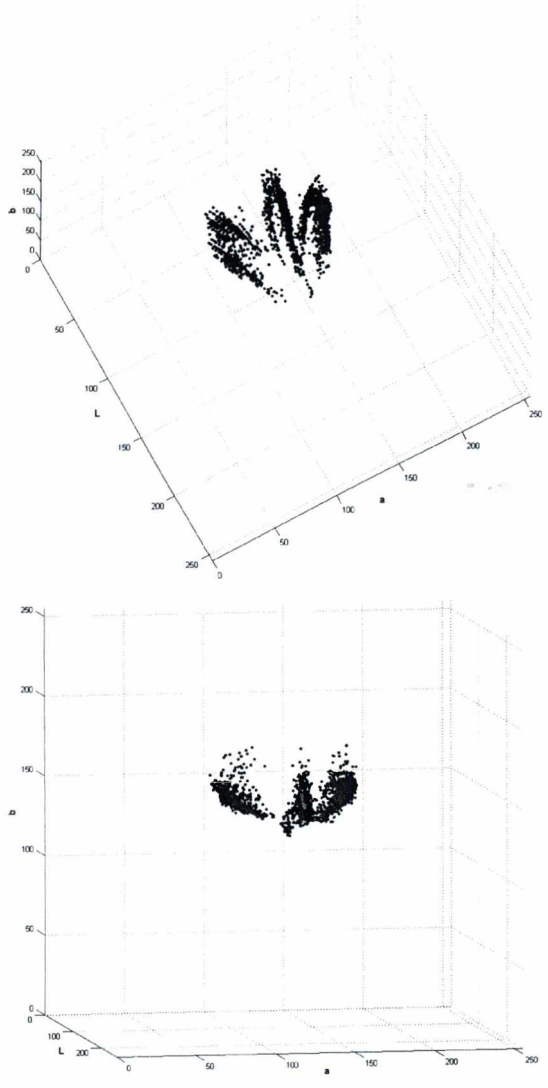


Figure 2: *Color bars in LAB space*

2.2. Mahalanobis distance

The Mahalanobis distance is not a metric by definition, rather it is a statistical measure which gives information about how much a data point belongs to a given set of points. It approximates the data cloud as an ellipse in 2D (or ellipsoid in 3D).

The Σ covariance matrix is a common descriptor of an ellipsoid. (Please note that in all following equations, vector quantities are, by default, column vectors.) If the dimension of our data set is n, and the data points are in the form of $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, where $x_i, (i = 1, \ldots, n)$ is a random variable with finite variance, then the elements of Σ can be calculated using:

$$\Sigma_{ij} = E\left[(x_i - \mu_i)\left(x_j - \mu_j\right)\right] \tag{1}$$

, where $\mu_i = E(x_i)$, the expected value of the i-th variable.

The eigenvectors of this matrix give the orientation of the main axes of the ellipsoid, while the eigenvalues give information about their length (they are equal to the variance of the x_i random variables).

Calculation of the Mahalanobis distance for an arbitrary $\mathbf{x_r}$ is the following, if we know $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu} = E(\mathbf{x})$:

$$D_M(\mathbf{x_r}) = \sqrt{(\mathbf{x_r} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x_r} - \boldsymbol{\mu})} \tag{2}$$

A new addition compared to the standard Euclidean distance is the multiplication by the inverse of Σ. An intuitive explanation for this is that we divide the distance from the mass center of the data set by the size of the ellipsoid in the direction of the observed point – which is the variance in that direction.

3. Segmentation algorithm

The algorithm can determine whether a given pixel has a predetermined color or not. Thus, the colors need to be specified prior to online execution. The descriptors are created by taking pictures of the resistors, and then sampling the pixels corresponding to a color band. We transform the sampled data set to the $L^*a^*b^*$ color space, and we compile its $\boldsymbol{\Sigma}$ covariance matrix, calculate $\boldsymbol{\mu}$, the center of mass, and we also determine a d distance threshold. This $(\boldsymbol{\Sigma}, \boldsymbol{\mu}, d)$ set specifies an ellipsoid of given size and shape in the 3D space.

During execution, the Mahalanobis distance of each pixel is calculated from the dataset specified by $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$, and if the resulting value is lower than d, it is deemed to be part of the color corresponding to the dataset. In essence, this is a thresholding method – see [8] – we get a binary image for each possible color.

This technique however, is highly susceptible to variations of the hue, intensity, and focus of the background illumination. If, for example, the reference pictures were taken in a brightly lit laboratory room, the segmentation algorithm will fail if run in the dim natural light of a window on a cloudy day. To increase the environmental lighting invariance, we propose a form of white balance correction.

3.1. White balance

White balance correction is a broadly used technique in photography. Its aim is to reproduce the true color of a picture element, regardless of the illumination when the image was taken. Our method requires an object to be present in the environment, which is considered white. A picture of this

object is taken, and its color properties are determined. This is done by compiling a histogram of the R, G and B channels. The bins containing the most pixels are selected for each channel, and the color defined by the corresponding R_w, G_w, B_w values is deemed to be the *white* color of the given environment (this method virtually finds the modus of the R, G and B channels, the resolution depending on the amount of bins).

In our application, it is important to normalize not only the color, but the intensity as well. To achieve this, we predetermine a bright white color with the RGB values of (I_w, I_w, I_w). (In the actual implementation the value of I_w is 240 on a 8-bit scale.) Then, we calculate the weighing values for the colors like this:

$$w_R = \frac{I_w}{R_w}, \qquad w_G = \frac{I_w}{G_w}, \qquad w_B = \frac{I_w}{B_w} \tag{3}$$

The normalization is then performed by multiplying each color channel of each pixel by the resulting w_R, w_G and w_B values.

This technique is employed on each picture used for creating the color descriptor sets, and also each time prior to running the detection algorithm.

3.2. Determining the threshold value

Another element which is key to the success of the algorithm, is setting an appropriate d threshold value. We use the following method to determine this. We perform the segmentation algorithm for a given color on a picture, with linearly increasing d values. We use *connected-component labeling* and *image moments* to select the largest connected blob in each resulting binary image. After his, we create a function which shows the area of the largest blob with regards to the value of d. With increasing d, the area also increases. Then, at a certain threshold, a small plateau region appears, where there is only little observable growth in the area. We select the initial value from this plateau region. If necessary, further fine-tuning is performed by hand.

4. Detection of the resistor body

If the resistor to be analyzed is in an environment, where there may be other resistors, or distracting background, it is important to be able to separate and highlight it prior to running the color segmentation algorithm in order to increase run speed, and to be able to handle images with multiple resistors. We propose the following method for this which works well on relatively homogeneous backgrounds with low saturation. Most of the operations are performed using OpenCV for Android in the implementation.

4.1. Selecting the resistor

On the camera preview feed, the user can at any time tap on a resistor to initiate detection. The program takes a 3MP image, and crops it to quarter resolution, centered at the tap coordinates. This method provides a simple yet effective way for selecting of a single resistor on the image.

4.2. Thresholding

Next, we use two different thresholding techniques in order to separate the resistor from the background. The cropped image is converted to the HSV color space, where the *saturation* channel is thresholded – see Figure 3. The threshold value was determined via hand tuning. Another binary image is created by first converting the original color image to gray scale, applying the separable 3×3 Sobel edge detection filter, then thresholding the resulting image, see Figure 4.

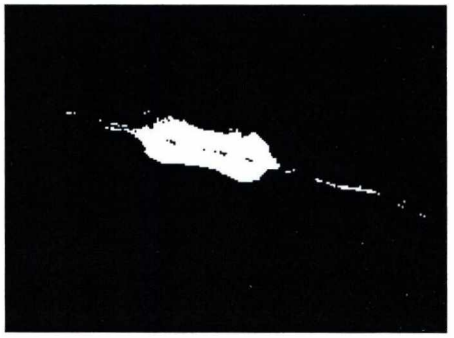


Figure 3: *Result of saturation thresholding*

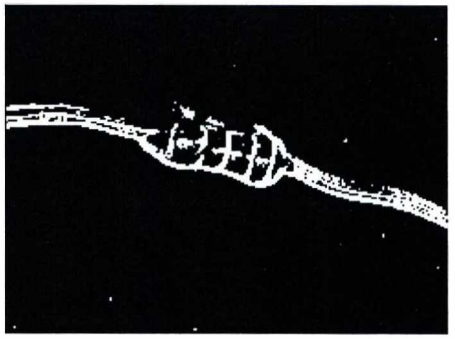


Figure 4: *Result of Sober filtering and thresholding*

The saturation thresholding alone gives good results most in of the cases when using a blank, low saturation background, there are times, however, when in order to get good focus, the device needs to be held close to the resistor. Then, the shadows cast by the device and the hand of the user can have a high enough saturation value to cause problems. On the other hand, the edges of the shadows are soft enough to not be detected by the Sobel thresholding if the limit is set properly. The edge-detected image, however, will not have active pixels inside the resistor body, only on its perimeter -

we need to fill these voids to get a binary mask covering the whole body.

4.3. Morphological post-processing

Usage of contour-detection algorithms did not lead to an outline which encompassed the entire resistor body. The issue stemmed from the fact that the Sobel edge detection filter did not continuously detect the edge of the body, there were some gaps, and other noise. Instead of this approach, we use morphological closing. For a general overview of morphology see [7].

In this process, first a set number of morphological dilatations are performed on the image, followed by erosions applied the same amount of times with the same kernel. After proper adjustment, we have found that this method gave good results for the Sobel-thresholded image: homogeneously filled binary masks, which resemble the original shape of the resistor – see Figure 5. Since we need the edge detection only to filter out shadows and highlights in the background, not for exact detection of the outline of the resistor, the slight deformation due to the morphology is acceptable. The morphological closing is also applied on the saturation thresholded image to fill any potential gaps.

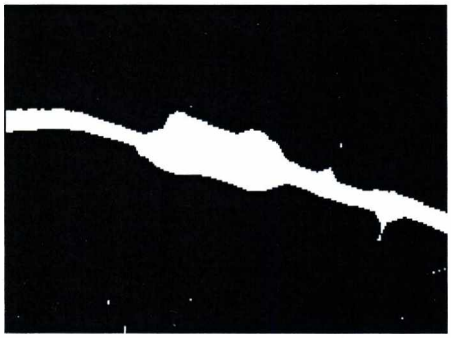


Figure 5: *Result of morphological closing on Sobel image*

After performing the morphological steps, the final mask is created by performing a binary `AND` operation on the thresholded images.

4.4. Removing the wires

In order to focus only on the body of the resistor, where the color bands are, the wires need to be removed. For this, we employ *ultimate erosion*. We start with the binary mask from the previous step, and perform erosions on it, saving each resulting image. After k erosions, we reach a state when all foreground pixels are removed. Then, we go back n steps in the memory, where we have an image which is already without pixels belonging to the wires, yet the body remains – see Figure 7(a). After this, we perform $k-n$ dilatations,

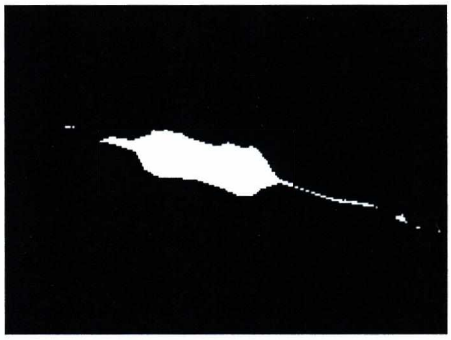
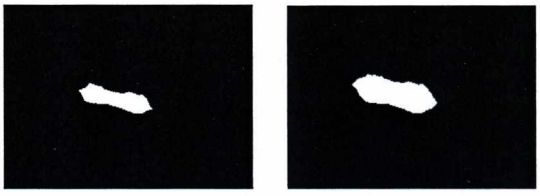


Figure 6: *Result of binary AND operation*

using the same kernel. The end result is a mask which fits the resistor body closely, and does not contain the wires – see Figure 7(b).

For this method to produce good results, it is key to ensure a fix ratio between the morphological kernel size and the resolution of the image. We use fix kernel size, and scale the image to a predetermined size.

Following these steps, the original image is cropped to the bounding box of the previously generated mask, and all original pixels where the mask is inactive are given the value $(0,0,0)$. See Figure 8.



(a) After $n-k$ erosions (b) End result

Figure 7: *Ultimate erosion*

Figure 8: *Masked resistor (still in RGB)*

4.5. Size normalisation

The size of the bounding box in terms of pixels varies depending on how large the resistor is on the image. The run time of the Mahalanobis-based thresholding algorithm depends linearly on the amount of pixels need to be processed, thus it is necessary to adjust this to a preset value prior to starting the algorithm.

The aspect ratio of the mask is not fixed - it depends on the orientation of the body of the resistor on the image and on the camera angle. Because of this, we could not aim for a preset resolution. Instead, we have set a pixel count N, and calculated the new (x_n, y_n) dimensions of the original (x_o, y_o) sized image the following way:

$$R^2 = \frac{N}{(x_o \cdot y_o)} \tag{4}$$

$$x_n = \frac{x_o}{R} \tag{5}$$

$$y_n = \frac{y_o}{R} \tag{6}$$

5. GPU implementation of the Mahalanobis-thresholding

After we have the image containing only the resistor body, and scaled to the predetermined size, we can do the Mahalanobis distance based thresholding. Since all contemporary Android-based devices have a graphics processing unit, we utilize this to enhance speed. Android provides the `GLSurfaceView` class for comfortable interfacing with OpenGL ES, which is currently the only way to directly control the GPU. We use this class as a template with an additional `GLSurfaceView.Renderer` descendant class in our implementation.

The masked image is converted to the $L^*a^*b^*$ color space using OpenCV, and loaded into GPU memory into a `GL_TEXTURE_2D` texture object. We also specify a frame buffer with its own texture object, since the output image does not carry interesting information for the user, and thus needs not to be visualized.

5.1. Vertex shader

We use the vertex and fragment shaders in our shader pipeline. A four-element triangle strip with the coordinates $(-1,-1),(-1,1),(1,-1),(1,1)$ is passed to the vertex shader, which covers the entire viewing area. These same coordinates are also passed in as texture coordinates, which are then interpolated for use by the fragment shader. Since the goal is to work on a flat, 2-dimensional image in the fragment shader, the Model-View-Projection matrix used is the identity matrix - no actual multiplication is performed.

5.2. Fragment shader

The thresholding is done in the fragment shader. Since we need to calculate binary images for 10 different colors (the color code contains 10 possible homogeneous colors), we need multiple passes. In OpenGL ES 2.0 it is guaranteed that data can be read back to CPU memory from a `GL_COLOR_ATTACHMENT0` binding point of a frame buffer. Access to the stencil and depth buffers may vary from device to device. Due to this we only use the color attachment, from which we can get 4 binary images per render pass, which are traditionally the *RGBA* channels. Thus, for 10 two-level images, we need 3 passes.

In between the passes, parameters describing the colors need to be adjusted. Each color is specified - as mentioned in section 3 - by the $(\mathbf{\Sigma}, \boldsymbol{\mu}, d)$ set. These are stored in a text resource in the application package, and are read in upon launching the app. In the OpenGL shader program, they are in `uniform` type variables, thus can be easily changed from the host application.

During calculation of the Mahalanobis distance, only $\mathbf{\Sigma}^{-1}$ is used in multiplication, so this inverse matrix is stored and passed into the shader program instead of $\mathbf{\Sigma}$ to avoid the on-line matrix inversion.

The fragment shader program fist samples a pixel using `texture2D()`, the input image (a `sampler2D` type uniform) and the interpolated texture coordinates provided by the vertex shader. Then, the pixel is checked if it is not pitch black (pitch black means having a $(0,0,0)$ value). In practice such pixels virtually never exist in photos taken by the device, so this modeling of the background does not introduce errors. If the pixel is not completely black, Mahalanobis distance calculation is performed.

The calculation for one color is separated into three steps:

$$\boldsymbol{\delta} = \mathbf{x} - \boldsymbol{\mu} \tag{7}$$

$$\mathbf{k} = \boldsymbol{\delta}^T \mathbf{\Sigma}^{-1} \tag{8}$$

$$\Delta = \mathbf{k} \cdot \boldsymbol{\delta} \tag{9}$$

, where $\mathbf{x} \in \mathbb{R}^3$ is the color of the pixel, and Δ is the resulting Mahalanobis distance. All three operations can be performed in a single GPU clock cycle. Then, if $\Delta < d$, the output pixel is set to active (1 value), otherwise it is inactive (0). We will refer to the resulting ten binary images as color masks.

5.3. Speed

Using the GPU for the distance calculation proved to improve algorithm speed dramatically. The tests were carried out using a Samsung Galaxy Tab 2 7.0 tablet device, running Android version 4.1. For comparison, we tried using the Mahalanobis distance calculator function implemented in OpenCV, and measured the run time. This function has to be called for each pixel, and thus means sequential execution on the CPU. The technique used for speed measurements was sampling the system uptime before and after the observed steps, this has a resolution of 1ms. The run time averaged for 2938ms with a standard deviation of 35ms. This

means that for 10 colors, the calculation takes around half a minute.

On the other hand, the first pass of the shader program on a new image takes 21ms on average, with 6ms deviation, and the second and third passes take between 1 and 2 ms. The combined run time of the three runs never exceeds 30ms. Based on these measurements, the achieved speed increase is over 100-fold when using the GPU.

6. Detecting the color bars

Alongside speed, high precision was a target of our work. In order to test this, we implemented a basic set of steps to solve the resistor code based on color masks.

6.1. Preconditioning

First, to fill out potential voids in the binary images, we employ morphological closing again. Since the pixel count of the images are already set (see 4.5) we can use a fixed size kernel.

After the morphology, we get the contours of each connected component in each color mask. In order to filter out small blobs of falsely detected color, we first find the largest blob among all the colors, which has an area of M pixels. Then, we go through all blobs, and remove the ones which have a size smaller than $\alpha \cdot M$. The value of α was set to 0.2 after fine-tuning.

6.2. Calculating the orientation of the body

The connected components we got after the last step all belong to a color band in the original image. But, in order to get their distance, we need more information. The blobs' shape can vary widely, and usually does not cover the entirety of the actual color band which they correspond to. Due to lighting irregularities and highlights, they can appear in the centerline of the body of the resistor, or far from it, towards its edge.

Our approach to get reliable distance information was to store the area and centroid of each blob, and project the centroids onto a line which is parallel to the body of the resistor. In order to do this, we need the orientation of the resistor body mask. First we calculate the following μ'_{ij} moments from the appropriate central moments:

$$\mu'_{11} = \frac{\mu_{11}}{\mu_{00}} \tag{10}$$

$$\mu'_{20} = \frac{\mu_{20}}{\mu_{00}} \tag{11}$$

$$\mu'_{02} = \frac{\mu_{02}}{\mu_{00}} \tag{12}$$

From here, we can get the θ_p orientation:

$$\theta_p = \frac{1}{2} \arctan\left(\frac{2\mu'_{11}}{\mu'_{20} - \mu'_{02}}\right) \tag{13}$$

Since the arctan function gives results in the $[-\pi/2 \quad \pi/2)$ interval, we need additional information. The value of μ_{11} is positive, if the orientation deviates from the vertical in the positive mathematical direction, and negative otherwise. Thus, by checking its sign, we can get the true orientation of the resistor body:

- $\mu_{11} > 0$:
  - $\theta_p \geq 0$: $\theta = -\theta_p$
  - $\theta_p < 0$: $\theta = \frac{\pi}{2} + \theta_p$
- $\mu_{11} < 0$:
  - $\theta_p \geq 0$: $\theta = \frac{\pi}{2} - \theta_p$
  - $\theta_p < 0$: $\theta = -\theta_p$

From here, we can project the $\mathbf{q_i}$ centroids of the color blobs. First, we build vector $\mathbf{b} = \begin{bmatrix} 1 \\ \tan\theta \end{bmatrix}$, which is parallel to the centerline of the body. Then:

$$\mathbf{P} = \frac{1}{\mathbf{b^T b}} \mathbf{b b^T} \tag{14}$$

$$\mathbf{q_{i,proj}} = \mathbf{P q_i} \tag{15}$$

6.3. True band centroids

Sometimes a color band produces two separate connected components in the appropriate color band. These need to be merged. Also, there are some artifacts which are not filtered out in the preconditioning step - these are incorrectly detected parts of color bands, whose centroids are close to the larger, correctly detected regions (a common example is small orange blobs at the top and bottom edges of a red band).

To do the merging, we first order all the projected centroids by increasing x coordinate, and get the d_i distances between neighbors. Then, we select the highest, D distance, and merge together all blobs of the same color which have less then $\alpha_d \cdot D$ distance from each other. The value of α_d was also set by hand tuning. The centroid of the new, merged blob is the algebraic average of the parents' centroids, and their area is summed.

Then, in order to remove the incorrectly detected color blobs at the edges of the bands, we check blobs of differing color, which are within $\alpha_d \cdot D$ range of each other, and we remove all, but the one with the largest area.

6.4. Decrypting the color code

After these steps, we have a list of points along a line which correspond to an actual color band on the resistor with a high probability. To read the color code, we need to know which direction to start from, since the result depends on the order we read each color band. On all observed resistors, the band to start with was the outer one whose distance from its neighbor is the larger. After we have determined this starting color band, reading the code is straightforward.

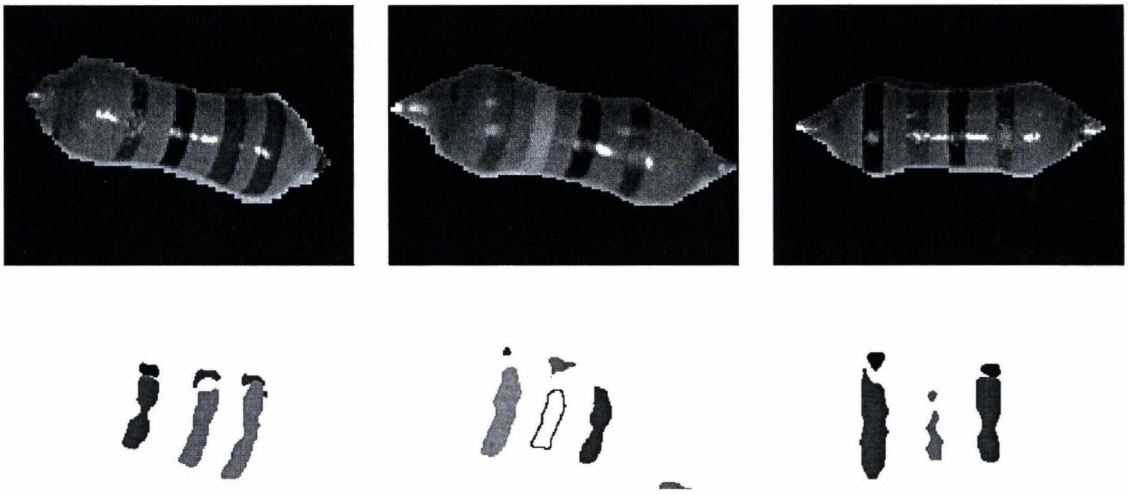


Figure 9: *Some resistors and results*

7. Testing

For testing we used resistors of the E24 series (5% tolerance) in the $[1k\Omega, 10k\Omega]$ range. This means 23 different pieces, with all ten possible colors represented. We first compiled the color descriptor set from 2 images per each color. In one image, we used a warmer shade ambient lighting, in the other, a more directed, bluish tint neon light, and applied the white balance correction in all cases.

Then, we did 46 test-runs, all resistors in both lighting environments. The result resistor value was correct in 39 cases, this means an accuracy rate of almost 85%.

To get more information about the failures we also visualized the detected color masks and the resistor mask after running the algorithm. Six failures were due to the incorrect masking of the resistor body – either only a part of the whole package, or a part of the wire was visible. The other error was due to an orange band largely being detected as red, and the post processing could not successfully filter this out.

Figure 9 shows some masked resistor images in RGB, and the corresponding segmentation results. The result images were compiled from all color masks with active pixels, and hand colored for better visualization.

8. Summary of results and future work

In summary, the Mahalanobis distance based thresholding shows impressive results in terms of speed when implemented on the graphics processing unit. Furthermore, its precision is adequate to enable more complex post processing tasks to work, such as determining the minuscule difference in relative distances of the color bands on a resistor body.

These qualities make it a good candidate for being the basis of various machine vision tasks, such as visual servoing. Other applications where artificial color markers are used are also possible.

There are also further opportunities for improving the algorithmic steps connected to resistor color code. One way of doing this could be the further tuning of the current parameters of the resistor body detection and post processing, and adding new rules, such as taking into account the expected shape of a detected color blob.

Another path can be using a feature point based algorithm (such as SIFT) in order to detect the resistor body. This would improve robustness in cases when the background is more complex, such as a circuit board with wiring and other electronic components.

An additional alternative for improving ambient lighting invariance could be using the method Kim et al. proposed in [10]. This would require some reworking of the GPU-based implementation to maintain high speed, since the color descriptor they employ is based on similar foundations, but contains more data – a different 2D ellipse for different intensity values.

Acknowledgements

This work was made possible by the help and insight of Michèle Gouiffes of Polytech Paris-Sud.

References

1. Jun Tang, "A color image segmentation algorithm based on region growing", *2nd International Conference on Computer Engineering and Technology (ICCET)*, **6**:634-637, 2010.
2. Hernandez, S.E., Barner, K.E., "Joint region merging criteria for watershed-based image segmentation", *2000 International Conference on Image Processing*, **2**:108-111, 10-13 Sept. 2000.
3. Loog, M., Ginneken, B., "Segmentation of the posterior ribs in chest radiographs using iterated contextual pixel classification", *IEEE Transactions on Medical Imaging*, **25**(5):602-611, May 2006.
4. Celik, T.; Ozkaramanli, H., Demirel, Hasan, "Fire Pixel Classification using Fuzzy Logic and Statistical Color Model", *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007.*, **1**:1205-1208, 15-20. April 2007.
5. International Electrotechnical Commission, Marking codes for resistors and capacitors. IEC 60062 ed5.0, 2004.
6. Xuan Guo, Baoping Guo, "Color image morphology based on distances in the HSI color space", *CCCM 2009. ISECS International Colloquium on Computing, Communication, Control, and Management, 2009.*, **3**:264-267, 8-9 Aug. 2009.
7. G. J. F. Banon, "Characterization of linear and morphological operators", *XII Brazilian Symposium on Computer Graphics and Image Processing, 1999. Proceedings.*, pp. 245-, 1999.
8. J. C. Russ, "Thresholding" in *The image processing handbook* (6th edition), New York: Taylor & Francis, pp. 395-398, 2011.
9. S.L.Phung, A. Bouzerdoum, and D. Chai, Sr., "Skin segmentation using color pixel classification: analysis and comparison", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(1):148-154, 2005.
10. Chi-Ho Kim, Bum-Jae You, Hagbae Kim, "Color Segmentation Robust to Brightness Variations by Using B-spline Curve Modeling", *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4874-4879, 9-15 Oct. 2006.

Creating 3D Models of Buildings by Car-Mounted LIDAR

Dmitry Chetverikov and Iván Eichhard

Institute for Computer Science and Control, Budapest, Hungary
Eötvös Loránd University, Budapest, Hungary

Abstract

Automatic reconstruction of large-scale outdoor objects like house facades is an important component of mixed-reality systems that model and visualise real world at different level of detail. The authors are involved in a project that utilises a car-mounted LIDAR to acquire a sequence 3D point clouds representing facades in a street. No GPS or IMU is used. Hundreds of point clouds need to be automatically aligned to obtain a realistic surface model of facades. In this paper, we present and compare two solutions to this complex registration problem. Our methods are based on two different, widely used techniques for registering two partially overlapping point clouds in presence of outliers. The proposed algorithms are capable of automatically detecting occasional misalignments. We analyse the operation of the algorithms paying special attention to the robustness, speed and optimal parameter setting.

1. Introduction

The Integrated 4D (i4D) project [1] by Institute for Computer Science and Control (MTA SZTAKI) aims at reconstructing, editing and visualising dynamic real-world scenes at varying level of detail and by fusing different kinds of input data. An important ingredient of such mixed-reality systems is the unit that builds 3D models of large-scale environments and scenes.

We contributed to the project by developing and testing two methods for automatic alignment of a long sequence of measured 3D point clouds acquired by a car-mounted LIDAR device carried along a street. (LIDAR stands for Laser Imaging, Detection and Ranging.) The complete aligned point set represents the street facades. Good alignment results in a high-quality surface model that can be efficiently textured by facade images taken simultaneously or separately.

Point cloud sequence alignment is a popular task in field robotics, remote sensing, spatial information sciences and computer vision. This problem is also addressed in urban area reconstruction [9] when the input data is often provided by airborne or car-mounted LIDAR devices. Compared to image-based reconstruction, reconstruction from LIDAR measurements is more robust to changing illumination conditions and lack of surface texture.

Aerial LIDAR data is used to reconstruct complete residential areas [15], while the car-mounted devices are usually applied to smaller scenes such as houses in streets and squares. Depending on the conditions of measurement, auxiliary sensors such as GPS (Global Positioning System) or IMU (Inertial Measurement Unit) can be used to support the alignment.

Many algorithms have been developed to solve the problem of automatic *registration of two point sets*. Some of them have been successfully used to align a long sequence of point clouds. One of such algorithms is the Normal Distributions Transform (NDT) registration algorithm [7]. Its efficiency and accuracy were studied in [14]. Another popular tool for registration of two point sets is the Iterative Closest Point algorithm and its variants [2,13] whose performance was investigated in [10].

The study [8] evaluates and compares ICP and NDT on LIDAR data. Since the original version of ICP is not applicable to partially overlapping point sets, the authors use a robustified version of the algorithm. This version imposes an upper limit on the distance between the points that can be matched: larger distances are simply discarded. In practice, such limit is hard to set, especially when there is a relatively large rotation between scans, or when point density varies significantly within a scan. In our LIDAR application, the rotation is limited, but point density variation can be large, which would make such ICP-variant impractical.

The Trimmed Iterative Closest Point (TrICP) registration

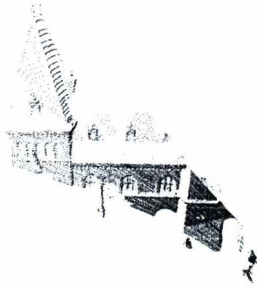


Figure 1: *A sample point cloud.*

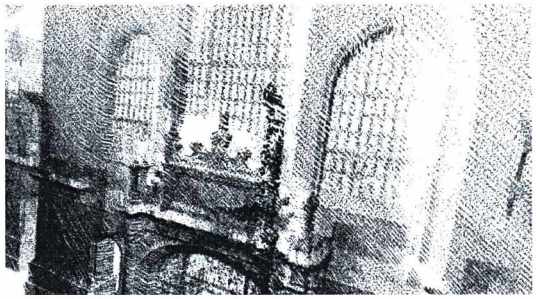


Figure 2: *A part of complete aligned point cloud.*

algorithm [2] is robust and free of such limitations. The comparative study [10] tests it as the standard, most widely applied robustification of ICP. TrICP has been successfully used to register different kinds of measured spatial data.

The main motivation of our study was to compare the two efficent point set registration techniques, NDT and TrICP, in the context of our application. In this paper, we present a point cloud sequence alignment method based on NDT and a method based on TrICP. The major contributions of the paper are the two methods and a discussion of our experience gained while applying the methods to real LIDAR sequences obtained for different streets and buildings.

The layout of the paper is as follows. In section 2, we describe the data acquisition procedure and introduce the alignment problem. Sections 3 and 4 are devoted to the proposed methods. Test results are presented in section 5, followed by discussion and conclusion in the final section 6.

2. Data Acquisition and the Alignment Problem

The 3D data is supplied by the Velodyne HDL-64E RMB-LIDAR device mounted on a car travelling along a street. The rotating multi-beam LIDAR device records 360°-view angle range data sequences of irregular clouds of unoriented points. Due to the intrinsic anisotropy of the data acquisition process, point density decreases with altitude and distance, which makes scanning and reconstruction of high or distant structures problematic.

When a wide facade or a short street is scanned, hundreds of 3D point clouds are stored. Figure 1 gives an example of point cloud. A cloud typically contains 30000−40000 points. The measurements are noisy, and they are always spoiled by outliers that make the alignment more difficult. These data are to be registered and aligned in a single point set, as illustrated in figure 2. The complete aligned point set is then triangulated to obtain a surface mesh. The mesh makes much better visible the misalignments 'hidden' in the final point set. The misalignments must be automatically detected and corrected.

Currently, we do not use any auxiliary sensor (GPS, IMU) to support alignment. Before doing that, one has to carefully analyse the precision of the sensor. If the precision is not sufficiently high, adjusting point clouds according to the sensor data may be counterproductive. For example, a small angular error may result in a significant position error at large distance.

In the context of the car-mounted LIDAR data registration problem, the main sources of outliers are moving cars and pedestrians, surfaces with unstable reflectance, interior of buildings, as well as curtains and windows.

Data points resulting from moving objects deteriorate the reconstruction of static environments. Their number depends on the density of the traffic. Such objects can be detected easily as their height and size vary in a limited, well-defined range.

Outliers resulting from objects with unstable reflectance, such as trees in the wind, are relatively rare. However, vegetation can occlude significant portions of buildings and generate unstable point clouds.

Interiors of buildings supply data whose character can change suddenly. In one view the measurements may cover a large area and be useful for alignment, while in the subsequent scans the usable area may decrease or disappear entirely. Curtains and windows pose a similar problem as the laser beam may be reflected from them in one view and penetrate it in the next view. Usually, these two categories of outliers in LIDAR scans appear as remote points behind the actual visible surface, and they can be detected based on this property.

Pre-conditioning of the measured data is very desirable since it improves the robustness of alignment and enhances the quality of the resulting mesh. Both of our methods presented below try to detect and remove the outliers prior to point cloud registration.

3. NDT-based Method

Our first point cloud sequence alignment method is based on the 3D NDT registration algorithm [7]. Before registration,

the acquired LIDAR data is pre-segmented by classifying each point as belonging to ground, short field object (vehicle, pedestrian), tall structure object (wall, roof, lamps post), or clutter. To achieve this, we use statistical features in the grid-based approach [4]. The procedure runs in real time as it uses only a few simple features of each grid cell such as the height difference between points within a particular cell.

The aim of the pre-segmentation is to select static, stable points that do not change their positions between scans. Such regions are, i.e., walls that are visible from large distances and appear in several scans as the car passes them by. The benefit of this is two-fold. First, the registration speeds up since less points are processed once the irrelevant points have been removed. Second, the preserved 3D data are the most useful regions of the scans. The regions that are hard or impossible to register, such as moving cars and pedestrians, are discarded.

The 3D NDT registration algorithm [7] uses a 3D voxel-based approach to match subsequent point clouds. We apply the 3D NDT to find the optimal rigid transformation between two neighbouring scans and validate the result since the registration may occasionally fail.

Since the 3D data is recorded in real-world environments, we can set up constraints on the transformation matrices. In typical urban traffic, the car carrying the LIDAR travels about 40–80 centimeters between consecutive scans; its speed cannot change drastically between scans. This limits the translation vector. Also, rotation should be reasonably small since the car stays on the road and turns only with a limited speed.

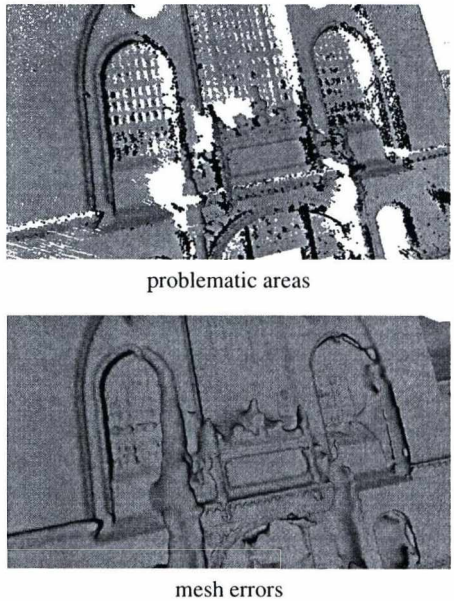
The validation procedure checks the obtained transformation matrix. If either the translation vector or the rotation matrix is unrealistic, the procedure rejects the result and skips the processed scan.

4. TrICP-based Method

Our second point cloud sequence alignment method is based on the TrICP registration algorithm [2]. Prior to point cloud registration, the input data are filtered to remove outliers in order to robustify and speed up the registration process. The procedure also provides surface normals that are used when a mesh is obtained from the point cloud sequence aligned by any of the two methods. In this section we, briefly discuss the main components of the TrICP-based method.

4.1. Data filtering and normal calculation

Similarly to the NDT-based method, the TrICP-based algorithm needs pre-processing to condition the input data, as discussed in section 2. The alignment block receives data processed by the pre-segmentation algorithm presented in section 3. In addition, we have developed a relatively simple but robust filtering procedure that removes other unreliable



problematic areas

mesh errors

Figure 3: *Problematic areas and resulting mesh errors.*

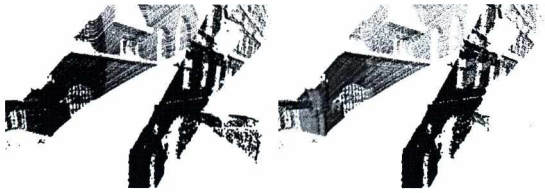
data such as occasionally measured interior structures of the buildings.

The filtering procedure also assigns normal vectors to the remaining points in a consistent way, alleviating the problem of the 'flipped' normals typical for many normal calculation algorithms. The Poisson surface reconstruction algorithm [6] we use assumes consistent normals. In the flipped areas, the normals point in the opposite direction which results in mesh errors, as illustrated in figure 3. In the figure, flipped normal areas are shown in dark, unmeasured areas in light.

The data filtering method is based on the Hidden Point Removal (HPR) operator [5] that provides a theoretical basis for testing visibility on unordered, unoriented point sets. Although the name reminds the hidden surface removal, the HPR does not use surfaces or surface reconstruction. Instead, the operator applies a view-based point set inversion method and a convex hull calculation algorithm on the inverted point set. The triangle connectivity of the convex hull transferred to the original point set provides consistent surface normals. In our solution, we selected the 'spherical flipping' [5] as the inversion method. Figure 4 illustrates the operation of our filtering and normal calculation method.

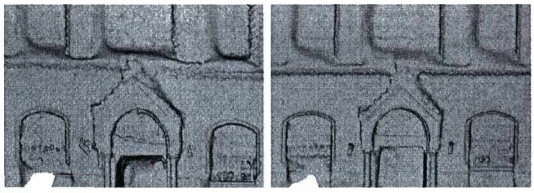
4.2. Data alignment

In a long sequence of point clouds, we select each K-th cloud and apply TrICP to register each selected cloud to the next one in the subsequence. The typical values of the temporal step K are 2–4. Each partial registration is characterised by a Mean Square Error (MSE) value, the mean square distance



input point set output set with normals

Figure 4: *Removing an outlier internal structure by the HPR operator.*



NDT result TrICP result

Figure 6: *Detail of Market Hall reconstruction.*

between the corresponding points in the two registered sets. Although the point-to-surface distance is often preferred to the point-to-point distance, in this particular application we still use the latter. The reason is that our data is dominated by flat surfaces of facades. As discussed in [11], the point-to-point metric is preferable for data with predominantly low or constant curvature.

A key parameter of the algorithm is the expected overlap of the two point sets. TrICP can search for the optimal value of this parameter and set it automatically, which needs additional computation. This makes sense when the overlap varies across the sequence. In the car-mounted LIDAR data, the overlap is sufficiently stable and easy to set. For this reason, we use a fixed overlap for each value of the parameter K.

The series of registrations results in a series of MSE values. We analyse these values and detect poor registrations as large outliers in the MSE array. A standard robust outlier detection rule [12] is used which is based on the median absolute deviation from the median value.

Each poor registration is discarded and substituted by a short sequence of registrations with the unit temporal step $K = 1$. In other words, a gap in the registration chain is bridged by aligning all clouds within the gap. The chance that this will repair the chain is high as we observed that misregistrations for different values of K do not correlate and appear in different places. At the same time, using $K = 1$ for the whole sequence is not a good solution as it is slower and often results in an even larger number of errors than with $K = 2$ or 3.

Finally, the transformation matrices of the complete repaired registration chain are multiplied, and each initial point cloud is registered to the reference cloud which is the last cloud of the sequence. In principle, it would be possible to 'dissipate' the registration errors in the chain [11] and smooth the overall alignment. However, given the large size of the data, this would require excessive computation, so we decided to omit this step. Currently, typical execution time of the algorithm is 20–30 minutes for a sequence of 300 point clouds.

5. Test Results

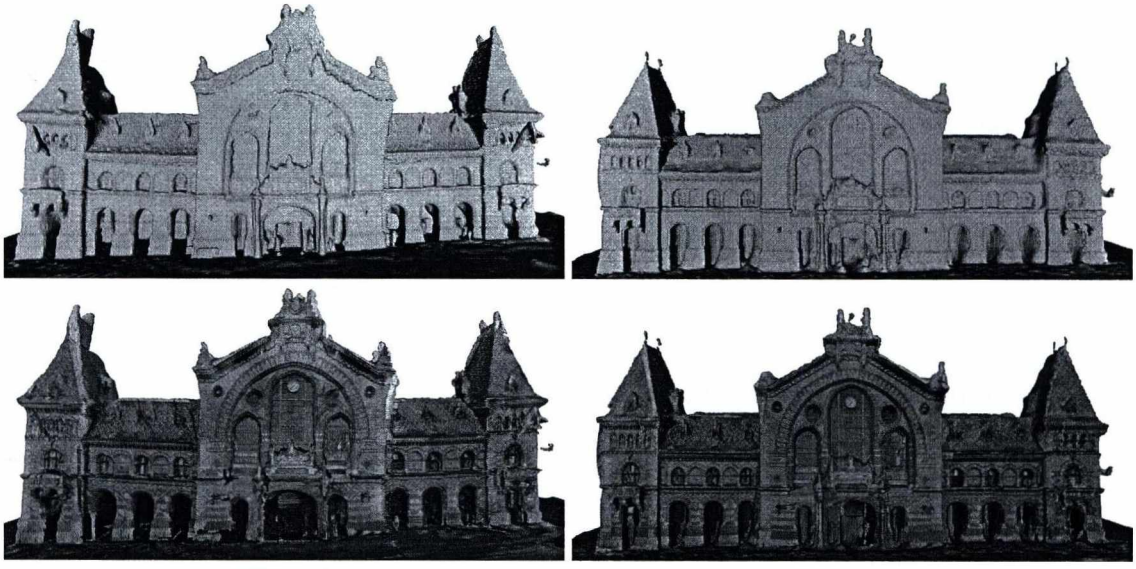
In this section, we demonstrate and compare sample results of reconstruction from car-mounted LIDAR data. The aligned point sets are converted to triangular meshes using the Poisson Surface Reconstruction algorithm [3, 6] which works on oriented point sets. As discussed in section 4.1, the normals are assigned to points by our data filtering algorithm. Alternatively, we could use the normal calculation algorithm provided by the Meshlab package [3]. However, we have experienced that this algorithm is slower and less robust; in particular, it can produce flipped normals.

The test results form two distinct groups. Section 5.1 is devoted to the Market Hall data acquired along a facade of the Central Market Hall of Budapest (in Hungarian, Központi Vásárcsarnok). This building features a number of characteristic architectural elements, which can facilitate the alignment but makes the potential reconstruction errors more visible. The Kende Street data discussed in section 5.2 represents most of this short street and includes 7–8 buildings on each side. Most of the buildings have simple, featureless facades that are significantly higher than those of the Market Hall.

5.1. Market Hall

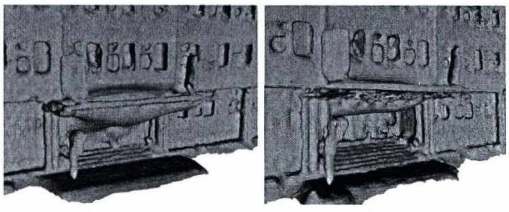
Figure 5 shows the front facade of the Market Hall reconstructed by the two methods. Both textureless and textured versions are presented. The TrICP-based alignment algorithm processed each third of the original 300 point clouds ($K = 3$) with the overlap value of 85%. Five bad registrations were detected and successfully repaired using $K = 1$ and overlap 95%. The overall quality of the reconstruction is good as the global geometry and the fine details are correct.

For the Market Hall data, the NDT-based alignment method yields worse results. In addition to the global bendings and visible surface roughness, some structural elements and details appear distorted and blurred, as illustrated in figure 6 where a vicinity of a market entrance is shown enlarged. The TrICP alignment better preserves the geometry and the details.



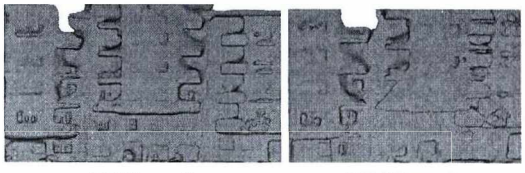
NDT result TrICP result

Figure 5: *Reconstruction of Market Hall front facade.*



NDT result TrICP result

Figure 8: *Reconstruction of the entrance of MTA SZTAKI.*



NDT result TrICP result

Figure 9: *Part of Kende Street reconstruction.*

5.2. Kende Street

For the Kende Street data, the outcome of the test is just the opposite. Here, the Trimmed ICP often fails to cope with featureless and sparse pieces of data, while NDT yields satisfactory results demonstrated in figure 7. The overall quality is nevertheless much lower than for the Market Hall data.

When the 3D data contains distinct features and is sufficiently dense (e.g., at low altitudes), TrICP can still produce reasonable output. Figure 8 compares the two results at the vicinity of the entrance of the main building of MTA SZTAKI situated in the street. There is no big difference in the quality.

However, in several other areas of the street the performance of TrICP is poor. Figure 9 gives an example when the reconstruction by TrICP exhibits severe global and local distortions, while the result by NDT is visibly better, although not perfect.

6. Discussion and Conclusion

We have presented two methods for aligning long sequences of point clouds acquired by a car-mounted LIDAR device measuring facades in a street. Our current experience with the methods can be summarised as follows.

Due to nature of the LIDAR scans and the fact that the NDT algorithm works in discrete space, this registration method is sensitive to the discretisation parameter. For challenging scenes such as facades with repeating patterns (i.e., rows of similar windows), this parameter may need to be increased to 50–70 centimeters which can result in distorted registrations. Fine details can be blurred. Otherwise, the method is suitable for reconstructing single facades as well as sequences of facades. It is less sensitive to featureless and sparse areas than the TrICP-based method.

The execution times of the two algorithms are comparable. However, the speed of NDT decreases drastically as the spatial resolution increases. The speed of TrICP is less sensitive to the setting of its parameters.

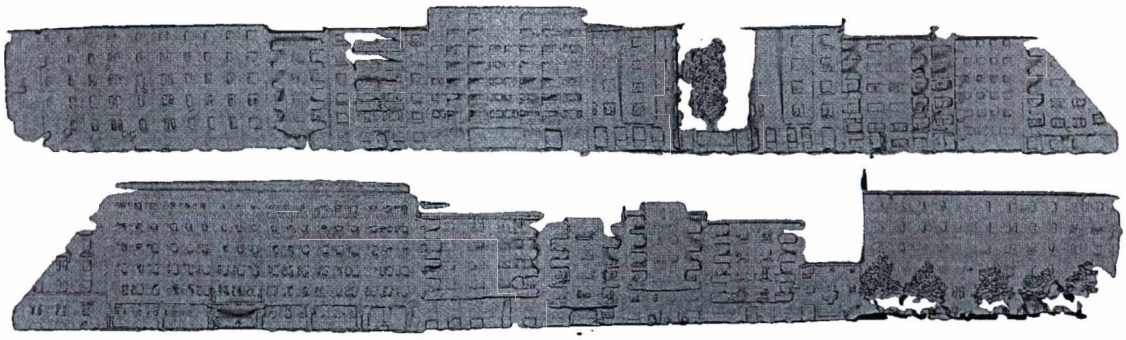


Figure 7: *NDT reconstruction of Kende Street facades. Top: East side. Bottom: West side.*

TrICP is applicable to, and produces superior results for, surfaces containing characteristic features. It can cope with larger rotations between scans then NDT. However, when featureless or sparse areas dominate, TrICP works at the limit of its capabilities and produces multiple misalignments that cannot be corrected by the proposed procedure.

In future, we plan to investigate the role of LIDAR sampling, in general, and its influence on the systematic registration errors, in particular. We have observed that both methods are sensitive to the order in which the point clouds are processed. This may result from the asymmetry of their cost functions w.r.t. the two clouds, as well as from the anisotropy of data sampling by LIDAR.

The possibility and efficiency of using auxiliary sensors in our street scenario will also be studied. If the answer is positive, such sensors will be applied. However, the precision of today's sensors does not seem to be sufficient for our purposes.

The TrICP-based method may profit from the prior knowledge of limited rotation and shift, which currently is not utilised. The NDT-based method can be enhanced by the filtering procedure introduced in section 4.1.

Both methods will be applied to the reconstruction of complete models of buildings and quarters. Such reconstruction will need fast and efficient algorithms for alignment error dissipation within a circular data sequence, to avoid error accumulation and global misalignment when the registration loop terminates.

Acknowledgment

This work was supported by an internal grant of the Institute for Computer Science and Control. The authors acknowledge the contribution of Oszkár Józsa who implemented and tested the NDT-based registration algorithm.

References

1. C. Benedek, Z. Jankó, C. Horváth, D. Molnár, D. Chetverikov, and T. Szirányi. An integrated 4D vision and visualisation system. In *Computer Vision Systems*, volume 7963 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2013.
2. D. Chetverikov, D. Stepanov, and P. Krsek. Robust Euclidean alignment of 3D point sets: the Trimmed Iterative Closest Point algorithm. *Image and Vision Computing*, 23:299–309, 2005.
3. P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Proc. Eurographics Italian Chapter Conference*, pages 129–136, 2008.
4. O. Józsa, A. Börcs, and C. Benedek. Towards 4D virtual city reconstruction from LiDAR point cloud sequences. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W1:15–20, 2013.
5. S. Katz, A. Tal, and R. Basri. Direct visibility of point sets. In *ACM SIGGRAPH*. ACM, 2007.
6. M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. Fourth Eurographics Symposium on Geometry Processing*, pages 61–70. Eurographics Association, 2006.
7. M. Magnusson. *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* PhD thesis, Örebro University, December 2009.
8. M. Magnusson, A. Nuchter, C. Lorken, A.J. Lilienthal, and J. Hertzberg. Evaluation of 3D registration reliability and speed – a comparison of ICP and NDT. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3907–3912, 2009.
9. P. Musialski, P. Wonka, D.G. Aliaga, M. Wimmer, L. Gool, and W. Purgathofer. A survey of urban reconstruction. In *Computer Graphics Forum*, volume 32, pages 146–177, 2013.

10. F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, pages 1–16, 2013.

11. K. Pulli. Multiview registration for large data sets. In *Proc. Second International Conference on 3-D Digital Imaging and Modeling*, pages 160–168, 1999.

12. P.J. Rousseeuw and A.M. Leroy. *Robust regression and outlier detection*. Wiley, 2005.

13. S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

14. T. Stoyanov, M. Magnusson, H. Almqvist, and A.J. Lilienthal. On the accuracy of the 3D normal distributions transform as a tool for spatial representation. In *Proc. IEEE International Conference onRobotics and Automation*, pages 4080–4085, 2011.

15. Q.-Y. Zhou and U. Neumann. Complete residential urban area reconstruction from dense aerial LiDAR point clouds. *Graphical Models*, 75:118–125, 2013.

Non-rigid Face Reconstruction and Head Pose Estimation

Ákos Pernek[1,2] and Levente Hajder[2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary
[2] Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Budapest, Hungary

Abstract

Robust human face recognition is one of the most important open tasks in computer vision. This study deals with a challenging subproblem of face recognition: the aim of the paper is to give a precise estimation for the 3D head pose. The main contribution of this study is a novel non-rigid Structure from Motion (SfM) algorithm which utilizes the fact that the human face is quasi-symmetric. The input of the proposed algorithm is a set of tracked feature points of the face. In order to increase the precision of the head pose estimation, we improved one of the best eye corner detectors and fused the results with the input set of feature points. The proposed methods were evaluated on real and synthetic face sequences. The real sequences were captured using regular (low-cost) web-cams.

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Image Processing and Computer Vision]: Scene Analysis

1. Introduction

The shape and appearance modeling of the human face and the fitting of these models have raised significant attention in the computer vision community. Till the last few years, the state-of-the-art method used for facial feature alignment and tracking was the active appearance model (AAM) [1,2]. The AAM builds a statistical shape [3] and grey-level appearance model from a face database and synthesizes the complete face. Its shape and appearance parameters are refined based on the intensity differences of the synthesized face and the real image.

Recently, a new model class has been developed called the constrained local model (CLM) [4,5,6]. The CLM model is in several ways similar to the AAM, however, it learns the appearance variations of rectangular regions surrounding the points of the facial feature set.

Due to its promising performance, we utilize the CLM for facial feature tracking. Our C++ CLM implementation is mainly based on the paper [6], however, it utilizes a 3D shape model.

The CLM (so as the AAM) requires a training data set to learn the shape and appearance variations. We use a basel face model (BFM) [7]-based face database for training data set. The BFM is a generative 3D shape and texture model which also provides the ground-truth head pose and the ground-truth 2D and 3D facial feature coordinates. Our training database consists of 10k synthetic faces of random shape and appearance. The 3D shape model or the so-called point distribution model (PDM) of the CLM were calculated from the 3D facial features according to [3].

During our experiments we have identified that the BFM-based 3D CLM produces low performance at large head poses (above 30 degree). The CLM fitting in the eye regions showed instability. We propose here two novelties: (i) Since the precision of eye corner points are of high importance for many vision applications, we decided to replace the eye corner estimates of the CLM with that of our eye corner detector. (ii) We propose a novel non-rigid structure from motion (SfM) algorithm which utilize the fact that human face is quasi-symmetric (almost symmetric).

2. EYE CORNER DETECTION

One contribution of our paper is a 3D eye corner detector inspired by [8]. The main idea of our method is that the 3D information increases the precision of eye corner detection. (In our case, it is available due to 3D CLM fitting.) We created a 3D eye model which we align with the 3D head pose and utilize to calculate 2D eye corner location estimates. These estimates are further developed to generate the expected values for a set of features [8] supporting the eye corner selection.

2.1. Related Work

The eye corner detection has a long history. Several methods have been developed in the past years. A promising method is described in [8]. This method applies pre-processing steps on the eye region to reduce noise and increase robustness: a horizontal rank filter is utilized for eyelash removal and eye reflections are detected and reduced as described in [10]. The method acquires the pupil, the eyebrow and the skin regions by intensity-based clustering and the final boundaries are calculated via region growing [9]. It also performs sclera segmentation based on the histogram of the saturation channel of the eye image [8]. The segmentation provides an estimate on the eye region and thus, the lower and upper eyelid contours can be estimated as well. One can fit an ellipse or as well as polynomial curves on these contours which provide useful information for the real eye corner locations. The method generates a set of eye corner candidates via the well-known Harris corner detector [12] and defines a set of decision features. These features are utilized to select the real eye corners from the set of candidates. The method is efficient and provides good results even on low resolution images.

2.2. Iris Localization

To localize the iris region, we propose to use the intensity based eye region clustering method of [9]. However, we also propose a number of updates to it. Tan et al. orders the points of the eye region by intensity and assigns the lightest $p_1\%$ and the darkest $p_2\%$ of these points to the initial candidate skin and iris regions, respectively. The initial candidate regions are further refined by means of region growing. The method is repeated iteratively until all points of the eye region are clustered. The result is a set of eye regions: iris, eyebrow, skin, and possibly degenerate regions due to reflections, hair and glass parts. In order to make the clustering method robust, they apply the image pre-processing steps described in Sec. 2.1 as well.

Our choice for the parameter p_1 is 30% as suggested by [9]. However, we adjust the parameter p_2 adaptively. We calculate the average intensity (i_{avg}) of the eye region (in the intensity-wise normalized image) and set the p_2 value to $i_d * i_{avg}$ where i_d is an empirically chosen scale factor of value $\frac{1}{12}$. The adaptive adjustment of p_2 showed higher stability during test executions on various faces than the fixed set-up.

Another improvement is that we use the method of [11] for iris detection. The method is robust and operates stable on eye images of various sources. We assign the central region of the fitted iris to the iris region to improve the clustering result.

The result of the iris detection and the iris center and the eye region clustering is shown in Figure 1. Note that we focus on the clustering of the iris region and thus, only the iris and the residual regions are displayed.

Figure 1: *Iris and its center (of scale 0.4), initial/final iris, initial/final residual region (left to right)*

2.3. Sclera Segmentation

The human sclera can be segmented by applying data quantization and histogram equalization on the saturation channel of the noise filtered eye region image [8]. We adopt this method with some minor adaptations: we set the threshold for sclera segmentation as a function of the average intensity of the eye region (see Sec. 2.2). In our case, the scale factor of the average intensity is chosen as $\frac{1}{8}$.

We also limit the accepted dark regions to the ones which are neighboring to the iris. We have defined rectangular search regions at the left and the right side of the iris. Only the candidate sclera regions overlapping with these regions are accepted. The size and the location of the search regions are bound to the ellipse fitted on the iris edge [11]. The sclera segmentation is displayed in Figure 2.

Figure 2: *Homogenous sclera, candidate sclera regions and rectangular search windows, selected left and right side sclera segments (left to right)*

2.4. Eyelid Contour Approximation

The next step of the eye corner detection is to approximate the eyelids. The curves of the upper and lower human eyelids intersect in the eye corners. Thus, the more precisely the eyelids are approximated, the more information we can have on the true locations of the eye corners.

The basis of the eyelid approximation is to create an eye mask. We create an initial estimate of this mask consisting of the iris and the sclera regions as described in Sections 2.2 and 2.3. This estimate is further refined by filling: the unclustered points which lay horizontally or vertically between two clustered points are attached to the mask. The filled mask is extended: we apply vertical edge detection on the eye image and try to expand the mask vertically till the first edge of the edge image. The extension is done within empirical limits derived from the eye shape, the current shape of the mask and the iris location [11].

The final eye mask is subject to contour detection. The eye mask region is scanned vertically and the up- and down most points of the detected contour points are classified as the points of the upper and lower eyelids, respectively.

Figure 3: *Eye mask, filled eye mask, vertical edge based extension, final eye mask, upper and lower eyelid contours (left to right)*

2.5. Eye Corner Selection

We use the method of Harris and Stephens [12] to generate candidate eye corners as in [8]. The Harris detector is applied only in the nasal and temporal eye corner regions (see Sec. 2.7). The detector is configured with low acceptance threshold ($\frac{1}{10}$ of the maximum feature response) so that it can generate a large set of corners. These corners are ordered in descending order by their Harris corner response and the first 25 corners are accepted. We constrain the acceptance with considerations of the Euclidean distance between selected eye corner candidates. A corner is not accepted as a candidate if one corner is already selected within its $1px$ neighborhood.

The nasal and the temporal eye corners are selected from these eye corner candidate sets. The decision is based on a set of decision features. These features are a subset of the ones described in [8]: Harris pixel weight, internal angle, internal slope, relative distance, and, intersection of interpolated polynomials.

These decision features are utilized to discriminate false eye corner candidates. We convert them into probabilities indicating the goodness of an eye corner candidate. The goodness is defined as the deviation of the feature from its expected value. Finally, an aggregate score for each candidate is calculated with equally weighted probabilities except for the internal slope feature which we overweight in order to try selecting eye corners located under the major axis of the ellipse. One important deviation of our method from that of [8] is that we don't consider eye corner candidate pairs during the selection procedure. We found that the nasal eye corner is usually lower than the temporal one thus the line passing through them is not parallel to the major axis of the fitted ellipse.

2.6. 3D Enhanced Eye Corner Detection

One major contribution of our paper is that our eye corner detector is 3D enhanced. A subset of the decision features (internal angle, internal slope and relative distance) in Sec. 2.5 requires the expected feature values in order to discriminate the false eye corner candidates. We define a 3D eye model and align it with the 3D head pose. We utilize the aligned model to calculate precise expected 2D eye corner locations and thus, expected features values as well.

Our 3D eye model consists of an ellipse modeling the one fitted on the eyelid contours and a set of parameters: p_1, p_2, p_3, p_4, and, b_a. Parameters p_1, p_2, p_3, and, p_4 denote the scalar projection of the eye corner positions w.r.t. ellipse center and the major and minor axes. Parameter b_a defines the bending angle: the expected temporal eye corner is rotated around the minor axis of the ellipse. Let us denote head yaw and pitch angles as: lr_a and ud_a, respectively (note that we do not model head roll). Assuming that the ellipse center is the origin of our coordinate system, the expected locations of the temporal and the nasal eye corners (of the right eye) can be written as: $c_t = (p_1 cos(lr_a - b_a)A, p_3 cos(ud_a)B)$ and $c_n = (p_2 cos(lr_a)A, p_4 cos(ud_a)B)$, respectively.

The ratio of the major A and minor B axes is a flexible parameter r_a and is unknown. However, it can be learnt from the first few images of a face video sequence (assuming frontal head pose).

In our framework the parameters p_1, p_2, p_3, p_4, and, b_a are chosen as -0.9, 0.9, -0.15, -0.5, and, $\frac{\pi}{12}$, respectively.

The eye model is visualized in Figure 4.

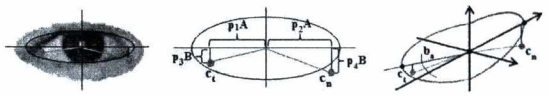


Figure 4: *Eye corners and fitted ellipse, 2D eye model (b_a = 0), 3D eye model (left to right)*

2.7. Enhanced Eye Corner Regions

Our method applies an elliptic mask in order to filter invalid eye corner candidates. We rotate this elliptic mask in accordance with the 3D head pose and we also shift the he rectangular eye corner regions vertically in accordance with the slope of the major axis of the ellipse (fitted on the eyelid contours). This allows us a better model for the possible location of the candidate eye corners (see Figure 5).

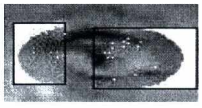


Figure 5: *Rectangular eye corner regions masked by the 3D elliptic mask. The white dots denote the available eye corner candidates.*

3. NON-RIGID STRUCTURE FROM MOTION

The other major contribution of our paper is a novel non-rigid and symmetric reconstruction algorithm which solves the structure from motion problem (SfM). Our proposed algorithm incorporates non-rigidity and symmetry of the object to reconstruct. The proposed method is applicable for

both symmetric or quasi-symmetric (almost symmetric) objects.

This section summarizes the main aspects of the non-rigid reconstruction. The input of the reconstruction is P tracked feature points of a non-rigid object across F frames. (In our case, they are calculated by 3D CLM tracking and the proposed 3D eye corner detection method.)

Usually, the SfM-like problems are solved by matrix factorization. For rigid objects, the well-known solutions are based on the classical Tomasi-Kanade factorization [13]. Our approach, similarly to the work of Tomasi and Kanade [13], assumes weak-perspective projection. We proposed an alternation-based method [14, 15] in 2008 that divides the factorization method into subproblems that can be solved optimally. We extend our solution to the nonrigid case here.

3.1. Non-rigid Object Model

A rigid object in the SfM methods is usually modeled by its 3D vertices. We model the non-rigidity of the face by K so-called key (rigid) objects. The non-rigid shape of each frame is estimated as a linear combination of these key objects.

The non-rigid shape of an object at the j^{th} frame can be written as:

$$S^j = \sum_{i=1}^{K} w_i^j S_i \tag{1}$$

where w_i^j are the non-rigid weight components for the j^{th} frame and the k^{th} key object ($k = [1 \,..\, K]$) is written as:

$$S_k = \begin{bmatrix} X_{1,k} & X_{2,k} & \cdots & X_{P,k} \\ Y_{1,k} & Y_{2,k} & \cdots & Y_{P,k} \\ Z_{1,k} & Z_{2,k} & \cdots & Z_{P,k} \end{bmatrix} \tag{2}$$

3.2. Weak-Perspective Projection Model

To estimate the key objects and their non-rigid weight components, the tracked $2D$ feature points has to be linked to the $3D$ shapes. This link is the projection model. Due to its simplicity, the weak-perspective projection is a good choice to express the relationship between the $3D$ shape and the tracked $2D$ feature points. It is applicable when the depth of the object is significantly smaller than the distance between the camera and the object center. Thus, the weak-perspective projection is applicable for web-cam video sequences, which is in the center of our interest.

The weak-perspective projection equation is written as follows:

$$\begin{bmatrix} u_i^j \\ v_i^j \end{bmatrix} = q^j R^j \begin{bmatrix} X_i^j \\ Y_i^j \\ Z_i^j \end{bmatrix} + t^j \tag{3}$$

where q^j is the scale parameter, R^j is the 2 x 3 rotation matrix, $t^j = [u_0^j, v_0^j]^T$ is the 2 x 1 translation vector, $[u^j, v^j]^T$ are the projected $2D$ coordinates of the i^{th} $3D$ point $[X_i^j, Y_i^j, Z_i^j]$ of the j^{th} frame.

During non-rigid structure reconstruction, the q^j scale parameters can be accumulated in the non-rigid weight components. For this reason we introduce the notation $c_i^j = q^j w_i^j$. Utilizing this assumption, the weak-perspective projection for a non-rigid object in the j^{th} frame can be written as:

$$\begin{aligned} W^j = \begin{bmatrix} u_1^j & \cdots & u_P^j \\ v_1^j & \cdots & v_P^j \end{bmatrix} &= R^j S^j + t^j \\ &= R^j \left(\sum_{i=1}^{K} c_i^j S_i \right) + t^j \end{aligned} \tag{4}$$

where W^j is the so-called measurement matrix.

The projection equation can be reformulated as

$$W = MS = [R|t][S, 1]^T \tag{5}$$

where W is the measurement matrix of all frames: $W = \begin{bmatrix} W^{1T} \dots W^{FT} \end{bmatrix}^T$. R is the non-rigid motion matrix and t the translation vector of all frames:

$$M = \begin{bmatrix} c_1^1 R^1 & \cdots & c_K^1 R^1 \\ \vdots & \ddots & \vdots \\ c_1^F R^F & \cdots & c_K^F R^F \end{bmatrix} \quad t = \begin{bmatrix} t_1 \\ \vdots \\ t_F \end{bmatrix} \tag{6}$$

and M is the non-rigid motion matrix of all frames.

and S is defined as a concatenation of the K key objects: $S = \begin{bmatrix} S_1^T & \dots & S_K^T & 1 \end{bmatrix}^T$

3.3. Optimization

Our proposed non-rigid reconstruction method minimizes the so-called re-projection error:

$$\|W - MS\|_F^2 \tag{7}$$

The key idea of the proposed method is that the parameters of the problem can be separated into independent groups, and the parameters in these groups can be estimated optimally in the least squares sense.

The parameters of the proposed algorithm are categorized into three groups: (i) camera parameters: rotation matrices (R^j) and translation parameters (t^j), (ii) key object weights (c_i^j), and (iii) key object parameters (S_k). These parameter groups can be calculated optimally in the least square sense. The method refines them in an alternating manner. Each step reduces the reprojection error and is proven to converge in accordance with [15]. The steps of the alternation are described here, the whole algorithm is overviewed in Alg. 1.

3.3.0.1. Rt-step The Rt-step is very similar to the one proposed by Pernek et al. [15]. The camera parameters of the frames can be estimated one by one: they are independent

of each other. If the j^{th} frame is considered, the optimal estimation can be given computing the optimal registration between the 3D vectors in matrices W and $\sum_{i=1}^{K} c_i^j S_i$. The optimal registration is described in [16]. A very important remark is that the scale parameter cannot be computed in this step contrary to the rigid factorization proposed in [15].

3.3.0.2. S-step The cost function in Eq 7 depends linearly on the values of the structure matrix S. The optimal solution for S is $S = M^{\dagger}W$ where $\dagger$ denotes the Moore-Penrose pseudoinverse. However, this is true only for non-symmetric points. We assume that many of the face feature points has a pair. If $s_{i,k}$ and $s_{j,k}$ are feature point pairs then $s_{i,k}^x = -s_{j,k}^x$, $s_{i,k}^y = s_{j,k}^y$, and $s_{i,k}^z = s_{j,k}^z$ if the plane of the symmetry is $x = 0$. ($s_{i,k}^x$, $s_{i,k}^y$, $s_{i,k}^z$ denotes the coordinates of the i^{th} point in key object k.) The corresponding parts of the cost function: $\left|\left|W_i - [m_1, m_2, m_3][s_{i,x}, s_{i,y}, s_{i,z}]\right|\right|$ and $\left|\left|W_i - [-m_1, m_2, m_3][s_{i,k}^x, s_{i,k}^y, s_{i,k}^z]\right|\right|$, where m_1, m_2, and m_3 are the columns of motion matrix M, and W_i and W_j the corresponding row pairs of measurement matrix W. The optimal estimation can be computed as

$$s_i = \begin{bmatrix} m_1 & m_2 & m_3 \\ -m_1 & m_2 & m_3 \end{bmatrix}^{\dagger} \begin{bmatrix} W_i \\ W_j \end{bmatrix} \tag{8}$$

$s_{i,x} = 0$ for non-symmetric points, thus, the linear estimation is simpler with respect to common rigid factorization since only two coordinates have to be calculated. Remark that S-step must be repeated for all key object.

3.3.0.3. c-step The goal of the c-step is to compute parameters c_i^j optimally in the least squares sense if all the other parameters are known. Fortunately, this is a linear problem, the optimal solution can be easily obtained by solving an overdetermined one-parameter inhomogeneous linear system. [17]. Remark that the weight parameters for frame j must be calculated independently from those of other frames.

Algorithm 1 Non-rigid And Symmetric Reconstruction

```
k ← 0
R,t,c,S ← Initialize()
R ← Complete(R)
S ← MakeSymmetric(S)
S ← CentralizeAndAlign(S)
repeat
  k ← k + 1
  S ← S-step(W,R,t,c)
  c ← c-step(W,R,t,S)
  (R,t) ← Rt-step(W,c,S)
  W ← Complete(W,R,t,c,S)
until Error(W,R,w,S,t) < ε or k ≥ k_max
```

3.3.0.4. Completion Due to the optimal estimation of the rotation matrix, an additional step must be included before every step of the algorithm as it is also carried out in [15]. The Rt-step yields 3×3 orthogonal matrices, but the matrices R^j used in non-rigid factorization are of size 2×3. Thus, the 2×3 matrix has to be completed with a third row: it is perpendicular to the first two rows, its length is the average of those. The completion should be done for the measurement matrix as well. Let r_3^j, w_3^j, and, t_3^j denote the third row of the completed rotation, measurement, and, translation at the j^{th} frame, respectively. The completion is written as:

$$w_3^j \leftarrow r_3^j \left(\sum_{i=1}^{K} c_i^j S_i \right) + t_3^j \tag{9}$$

3.4. Initialization of Parameters

The proposed improvement is an iterative algorithm. If good initial parameters are set, the algorithm converges to the closest (local or global) minimum, because each step is optimal w.r.t. reprojection error defined in Eq. 5. One of the most important problem is to find a good starting point for the algorithm: camera parameters (rotation and translation), weight components, and, key objects.

We define the structure matrices of the K key objects w.r.t. the rigid structure as $S_1 \approx S_2 \cdots \approx S_K \approx S_{rig}$, where S_{rig} denotes the rigid structure. In our case S_{rig} is the mean shape of the 3D CLM's shape model. The approximation sign '$\approx$' means that a little random noise is added to the elements of S_i with respect to S_{rig}. This is necessary, otherwise the structure matrices remain equal during the optimization procedure. We set w_i^j weights to be equal to the weak-perspective scale of the rigid reconstruction. The initial rotation matrices R^j are estimated via calculating the optimal rotation [16] between W and S_{rig}.

The CLM based initialization is convenient for us, however, the initialization can be performed in many ways such as the ones written in [15] or [18].

We also enforce the symmetry of the initial key objects. We calculate the symmetry plane of them and relocate their points so that the single points lay on, the pair points are symmetrical to the symmetry plane. The plane of the symmetry is calculated as follows. The normal vector of the plane should be parallel to the vector between the point pairs, and the plane should contain the midpoint of point pairs. Therefore, the normal vector of the symmetry plane is estimated as the average of the vectors between the point pairs, and the position of the plane is calculated from the midpoints. Then the locations of the feature point of key objects are recalculated in order to fulfill the symmetricity constraint. (And the single points are projected to the symmetry plane.)

4. TEST EVALUATION

The current section shows the test evaluation of the 3D eye corner detection and the non-rigid and symmetric reconstruction.

For evaluation purposes we use a set of real and synthetic video sequences which contain motion sequences of the human face captured at a regular face - web camera distance. The subjects of the sequences perform a left-, a right-, an up-, and, a downward head movement of at most 30-40 degrees.

The synthetic sequences are based on the BFM [7]-based face database.

4.1. Empirical Evaluation

This section visualizes the results of the 3D eye corner detection on both real and synthetic (see Figure 6) video sequences. The section contains only empirical evaluation of the results. The sub-figures display the frontal face (first column) in big, and the right (middle column) and left (right column) eyes in small at different head poses.

The frontal face images show many details of our method: the black rectangles define the face and the eye regions of interest (ROI). The face ROIs are detected by the well-known Viola-Jones detector [19], however, they are truncated horizontally and vertically to cut insignificant regions such as upper forehead. The eye ROIs are calculated relatively to the truncated face ROIs. The blue rectangles show the detected [19] eye regions and the eye corner ROIs as well. The eye region detection is executed within the boundaries of the previously calculated eye ROIs. The eye corner ROIs are calculated within the detected eye regions with respect to the location and size of the iris. The red circles show the result of the iris detection [11] which is performed within the detected eye region. Blue polynomials around the eyes show the result of the polynomial fitting on the eyelid contours. The green markers show the points of the 3D CLM model. The yellow markers at eye corners display the result of the 3D eye corner detection.

The right and the left eye images of the sub-figures display the eyes at maximal left, right, up, and, down head poses in top-down order, respectively. The black markers show the selected eye corners. The grey markers show the available set of candidate eye corners.

The test executions show that the 3D eye corner detection works very well on our test sequences. The eye corner detection produces good results even for blurred images at extreme head poses.

4.2. 2D/3D Eye Corner Detection

This sections evaluates the precision of the eye corners calculated by the 3D CLM model, our 3D eye corner detector and its 2D variant. In the latter case we simply fixed the (rotation) parameters of our 3D eye corner detector to zero in order to mimic continuous frontal head pose.

To measure the eye corner detection accuracy, we used 100 BFM-based video sequences . Thus, the ground-truth 2D eye corner coordinates were available during our tests.

The eye corner detection accuracy we calculated as the average least square error between the ground-truth and the calculated eye corners of each image of a sequence. The final results displayed in Table 1 show the average accuracy for all the sequences in pixels and the improvement percentage w.r.t the 3D CLM error.

Table 1: *Comparison of the 3D CLM, and the 2D/3D eye corner (EC) detector*

Type	3DCLM	2DEC	3DEC
Accuracy	0.5214	0.4201	0.4163
Improve	0.0	19.42	20.15

The results show that the 3D eye corner detection method performs the best on the test sequence. It is also shown that both the 2D and the 3D eye corner detectors outperform the CLM method. This is due to the fact that our 3D CLM model is sensitive to extreme head pose and it tends to fail in the eye region. An illustration of the problem is displayed in Figure 7.

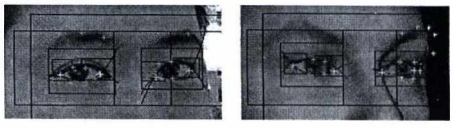


Figure 7: *CLM fitting failure (green markers around eye and eyebrow regions) at extreme head poses*

4.3. Non-rigid Reconstruction

In this section we evaluate the accuracy of the non-rigid and symmetric reconstruction. For our measurements, we use the same synthetic database as in Section 4.2. The basis of the comparison is a special feature set. This feature set consists of the points tracked by our 3D CLM model. However, due to the eye region inaccuracy described in Section 4.2, we drop the eye points (two eye corners and four more points around the iris and eyelid contour intersections) and use the eye corners computed by our 3D eye corner detector.

The non-rigid reconstruction yields the refined cameras and the refined 2D and 3D feature coordinates of each image of a sequence. The head pose can be extracted from the cameras. We selected the head pose and the 2D and 3D error as an indicator of the reconstruction quality. The ground-truth head pose, 2D and 3D feature coordinates are acquired from the BFM.

We calculated the head pose error as the average least square error between the ground-truth head pose and the calculated head pose of each image of a sequence. The 2D and 3D error we define as the average registration error [16] of the

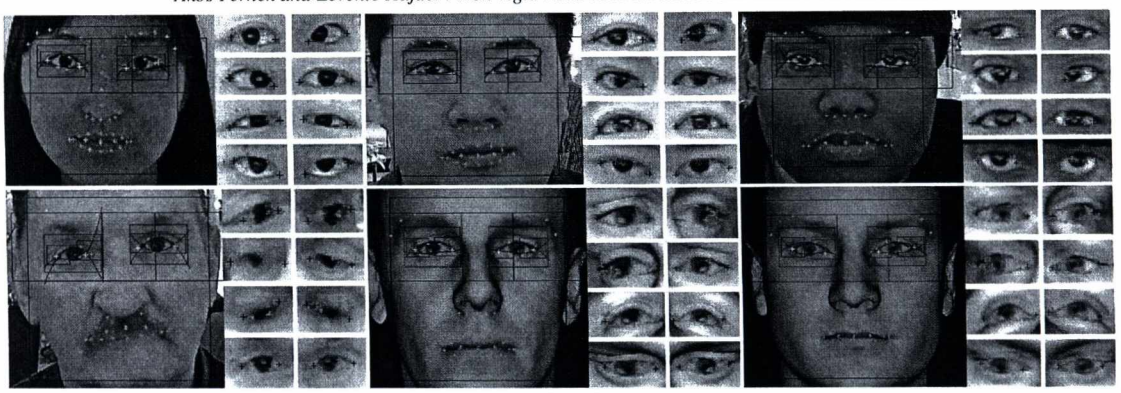


Figure 6: *Real and synthetic test sequences*

Table 2: *Comparison of the 3D CLM, the symmetric and non-rigid and the generic non-rigid reconstruction*

Type	3DCLM	Gen (K=1)	Gen (K=5)	Gen (K=10)	Sym (K=1)	Sym (K=5)	Sym (K=10)
2D Err.	2.73162	2.72951	2.77952	2.78255	2.72853	2.72853	2.72853
2D Impr.	0.0	0.0772	-1.7535	-1.8644	0.1131	0.1131	0.1131
3D Err.	1.03933	0.89338	4.56524	2.50865	0.880928	0.880915	0.880910
3D Impr.	0.0	14.0427	-339.24	-141.37	15.2407	15.2420	15.2425
Pose Err.	0.3443	0.2756	0.5317	0.5974	0.2829	0.2807	0.2908
Pose Impr.	0.0	19.9535	-54.429	-73.5115	17.8332	18.4722	15.5387

centralized and normalized ground truth and the computed 2D and 3D point sets of each image of the sequence.

The compared methods are the 3D CLM, our non-rigid and symmetric reconstruction and its generic non-rigid variant (symmetry constraint not enforced).

The results displayed in Table 2 show the average accuracy for all the test sequences in degrees and the improvement percentage w.r.t the 3D CLM model. The generic (Gen) and the symmetric (Sym) reconstruction methods have been evaluated with different number of non-rigid components (K) as well.

It is seen that by optimizing a huge amount of parameters, lower reprojection error values can be reached, however, without the symmetry constraint this can yield an invalid solution. Our proposed symmetric method keeps stable even with a high number of non-rigid components (K).

One can also see that the head pose error of our proposed method outperforms the 3D CLM, however, the generic rigid reconstruction (Gen (K=1)) provides the best results. We believe that the rigid model can better fit to the CLM features due to the lack of the symmetry constraint.

On the other hand the best 3D registration errors are provided by our proposed method. It shows again that the symmetry constraint does not allow the reconstruction to converge toward a solution with less reprojection error, but with a deviated 3D structure.

The table also shows that the 2D registration is best by our proposed method, however, the gain is very little and the performance of the methods are basically similar.

5. CONCLUSIONS

It has been shown in this study that the precision of the human face pose estimation can be significantly enhanced if the symmetric (anatomical) property of the face is considered. The novelty of this paper is twofold: we have proposed here an improved eye corner detector as well as a novel non-rigid SfM algorithm for quasi-symmetric objects. The methods are validated on both real and rendered image sequences. The synthetic test were generated by the basel face model, therefore, ground truth data have been available for evaluating both our eye corner detector and non-rigid and symmetric SfM algorithms. The test results have convinced us that the proposed methods outperforms the compared ones and a precise head pose estimation is possible for real web-cam sequences even if the head is rotated by large angles.

References

1. T. F. Cootes and G. J. Edwards and C. J. Taylor Active Appearance Models *PAMI*, 484–498, 1998.
2. I. Matthews and S. Baker Active Appearance Models Revisited *IJCV*, **60**(2):135–164, 2004.
3. T. F. Cootes and C. J. Taylor and D. H. Cooper and J. Graham Training Models of Shape from Sets of Examples *BMVC*, 9–18, 1992.
4. D. Cristinacce and T. F. Cootes Feature Detection and Tracking with Constrained Local Models. *BMVC*, 929–938, 2006.
5. Y. Wang and S. Lucey and J. Cohn Enforcing Convexity for Improved Alignment with Constrained Local Models *CVPR*, 2008.
6. J. M. Saragih and S. Lucey and J. Cohn Face Alignment through Subspace Constrained Mean-Shifts *ICCV*, 2009.
7. P. Paysan and R. Knothe and B. Amberg and S. Romdhani and T. Vetter A 3D Face Model for Pose and Illumination Invariant Face Recognition *AVSS*, 2009.
8. G. M. M. Santos and H. Proença A Robust Eye-corner Detection Method for Real-world Data. *IJCB*, 1–7, 2011.
9. T. Tan and Z. He and Z. Sun Efficient and Robust Segmentation of Noisy Iris Images for Non-cooperative Iris Recognition *IVC*, **28**(2):223–230, 2010.
10. Z. He and T. Tan and Z. Sun and X. Qiu Towards Accurate and Fast Iris Segmentation for Iris Biometrics *PAMI*, **31**(9):1670–1684, 2009.
11. Zs. Jankó and L. Hajder Improving Human-Computer Interaction by Gaze Tracking *Cognitive Infocommunications*, 155–160, 2012.
12. C. Harris and M. Stephens A Combined Corner and Edge Detector *Fourth Alvey Vision Conference*, 147–151, 1988.
13. C. Tomasi and T. Kanade Shape and Motion from Image Streams under Orthography: A Factorization Approach *IJCV*, 137–154, 1992.
14. L. Hajder and Á. Pernek and Cs. Kazó Weak-perspective Structure from Motion by Fast Alternation *The Visual Computer*, **27**(5):387–399, 2011.
15. Á. Pernek and L. Hajder and Cs. Kazó Metric Reconstruction with Missing Data under Weak Perspective. *BMVC*, 2008.
16. K. S. Arun and T. S. Huang and S. D. Blostein Least-squares Fitting of Two 3-D Point Sets *PAMI*, **9**(5):698–700, 1987.
17. R. I. Hartley and A. Zisserman Multiple View Geometry in Computer Vision 2003.
18. J. Xiao and J.-X. Chai and T. Kanade A Closed-Form Solution to Non-rigid Shape and Motion Recovery *ECCV*, 573–587, 2004.
19. P. Viola and M. Jones Rapid Object Detection using a Boosted Cascade of Simple Features *CVPR*, 511–518, 2001.

3D mesh generation from aerial LiDAR point cloud data

Péter Polcz and Csaba Benedek

Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI)
polcz.peter@sztaki.mta.hu, benedek.csaba@sztaki.mta.hu
http://web.eee.sztaki.hu/remotesensing

Abstract

Three dimensional urban scene modelling became important issue in the last few years. Beside visual experience, 3D city modelling has gained a significant function in diverse analysing tasks, however the amount of data requires a high level of automation of model generation. In this work, we introduce an automatic and robust algorithm which produces detailed 3D virtual city models by analysing high resolution airborne LiDAR point clouds. Using the idea of the surface normal based roof segmentation we have designed a procedure, which takes into account the boundaries of each roof segment, so that the adjacent segments connect without gaps. We have developed an algorithm to detect 3D edge lines of the rooftops. Since the applied triangulation methods operate on the whole convex hull of the input points, hollow outer parts of the roof segments are filled in with false triangles. To solve this problem, we have proposed a method using a Markov Random Field, in which we filter out the incorrect triangles lying on the concave parts.

Categories and Subject Descriptors (according to ACM CCS): I.4.5 [Computer vision]: Reconstruction

1. Introduction

In the last decade, LiDAR (Light Detection and Ranging) has been widely used in various remote sensing application fields. LiDAR is an optical remote sensing technology that can measure the distance of targets from the scanner by illuminating the target with laser light and analysing the backscattered light, therefore a such a laser scanner yields a 3D point cloud representing the objects around the scanner.

A specific type of these sensors can be mounted on airplanes, and the provided scans are appropriate for creating digital terrain models (DTM) and digital elevation models (DEM). These models are efficient and detailed descriptions of fields, valleys, mountains or other desert areas. These irregular, rough and mountainous terrain types cannot be represented as a set of regular shapes.

On the other hand, in case of cities or other urban settlements polygon reconstruction constitutes another alternative solution for modelling. The main targets of the reconstruction are the buildings having regular geometrical shapes introducing the possibility to approximate them with several three-dimensional polygons. Worldwide projects (Google maps 3D, Nokia maps) are devoted to this topic.

As input, we have used high resolution LiDAR records of Budapest city center, which have been provided by Infoterra Astrium GEO-Information Services Hungary.

In this paper we intend to present our approaches of aerial point cloud processing, three-dimensional city reconstruction and urban scene modelling.

2. Previous Work

This research domain has considerably progressed during the last decade. Many of computer vision researchers have developed new techniques and algorithms in order to create not only realistic but also simple[1] city-models at the same time. Regarding the latest publications, significant results have been encountered by Lafarge et al. [1, 2], Zhou et al. [3–5], Huang et al. [6, 7] and Verma et al. [8]. Zhou's approach consists in geometrical and topological corrections of an initial mesh on the basis of local observations of the buildings' orientation. Whereas Lafarge and Huang defined geometric 3D primitives to fit them to the different building types and rooftop shapes appearing in the point cloud.

[1] in the means of reduced number of facets

Lafarge et al. [1, 2] also handled non-planar primitives as cylinders, spheres and cones.

Huang et al. [6, 7] created complete roof models (composed by planar primitives) and attempted to fit them to the cloud regions classified as buildings using different statistical methods (in particular likelihood function maximization). After a geometrical adjustment, the primitives were "merged" into a plausible model.

Verma et al. [8] also used a statistical approach, by building a dual graph from the roof segments. However, this technique only worked for planar roof models.

3. A brief description of our proposed reconstruction algorithm

The workflow includes four steps, from which the first three steps are illustrated in Figure 1. First, the point cloud is classified using an unsupervised method based on the work of Börcs and Horváth [9] and the method introduced by Lafarge et al. [1, 2] and Zhou [5], in which the algorithm distinguishes four different classes: ground, building, vegetation and clutter. Then the point cloud regions classified as buildings are divided into several parts in order to reduce the complexity of the further steps. Each part of the cloud will contain a reduced number of points belonging to a single complex rooftop.

Secondly, the proposed algorithm approximates each rooftop by planar shaped faces. These planar roof segments are extracted by a robust method detailed in Section 4. The points of a roof component determine a plane, which is calculated through minimizing the sum of squared distances of the points from the plane.

The third step is described in Section 5, which consists in generating a 3D skeleton model for each building block by detecting the roof's edge lines. After triangulating the endpoints of the edge lines we retain several, approximately planar shaped polygon meshes which will form together a 3D building model.

The last step constitutes the main part of our contribution, in which we introduce a new method for concave triangle mesh generation which solves the problem of concave shaped roof segments.

4. Roof segmentation

In this section separate planar roof segments on the basis of their orientation. First of all, we estimate a local surface normal at every point of the roof cloud using the Point Cloud Library's [10] implementation of Moving Least Squares (MLS) algorithm (Figure 2 - left). Since we know every point's normal, we apply a clustering algorithm to detect the representative directions in which the planar roof components face (black vectors in Figure 2 - right). These

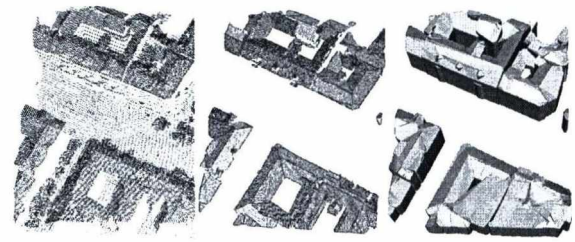


Figure 1: The first three steps of the proposed method: initial classification (left), roof segmentation and edge detection (middle), triangle mesh generated from endpoints of the edge lines.

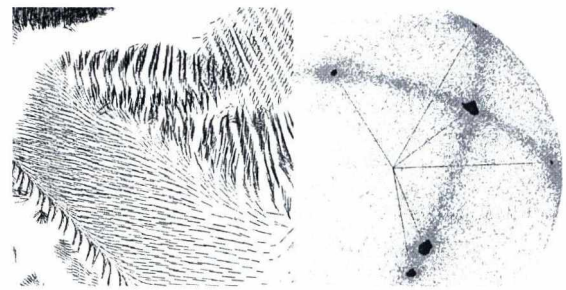


Figure 2: Illustration of our surface normal based clustering. First the normals are calculated using an MLS algorithm (left), after that the normals are translated into the origin. The second image illustrates the endpoints of the normal vectors, from which dense regions are extracted and than clustered.

few directions will represent separate clusters with different labels. In the following, every point will be assigned an appropriate label (i.e. color), depending the point's normal. As a result, the points of every roof segment having similar orientation will be given the same label. Afterwards, a region-growing is applied on the cloud knowing the labels that the roof points belong to. Since the segmentation produces a slightly noisy label mask, we adopt a further smoothing step, which uses the K-nearest neighbors smoothing algorithm. At the end we retain the final labeling, in which every planar continuous roof component are distinguished by a unique segment label, and then we will be ready to perform the polygon approximation for every roof segment.

5. 3D edge detection and triangulation

3D edge lines are detected in two different ways. Let us call *inner edges* those, which lie alongside the connection of two neighboring faces of the roof (ridges). These lines are identified as the intersections of the respective neighboring planar roof components.

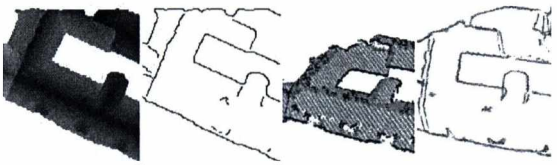


Figure 3: Outer edge detection's assembly - projection (z-image) - edge detection - elevation into the 3D space - segment fitting.

Figure 4: Concave problem - points of a concave roof segment (left) - triangulation on the whole convex hull (middle) - triangle mesh generated by our method (right)

On the other hand, we call *outer edges* the lines, which constitute the outer boundaries of the rooftops (eaves). Since in the airborne LiDAR point clouds, we usually have no reflection from the vertical walls, outer edges are calculated through image processing techniques, as illustrated in Figure 3.

First the roof cloud is projected onto the xy horizontal plane so that each pixel will get the respective 3D point's z elevation value, as its grayscale color value. Henceforth, we call this projected image as z-image. After adopting an edge detection algorithm on the z-image, we retain an edge image in which high elevation differences are highlighted. Using the z-image and the edge image we elevate the edge points into 3D, and we fit 3D lines to the 3D edge points. The generated 3D lines constitute the *outer* edge lines of the rooftops.

The rooftops' planar faces are generated by triangulating the endpoints of the edge lines. Vertical outer walls are produced using the outer triangle sides of the previously generated triangle meshes.

6. Concave triangle mesh generation

Concave shaped roof segments appear frequently, however several well established triangulation methods, such as the used Delanuay triangulation[2] provided by CGAL [12], generate triangle meshes on the whole convex hull of the given points[3]. Consequently, as shown in Figure 4 (middle), important architectural features of the building may be filled in with false roof components. Therefore we designed a procedure in which triangles lying on the concave parts of the Delaunay convex mesh will be erased preserving smooth boundaries in the final concave mesh. The procedure is based on a probabilistic graphical model.

According to Gansner, Hu and Kobourov [13] a triangle mesh is defined by the included triangles (S) and the neighborhood connections ($\mathcal{N}$) between them, hence it can be modeled as an undirected graph (Figure 5) where each triangle is considered as a separate vertex ($s \in S$), and each neighborhood connection as an undirected link ($\{s,r\} \in \mathcal{N}$) between two vertices ($s,r \in S$) corresponding to the neighboring triangles. Furthermore, some of the triangles in the mesh need to be deleted because they are lying on the concave parts of the roof segment. As a consequence, we have to assign a random variable (Ω_s) to each vertex that marks the fact whether the corresponding triangle needs to be eliminated or not. After classification, we call a given triangle as *relevant triangle*, if it should be kept in the final mesh. These Ω_s variables are defined on the set of vertices (S), therefore they form a $\Omega = (\Omega_s)_{s \in S}$ random field with respect to $\mathcal{N}$.

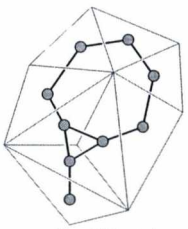


Figure 5: 3D triangle mesh and its corresponding undirected triangulation graph

6.1. Markov property

In our case *planar graph* models can also be used, since the considered meshes are open[4] (i.e. they have at least three triangles having neighbors fewer then three), and do not contain any wholes. *Planar graphs*[5] can be drawn on the plane, so that their edges intersect only at their endpoints. It is convenient to use them, since they satisfy the below constraints[6]:

- we have an upper limit for the number of edges: $e \leq 3v - 6$, where e is the number of edges and v is the number of vertices.
- the maximum number of vertices of a fully connected (complete[7]) sub-graph is 4.

With reference to Gansner, Hu and Kobourov [13] (Lemma 1. on pg. 5.) we cannot draw on the plane four triangles

[2] described in detail by Gallier et al. [11] in Section 8.3 *Delaunay Triangulations*

[3] "there is an intimate relationship between convex hulls and Delaunay triangulations", pronounced by Gallier et at. [11] in Section 8.4 *Delaunay Triangulations and Convex Hulls*

[4] they are open 2-manifolds (Smith et al. [14] on pg. 14.)

[5] see definition given by Balakrishnan et al. [15] (Definition 8.2.1 on pg. 175.)

[6] see theorems and their consequences formulated by Balakrishnan et al. [15] in Section 8.3 *Euler Formula and Its Consequences*

[7] see definition given by Balakrishnan et al. [15] (Definition 1.2.11)

which are all connected to each other, therefore the maximum number of vertices of a complete sub-graph is three. Accordingly, the vertices of the graph are usually connected with a reduced number of other vertices especially in their close proximity (Markov property). Furthermore, we presume that every triangle's label is conditionally independent of any other non-adjacent node's label, given the labels of all neighboring triangles. With regard to Stan Z. Li et al. [16], Ω is said to be a Markov Random Field (MRF) on S wrt. $\mathcal{N}$.

6.2. Prior and data model

MRFs are able to simultaneously embed a data model, reflecting the knowledge on the observation; and prior constraints, such as spatial smoothness of the solution. As for the ***prior model***, we used the following energy function:

$$E_s = \sum_{\{s,r\}\in\mathcal{N}} V(\omega_s, \omega_r)$$

where V implements a smoothing constraints, using the Kronecker delta: $V(\omega_s, \omega_r) = \delta(\omega_s = \omega_r)$. In case of a particular $s \in S$ triangle, our ***data model*** uses the following descriptors:

n_s : number of points projected into s

A_s : area of s

φ_s, ϑ_s : two arbitrary angles of s

Using these measures we will generate a single x_s fitness value for each triangle s, so that x_s will be approximately proportional with the likelihood of the fact that s constitutes a relevant element in the concave triangulation, hence it should not be deleted from the final mesh. As for the first feature, we calculate the density of the projected points in each triangle ($\frac{n_s}{A_s}$), and we divide the calculated value by a

$$\mathcal{K} = \max_{s_i \in S} \frac{n_{s_i}}{A_{s_i}}$$

normalization coefficient, so that we obtain a density feature in the interval $[0,1]$. Let us introduce the following notation for this descriptor (ρ stands for density):

$$\rho_s = \frac{1}{\mathcal{K}} \cdot \frac{n_s}{A_s} \in [0,1] \qquad (1)$$

Next, we use our observation that bays (i.e. internal regions of the mesh which should be likely eliminated) contain mainly acute-angled triangles, while micro concavities on the boundaries of the open mesh consist of long and thin obtuse triangles. As a consequence, we introduced a so-called *angle cost* that measures how much a given triangle is obtuse. Let us define *angle cost* as the product of each angle's cosine values in the triangle.

$$\alpha_s = \left(\cos(\varphi_s)\cos(\vartheta_s)\cos(\pi - \varphi_s - \vartheta_s) + 1\right) \cdot \frac{1}{1.125}$$

The angle cost gives its maximum value, if the triangle is equilateral ($\varphi = \vartheta = \pi/3$). Otherwise, the more a triangle is acute, the more its angle cost tents to 0.

Finally, a joint fitness value, i.e. a pseudo probability is defined as the product of ρ and $(1-\alpha)$, which indicates us, whether a given triangle is a relevant element of the mesh. These $x_s = \rho_s \cdot (1-\alpha_s)$ values will form our ***observed feature layer*** (Figure 6).

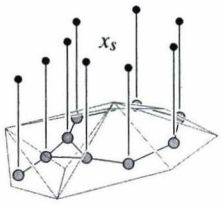


Figure 6: dependency graph, in which the filled dots stand for the x_s observed feature layer

However, during our experiments we perceived that excluding triangles just by their low x_s values using a given hard threshold can cause several false positive/negative triangles. In other words, the designed feature (x_s) is not enough in itself to decide whether a triangle belongs to the concave parts of the mesh or not. To overcome this limitation, we started to compare each triangle's class label with labels in its neighborhood, hence we took the advantage of the *prior model*. Just for illustration (Figure 7), let us color relevant triangles white and triangles able to be skipped gray. This color can also be interpreted as a label marking that the corresponding triangle is *relevant* or *removable*. If two or three neighboring triangles are gray the actual triangle is likely to be gray too, especially when the nearby triangles have larger area then the actual one.

For the $d_s(\omega_s) = d(x_s \mid \omega_s)$ *data cost* we have chosen the following functions:

$$d(x_s \mid \Omega_s = \text{relevant}) = f(1 - x_s)$$
$$d(x_s \mid \Omega_s = \text{removable}) = f(x_s) \qquad \forall\, s \in S$$

where $f(x)$ is the sigmoid function, operating as a soft threshold:

$$f(x) = \frac{1}{1 + e^{-n(x - x_0)}} \cdot (b - a) + a, \qquad f : [0,1] \longrightarrow [a,b]$$

with gradient $n = 20$, shift $x_0 = 0.5$, offset and scale $(a,b) = (0.2, 2)$. Thereafter the *data model* of the MRF has the following form:

$$p(x_s \mid \omega_s) = e^{-d(x_s \mid \omega_s)}$$

Note that the above quantities define pseudo probabilities, since they are unnormalized.

Using our defined *prior* and *data model*, we can determine the ***posterior*** likelihood $P(\omega|x)$ of every possible global labeling over the triangle-graph, and we have no other tasks but choosing the most probable global labeling that will point out the desired (*relevant*) triangles. To conclude, we have optimized the following energy function using graph cuts based optimization technique developed by

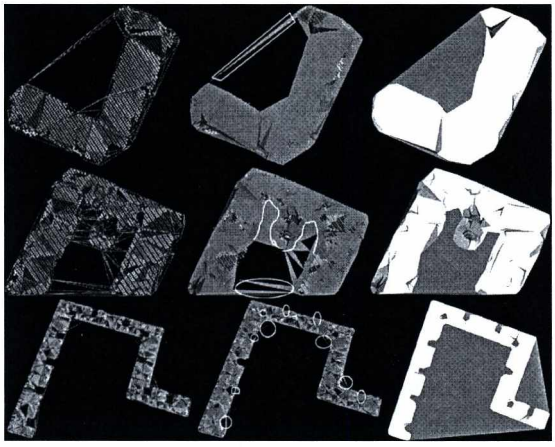


Figure 7: Removing triangles that do not belong to the concave hull. The first two columns demonstrate a hard threshold of the x_s feature, without the MRF smoothing constraint: we can observe several false triangles in the resulting meshes (2nd column). The 3rd column shows the optimal label mask where maximum probability is met. In the first column it can also be observed through coloring how the points are associated to the corresponding triangles.

Olga Veksler, using the libraries provided by Yuri Boykov and Vladimir Kolmogorov [17–20]:

$$\omega_{opt} = \arg\min_{\omega \in \Gamma} \left(\sum_{s \in S} d(x_s \mid \omega_s) + \lambda \sum_{\{s,r\} \in \mathcal{N}_s} \delta(\omega_s, \omega_r) \right)$$

The results are illustrated in Figure 7, which shows smooth features at the roof segments' boundaries in the same way false triangles are eliminated.

7. Results

As Figure 8 illustrate, the algorithm can be applied for a wide range of building types even though it solely estimates the geometry of objects by several planar elements (polygons). We have reconstructed city sites featuring urban civil apartment houses (Figures 1 and 8 - city site), buildings with complex architectural roof models (Figure 8 - Market Hall and BUTE K-building), large concave blocks of flat (Figure 4). The algorithm was tested on different point clouds containing together about three million points, covering an area of 102,000 m^2. The aggregate number of the generated triangles is about 200,000 without triangles of the outer walls. See Table 1 for the details.

8. Future plans and conclusion

Our work's primary objective was to design an automatic and robust method to process aerial LiDAR data and produce

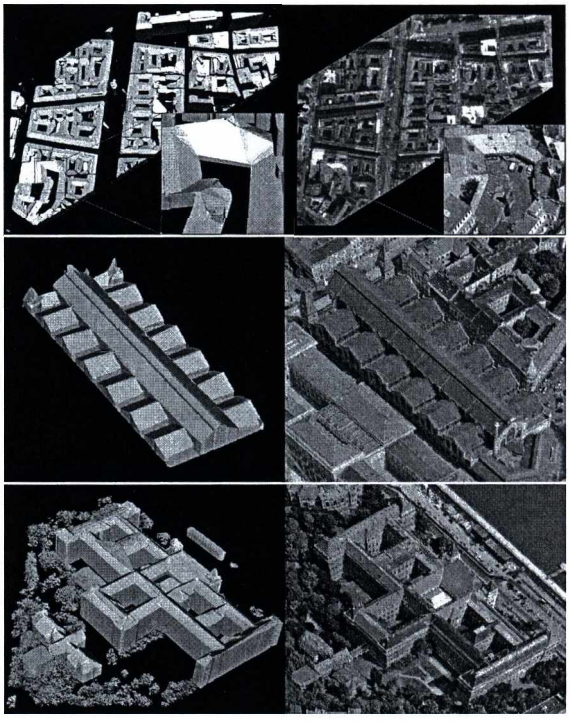


Figure 8: Our polygon reconstruction results (left), and the reference aerial photos (right) of various landmarks of Budapest. Civil apartment houses - Budapest's site between Mária St., Nap St., Futó St. and Baross St. (top), Vásárcsarnok Market, Vámház körút (middle), Budapest University of Technology and Economics (BUTE) - central building (bottom)

three-dimensional geometric models from them. The obtained three-dimensional models will be compared with optical images taken from the space in different times, analysing the possibilities of adaptive texturing and change detection.

9. Acknowledgement

We thank Infoterra Astrium GEO-Information Services ©, who gave us the permission to test our system on their aerial LiDAR records of Budapest. This work was founded by the Government of Hungary through a European Space Agency (ESA) Contract under the Plan for European Cooperating States (PECS), and by the Hungarian Research Fund (OTKA #101598).

References

1. Florent Lafarge and Clément Mallet. Creating large-scale city models from 3d-point clouds: A robust approach with hybrid representation. *International Journal of Computer Vision*, 99(1):69–85, August 2012.

city site	nr. of points	nr. of roof points	nr. of triangles[1]	$\frac{\text{nr. roof points}}{\text{nr triangles}}$	area (m^2)	proc. time	in Figure
Móricz sqr.	147,120	69,643	3,950	17.63	7,956	$\sim$ 2 min	1
Baross str.	1,927,656	948,461	160,684	5.90	35,954	$\sim$ 20 min	8 (top)
Market Hall	151,319	151,319	6,307	23.99	9,000	$\sim$ 1 min	8 (middle)
BUTE	875,106	362,024	38,544	9.39	49,163	$\sim$ 10 min	8 (bottom)

Table 1: Quantitive results of our mesh generator algorithm.
[1] without triangles of outer walls

2. Florent Lafarge and Clément Mallet. Building large urban environments from unstructured point data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1068–1075, Barcelona, Spain, 2011.
3. Qian-Yi Zhou and Ulrich Neumann. Complete residential urban area reconstruction from dense aerial LIDAR point clouds. *Graphical Models*, 75(3):118–125, 2013.
4. Qian-Yi Zhou and Ulrich Neumann. Modeling residential urban areas from dense aerial LIDAR point clouds. In *Proceedings of the First international conference on Computational Visual Media*, CVM'12, pages 91–98, Berlin, Heidelberg, 2012. Springer-Verlag.
5. Qian-Yi Zhou. *3D urban modeling from city-scale aerial LIDAR data*. Phd thesis, Faculty of the Graduade School University of Southern California, 2012.
6. Hai Huang, Claus Brenner, and Monika Sester. 3d building roof reconstruction from point clouds via generative models. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 16–24, New York, NY, USA, 2011. ACM.
7. Hai Huang, Claus Brenner, and Monika Sester. A generative statistical approach to automatic 3d building roof reconstruction from laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 79(0):29–43, May 2013.
8. Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial LIDAR data. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2 of *CVPR '06*, pages 2213–2220, Washington, DC, USA, 2006. IEEE Computer Society.
9. Attila Börcs and Csaba Horváth. Városi környezet automatikus analízise és rekonstrukciója légi LiDAR mérések alapján, TDK dolgozat, Pázmány Péter Katolikus Egyetem Információs Technológiai Kar. 2011.
10. Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011.
11. Jean Gallier. Notes on Convex Sets, Polytopes, Polyhedra Combinatorial Topology, Voronoi Diagrams and Delaunay Triangulations. Rapport de recherche RR-6379, INRIA, 2007.
12. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.
13. Emden R. Gansner, Yifan Hu, and Stephen G. Kobourov. On touching triangle graphs. *CoRR*, abs/1001.2862, 2010.
14. Colin Smith. *On vertex-vertex systems and their use in geometric and biological modelling*. PhD thesis, Calgary, Alta., Canada, Canada, 2006. AAINR19574.
15. R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. Universitext - Springer-Verlag. Springer, 2012.
16. Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer Publishing Company, Incorporated, 3rd edition, 2009.
17. Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Aseem Agarwala, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. In *In ECCV*, pages 16–29, 2006.
18. Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.
19. Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:65–81, 2004.
20. Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, sep 2004.

Perfecting 3D computer models reconstructed from measured data

István Kovács, and Tamás Várady

Budapest University of Technology and Economics

Abstract

The goal of digital shape reconstruction is to create computer representations from measured data. The majority of existing 3D reconstruction methods can only closely approximate the 'ideal' bounding surfaces. Inaccuracies occur due to noisy data and the numerical nature of the algorithms used for fitting. In this paper we present algorithms to eliminate these inaccuracies and create a "perfect" model for CAD/CAM systems. We extend a formerly published technology[1] in two areas. We propose methods to automatically set up hypotheses for various geometric constraints and select the most likely ones instead of requesting this information from the user. We also compute global constraints related to the whole object. This includes determining (i) an optimal axis orientation and a grid cell size for the coordinate system in which the object might have been defined; and (ii) the best - full or partial - axes of symmetries. We investigate planar contours with constraints, however, to extend this technology to 3D is hoped to be straightforward. Related theoretical and numerical problems will be illustrated by several test examples.

1. Introduction

Digital shape reconstruction (reverse engineering) is an expanding, challenging area of Computer Aided Geometric Design[10]. This technology is utilized in various applications where a given physical object is scanned in 3D, and a computer representation is needed in order to perform various computations. A wide range of applications emerges in engineering, medical sciences, and to preserve the cultural heritage of mankind[5]. Examples include redesigning and remanufacturing old mechanical parts, creating surface geometries from clay models, or producing surfaces matching human body parts for hearing aids, dentures, prosthetics, etc.

1.1. Digital shape reconstruction

Digital shape reconstruction consists of the following technical phases: (1) 3D data acquisition (scanning), (2) filtering and merging point clouds, (3) creating triangular meshes, (4) simplifying and repairing meshes, (5) segmentation (partitioning into disjoint regions), (6) region classification, (7) fitting surfaces, (8) fitting connecting surfaces (e.g. fillets), *(9) perfecting surfaces* (including constrained fitting and surface fairing), (10) exporting to CAD–CAM systems for downstream applications.

Assume segmentation has taken place, and classification produced a surface type for each region that will best approximate the related data points. The conventional approach is to fit surfaces individually. Let us denote the surfaces by $\{s_i\}$, and the corresponding point clouds by $\{p_{ij}\}$. Our goal is to minimize the average square distances between the surfaces and the point clouds. Let $\mathbf{x}$ contain the parameters of the surfaces. Then the problem is

$$f_i(\mathbf{x}) = \sum_j d(p_{ij}, s_i)^2, \qquad f_i(\mathbf{x}) \to \min.$$

Fitting simple surfaces is generally based on solving eigenvalue problems[2,9]; for more complex surfaces efficient numerical methods exist[8]. Fitting surfaces *separately* is likely to produce inaccuracies; fortunately, the model quality can be perfected, if we recognize and enforce various geometric constraints and then fit groups of related surfaces *simultaneously*.

1.2. Constrained fitting

Geometric constraints define relationships amongst various entities. This is a key issue in engineering design; orthogonality, parallelism, tangency, symmetry, etc. can be best pre-

scribed by means of constraints, which are expressed in the form of various algebraic equations.

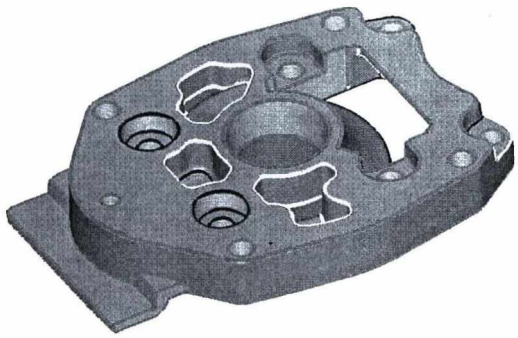


Figure 1: *An engineering object with many self-contained constraints.*

We may distinguish between constraints that has *local* effect related to pairs of curves and surfaces, such as, lines, circles, planes, cylinders, cones, extruded and rotational surfaces, and more complex constraints that *globally* determine groups of surfaces. The most frequent local constraints include

- orthogonal/parallel curves and surfaces,
- concentric curves and surfaces,
- tangential curves and surfaces,
- rounded numerical values,
- fixed numerical values.

The most frequent global constraints include

- common direction for extrusions,
- common rotational axes,
- global grid,
- global axis of symmetry,
- global rotational symmetry.

The scanned data – in itself – do not carry information about the structure and the constraints of high-level geometric entities; and these need to be set either explicitly by the user, or recognized and set by some "intelligent" algorithm. After individual fitting — due to noise and numerical inaccuracies — constraints will be satisfied only within some tolerances; an example with inaccurate values is shown in Figure 2. If we can set a constraint system, we can enhance the model and *refit* the surfaces accordingly. This process is called *constrained fitting*.

Our goal is to minimize the average square distances between the point clouds $\{p_{ij}\}$ and the surfaces $\{s_i\}$

$$f(\mathbf{x}) = \sum_i \alpha_i \sum_j d(p_{ij}, s_i)^2,$$

while the constraint system is satisfied. The α_i weights are typically set proportional to the area of the surfaces.

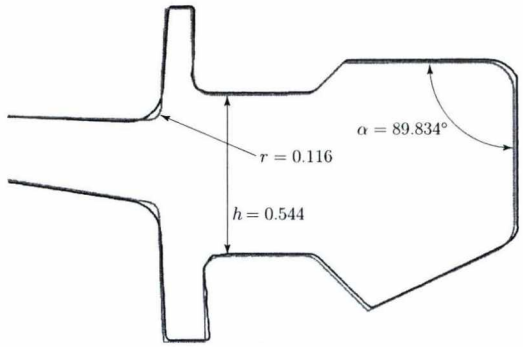


Figure 2: *Inaccurately reconstructed contour without (black) and with (red) constraints.*

1.3. Previous work

Numerical methods to solve this problem have been published earlier[12,7]. An important paper on constrained fitting was published by Langbein et al.[3], where partial symmetries on point sets are detected based on an algebraic concept. In the paper of Mitra et al.[6] it is shown how partial global symmetries on 3D models by feature points can be detected. Our paper expands a numerical technique originally suggested by Benkő et al.[1], that can handle under- and over-constrained systems with priorities, applying a special extension of Newton's method.

1.4. Outline

In this paper we focus on algorithms, that substitute user driven, manual constrained fitting by automatic techniques. After presenting the basic algorithm of Benkő et al.[1] in Section 2, we present how to detect and enforce various hypotheses for likely *local* geometric constraints in Section 3. Then we continue with methods to detect *global* constraints, including best fit grids and optimal axes of symmetries - see Section 4 and 5, respectively. Results will be illustrated by several examples using 2D point sets and related constraints for planar curves.

2. Constrained fitting – basics

2.1. A simple example

Consider the profile curve in Figure 3.[11] If we fitted these circles independently, the tangential constraints would not be satisfied, however, constrained fitting provides an appropriate solution. In this example, let c_i denote the circles with parameters (A_i, B_i, C_i, D_i) , and the corresponding equations are $A_i(x^2+y^2)+B_ix+C_iy+D_i=0$. The average squared distance to be minimized is

$$f(\mathbf{x}) = \sum_{i,j} d_{i,j}^2 = \frac{1}{n_i} \sum_j (A_i(x_{ij}^2+y_{ij}^2)+B_ix_{ij}+C_iy_{ij}+D_i)^2.$$

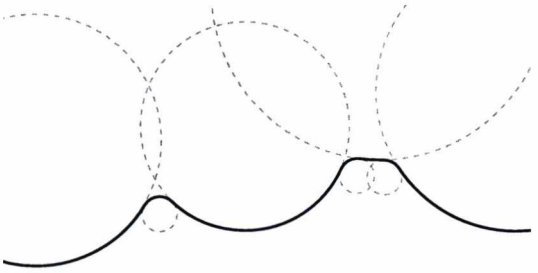


Figure 3: *Profile curve*

The constraint system includes

- normalization constraints ($B_i^2 + C_i^2 - 4A_iD_i = 1$); these assure that the distances from the points closely approximate the Euclidean distances, and
- tangential constraints ($2A_jD_i + 2A_iD_j - B_iB_j - C_iC_j \pm 1 = 0$); these assure that each pair of adjacent circular arcs shares a common endpoint and tangent.

2.2. The numerical method

Let us continue with presenting the method of Benkő et al.[1], this is necessary to understand the rest of the paper. We use the previous notations, where $d(p_{ij}, s_i)$ denotes the distance between surface s_i and point p_{ij}. Let α_i-s be the positive weights assigned to the i-th surface. Let $\mathbf{x}$ contain the parameters of all surfaces, and let us define the constraint equations in the form of $c_k = 0$. Then we can write the global system of equations as

$$\mathbf{c}(\mathbf{x}) = 0.$$

We minimize the average square distance while the constraints are satisfied:

$$f(\mathbf{x}) = \sum_i \alpha_i \sum_j d(p_{ij}, s_i)^2$$

Let $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), ..., c_k(\mathbf{x}))$, where the constraints are ordered by priority and suppose that $f(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$ are smooth enough (at least C^2). Here we have a highly non-linear system of equations, that we are going to solve using a special Newton iteration. We approximate $\mathbf{c}$ in first order, and f in second order. The Taylor approximations of $\mathbf{c}$ and f around $\mathbf{x}_0$ are the following:

$$\mathbf{c}(\mathbf{x}_0 + \mathbf{d}) \approx \mathbf{c}(\mathbf{x}_0) + \mathbf{c}'(\mathbf{x}_0)\mathbf{d}$$

$$f(\mathbf{x}_0 + \mathbf{d}) \approx f(\mathbf{x}_0) + f'(\mathbf{x}_0)\mathbf{d} + \frac{1}{2}\mathbf{d}^T f''(\mathbf{x}_0)\mathbf{d}$$

In each step we want to determine a small difference vector $\mathbf{d}$. Using the above equations, the problem can be written locally in the form

$$C\tilde{\mathbf{d}} = 0 \qquad (1)$$

$$\tilde{\mathbf{d}}^T A \tilde{\mathbf{d}} \to \min,$$

where $\tilde{\mathbf{d}} = (d_1, ..., d_n, 1)$, $C = [\mathbf{c}'(\mathbf{x}_0)|\mathbf{c}(\mathbf{x}_0)]$ and A is an $(n+1) \times (n+1)$ size matrix, as follows:

$$A = \left[\begin{array}{c|c} f''(\mathbf{x}_0) & f'(\mathbf{x}_0) \\ \hline f'(\mathbf{x}_0)^T & 0 \end{array}\right].$$

In order to calculate $\tilde{\mathbf{d}}$ we have to reduce it to a lower dimensional vector $\mathbf{d}^*$ by (1), such that $\mathbf{d}^*$ has only *independent* coordinates. We calculate a matrix M, such that $\mathbf{d} = M\mathbf{d}^*$, and $CM = 0$. Now the dimension of $\mathbf{d}^*$ gives us, how many independent variables exist in the system. Finally, we can solve ${\mathbf{d}^*}^T A^* \mathbf{d}^* \to \min$ without constraints, where $A^* = M^T A M$, and this can be solved as a simple system of linear equations.

The way of calculating M is very similar to Gauss elimination. During elimination, we can check if some of the constraints contradict to each other, or if the system is overdetermined (see details in the original paper[1]).

2.3. Auxiliary objects

The use of the so-called *auxiliary objects* is an important idea in constrained fitting. We illustrate this through a simple example. Take three lines that are supposed to meet in a common point (see Figure 5(a)). We can formulate the related constraints by taking line 1 and 2, compute their intersection and constrain this intersection point to lie on line 3; then we take line 2 and 3, and line 3 and 1 with similar constraints. This set of equations defines a relatively simple problem in a very complicated way.

An alternative solution is to introduce an auxiliary point p. This is also an unknown entity, but now we can define our constraints by three simple equations: i.e. all three lines must pass through p. Clearly, we have increased the number of unknowns in the parameter vector $\mathbf{x}$, but the system of equations – and all related Taylor approximants – have become much simpler. Note, the unknown surface parameters are generally associated with a corresponding point sets, but for auxiliary objects such a data point has no meaning. Typical auxiliary entities include a point, a point and a normal, a distance, etc.; their exclusive role is to simplify the system of equations and thus our computations.

3. Automatic detection of local constraints

Let us start with a simple example. We wonder whether pairs of lines are perpendicular or not, and we wish to incorporate additional constraints into our system, if the likelihood of being perpendicular is high. This can clearly be controlled by a user defined angular tolerance, and extra constraints will be added automatically, if two lines span an angle between $90 \pm \varepsilon$.

Formally: let $c(\mathbf{x}) = 0$ a simple constraint between two

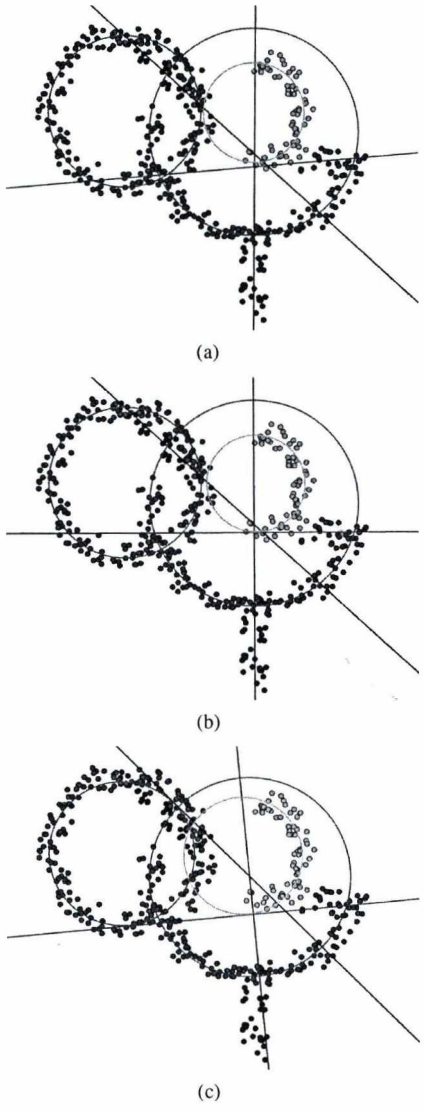


Figure 4: *Automatically detected constraints: (a) — initial state; (b) and (c) — different configurations created by different tolerance levels*

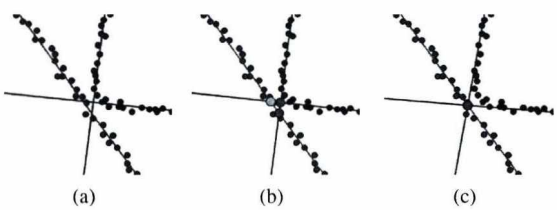


Figure 5: *Three lines meet at a common point: (a) initial state; (b) pairwise intersection (auxiliary points); and (c) enforce the 'three points are equal' constraint.*

objects. Let ε be a tolerance level. The c constraint is within tolerance if and only if $|c(\mathbf{x})| < \varepsilon$, and we want to validate whether the constraint holds. For this, we introduce the following function:

$$s_\varepsilon(x) := \begin{cases} x & \text{if } |x| < \varepsilon \\ 0 & \text{otherwise.} \end{cases}$$

We observe that, if $c(\mathbf{x})$ is out of tolerance, then $s_\varepsilon(c(\mathbf{x}))$ vanishes, and the constant zero constraint will not modify our system, otherwise $s_\varepsilon(c(\mathbf{x}))$ reproduces $c(\mathbf{x})$.

A necessary condition for c is, that $c(\mathbf{x})$ represents a so-called *faithful* representation for the distance used for a given entity. (Faithful means that the true Euclidian distance or a close approximation is computed in the vicinity of the curve/surface to be fitted, see details[1]). For example, for the *line meets point* constraint, we must use a normalized line-point distance function

$$c_{pl}(\mathbf{x}) = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}.$$

Also note that s_ε is not continuous, but a piecewise continuous function, so if we calculate the derivative numerically, we need to make it piecewise, as well.

To detect constraints automatically, we take the modified constraints for all object pairs. The numerical method will enable constraints only that are within the related tolerance level. The user typically defines different tolerances for different – parallel, perpendicular, tangential, concentric, etc. – constraints. Let us denote the set of objects by $S = \{s_i\}$, and the constraint types by $\{c_j\}$, such that $c_j(s_1, s_2)$ denotes an actual constraint between s_1 and s_2. Then for all c_j, consider the following constraint set:

$$C_j = \{s_{\varepsilon_j}(c_j(s_1, s_2)) | s_1, s_2 \text{ suitable for } c_j\}.$$

Thus the global constraint system includes the explicitly defined constraints, and the C_j-s, i.e. the 'likely' constraint set.

A somewhat artificial example with three circles and three lines can be seen in Figure 4 that shows different configurations created by different constraint tolerances. Compare cases (b) and (c). The angular tolerances of 'lines orthogonal' are (b):10, (c):10 degrees. The distance tolerances of 'line passes through the center of a circle' are (b):10, (c):10 units, and 'line is tangent to circle' are (b):15, (c):25 units, respectively.

We can also handle more complex local (i.e. not pairwise) constraints. For example, take the previously mentioned *three lines meet in a single point* constraint in Figure 5. We may create auxiliary intersection points for all three pairs of lines, and by means of a corresponding *line close to point* constraints the algorithm can detect, whether the three intersection points are "likely to be" coincident or not. In the former case the three lines will be fitted simultaneously, enforcing a common point of intersection.

4. The best fit global grid

In this section, we investigate how to detect and create a best fit *'grid'* object and set the corresponding constraints.

The grid is represented as a 5 dimensional auxiliary object with the following parameters:

- the orientation of the grid (n),
- the origin of the grid (p_0),
- and a positive constant, the width of the cells (d).

Note that, the above parameters are not uniquely defined, since we can select all intersection points of the grid as origin, and we have four ways to define the orientation.

We can define constraints for the grid in a similar way, as earlier. For example, the constraint of a line ($Ax+By+C=0$) is orthogonal/parallel to the grid can be given as $c(\mathbf{x}) = \min(|An_1 - Bn_2|, |An_2 + Bn_1|) = 0$. We can define the point meets grid constraint as $\langle p - p_0, n\rangle / d$ and $\langle p - p_0, n^{\perp}\rangle / d$ are integers. So the most important constraints are the following:

- certain parameters (n, p_0, d) are fixed,
- points are contained in the grid,
- lines are orthogonal/parallel to the grid,
- lines lie on the grid lines.

For detecting the above constraints, we can use the automatic methods shown earlier, but this will work only, if the grid has been initialized 'almost perfectly'. Alternatively, we suggest an algorithm based on the following four basic steps (see Figure 6):

1. determine the best orientation,
2. fit the corresponding lines,
3. determine the best width parameter,
4. refit the objects matching the enhanced grid.

4.1. Determine the best orientation

Assume that, each line belongs to a straight section. Let us denote the length of l_i as $h(l_i)$, and the angle from x-axis of its normal vector l_i in radian as $\angle(l_i)$. With respect to the grid, α and $\alpha + \pi/2$ have the same orientation, so we work with angles modulo $\pi/2$. The solution is given by clustering the angular values. We associate a radius with each cluster, that depends on a tolerance level and a weight by $w(S) = \sum_{l \in S} h(l)$. We select the best cluster (where $w(S)$ is maximal), and the weighted average of the angles will yield the best orientation.

4.2. Determine the best cell size

After setting up the best oriented grid, we fit the corresponding lines by the automatic method presented in Section 3. The next problem is determine the best common divisor of the distances between the parallel lines.

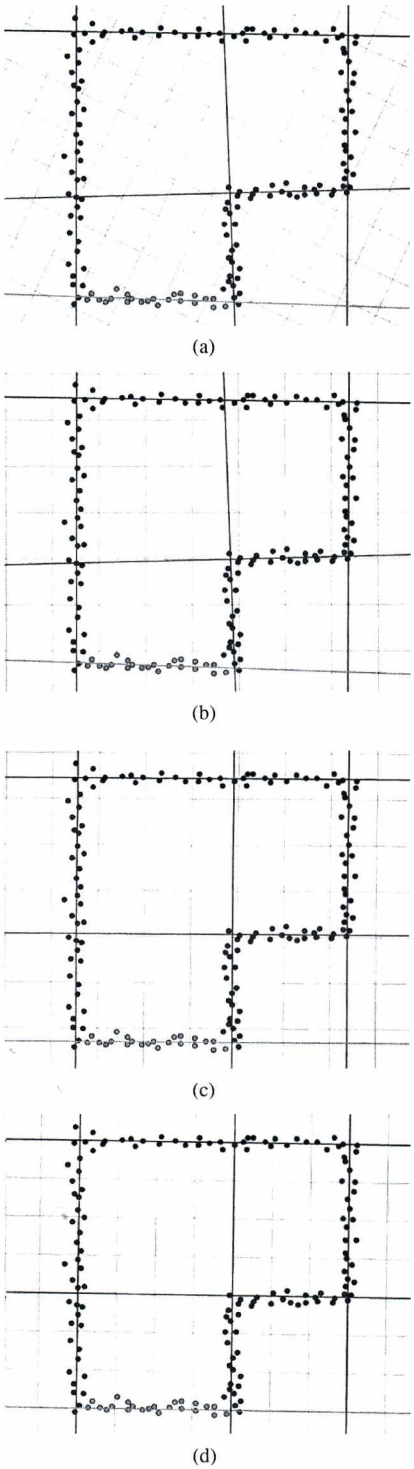


Figure 6: *Detect grid: (a) initial state (independent fittings); (b) optimal orientation; (c) re-fitted lines; and (d) final fitting with optimal width.*

Let us denote the lines fitted according to the optimal orientation by $\{l_i\}$, and the absolute distances between the parallel lines by $\{d_{ij}\} = \{n_l\}$. If d is a suitable width of the grid cells, then the average remainder by $\{n_l\}$ is small. The sum of remainders can be written in the form

$$\delta(d) = \sum_l \min\left(\left\{\frac{n_l}{d}\right\}, 1 - \left\{\frac{n_l}{d}\right\}\right),$$

where $\{x\}$ denotes the fraction part of x.

Now the goal is to find the minimum of $\delta(d)$ in the $[d_{min}, d_{max}]$ interval. It is easy to see, that the function δ is piecewise monotone, and the monotonicity drops at numbers being in the form of n_l/k, for certain n_l-s, where k is a positive integer. Therefore, we need to search for the minimum only at these points, and we can find this in $O(N^2 n_{\max}/d_{min})$ steps, where N is the number of distances, and $n_{\max} = \max_l n_l$.

After setting up the optimal grid size, we can automatically detect the lines that satisfy all the grid constraints.

5. Estimate global symmetries

The second area of setting global constraints is the computation of axes of symmetry. We investigate algorithms for curves consisting of straight segments and circular arcs. First we determine all potential axes that may occur, then evaluate and prioritize them, and finally select the best one(s). Let $P = \{p_i\}$ denote the endpoints of the segments and the centers of the circles, and $L = \{l_i\}$ the lines.

The main steps of the algorithm are the following (see also Figure 7):

1. Collect all perpendicular bisectors between the points of P, and all angular bisectors between the lines of L. These bisector lines are called *auxiliary lines*.
2. Determine clusters of the auxiliary lines.
3. Evaluate the clusters (i.e. compute the corresponding axes and evaluate their 'measure' of symmetry).
4. Select the best axis (axes), and apply constrained fitting accordingly.

The set of *auxiliary lines* A contains the perpendicular bisectors between the points of P: $A_1 = \{\text{PBisector}(p_i, p_j) : i < j\}$, and the angular bisectors between the lines of L: $A_2 = \{\text{ABisector}(l_i, l_j) : i < j\}$. Now we cluster these in two steps. First by the argument of the normal vectors (modulo π), then by the distances from the origin. For each cluster C, let l_C denote the average of C, these are the *axis candidates*. The number of elements in a given cluster is not necessarily the best quantity to measure the extent of symmetry; it is better to locate the corresponding symmetric parts by the related axis candidate and compute their arc lengths. With other words, symmetries amongst many small segments will be considered less important than those of a few large segments.

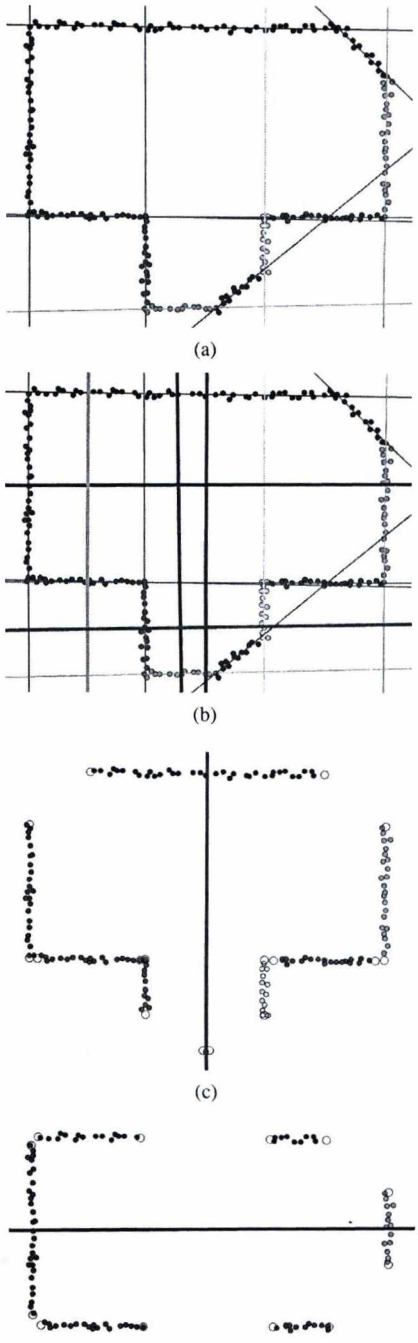


Figure 7: *Detecting axis of symmetry: (a) initial state; (b) axis candidates; (c) the best axis (measure:* 70%*); and (d) the second best axis (measure:* 41%*).*

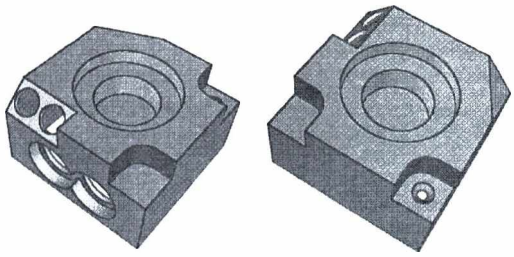


Figure 8: *Segmented object from measured data*

The clusters also provide information about symmetries of circular arcs, which help to enhance these computations. For each pair of arcs, we determine the corresponding arcs, or parts of arcs, that can really be considered as symmetric. The sum of these arcs yield additional weights to qualify the axis candidates.

We generally define constraints for axis of symmetry using auxiliary objects, as well. For example,

- *an axis is a perpendicular bisector of a segment,*
- *axis is an angular bisector of two lines*, etc.

Finally we perform constrained fitting according to the best axis of symmetry.

6. Conclusion

It is a crucial issue in the course of reconstruction from measured data to perfect engineering objects, since having only rough approximations for perpendicularity, parallelism, concentricity, etc, would not be acceptable for the majority of downstream CAD applications. In this project we have investigated *perfecting techniques* to automatically set up local and global geometric constraints. We have tested our methods for planar point data sets, representing straight and circular curve segments; at the same time these algorithms can be generalized to 3D objects in a reasonably straightforward way; this is subject of ongoing research (a test example is shown in Figure 8). In the future, we plan to detect other type of global symmetries, such as, rotational and translational symmetries and apply these techniques, when conventional and free-form curves and surfaces need to be coupled with constraints.

Acknowledgements

We would like to thank the other members of our research team – Péter Salvi, György Karikó and Pál Benkő – for important technical discussions. This research is being supported by the Hungarian Scientific Research Fund (OTKA No.101845).

References

1. P. Benkő, G. Kós, T. Várady, L. Andor, and R. R. Martin. Constrained fitting in reverse engineering. *Computer Aided Geometric Design*, 19(3):173–205, 2002.
2. I. Coope. Circle fitting by linear and nonlinear least squares. *Journal of Optimization Theory and Applications*, 76(2):381–388, 1993.
3. M. Li, F. C. Langbein, and R. R. Martin. Detecting approximate incomplete symmetries in discrete point sets. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pp. 335–340. ACM, 2007.
4. G. Lukács, R. R. Martin, and D. Marshall. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In *Computer Vision-ECCV'98*, pp. 671–686. Springer, 1998.
5. P. Marks. Capturing a Competitive Edge Through Digital Shape Sampling & Processing (DSSP). *SME Blue Book Series*, 2005.
6. N. J. Mitra, L. J. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Transactions on Graphics (TOG)*, 25(3):560–568, 2006.
7. J. Porrill. Optimal combination and constraints for geometrical sensor data. *The International Journal of Robotics Research*, 7(6):66–77, 1988.
8. H. Pottmann, S. Leopoldseder, and M. Hofer. Approximation with active B-spline curves and surfaces. In *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on*, pp. 8–25. IEEE, 2002.
9. V. Schomaker, J. Waser, R. T. Marsh, and G. Bergman. To fit a plane or a line to a set of points by least squares. *Acta crystallographica*, 12(8):600–604, 1959.
10. T. Várady and R. R. Martin. Reverse engineering. *G. Farin, J. Hoschek, M. S. Kim, Handbook of Computer Aided Geometric Design, Chapter 26*, Elsevier, 2002.
11. T. Várady, P. Salvi, *3D Geometric Modelling and Digital Shape Reconstruction, Lecture Notes*, Budapest University of technology and Economics, BME IIT, 2013.
12. N. Werghi, R. Fisher, C. Robertson, and A. Ashbrook. Modelling objects having quadric surfaces incorporating geometric constraints. In *Computer Vision-ECCV'98*, pp. 185–201. Springer, 1998.

Optimizing State Changes in Rendering Engines

Dániel Bányász and László Szécsi

Budapest University of Technology and Economics, Budapest, Hungary

Abstract

This paper presents an algorithm for ordering state change operations—including shader context changes and input/output bindings—necessary to render a frame in an interactive application. We expand on the context of the render queue*, but instead of sorting renderable primitives only by material, we propose a flexible framework grouping together drawing calls that share any of the conceivable render states, minimizing the number of CPU-GPU communication instances required to set them.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Graphics systems

1. Introduction

A modern graphics engine has numerous requirements to fulfill. From the consumer viewpoint, it has to create visually appealing images at a constant frame-rate. From the developer viewpoint, it should couple numerous subsystems and allow for flexible addition of new functionality and unrestricted combination of existing features[4]. Game design and content integration should also be easy. Meeting all these criteria simultaneously can be challenging.

Engines are inherently complex software systems that serve as middleware connecting graphics hardware and game logic. Graphics hardware itself has sophisticated architecture running half a dozen shader programs, at least a dozen differently accessed memory constructs, and a good number of fixed-function elements, the operation of which is still highly customizable. Game logic and scene management can be similarly complex, but organized along different principles, and usually not entirely known at the time of engine development. This calls for a software design that is manageable, extendable, and scalable for programmers, even when all the underlying flexibility is exposed. In practice, for programmers with sufficient skills and understanding, programming tasks should not snowball in complexity as the system grows[8].

This level of expandability is not adequately supported by basic instruments of object-oriented design, as encapsulation and inheritance[4]. In order for the addition of new features to happen without compromising the already existing functions in a major way, the engine needs to be very modular. This way changing or completely remaking specific elements of the main engine will not affect other parts of it.

Engines need to support real-time high-fidelity photorealistic visualization of extremely detailed game worlds[11]. Much of the visual experience is dependent on the artistic content, for which a complete pipeline of content creation and integration, with numerous editing tools, is usually supplied. However, it is the rendering capabilities of the engine that define the limits of what game world contents can be. Immersion into a game is reliant on continuous animation, so image frames have to be rendered at consistently high speed[1]. Thus, rendering performance and accommodation of a highly customizable effect range are the two factors that can define the limits of gaming experience.

Unfortunately, flexibility contradicts performance in practice. Both from a programming or a rendering perspective, flexibility translates to a wide and extendable set of features that can be freely combined in the definition of game world objects. Optimization for performance, on the other hand, means the elimination of multiple executions of the same task through exploiting uniformity. Consequently, almost every optimization technique depends on restricting what the engine can do. For example, grouping some of the game object properties into *materials*, then drawing objects sharing the same material together is a natural and widely accepted technique. However, what exactly constitutes a material depends on design decisions, where choosing a too narrow subset results in little reuse, while a too wide subset either leads to too much uniformity or too many materials. Multi-level material systems could address this problem, but then the hi-

erarchy of material parameters already assumes a lot about their usage. Even if this is based on well-established best practice, it is a restriction imposed on how the content should be prepared for optimal performance.

In this paper we describe the outlines of a flexible graphics engine, focusing on the capabilities of optimized rendering without assuming any pre-established classification, uniformity or hierarchy of game world objects. This is to provide the means of high-end rendering quality while maintaining complete freedom in the design of material or shader systems and game object construction. The main contribution of the paper is a run-time algorithm to order all visible game world objects for rendering so that state change overheads are minimal.

Our optimization algorithm is a greedy one, which means it is not always optimal. Greater performance increase can be achieved using batching of elements or instancing. However, this algorithm can easily work with batched resources and instancing, is independent from the concrete graphics API and development environment and can easily be modified and expanded to the needs of any certain engine.

2. Component-based engine architecture

The choice of a *game object model* or *entity model* is a defining feature of game systems[4]. Small games fare well with the classic *monolithic model* that relies on object-oriented inheritance hierarchy. Every different type of entity is implemented in a class, with common features extracted into superclasses. This would imply multiple inheritance where entity types share features along multiple taxonomies. Due to the inherent design challenges of multiple inheritance, it is only used with restrictions. Interfaces and *mix-in* classes add some flexibility to a monolithic inheritance tree.

The most severe issue with an inheritance based game object model is that of adaptability and extensibility. Game development is often an organic process where technological features may inspire gameplay elements[4]. If a new feature is added to an entity class, it is very likely that sooner or later other, formerly unrelated entities could also use the feature. Moving shared functionality to the lowest common base class results in bloated classes at the top of the inheritance hierarchy, and requires switches to turn off functionality where not needed. This defeats the purpose of object-oriented decomposition. Mix-ins solve this problem only for simple bits of functionality that do not require further decomposition.

Another, related problem is the strong coupling of interfaces. Whenever an entity class requires that the interface is changed, this can affect a complete subtree of the inheritance hierarchy.

The need for integration of several subsystems in an engine rapidly and naturally leads to component-based systems, where entities are defined as collections of components. The idea lends itself to implementations ranging in complexity and flexibility from simple *composition over inheritance* schemes to run-time configurable object compositions. *Property-based* systems can be seen as an extreme case, where all entities are disassembled into atomic components, and behavior is no longer defined by components themselves, but the logic linking those components. Operations are dispatched not based on the type of one component, but on the occurrence of certain sets of components.

The decomposition of game objects into components can easily be extended to features that are parts of virtual worlds, but not readily viewed as game objects in simple systems. Light sources, cameras, trigger zones, and audio sources may very well share functionality with renderable objects.

A component-based architecture allows simple, yet powerful and flexible couplings between entity aspects and the ability to create or change these connections very easily. The more abstract the component management is, the more machinery it requires to glue a component into the system, but also the less existing components constrain what new components can do.

3. Testbed implementation

In our implementation we opted for a strong and strict decoupling of subsystem functionalities into rendering, physics, etc. components. Variations within those components are handled by inheritance, e.g. there is a common interface for physics components allowing creation, destruction, and simulation. However, how simulation results are transferred to rendering components is not defined in either physics or rendering components: they are independent. Also, there is no entity class storing components that belong to the same entity, it is only the components themselves that store an entity identifier. Entities exist only as components sharing the same identifier.

As in the rendering and physics components example, components have to communicate, and they need to do so without compromising loose coupling. We need an object in the engine that knows about all the different types of components and can invoke the specific behaviors from these elements. These objects are *visitors*, and they implement the Visitor Pattern[7, 6]. For every action (render, update, release resources etc.) we create a specific visitor object that knows how to handle given components. This way the knowledge about the concrete coupling and the communication between components is comprised in the visitors and not scattered throughout the engine. Implementing visitors is not substantially different from implementing virtual functions in a monolithic inheritance engine, but visitors can operate on multiple classes.

4. Dependency injection

This approach incurs some communication overhead, as visitors need to look up objects based on entity identifiers and discern their types, but this is easily acceptable for game logic resolution. When it comes to rendering, however, these loose connections can be a burden. In a monolithic engine, all data for drawing an entity would be encapsulated in one class, and a single render method would be responsible for drawing the entity. With components, the visitor must gather all the parameters, variables and resources necessary.

This overhead can be eliminated if connections between critical components are made more direct. In order to achieve this without undermining component separation, we employ the *dependency injection* pattern[3, 9]. This inverts the control flow between various components supplying rendering parameters (e.g. model, camera, light poses) and the component performing rendering. While originally the render-performing component should locate and query all other components, with dependency injection it is the other components that register themselves.

5. Shader components

The render-performing component in our implementation is called the *shader component*. None of the data requirements for a shader are hard-coded. Instead, they are acquired runtime using *shader reflection* API[13]. After loading a shader program, all of its parameter and resource binding points, along with their names used in the shader programs, are gathered and saved. A shader component stores every parameter by name. Visitors can connect data providers and shader components by querying provided items and setting references to them into named shader parameters. A visitor might contain the logic linking data items to names, but if shader program authoring is performed at the same time as component design, it is a clearer solution to have the provider components use the same naming convention as the shader programs.

After the shader components have been populated with appropriate settings, rendering does not require any communication between components. As data provider components pass data references to shader components, no synchronization is necessary when the data changes.

6. Related work

A high number of render components share a large subset of parameters. The camera properties are the same for all of the objects in the frame, texture samplers can also be common between lots of elements, and the lights can affect groups of objects in the virtual world. We may be drawing a lot of entities with the same texture or with the same world matrix, if one object consists of multiple render components. What we need to do is find the overlapping resources and set them only once for a large number of objects, minimizing the state changes and the communication between CPU and GPU, maximizing the utilization of the PCI-Express bandwidth, and therefore optimizing the rendering process.

The most efficient way of reducing CPU-GPU communication is reducing the number of draw calls, or *batches*, themselves[12]. This can be done if several entities can be drawn at once. As no state changes during the processing of a batch are possible, any variation in drawn geometry must be accommodated for dynamically in shaders. This requires pre-transformed static objects stored in common geometry buffers, the use of texture arrays instead of switching textures, and generic shaders accommodating different material models. These optimizations can be performed when designing the shader library and the scene management for the engine.

A special, hardware-supported case of batching is *geometry instancing*[5]. This allows rendering the same geometry several times, with shaders able to process them with different parameters.

For the purposes of run-time performance optimization, we assume batching has already been done, and the entities may already constitute several game objects batched together. In other world, we define an entity as what can be drawn in a single batch.

Minimizing the number of state changes can be accomplished by ordering batches so that those using similar state settings are performed one after the other. The tool for this is the *render queue*[2], which accumulates batches, and executes them only after ordering. This solution is employed generally, but the ordering strategy is often arbitrary, or manually tuned to a certain purpose. Typically, engines have a material system, where materials encompass a subset of all render states. How this subset is chosen is based on industry best practices, but it is not adaptive on the momentary runtime set of entities. The batches are ordered according to material IDs, which are statically assigned.

Our algorithm optimizes draw calls not by materials but by used states and resources. We do not differentiate between render state defined at material or entity level, and dynamically adjust the ordering strategy depending on the composition of entities in the scene.

7. The rendering optimization problem

The virtual world, or *scene*, is a collection of constituent *entities*. From the rendering point of view, we consider those entities that are *renderable*, meaning that they can be drawn into a render target image. For each of these entities there exists a set of various *render components* that define the rendering behavior. As a simple, but typical case, these could be a *pixel shader component*, a *vertex shader component* and an *indexed geometry component*. Such a setup allows multi-

material meshes if they are instantiated as several single-shader entities.

The objective is to determine how and when to upload or bind data to the GPU in order to minimize CPU-GPU data transfer and communication latency. Several entities need to be rendered using the same resources, shader programs and state blocks, and thus, re-binding—or even re-uploading—these could needlessly stall operation. For simple scenarios, this could be addressed manually in the content creation phase, using a material system that orders materials so that similar ones follow each other. Then, entities can be sorted by material ID in a render queue, in run-time, before rendering. However, content-creation-phase ordering might not be optimal for the actual set of entities that are present at run-time. Furthermore, if rendering behavior is not merely defined by a material, but by an ever extendable set of components, then the render queue approach is insufficient. For example, a future gameplay decision may dictate that the player is given a gun that changes the texture of entities fired upon. This can easily be solved by creating a new render component class that imposes a texture setting when visited, overloading the material default. However, this could break the material system and render queue ordering, calling for a custom workaround to be implemented. In order to avoid development overheads of this kind, we aim to determine the optimal render ordering at run-time.

An exact method can be easily imagined using a graph, every node of which is an entity. The graph is a complete graph, and going from one node to another means rendering one entity after the other. This graph is also a weighted one, where the edge weights between nodes represent the severity of state-changes between GPU states when rendering one entity after the other. Before we render an object, we bind (or even need to upload) all the necessary resources, set the render states, and when we are finished with the rendering, we have to update some or even all of the parameters in order to prepare for the drawing of the next object. When the weight of an edge is low, it means we barely have to change anything. When this number is high the number of differences between used resources is significant, meaning we need a lot of data transferred to the GPU before the actual rendering begins. The edge weights should also represent the differences between the performance impacts of the API calls.

After the graph is ready, we need to find a path that includes every node once and only once, and has minimal weight, i.e. a minimal weight Hamiltonian path. This would mean that we have rendered every entity with minimal or no overhead. Finding a Hamiltonian path in a complete graph is trivial, but finding one with minimal weight (known as the *travelling salesman* problem) is known to be NP-hard[10]. There are various fast approximate algorithms, however. Our solution can be seen as another such algorithm, also exploiting our knowledge about the structure of the edge weights.

8. The algorithm

There is a high number of configuration items that influence GPU operation. These include render states, input/output resource bindings, dynamic resource contents, shader programs or dynamic shader linking parameters. For the sake of brevity, we will refer to all of these as render states, even though setting costs and the implied amount of CPU-GPU data transfer might vary greatly for different types.

Render states are set using graphics API methods. An atomically settable render state is something that can be changed by invoking a single method. The basic intuition of our algorithm is that if there are two atomically settable render states with similar setting costs, one of which takes just a few different values in the course of rendering, but the other one is likely to be different for all entities, then it is more beneficial to order entities according to the first one.

To explain this, let us assume we need to order n entities. Let M be one of the graphics API methods, associated with a render state. The cost of invoking M is c_M, and there are v_M different values of the render state used. Then, the cost for invoking M individually for every entity is $c_{\text{base}} = c_M n$. If sorted by the values of the render state, the cost will be $c_{\text{opt}} = c_M v_M$. The *gain* is $c_{\text{base}} - c_{\text{opt}} = c_M(n - v_M)$. Thus, the gain is more significant if there are fewer possible values or if setting the state is more expensive.

Thus, our algorithm will find render states that are used by a high number of entities and are expensive to change. We set these early and keep the GPU and its memory in the correct state while all corresponding entities are drawn. To achieve this, we count the number of different parameter sets for every graphics API method that causes data transfer or state change. We refer to a method call with a certain set of parameters as a *task*. Two calls with the same set of parameters are not considered separate tasks. If a method is invoked by numerous entity components, but repeating only a few tasks, then a high number of entities share the render state setting.

Our algorithm first selects a render state, ordering along which offers the maximum gain. For this we need to know the number of different parametrizations the methods setting render states are called with. In other words, this is the number of tasks pertaining to the method. Also required is the cost of invoking a method over the API. When the maximum gain render state has been selected, the entities are grouped by the value of the render state they require for rendering—in other words, which task of the method they invoked. The resulting groups consist of entities for which the render state needs only be set once, if the group is rendered together. Such a group of entities can further be divided choosing another render state (i.e. API method) to discriminate by. This can be repeated recursively, until there remains no other API method but the final draw call, which has to happen for every entity.

8.1. Optimizer

The central construct of our algorithm is the *optimizer*. This provides an interface equivalent to that of the graphics library used to control GPU operation, but for all method calls, the calling entity also has to be specified. The optimizer does not immediately execute the graphics library method calls, but records them, and generates an optimized list of *commands*, that can be played back to the same effect.

Thus, all shader objects and other rendering components of entities issue parametrized calls to the optimizer, instead of the graphics API. In order to explain the optimizer, let us introduce the following nomenclature:

- A *method* is a graphics API function exposed on the optimizer's interface.
- The *method cost* is the overhead incurred by calling the graphics API function.
- A *task* is a graphics API call with a given parametrization, i.e. a method–parameter-set pair. Invocations of the same method with identical parameters are not considered separate tasks.
- The *callers of a task* are those entities, the components of which have invoked the task's method with the task's parameter list.
- An *entity set* is a set of entities to be drawn.
- A *bin* is a set of the tasks that refer to the same method.
- A *method of the bin* is the method all tasks of the bin refer to.
- The *bin gain* is the reduction in state setting overhead achievable by grouping entities according to their tasks in the bin. It is computed as the number of entities minus the number of tasks in the bin, weighted by the the method cost.
- A *command* represents one execution of a task. Thus, it is a wrapper for a graphics API call with a given parameter set. By extension, API draw calls can also be wrapped in commands.
- The *command list* is a replayable sequence of API calls.

The optimizer stores all tasks into bins, one for every method. We refer to the bin that has the highest gain as the *best bin*—the choice is arbitrary in case of a tie. For every task, the parameters and the list of callers are also stored.

The optimizer maintains this data structure when receiving invocations from rendering components of entities. If a method is called with a parameter set that differs from every previous, a new task is created, and added into the appropriate bin. If the parameter set used by the calling entity was already associated with a task, we just register the entity as one of its callers.

Draw calls—the graphics API function calls that initiate pipeline operation with current bindings—are not wrapped or stored as tasks. These cannot be eliminated by ordering, as a draw call must happen for every entity. (As discussed in Section 6, optimizing the number of draw calls using instancing or batching is out of the scope of our investigation.) On the other hand, the output optimized render list must contain draw commands. Thus, the optimizer maintains an entity set, containing all entities that have to be drawn. This set is identical to the union of callers of all tasks, and thus redundant in information to the task bins. However, it is much more useful for entity set partitioning operations.

8.2. Optimization

When entities are created, rendering components are also instantiated. The core functionality is encapsulated in the shader component class. Shader components manage all program and resource bindings of the GPU pipeline. Initially, however, shader components are devoid of bindings, and it is the role of the visitors to fill these as dictated by other components. Thus, components are decoupled, and linked only by visitor logic. During the process, shader components receive memory locations for all the shader variables' desired values. After these bindings are made, the engine can start creating the optimized render list.

First, the optimizer is instantiated, with empty bins for all methods, and an empty entity set.

Then, we emulate the rendering process, setting required resource bindings and render states, and rendering the geometry through the optimizer. This is accomplished by visiting the shader components, that inform the optimizer about required settings, per entity. The optimizer receives these calls, and creates, bins and updates tasks as described in Section 8.1. Also, the caller is inserted into the entity set, if it was not a member yet.

By the end of this step, the optimizer has been initialized, and the OPTIMIZE function (Algorithm 1) can be called with the set of all bins and the complete entity set as inputs, returning a list of commands (see Figure 2 for an overview).

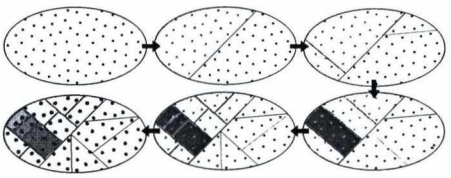


Figure 1: *Recursive subdivision of the render state space.*

The function partitions the entity set into groups recursively along a set of methods (Figure 1), represented by the bins of their tasks. For an empty group E, there is no action required (lines 2-4). Otherwise, the list of commands to render the group must be returned, assuming all appropriate settings have already been applied, apart from those using the methods of $\mathcal{B}$, which are currently subject to optimization. The necessary commands are appended to an initially

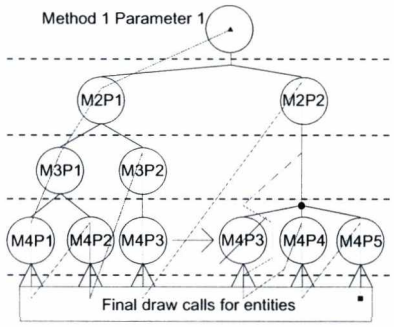


Figure 2: *The algorithm recursively partitions entity sets along tasks, implicitly creating a tree. The green polyline indicates the order in which the tree is traversed to find the output render command list.*

Algorithm 1 Optimize function

```
1:  function OPTIMIZE(set of bins B, set of entities E)
2:      if E is empty then
3:          return empty list of commands
4:      end if
5:      R ← empty list of commands
6:      if B is empty then
7:          for each entity e in E do
8:              R ← R + new command(draw e)
9:          end for
10:     else
11:         M ← best bin in B
12:         for each task p in M do
13:             R ← R + new command(p)
14:             I ← {callers of p} ∩ E
15:             D ← empty set of bins
16:             for each bin B in (B \ M) do
17:                 D ← D ∪ {q | q ∈ B ∧ (caller of q) ∈ I}
18:             end for
19:             R ← R + OPTIMIZE(D, I)
20:         end for
21:     end if
22:     return R
23: end function
```

empty list R (line 5). We refer to adding commands performing tasks into the render list as *compiling* the tasks. If the bin set $\mathcal{B}$ is empty, then all settings are already compiled into the command list, and only the draw calls for all entities must be issued (lines 6-9). Otherwise, a best bin M can be selected (line 11).

Then, we loop over all tasks in the best bin (lines 12-20). Processing one task should compile the setting of a render state and the rendering of all entities using it. Therefore, we append the command executing the task p to the command list R (line 13). Then, all the entities that use p are selected into set I (line 14). These entities must be compiled, but without calling the method of M, as that has already been properly applied. This is accomplished by calling OPTIMIZE with entity set I, and a reduced set of bins $\mathcal{D}$ (line 19). The returned command list is appended to R.

Set $\mathcal{D}$ is populated from bins of $\mathcal{B}$ other than M, copying all tasks that belong to an entity in I (lines 15-18). Note that bin M has been eliminated from the recursion, because the entities we are left with share the same setting with respect to that method. Tasks that concern entities other than the ones in the processed group are also eliminated, so that the gain computation for the bins remains correct.

After all tasks of M have been processed, the command list is complete and can be returned (line 22).

The top level call of OPTIMIZE returns a list of render commands that is as close to the optimum as the greedy approach allows.

However, it is possible that there is some unnecessary duplication amongst method calls. Suppose we render three entities, using two methods A and B, issuing tasks a_1, a_2, b_1 and b_2. The three entities e_1, e_2, e_3 use settings (a_1, b_1), (a_1, b_2) and (a_2, b_2), respectively. Partitioning by A separates the first two entities, resulting in the command sequence (set a_1, set b_1, draw e_1, set b_2, draw e_2). On the other branch, we get (set a_2, set b_2, draw e_3), where (set b_2) is superfluous. This suboptimal behavior is possible because separate branches are processed independently, and no considerations for minimizing state changes between branches are taken. While this concession has to be made in favor of the fast greedy evaluation scheme, a trivial elimination of ineffective commands can improve our solution in simple cases like the one presented above. We could see these as lucky coincidences, where the rendering of one group of entities happens to end in a render state in which rendering the next group should start—but these lucky coincidences are quite probable if only a few possible states exist.

Visiting rendering components again is unnecessary, as the render command list contains every method call with a full parameter list, with pointers to resources originally specified by the render components. Thus, no additional overhead other than the execution of the tasks is incurred when the command list is played back for rendering. If no entities are created or destroyed, or their components otherwise altered, the command list does not need to be re-generated in every frame. Dynamically uploaded data, like the contents of constant buffers storing camera and model transformations is allowed to change as long as the data resides at the same memory location. Thus, even for a dynamic scene, every frame can be rendered by playing back an optimized sequence of graphics API calls.

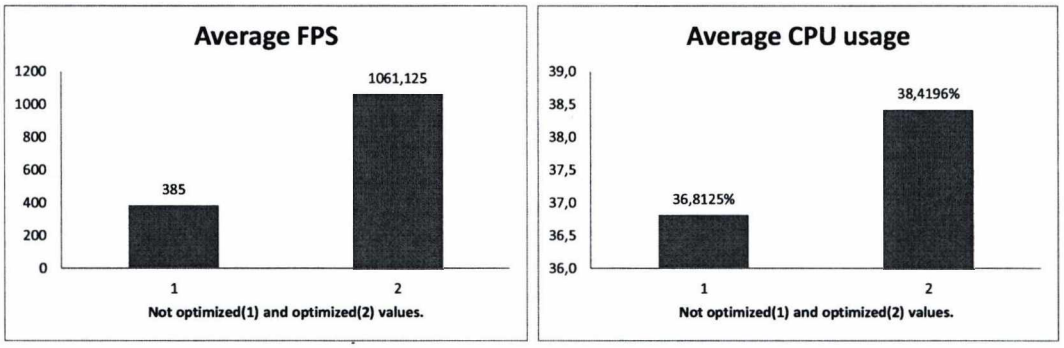


Figure 3: *Optimized CPU-GPU communication resulted in higher render speeds and increased CPU utilization.*

9. Results

We tested our algorithm on an Intel Core i5 3570K CPU with 8 GB RAM and an MSI N670 PE 2GD5/OC graphics card. With only a small initial computational overhead of creating the render lists, the speed of the actual rendering increased significantly. The differences between the optimized and unoptimized runs were apparent on the test machine.

These tests were conducted using a simple and carefully constructed virtual scene (see Figure 4). A small number of objects were placed in the world, some of them forming optimizable groups by using several common resources. With a test scene this size, the algorithm was fully observable.

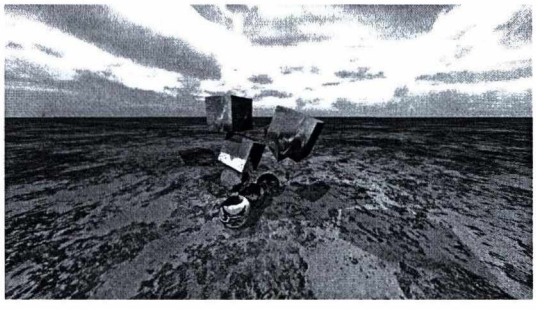


Figure 4: *Test scene with deferred shading and depth map shadows.*

We compared two methods of drawing this virtual world. One was with the suboptimal rendering, which draws entity by entity, collecting components and setting parameters individually. The other was the optimized method which uses the render list created by the algorithm. There are multiple light sources casting shadows using shadow maps, so each of them has its own render pipeline, meaning they all have to render the scene from their point of view. We use deferred shading, so each of the lights contributions are blended together in a final draw sequence. In conclusion, the scene will be rendered multiple times using separate render queues, separate optimized passes, each pass manifested in their own optimized method list.

The engine has some built-in tools to measure specific aspects of performance. We created the scene and the engine environment in order to quantify the gain of using our optimizer algorithm. In the first test, we compared the optimal and the suboptimal renders with no other restrictions on the engine, meaning we disabled V-Sync and any other frame rate limits. This way the GPU ran at 100% capacity generating as many frames a second as it possibly could. We ran the engine for a few seconds measuring at a steady interval the CPU usage and the rendered frames per second count. The numbers showed that using the optimized method list to draw the scene generated nearly three times the frames a second, than rendering without the optimized list (Figure 3).

The CPU usage was slightly higher during the optimized rendering. This can be explained by the fact that without optimization, the GPU communication overhead blocks the CPU from getting higher amount of work done. During the tests, the engine had nothing else to do but to render. It processed some messages from the operating system in order to allow the user to move the camera and close the program window, but apart from that, it only prepares the scene and uploads the data to the GPU. Faster render means more CPU work, because the engine will require more work to be done for the rendering process from the CPU side.

In an effort to determine the accurate gain in CPU power we tried to level the playing field in terms of CPU usage, and framelimited the engine. So at the second round of measurements, the FPS was capped at 60. This produced very low CPU load, probably because of the small size of the virtual space. The CPU usage while rendering from the optimized list however is nearly half with this simple scene than when we render the entities one by one (Figure 5). Hopefully even with a more complex virtual world this gain will have a huge impact on the overall performance.

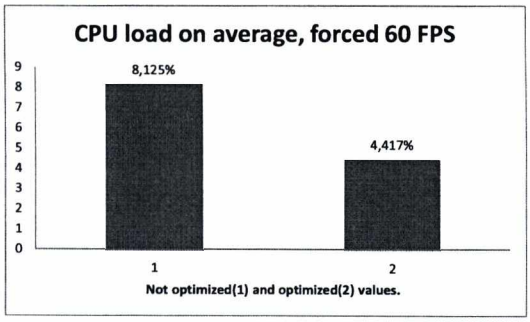


Figure 5: *CPU usage for the optimized and non-optimized cases over several frames with limited frame rate.*

10. Future work

The proposed solution needs further verification for more extensive scenes and full, plausible gaming scenarios. More quantitative comparison with material-based render queues are required, and a study of circumstances where the theoretical merits of the proposed algorithm manifest most strongly. We need to investigate further on the issue of when the render command list needs to be rebuilt, and whether it is possible and advisable to keep parts of the list and update others, trading optimality for cheaper list updates. In a related issue, the effect of visibility algorithms removing and adding entities for the set to be drawn should be addressed. We have not yet investigated another important option for increasing rendering performance, which is sorting entities by approximate depth. We plan to examine whether this sorting strategy can be combined with our optimization algorithm. Also, it would be useful to know when is one or the other more beneficial.

11. Acknowledgements

This work has been supported by OTKA PD-104710 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

1. Kajal T Claypool and Mark Claypool. On frame rate and player performance in first person shooter games. *Multimedia systems*, 13(1):3–17, 2007.
2. Christer Ericson. Order your graphics draw calls around! `http://realtimecollisiondetection.net/blog/?p=86`, 2008.
3. Martin Fowler. Inversion of control containers and the dependency injection pattern. `http://martinfowler.com/articles/injection.html`, 2004.
4. Jason Gregory, Jeff Lander, and Matt Whiting. *Game engine architecture*. AK Peters, 2009.
5. Emmett Kilgariff and Randima Fernando. The geforce 6 series gpu architecture. In *ACM SIGGRAPH 2005 Courses*, page 29. ACM, 2005.
6. Robert Cecil Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.
7. Bertrand Meyer and Karine Arnout. Componentization: the visitor example. *Computer*, 39(7):23–30, 2006.
8. Bob Nystrom. *Game programming patterns*. 2013.
9. Erick B Passos, Jonhnny Weslley S Sousa, Esteban Walter Gonzales Clua, Anselmo Montenegro, and Leonardo Murta. Smart composition of game objects using dependency injection. *Computers in Entertainment (CIE)*, 7(4):53, 2009.
10. Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994.
11. L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.
12. Matthias Wloka. Batch, batch, batch: What does it really mean? In *Game developers conference*, 2003.
13. Jason Zink. Direct3d 11 shader reflection interface. `members.gamedev.net/JasonZ/Heiroglyph/D3D11ShaderReflection.pdf`, 2009.

Hatching Animated Implicit Surfaces

László Szécsi and Ferenc Tükör

Budapest University of Technology and Economics, Budapest, Hungary

Abstract

This paper presents a highly parallel algorithm for the stylized, real-time display of fluids and smoke. We use metaballs to define a fluid surface from a particle-based fluid representation, but instead of the costly complete reconstruction of this surface, we only trace the motion of random seed points on it. Hatching strokes are extruded along the lines of curvature. We propose methods for hidden stroke removal and density control that maintain animation consistency.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image GenerationLine and curve generation

1. Introduction

Hatching is an artisitic technique that is often emulated in stylistic animation. Implicit surfaces are becoming increasingly important in real-time applications for visualizing fluids simulated by particle-based methods. Thus, we aim to extend real-time hatching to deforming implicit surfaces, and specifically to metaballs. In addition to enabling fluid rendering in hatching-stlye NPR, we also consider this approach a more feasible alternative to expensive polygonization[12, 18], ray casting[15, 3], or screen-space filtering[20], when visualizing scientific isosurface or fluid simulation data.

2. Previous work

In pencil drawings artists convey the shape and illumination of objects with the density and orientation of thin hatch lines[23, 7]. To mimic this, we should define a *density* and a *direction field* in the image plane that is as close as possible to what an artist would use. Density is influenced by illumination, while the direction field is determined either by the *principal curvature directions*[6], the tone gradient[10], or in case of animation, the direction of motion.

Hatch strokes should appear hand-drawn, with roughly similar image-space width, dictated by brush size, but they should also stick to surfaces to provide proper object space shape and motion cues. Hatches can be generated into textures and mapped onto animated objects, with level-of-detail mechanisms to approximate image space behavior[17]. In absence of surface parametrization, this approach is not applicable to implicit surfaces.

Hatch strokes can be generated directly in image space[9, 11]. In order to avoid the disturbing *shower door effect*, lines can be moved along with an optical flow or image space velocity field, but placing new strokes on emerging, previously non-visible surfaces still poses problems. Especially if strokes are long, following curvature or feature curves of object surfaces, they should maintain this even when only tiny fractions have become visible. This cannot be assured when only using image space information. For implicit surfaces, it is often prohibitively expensive to render a full image, or even to find isosurfaces in some pixels.

Several works[13, 19] proposed the application of *particles* or *seeds* attached to objects, extruding them to hatch strokes in image space. The key challenge in these methods is the generation of the world-space seed destribution corresponding to the desired image-space hatching density. This approach is well-suited to implicit surfaces.

Much effort was directed at rendering implicit surfaces, esp. metaballs photorealistically in real time, based on ray casting[8]. This is computationally demanding as it requires a high number of field function evaluations to find the visible isosurface in every pixel. The stylisation of the result is straightforward only with image-space techniques, as no surface parametrization or visibility-independent object-space shape information is extracted.

Several apects of stylized rendering of implicit surfaces

have been studied. Brazil et al.[22] use *seed points* to generate *render points* on isosurfaces. They require the user to edit seed point distribution manually, excluding application for deforming surfaces. Elber[5] proposed the approach of first obtaining a Euclidean-space on-surface uniform point distribution, then extruding strokes along symbolically computed principal curvature directions[25, 16]. For the generation of uniformly distributed points on implicit surfaces, they refer the reader to Witkin[24], who proposes an adaptive resampling of deformed implicit surfaces, for purposes of sculpting, by the means of *repulsion forces*, *fissioning* and *killing* operating on a set of *floater particles*. Kooten et al.[21] employ a similar concept more specifically for isosurface rendering of metaball models. Both solutions require a full *self-spatial join* on surface particles to compute repulsion forces, and allow particles to float on surfaces. We consider this detrimental for our purpose of hatching stylization, as hatch lines not moving with the surface could provide inappropriate motion cues. A rejection-based density control approach from[19] does not require repulsion forces to achieve desired distrubution.

3. Implicit surfaces and metaballs

An implicit surface is defined as an isosurface at value L of field function $f(\mathbf{x})$ with the implicit equation $f(\mathbf{x}) = L$. The gradient $\mathbf{g}(\mathbf{x})$ is $\nabla f(\mathbf{x})$. Gaussian and mean curvatures K and H, the principal curvatures κ_1 and κ_2, principal curvature directions $\mathbf{t}_1, \mathbf{t}_2$ can be computed[1] using the Hessian $\mathbf{H}(\mathbf{x})$ (see in Appendix A). Where the determinant $D = H^2 - K$ is close to zero, the principal curvatures are not well defined, and we regard the surface point as umbilical.

Metaballs[14, 2] constitute a special case where the fluid is represented by a number of balls or *atoms* as

$$f(\mathbf{x}) = \sum_{j=0}^{M-1} f_j(\mathbf{x}) = \sum_{j=0}^{M-1} \rho_j\left(\left\|\mathbf{x} - \mathbf{a}_j\right\|\right), \qquad (1)$$

with M as the number of atoms, ρ_j the generator of *radial basis function* $f_j(\mathbf{x})$ for an *atom* centered at $\mathbf{a}_j$. If $\exists R_j \in \mathbb{R} : \forall r > R_j : \rho_j(r) = 0$, then we call R_j the *effective radius* of atom j. The gradient $\mathbf{g}(\mathbf{x})$ and Hessian $\mathbf{H}(\mathbf{x})$ can be computed as sums of atom gradients $\mathbf{g}_j(\mathbf{x})$ and atom Hessians $\mathbf{H}_j(\mathbf{x})$. See Appendix B for formula derivations for popular radial basis functions. When atoms are animated, they have velocity $\mathbf{q}_j(t)$ at time instance t.

Seeds are particles that we move along the deforming isosurface. The velocity $\mathbf{v}_k$ for a seed at position $\mathbf{s}_k$ is found as[21] (see Appendix C for details):

$$\mathbf{v}_k = \mathbf{v}_k^{\mathrm{fl}} - \qquad (2)$$

$$\frac{\mathbf{g}(\mathbf{s_k})\left[\mathbf{v}_k^{\mathrm{fl}} \cdot \mathbf{g}(\mathbf{s}_k) + (f(\mathbf{s_k}) - L)\Phi + \sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j\right]}{\|\mathbf{g}(\mathbf{s_k})\|^2},$$

where the flow velocity $\mathbf{v}_k^{\mathrm{fl}}$ is:

$$\mathbf{v}_k^{\mathrm{fl}} = \frac{\sum_{j=0}^{M-1} \left\|\mathbf{g}_j(\mathbf{s}_k)\right\| \mathbf{q}_j}{\sum_{j=0}^{M-1} \left\|\mathbf{g}_j(\mathbf{s}_k)\right\|}.$$

4. The algorithm

Our algorithm moves seeds along a metaball surface similar to[21], applies a screen-space approximate version of the density control approach from[19], and extrudes textured triangle strips along principal curvature directions. We propose a solution for the seed visibility problem based on the idea employed by *variance shadow maps*[4]. The algorithm performs the following steps in every frame of an animation:

1. Seed initialization.
2. Seed animation.
3. Seed filtering by visibility testing and rejection.
4. Curve extrusion from seeds.
5. Triangle strip extrusion from curves.
6. Stroke weighting and rendering.

Along the process, various weighting factors—w^{prox} for proximity to isosurface, w^{age} for age , w^{vis} for visiblity, w^{rej} for density control by rejection—for seeds are computed. The product of these w^{Π} is used in the final rendering step for opacity weighting, with the optimization that seeds with zero weight need not to be extruded into hatch strokes. The weight without density control, $w^{\mathrm{pre}} = w^{\mathrm{prox}} w^{\mathrm{age}} w^{\mathrm{vis}}$ is used for estimating *pre-rejection density*.

When seeds are initialized, they are placed randomly on atom-centered spheres within the effective radius. They are not guaranteed to be on the compound isosurface, and the isosurface-projected distribution might not be uniform. Those requirements are to be achieved by consequent seed animation and rejection steps, over the course of several frames. Seed points are re-initialized after a fixed lifetime to avoid excessive clustering. Seed point ages are evenly distributed, so that only a small percentage of seeds are re-initialized in every frame. Weight w^{prox} is computed as a smooth step function on the difference of the filed value at the seed point and the desired isosurface. This is to eliminate seeds not yet converged to the surface. Weight w^{age} fades to zero at the beginning and the end of the seed lifetime to avoid suddenly appearing and disappearing hatch lines.

Seed point animation is based on the technique proposed by[21], without using repulsion forces to achieve uniform density, thus eliminating the need for a self-spatial join on seeds. Seed animation according to Equation 2 requires the field value and the gradient. We compute these, and also the world space *stroke direction* along the isosurface. The computation of the stroke direction involves first finding the pure and mixed second derivatives forming the Hessian, the principal curvatures and curvature directions, the determinant D indicating whether the seed is at an umbilical point, the cosine of the view angle $\cos\Theta$ indicating whether the seed is near a

silhouette, and the local illumination normalized to a desired tone V at the seed.

Generally, the stroke direction is the principal curvature direction of the isosurface, but near umbilical points, we employ a custom direction, obtained as the cross product of the surface normal and a per-atom direction vector. The choice of this per-atom vector might be random, or subject to artistic consideration. In order to produce simple outlines, a different direction scheme is applied to lines near the silhouette: the stroke direction there is perpendicular to both the view direction and the surface gradient (see Figure 1). The three direction schemes are combined based D and $\cos\Theta$, so that there are no abrupt changes in the stroke direction. For any direction $\mathbf{t}$, the corresponding curvature κ can be found as $\kappa = \kappa_1 (\mathbf{t}\cdot\mathbf{t}_1)^2 + \kappa_2 (\mathbf{t}\cdot\mathbf{t}_2)^2$.

Seed points have to pass two filters to see if they should be extruded into hatch strokes. The first is the visibility test needed to decide if the seeds are seen from the camera. For this purpose, we render all seeds as isosurface-oriented billboards into a low-resolution buffer, outputting fragment depths and their squares. Using the idea of *variance shadow maps*[4], this low-resolution depth map is heavily filtered by two-pass separable Gaussian filtering. The resulting approximate variance depth map can be used for a smooth and lenient rejection of hidden seeds, producing visibility factor w^{vis}. As we are emulating the hand-drawn style, the error—from approximating the isosurface with billboards, using a low-resolution map, agressive filtering, and testing for visibility only at seeds—is not only acceptable, but welcome.

The second rejection step is to achieve an illumination-dictated screen space density of seed points (Figure 2). The *full cover density* Υ^{full} is an artistic parameter that specifies the seed density corresponding to surfaces devoid of illumination. This, modulated by seed tone V_k gives the desired on-screen density near a seed. Let us refer to the local density of all screen-projected seed points (weighted by w^{pre}) as Υ^{pre}. The ratio of $V_k\Upsilon^{\text{full}}/\Upsilon^{\text{pre}}$ gives the percentage of seed points to be kept. If all seed points have a random normalized priority value p_k, then those with priorities above the desired percentage should be rejected. The Υ^{pre} density is approximated by rendering all visible seeds, extruded into approximate hatch strokes, with additive blending, weighted by w^{pre} into a low-resolution buffer, and performing heavy filtering to eliminate rasterization artifacts. Note that what we get is not exactly the density of seeds, but an approximate density of hatching coverage. Thus, it helps to eliminate not only the clustering of seeds, but also the clustering of aligned strokes. Weight w^{rej} is computed as a smooth step function of $V_k\Upsilon^{\text{full}}/\Upsilon^{\text{pre}} - p_k$. Thus, rejection is performed smoothly, thus avoiding temporal visual artifacts, i.e. suddenly disapparing, appearing, or flickering hatch lines.

The seeds surviving visibility testing and rejection are extruded into curves. For short strokes, it is sufficient to use the local curvature at the seed, but longer lines require integration along the isosurface. In the latter case, visibility testing has to be performed for all samples. Curves are extruded into triangle strips to a uniform image space width. This width, and also the length of strokes, is an artistic parameters.

In the final rendering step the stroke is textured with an artist-drawn stroke image, with weights applied as opacity modifiers. We only discard the seeds if the weight would indeed be zero.

5. Implementation

The steps of our algorithm are implemented in five passes, depicted in Figure 3.

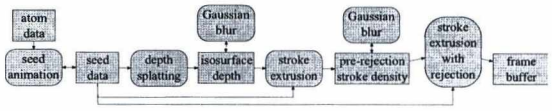


Figure 3: *Passes of the implementation.*

The first pass performs seed animation. All seed data is stored in textures used as data tables, where rows correspond to atoms, and the elements of the rows are individual seed points. Aging and re-initialization of seeds is performed by a rotating pipeline. In fact, in every texture row, seed attributes are shifted out to the right and reinitialized seeds shift in from the left, at a constant rate. The textures are also shifted vertically, to account for newborn and dying atoms, if so dictated by fluid simulation. For computation of quantities derived from the field function we used a regular grid space subdivision scheme to access relevant atoms.

The second pass produces the variance depth map of the isosurface to be used for a visibility filtering. Billboards are only extruded for seeds already converged to the surface to avoid unnecessary occlusion by seeds that are still trying to find their place. The depth values are blurred using a Gaussian filter, in accordance with the VSM technique, eliminating jagged edges in the depth map that could cause flickering hatch strokes in the final image.

The fourth pass is used to produce an image of Υ^{pre} values. These are needed for rejection of seeds later, to achieve uniform screen space density. It extrudes hatch strokes from all visible seed points, and applies the same opacity weighting to them—for visibility, age and proximity to the isosurface—, as would be when rendering on-screen strokes. Rejection for density, however, is not applied, since the goal is to approximate the hatcing density from all visible seeds. After visibility determination and curve extrusion, the hatch strokes are rendered, given color and opacity values that smoothly fall off towards the edges of the strokes. The output of this pass is rendered to a texture, using additive blending to generate high density values for high density areas on the screen.The Υ^{pre} density values also need to

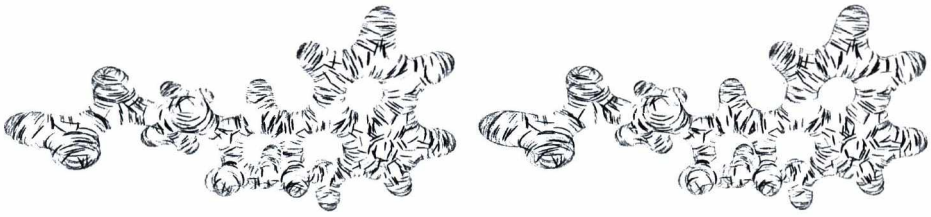


Figure 1: *Hatching of an LSD molecule discarding seeds near silhouettes (left) and rotating strokes to produce outlines (right).*

Figure 2: *Uniform hatching of an LSD molecule with and without illumination.*

be blurred, to avoid rasterization artifacts caused by jagged edges of approximate hatch strokes.

In the final pass, the process of rejection and opacity weighting based on visibility and hatch stroke extrusion is the same as it was during rendering the Υ^{pre} density. In addition, this pass also weights seed points using the Υ^{pre} values, and illumination values calculated on the fly, before extruding the hatch strokes themselves. If the compound weight of the seed is positive, the strokes are extruded, textured, and opacity is modulated by all weighting factors.

6. Results and future work

We ran our tests on a PC with an ATI5850 graphics card. At a resolution of 1024×768, with 65K seeds, which we deemed sufficient for rendering quality, and regardless of the number of atoms, we measured frame rates around 20 FPS.

Extruding long hatch curves requires several curvature samples on the isosurface, and as curves travel into zones of different curvature characteristics they tend to cross each other. Density estimation at seeds is also less accurate in this case. Therefore, we wish to investigate the possibility of using several linked seeds points for every hatch curve. Another limitation of the method is that the seed density cannot exceed that provided by rendering all seeds at unit weight. This is made worse if the distribution of seed points gets uneven because of seed motion. Thus, we plan to add seed fissioning and killing to improve performance and provide much wider level-of-detail support without increasing the seed count.

7. Acknowledgements

This work has been supported by OTKA PD-104710 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

1. Alexander G Belyaev, Alexander A Pasko, and Tosiyasu L Kunii. Ridges and ravines on implicit surfaces. In *Computer Graphics International, 1998. Proceedings*, pages 530–535. IEEE, 1998.

2. J.F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982.

3. N.K.R. Bolla. High quality rendering of large point-based surfaces. Master's thesis, International Institute of Information Technology, Hyderabad-500 032, INDIA, 2010.

4. William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165. ACM, 2006.

5. Gershon Elber. Interactive line art rendering of freeform surfaces. In *Computer Graphics Forum*, volume 18, pages 1–12. Wiley Online Library, 1999.

6. Ahna Girshick, Victoria Interrante, Steven Haker, and Todd Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings. In *Proceedings of the 1st international symposium on*

Non-photorealistic animation and rendering, pages 43–52. ACM, 2000.

7. A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

8. Y. Kanamori, Z. Szego, and T. Nishita. GPU-based fast ray casting for a large number of metaballs. In *Computer Graphics Forum*, volume 27, pages 351–360, 2008.

9. Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. Line-art illustration of dynamic and specular surfaces. In *ACM Transactions on Graphics (TOG)*, volume 27, page 156. ACM, 2008.

10. Yunjin Lee, Lee Markosian, Seungyong Lee, and John F Hughes. Line drawings via abstracted shading. In *ACM Transactions on Graphics (TOG)*, volume 26, page 18. ACM, 2007.

11. Zoltán Lengyel, Tamás Umenhoffer, and László Szécsi. Screen space features for real-time hatching synthesis. In *Proceedings of the 9th conference of the Hungarian Association for Image Processing and Pattern Recognition*, KEPAF '13, pages 82–94, 2013.

12. W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

13. Barbara J Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484. ACM, 1996.

14. H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modeling by distribution function and a method of image generation. *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, 68(Part 4):718–725, 1985.

15. T. Nishita and E. Nakamae. A method for displaying metaballs by using bézier clipping. In *Computer Graphics Forum*, volume 13, pages 271–280. Wiley Online Library, 1994.

16. Afonso Paiva, Emilio Vital Brazil, Fabiano Petronetto, and Mario Costa Sousa. Fluid-based hatching for tone mapping in line illustrations. *The Visual Computer*, 25(5-7):519–527, 2009.

17. Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581. ACM, 2001.

18. László Szirmay-Kalos, György. Antal, and Ferenc Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés*. ComputerBooks, Budapest, 2003.

19. T. Umenhoffer, L. Szécsi, and L. Szirmay-Kalos. Hatching for motion picture production. In *Computer Graphics Forum*, volume 30, pages 533–542, 2011.

20. W.J. van der Laan, S. Green, and M. Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pages 91–98. ACM, 2009.

21. K. van Kooten, G. van den Bergen, and A. Telea. Point-based visualization of metaballs on a GPU. *GPU Gems*, 3:123–148, 2007.

22. Emilio Vital Brazil, Ives Macêdo, Mario Costa Sousa, Luiz Velho, and Luiz Henrique de Figueiredo. Shape and tone depiction for implicit surfaces. *Computers & Graphics*, 35(1):43–53, 2011.

23. Georges Winkenbach and David H Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM, 1994.

24. Andrew P Witkin and Paul S Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 269–277. ACM, 1994.

25. Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High quality hatching. In *Computer Graphics Forum*, volume 23, pages 421–430. Wiley Online Library, 2004.

8. Appendix A - curvature computation

All quantities are functions of $\mathbf{x}$, which we will omit in the notation for easy of reading.

The Gaussian curvature K is

$$K = -\frac{1}{\|\mathbf{g}\|^4}\begin{vmatrix}\mathbf{H} & \mathbf{g}\\ \mathbf{g}^T & 0\end{vmatrix}. \tag{3}$$

With normal $\mathbf{n} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}$, and Laplacian $\Delta\mathbf{f} = \frac{\partial^2\mathbf{f}}{\partial x^2} + \frac{\partial^2\mathbf{f}}{\partial y^2} + \frac{\partial^2\mathbf{f}}{\partial z^2}$ the mean curvature H is

$$H = \frac{1}{\|\mathbf{g}\|}\left[\mathbf{n}^T\mathbf{H}\mathbf{n} - \Delta\mathbf{f}\right].$$

The principal curvatures are:

$$\kappa_1 = H + \sqrt{(H)^2 - K},$$

$$\kappa_2 = H - \sqrt{(H)^2 - K}.$$

We need to construct the matrix

$$\left(\mathbf{n}\cdot\mathbf{n}^T - \mathbf{I}\right)\mathbf{H} - \mathbf{I}\kappa_1\|\mathbf{g}\|,$$

where $\mathbf{I}$ is the indentity matrix, then take the maximum length one out of the three possible pairwise cross products of its rows. Normalized, it gives principal direction $\mathbf{t}_1$. Then, $\mathbf{t}_2 = \mathbf{t}_1 \times \mathbf{n}$.

Let us introduce the vectors of pure and mixed second-order partial derivatives as

$$\mathbf{p} = \left[\frac{\partial^2\mathbf{f}}{\partial x^2} \quad \frac{\partial^2\mathbf{f}}{\partial y^2} \quad \frac{\partial^2\mathbf{f}}{\partial z^2}\right]^T$$

and

$$\mathbf{m} = \left[\frac{\partial^2\mathbf{f}}{\partial x\partial y} \quad \frac{\partial^2\mathbf{f}}{\partial y\partial z} \quad \frac{\partial^2\mathbf{f}}{\partial z\partial x}\right]^T.$$

With these Hessian is

$$\mathbf{H} = \begin{bmatrix} p_x & m_x & m_z \\ m_x & p_y & m_y \\ m_z & m_y & p_z \end{bmatrix}.$$

Let us introduce the notation for a *swizzle* of a vector $\mathbf{y}$

$$\mathbf{y}_{yxz} = \begin{bmatrix} y_z \\ y_x \\ y_z \end{bmatrix},$$

and similarly for any order of elements. With this the determinant of equation 3 can be computed without explicitly constructing the matrix as

$$\begin{vmatrix}\mathbf{H} & \mathbf{g}\\ \mathbf{g}^T & 0\end{vmatrix} =$$

$$2(\mathbf{p}\circ\mathbf{m}_{yzx})\cdot(\mathbf{g}_{yzx}\circ\mathbf{g}_{zxy})$$

$$-(\mathbf{p}_{zxy}\circ\mathbf{p}_{yzx})\cdot(\mathbf{g}\circ\mathbf{g})$$

$$+(\mathbf{m}\circ\mathbf{m})\cdot(\mathbf{g}_{zxy}\circ\mathbf{g}_{zxy})$$

$$-2(\mathbf{m}_{xzy}\circ\mathbf{m}_{yxz})\cdot(\mathbf{g}_{xzy}\circ\mathbf{g}_{zyx}),$$

where $\circ$ is the Hadamard product operator.

9. Appendix B - gradients and Hessians

We continue using notation form Appendix A. In order to be able to evaluate the curvature forumulae, we need to compute the field function, its gradient, and Hessian. Those are all obtained as the sum of respective functions for metaball atoms. Here we give the formulae for the infinite support *Blinn* and finite support *Wywill* functions. We give all base functions, gradients and Hessians as functions of $\mathbf{d} = \mathbf{x} - \mathbf{a}$, where $\mathbf{a}$ is the atom position. This is to avoid having to subtract $\mathbf{a}$ at every instance of $\mathbf{x}$.

The Blinn base function is:

$$f^{\text{Blinn}}(\mathbf{d}) = \frac{1}{\|\mathbf{d}\|^2}.$$

The gradient is:

$$\mathbf{g}^{\text{Blinn}}(\mathbf{d}) = -\mathbf{d}\frac{2}{\|\mathbf{d}\|^4}.$$

The vector of pure second derivatives $\mathbf{p}(\mathbf{d})$, using $\mathbf{e} = \mathbf{d}\circ\mathbf{d}$ is:

$$\mathbf{p}^{\text{Blinn}}(\mathbf{d}) = \frac{6\mathbf{e} - 2(\mathbf{e}_{yzx} + \mathbf{e}_{zxy})}{\|\mathbf{d}\|^6}.$$

The vector of mixed second derivatives $\mathbf{m}(\mathbf{d})$ is

$$\mathbf{m}^{\text{Blinn}}(\mathbf{d}) = \frac{8\mathbf{d}\circ\mathbf{d}_{yzx}}{\|\mathbf{d}\|^6}.$$

The Wywill base function has finite support. Let R be the effective atom radius, and introduce the shorthand $\delta = \|\mathbf{d}\|/R$. With these, the Wywill base function is:

$$f^{\text{Wywill}}(\mathbf{d}) = \begin{cases} 0 & \text{if } \delta > 1, \\ 1 + \frac{-4\delta^6 + 17\delta^4 - 22\delta^2}{9} & \text{if } \delta \le 1. \end{cases}$$

With

$$G = \frac{4\left(6\delta^4 - 17\delta^2 + 11\right)}{9R^2},$$

the gradient is:

$$\mathbf{g}^{\text{Wywill}}(\mathbf{d}) = \begin{cases} \mathbf{0} & \text{if } \delta > 1, \\ -\mathbf{d}G & \text{if } \delta \le 1. \end{cases}$$

The vector of pure second derivatives $\mathbf{p}(\mathbf{d})$, using $\mathbf{e} = \mathbf{d}\circ\mathbf{d}$ is:

$$\mathbf{p}^{\text{Wywill}}(\mathbf{d}) = \begin{cases} \mathbf{0} & \text{if } \delta > 1, \\ \frac{4\mathbf{e}(17 - 12\delta^2)}{R^4} - \begin{bmatrix} G \\ G \\ G \end{bmatrix} & \text{if } \delta \le 1. \end{cases}$$

The vector of mixed second derivatives $\mathbf{m}(\mathbf{d})$ is:

$$\mathbf{m}^{\text{Wywill}}(\mathbf{d}) = \begin{cases} \mathbf{0} & \text{if } \delta > 1, \\ \mathbf{d}_{xyz} \circ \mathbf{d}_{yzx} \frac{4(12\delta^2 - 17)}{9R^4} & \text{if } \delta \leq 1. \end{cases}$$

10. Appendix C - seed motion explained

Seeds are particles moving along the deforming isosurface. There are three effects that contribute to this motion: fluid motion, field shift, and correction.

10.1. Fluid motion

The fluid medium itself is moving. Its motion is defined for atoms with atom velocities $\mathbf{q}_j$ How we construct the flow velocity at a point from these relies on the requirement that points on the isosurface should remain on the isosurface. How much the linear motion of an atom influences the isosurface depends on the length of the base function gradient at the isosurface point. Thus, linear atom velocities should be weighted with this gradient length to get the flow velocity:

$$\mathbf{v}^{\text{fl}}(\mathbf{s}) = \frac{\sum_{j=0}^{M-1} \|\mathbf{g}_j(\mathbf{s}_k)\| \mathbf{q}_j}{\sum_{j=0}^{M-1} \|\mathbf{g}_j(\mathbf{s}_k)\|}.$$

The seeds need to travel along the isosurface, so the fluid velocity must be projected on it. The component perpendicular to the surface is found as

$$\mathbf{v}_k^{\text{perp}} = \frac{\mathbf{g}(\mathbf{s}_k)\left(\mathbf{v}_k^{\text{fl}} \cdot \mathbf{g}(\mathbf{s}_k)\right)}{\|\mathbf{g}(\mathbf{s}_k)\|^2},$$

and thus the projected fluid velocity is

$$\mathbf{v}_k^{\text{pfl}} = \mathbf{v}_k^{\text{fl}} - \mathbf{v}_k^{\text{perp}} = \mathbf{v}_k^{\text{fl}} - \frac{\mathbf{g}(\mathbf{s}_k)(\mathbf{v}_k^{\text{fl}} \cdot \mathbf{g}(\mathbf{s}_k))}{\|\mathbf{g}(\mathbf{s}_k)\|^2}.$$

10.2. Surface pull

Seeds need to move towards the isosurface either because they are distant due to initial or accumulated error, or because the isosurface itself has moved. For both effects, we will be able to find the desired rate of change in field value at the seed $\delta = \frac{\partial f(\mathbf{s}_k)}{\partial t}$, and need to compute the seed velocity $\mathbf{v}_k^{\text{pull}} = \partial \mathbf{s}_k / \partial t$ from this. We move the seed along the gradient, so $\mathbf{v}_k^{\text{pull}} = \xi \mathbf{g}(\mathbf{s}_k)$ with some ξ. It must be true that

$$\delta = (\xi \mathbf{g}(\mathbf{s}_k)) \cdot \mathbf{g}(\mathbf{s}_k).$$

Solving this for ξ gives

$$\xi = \frac{\delta}{\|\mathbf{g}(\mathbf{s}_k)\|^2},$$

and then

$$\mathbf{v}_k^{\text{pull}} = \frac{\mathbf{g}(\mathbf{s}_k)\delta}{\|\mathbf{g}(\mathbf{s}_k)\|^2}.$$

10.2.1. Correction

As neither the temporal nor the spatial linearizations applied are accurate, the seeds positions would accumulate error and drift away from the isosurface. Also, when initialized, the seeds are are random positions and need to be drawn to the isosurface rapidly. Therefore, a correction term with boldness factor Φ is applied. The boldness factor Φ is the inverse of the time in which the seed is supposed to reach the isosurface. Thus, δ^{corr} is $(L - f(\mathbf{s_k}))\Phi$. However, large Φ values can lead to instabilities near strongly non-linear regions of the field function.

$$\mathbf{v}_k^{\text{corr}} = \frac{\mathbf{g}(\mathbf{s_k})(L - f(\mathbf{s_k}))\Phi}{\|\mathbf{g}(\mathbf{s_k})\|^2}.$$

10.2.2. Field shift

As atoms move, the field value at a $\mathbf{s}_k$ is going to increase or decrease. This change will make the isosurface of L move along the gradient. The rate of change at seed k due to atom j moving is:

$$\delta_j^{\text{shift}} = -\mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j,$$

and the total effect of all atoms is:

$$\delta^{\text{shift}} = -\sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j.$$

This makes the shift velocity:

$$\mathbf{v}_k^{\text{shift}} = -\frac{\mathbf{g}\sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j}{\|\mathbf{g}\|^2}.$$

10.3. Complete seed velocity

All terms, save for the unprojected fluid velocity, contain the gradient divided by its length squared. Their sum can therefore be written as:

$$\mathbf{v}_k = \mathbf{v}_k^{\text{fl}} - \frac{\mathbf{g}(\mathbf{s_k})\left[\mathbf{v}_k^{\text{fl}} \cdot \mathbf{g}(\mathbf{s}_k) + (f(\mathbf{s_k}) - L)\Phi + \sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j\right]}{\|\mathbf{g}(\mathbf{s_k})\|^2}$$

Seventh Hungarian Conference on Computer Graphics and Geometry, Budapest, 2014

Realtime, coherent screen space hatching

Zoltán Lengyel,[1] Tamás Umenhoffer[1] and László Szécsi[1]

[1] Budapest University of Technology and Economics, Department of Control Engineering and Information Technology

Abstract

In this paper we present a screen space hatching algorithm that provides time coherent placing of hatching lines relative to object surfaces. While with screen space techniques we can easily achieve consistent image space hatching density, it is hard to make hatching lines express surface features, and to make them follow the underlying geometry. Drawing individual textured lines can provide high quality results, but their direction and amount of bending should be calculated according to the 3D geometry. We propose a method that combines the illumination gradient with curvature based line direction calculation to support a wide variety of objects. To achieve surface position coherency during animation we use image space velocity maps to move the individual hatch lines. We use rejection sampling and low discrepancy sequences to filter out high density areas where the flow accumulates lines, and to fill in the vacant areas. The multi-pass algorithm is implemented entirely on the GPU using geometry shaders and vertex transform feedback.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

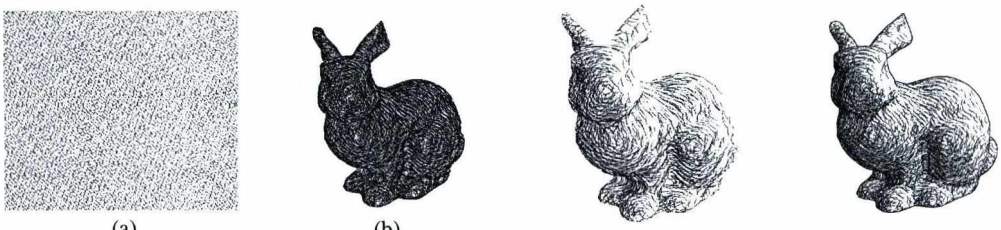
1. Introduction

A wide range of techniques in computer graphics target non-photorealistic (NPR) or illustrative image production, among them the synthesis of hand-drawn art. Like photorealistic techniques, NPR techniques are also based on three dimensional objects, but try to mimic the look of traditional media. The 3D modelling and rendering pipeline has a great efficiency benefit compared to manual image creation, but it is very hard to reproduce the visual look of classic techniques. 3D rendered images have restricted freedom regarding geometry borders, smooth lighting transitions and surface materials. Though several techniques exists that can well mimic special traditional techniques, their applicabilities are limited.

Animation movies focus less on rendering speed but on rendering quality, but this does not mean that render time is not important. Increased rendering time can have serious cost impact, thus NPR techniques made for offline rendering should also keep processing time low. An other important aspect is that preprocessing requirements should also be treated carefully, as in a production environment the modelling, the rendering, and the final compositing is done in separate packages, which have their own specialities, thus sharing data between them can be very hard. Huge preprocessing or per-frame geometry processing is also not feasible for realtime applications.

The biggest limitation of most NPR techniques both for realtime and for production purposes is time coherency. It is hard to ensure that stylized lines, illustrative surface details or lighting features follow the geometry during animation. If these features are fixed in image space it produces an annoying artifact called *shower door* effect, as it seems like the NPR effect is caused by a distorting glass door we are looking through. These artistic features should move with the geometry without any sudden change or flickering.

In this paper we focus on pen and ink illustration and pencil drawings, where the lighting and shadows and the shape of the objects are represented with the density and orientation of thin hatch lines. To mimic the traditional 2D technique we should ensure that hatching line density, line length and width is consistent in screen space. Hatch lines should be placed more densely in shadowed areas, and they should be completely missing in lit areas. Also their orientation and bending should well describe the underlying geometry.



(a) (b) (c) (d)

Figure 1: *Basic elements of the hatching rendering algorithm from left to right: hatch line positions in screen space (a), rotated and bent lines (b), rejected lines according to lighting (c), adding edge detected contours (d).*

2. Previous work

The most obvious way to add artistic look to our objects is to paint an artistic texture for them. This also works for hatching: we can use an image of hatch lines and map this as a repeated texture to our objects. Choosing the detail of this image is not as simple as in case of usual textures. As we move closer to the surface more lines should appear. Consistent image space line density can be achieved with mipmapping, multiple texturing and texture blending[6, 4, 5, 9, 8, 12]. One drawback is that the maximum detail is limited as the number of mipmap levels and texture resolution is also limited. The other problem is that a proper parametrization is needed, and not only to avoid texture seams, but to make the lines follow geometry features. Manual UV layout creation is rather time consuming, and seams can not be eliminated completely. Automatic methods need to calculate principal curvature directions from the geometry and orient smaller hatching patches usually placed in screen space.

The another main group of techniques generates new geometry for hatch lines. Here we distinguish between object space and image space methods. Object space methods place hatching lines directly onto the rendered surface in 3D space [11, 1, 3]. The lines are rendered as polygon strips. Principal curvature directions should also be calculated to make the lines follow the surface. The strength of these methods is that lines are tied onto the surfaces providing straightforward temporal coherence. On the other hand visibility calculation of the lines can cause biasing problems, and it is also hard to ensure uniform distribution of hatching lines in image space. With these techniques lines go through the same transform pipeline as all other rendered polygons.

Hatching lines can also be drawn in image space [2, 10]. These techniques work with uniformly placed hatch lines in screen space. Determining the direction of the lines needs special considerations. The lines should illustrate the underlying surface, thus a proper image space directional field should be created. Different approaches use different quantities to calculate this vector field. They might use the tone gradient of an input image, or use screen space principal curvature direction data. The latter can be calculated in screen space if some necessary information like camera space depth or normal vectors can also be rendered. Principal curvature directions could also be computed from the processed geometry and projected onto the image plane, but this requires geometry processing. Image space methods often suffer from temporal incoherence.

3. Motivation

Our goal is to develop a hatching rendering technique that defines individual lines in screen space, but keeps time coherency so no flickering or sudden change appears. Lines should have an even distribution in screen space, moving closer to objects should make more hatching lines appear. Line direction and curving should well express surface features, they should accurately fit onto the 3D surface. Animation should be supported by moving the lines with the objects in screen space but still maintaining even screen space density.

We should avoid complex geometry processing, moreover we should make our algorithm independent of the geometry, and rely only on standard outputs of common renderers like image buffers. This enables us to implement the algorithm in an interactive post processing framework. As the main goal is to use the technique interactively all calculations should be kept on the GPU.

4. Proposed algorithm

Our technique is based on our previous results[7]. Hatching lines are rendered as individual triangle strips. Even density of hatching lines can be easily achieved with placing the lines randomly on screen using a uniform distribution. Illumination can be depicted with filtering out particles at highlight areas. Figure 1 shows the main components of screen space hatching generation.

Our main contribution to the base work is to support time coherent line animation. Lines are moved according to a velocity map, which distorts the even distribution. To fix the uneven particle density we filter out dense regions and fill in sparse regions. The final steps are to calculate hatching

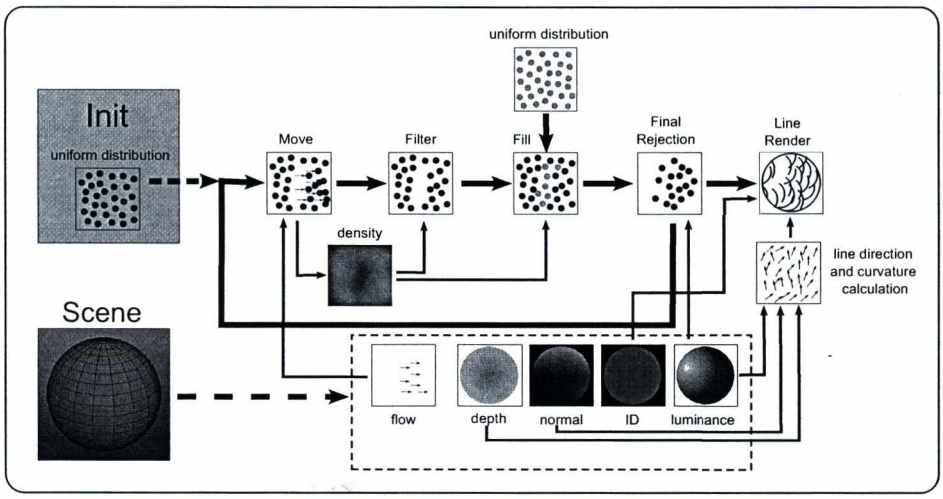


Figure 2: *Basic elements of the proposed algorithm. In initialization phase uniform screen space particle density is generated. In each frame buffers containing normals, depth, illumination, object ID, and screen space velocity are rendered. Particles are advected according to the velocity. The resulting density is not uniform which will be equalized with a filtering process followed by inserting in new random particles. Final screen space density is usually defined by a luminance map, so further particles are rejected. As a last step lines are drawn as textured line primitives.*

direction and blending for the particles and render them as textured line strips. The final particles after the fill and filter process will be the input of the next rendered frame. The inputs required by the algorithm are several buffers rendered in every frame. These buffers store data that is common both in realtime and in production environments. They are the illumination, the depth, normal, ID and velocity buffers. Figure 2 shows the main components of the proposed algorithm.

Our second contribution over our earlier work[7] is that we propose an automatic hatching direction calculation that combines several surface features. The following subsections cover the main components of the algorithm in detail.

4.1. Hatching particle generation

The first step is to define a set of particle positions (seed points) in screen space, at which final lines will be rendered. These particles should have a uniform distribution to evenly cover the image plane, and they should show no recognizable pattern to mimic the random behaviour of hand drawn hatch lines. This can be easily achieved with placing the lines randomly on screen using a uniform distribution. The artist-defined global density can be achieved simply with fewer or more random samples on screen.

Random particle generation is the initialization phase of our algorithm, where we create a huge buffer of random positions using Halton sequences. The size of this buffer is much bigger than the desired line number count, as later, during animation more and more random positions will be requested. If the buffer runs out, we start from the beginning. For each particle we also store an additional random number from the unit interval which we call priority. Priorities should also have a uniform distribution in image space. This can be easily achieved using low-discrepancy sequences like the Halton sequence, using the normalized sequence number as priority, but as we use only a part of the buffer and need priorities from the whole unit interval, it was easier to define an additional random sequence for priorities, too. For smoother animation, we can also store additional line data for each particle like line length, width, direction or curvature.

4.2. Particle movement

As the camera or any objects in the scene move, we should ensure that the seed positions will move with the surface. This is crucial to avoid the unwanted shower door effect. As we are working in image space, the most obvious solution is to use a screen space velocity map, or in other words: an optical flow. As our algorithm is based on 3D scenes we can produce this map quite easily and accurately. We can use the camera matrices and the world matrices for each object from the previous frame, transform the vertex positions with both the new and the previous transformations, from which movement can be calculated with a simple vector difference.

Using this flow map particles are advected. After moving particles, some will remain in their original positions, some will move to different image positions and some will fall

outside the screen. For most of the movements this results in an uneven particle distribution.

4.3. Filtering dense areas

After particle movement we should restore the even artist-defined particle density. Our first step is to filter out too dense areas. This filtering means that we should keep a particle only if drawing that particle will not make its local neighbourhood too dense. Refreshing density during each particle draw is not an option in a real-time environment as dependency between the rendering of each line can make parallel hardware implementation impossible. To overcome this we should make the rejection of a single line dependent only on a local desirable density but not on the influence of previously rendered lines. To achieve this we use the theory of rejection sampling.

The classic rejection sampling problem is when we would like to generate samples according to a desired distribution, but we can not use the inversion method. With rejection sampling we choose a well known distribution which is easy to sample and upper bounds the desired distribution with an appropriate scale. The simplest distribution to use is the uniform distribution. If we assign a random priority to each uniformly distributed sample and reject the sample if its priority is above the (scaled) desired density, the remaining samples will have the desired distribution.

On our specific case the desired distribution is a uniform distribution and our particles have an uneven distribution, the sample density of which locally exceeds the desired uniform distribution. To filter out the particles we should know the density in the neighbourhood of each particle. To do this we render a small disk shaped snippet at each particle with a linear falloff and blend them together with additive blending. The result is an approximate density function. The radius of the snippets and their power defines the locality and smoothness of the density map, these parameters are set empirically. Note that these parameters should depend on the resolution and the number of desired hatching lines. After parameter tuning we used the following expressions which worked well for different resolutions and line numbers:

$$\text{snippet\_size} = \sqrt{\frac{\text{number\_of\_pixels}}{\text{desired\_line\_number}}} * \frac{6.5}{\text{window\_resolution}},$$
$$\text{snippet\_power} = 0.04 * \text{linearFalloff}().$$

With the density map available we examine each particle and reject them if their priority scaled with the underlying density is above the desired uniform density. As priorities are uniformly distributed, lines will be rejected uniformly, thus the resulting density in high density areas will also match the original uniform distribution. After this filtering step particle priorities should also be mapped back to the unit interval, as high priority particles were filtered out. This will ensure the uniform priority distribution again.

4.4. Refilling sparse areas

After the filtering process we still need to fill in vacant areas. To do this we take an other set of random samples and try to fill the image with them. Here we also use the former density map and keep particles only if their priority is above the underlying density value. Thus at coarser areas we keep more particles, and where we already reached the desired density we reject all new particles. Again, particle priorities should be mapped back to the unit interval. The new particles are placed at the end of the former filtered particle buffer. At this stage we again have a uniform particle position and priority distribution in image space, but particles are moving with the surfaces wherever it is possible, which provides the illusion of them being defined in object space.

4.5. Illumination

Hatching density also depicts current lighting conditions, thus illuminated image regions should have coarser hatching density than areas in shadows. Here again we face the classic problem of rejection sampling. Our desired distribution is the illumination value, and we have a uniform distribution to reject samples from. Each particle, whose priority is below the inverted luminance value will be rejected. At this step particles are not removed, only their rendering is skipped, thus they will stay in the particle buffer which will be the input of the next frame.

We can make animation smoother if we do not let lines disappear and appear suddenly due to luminance change or flow filtering. We can store the line length for each particle, decrease this length if the particle is marked for removal and remove it only if its length reached zero. The same can be used for new lines: they start with a small line length and in each frame their length will be increased. For smooth luminance change we do not even need to store previous length values, they can be calculated on the fly by applying a smoothstep function centred at the rejection priority cutoff value. Note that the original solution is equivalent to a step function at the inverted luminance value.

4.6. Line direction and bending calculation

Before final rendering the lines, we need to define their direction and amount of bending. These features should be chosen in a way that they describe the underlying geometry well, and mimic the way an artist would orient them. Due to our experiments presented in our previous paper[7], we can say that no single feature exists that can be used for all types of geometry (see figure 3). On the other hand, principal curvature directions are commonly used in NPR techniques to orient lines, and artist also find it as a natural orientation.

Principal curvature can be calculated from the curvature tensor which is the Hessian of the depth field. As surface normals describe the depth gradient the Hessian matrix can

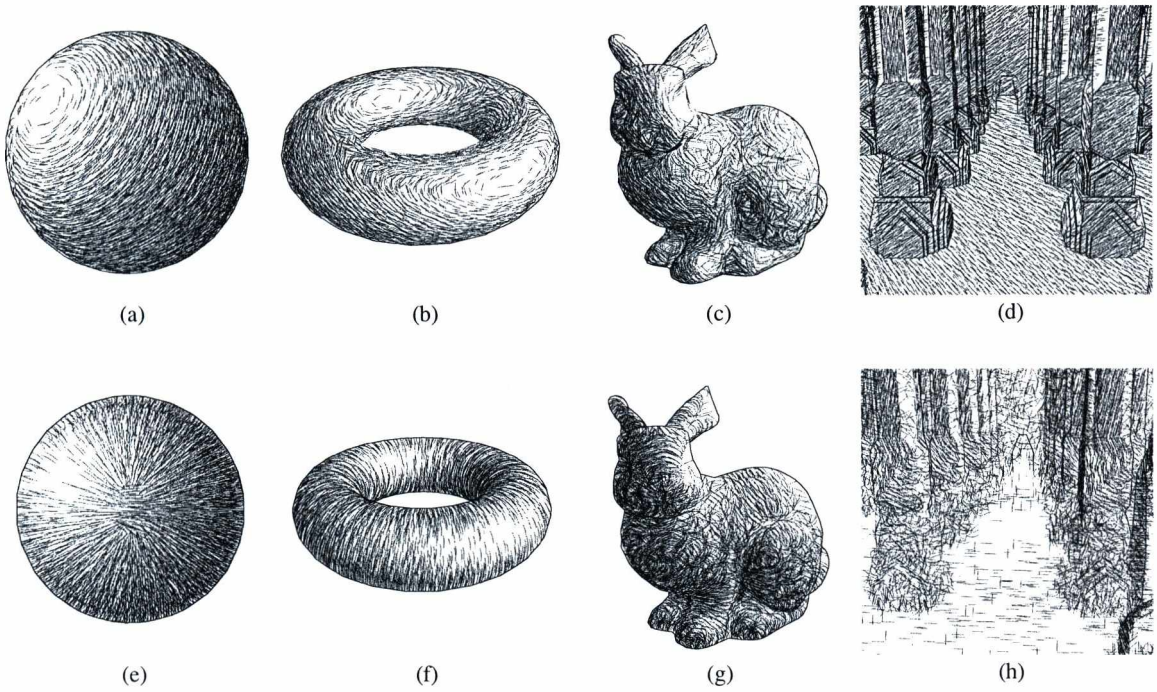


Figure 3: *Hatching using luminance (top) and principal curvature features (bottom). Different geometry types need different approaches to represent main surface features.*

be defined in terms of the directional derivatives of the surface normal:

$$H = \begin{pmatrix} \frac{\partial \vec{n}}{\partial \vec{u}} \cdot \vec{u} & \frac{\partial \vec{n}}{\partial \vec{v}} \cdot \vec{u} \\ \frac{\partial \vec{n}}{\partial \vec{u}} \cdot \vec{v} & \frac{\partial \vec{n}}{\partial \vec{v}} \cdot \vec{v} \end{pmatrix}$$

where $\vec{u}$ and $\vec{v}$ are orthonormal tangent vectors on the surface. In our case, they are the projections of the screen space unit $\vec{x}$ and $\vec{y}$ vectors to the surface. In other words, the Hessian is the screen space directional derivatives of the screen space normal vectors. The gradient is computed with a Sobel filter and its value is compensated with the projected length of the surface normal. The eigenvalues and eigenvectors of this matrix define the maximum and minimum normal curvature values and their corresponding directions, thus the principal curvature and principal curvature direction.

We should note that using the normal vectors as first order depth gradients will lead to clearly visible tessellation in the second order features, as interpolated normal vectors only linearly approximate those of a smooth surface. In some cases this is does not produce obvious artifacts as lines are not placed too densely to make these sudden changes conspicuous. On the other hand when moving closer to the surface the relative density gets higher, thus these triangle borders will be visible. We also found that during the animation of the particles, some of them can randomly move across of one of these borders, which makes flickering orientation changes. To overcome these problems we used an edge preserving smoothing filter on the normal vectors before derivation.

In our tests we found that the luminance gradient is also a good feature that can describe the geometry well, and can handle some special cases that principal curvature directions cannot. These cases include surfaces with no curvature like flat geometry, or with no principal curvature like a sphere. On the other hand some geometries, like a torus, can be better described with principal curvature directions (see figure 3).

To combine the advantage of both methods we calculate both features and use the luminance gradient based line direction where prineipal curvatures are uncertain. These are the cases when the maximal and the minimal curvature equals, and where no principal curvature can be calculated. Figure 4 shows our automatic feature selection technique on different types of geometries.

4.7. Final render

After line direction and amount of bending is calculated lines are drawn as curved and textured triangle strips with hardware blending enabled. The lines can be drawn directly to screen with orthographic projection, as image space samples do not need any 3D transformations, nor visibility testing.

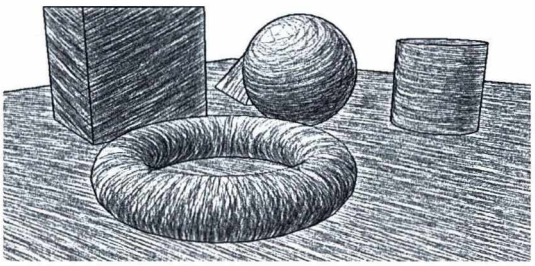


Figure 4: *Line direction and curving calculated on a per line decision between luminance and principal curvature features.*

Additionally we can clip lines on a per pixel basis using an ID map to prevent them from crossing object borders. This behaviour is useful in case of longer lines, but shorter lines can even pass these borders without visible artifacts, which makes the final rendering even more natural and hand drawn like.

5. Implementation

We implemented the hatching synthesis algorithm in a standalone application using OpenGL and GLSL shaders. Geometry buffers like depth and normal maps are rendered in each frame, and the proposed algorithm uses them as input. Note that this implementation can also be extended to support input from an image stream rendered with an offline renderer. Image processing of the buffer inputs like gradient calculation, principal curvature calculation or blurring was implemented with full screen quad fragment shaders and render to texture.

The main workflow of the algorithm is to process a dynamic array of particle data. This data processing can be implemented with geometry shaders. The input of the shaders are point primitives storing the particle data. A geometry shader instance processes one particle and alters this data if necessary, or it can completely reject it. The output of the geometry shader is also a point primitive, which can be sent to the rasterizer, or in our specific case it can be fed back to another buffer on the GPU without actual drawing. Output feedback can be directed to a specific position in the GPU buffer, so merging of two buffers is also possible (this is needed in the particle refill phase). The number of lines that was written to the buffers can be queried with the OpenGL API, so we can always know the currently valid line count in our buffers.

To get the current density map, a geometry shader extrudes small quads from each particle, and the pixel intensities are calculated in the fragment shader. During final line rendering a geometry shader creates bended line strips from the particle points and sends them to the rasterizer. Our implementation keeps all calculation and necessary data on the GPU from input data processing to final display.

6. Results

As our algorithm works completely in screen space the performance does not depend on the rendered geometry, only on screen resolution. We should note that creating the input buffers needs additional scene rendering passes, thus the combined performance is influenced by geometry complexity after all. This can be eased with the use of multiple render targets to output all necessary information in a single render pass. The main limitation factors are line count and screen resolution. We found that creating the density map is a critical part of the algorithm, as because of the blending a serious amount of pixel overdraw is present. This can be eased with choosing a smaller snippet radius, which makes the density estimation more local and bit more noisy. On the other hand using more lines does not increase the computational cost of this density rendering phase as more lines results in smaller snippet size which reduces the number of fragments processed (see the second column in table 1).

Choosing the snippet size rightly also influences line flickering as we can not create a completely uniform density map with drawing snippets, thus even without particle movement some areas will be filtered, while others will be filled with extra particles. This results in continuous disappearance and rebirth of particles even within a static environment which is an unwanted effect. To handle this we introduced a threshold range around the desired density. Within this range we treat the area as having the desired density. This threshold value together with density snippet size and intensity should be fine tuned to get good results.

The other aspect that greatly decreased the performance is the edge preserving blurring we applied to the buffers before taking their gradient (see the third column in table 1). The cost of this this step depends on the blur kernel and the screen resolution. If little flickering is not a problem, this step can be skipped.

7. Conclusion

We presented a real-time hatching rendering algorithm that randomly places textured hatch lines on the image plane. Lines are advected according to screen space velocity of the surfaces. Uneven image space particle density caused by particle movement is equalized using rejection sampling methods. Lines are rendered as textured line strips to the screen. The resulting algorithm eliminates the time coherency problem of screen space methods (see figure 5) but can keep their advantages like no visibility test is needed, uniform screen space particle density is maintained and the performance is independent of geometry complexity. Hatching

Number of lines	FPS without filtering	FPS with filtering
10000	195	80
20000	190	80
40000	180	75
100000	145	60
200000	100	50

Table 1: *Performance tests on a Geforce GTX 480 with 1280x720 screen resolution.*

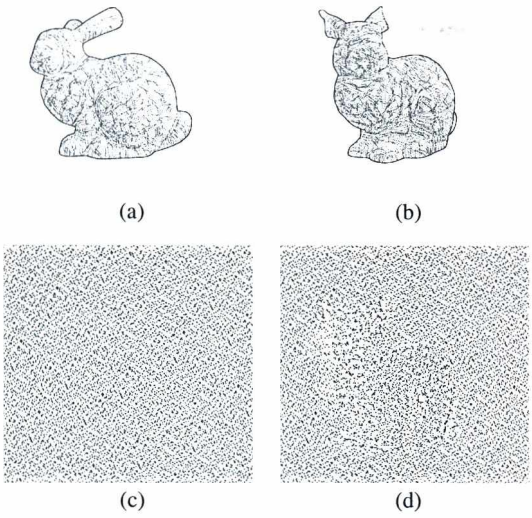


Figure 5: *Frames from an animated sequence. On the left, the initial frame and its corresponding particle distribution is shown. On the right, a later frame and its distribution is shown after rotating the camera. Uniform distribution is well preserved during animation. Some particle accumulation still occurs at object borders, but this is not visible in the final image.*

lines can have a wide variety of styles by adjusting line density, width, length, maximal bending and applying artistic textures. Our GPU implementation provides real-time performance in high resolution environments.

Acknowledgements

This work has been supported by OTKA PD-104710 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

1. Gershon Elber. Interactive line art rendering of freeform surfaces. *Comput. Graph. Forum*, 18(3):1–12, 1999.
2. Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *PROCEEDINGS OF SIGGRAPH 2000*, pages 517–526, 2000.
3. Matthew Kaplan, Bruce Gooch, and Elaine Cohen. Interactive artistic rendering. In *Non-Photorealistic Animation and Rendering 2000 (NPAR '00)*, Annecy, France, June 5-7,2000.
4. Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. Line-art illustration of dynamic and specular surfaces. *ACM Transactions on Graphics (SIGGRAPH ASIA 2008)*, 27(5), December 2008.
5. Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, NPAR '00, pages 13–20, New York, NY, USA, 2000. ACM.
6. Hyunjun Lee, Sungtae Kwon, and Seungyong Lee. Real-time pencil rendering. In Douglas DeCarlo and Lee Markosian, editors, *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 37–45. ACM, 2006.
7. Zoltán Lengyel, Tamás Umenhoffer, and László Szécsi. Screen space features for real-time hatching synthesis. In *Proceedings of the 9th conference of the Hungarian Association for Image Processing and Pattern Recognition*, KEPAF '13, pages 82–94, 2013.
8. Afonso Paiva, Emilio Vital Brazil, Fabiano Petronetto, and Mario Costa Sousa. Fluid-based hatching for tone mapping in line illustrations. *Vis. Comput.*, 25(5-7):519–527, April 2009.
9. Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *In Proceedings of SIGGRAPH 2001*, pages 579–584. ACM Press, 2001.
10. Mìchael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive penâĂŞandâĂŞink illustration. In *In Proceedings of SIGGRAPH âĂŹ94*, pages 101–108, 1994.
11. Tamás Umenhoffer, László Szécsi, and László Szirmay-Kalos. Hatching for motion picture production. *Comput. Graph. Forum*, 30(2):533–542, 2011.
12. Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High quality hatching. *Comput. Graph. Forum*, 23(3):421–430, 2004.

Improving Texture-based NPR

László Szécsi

Marcell Szirányi

Tamás Umenhoffer

Budapest University of Technology and Economics, Budapest, Hungary

Abstract

This paper presents a real-time procedural texturing algorithm for hatching parametrized surfaces. We expand on the concept of Tonal Art Maps *to define self-similar, procedural tonal art maps that can service any required level-of-detail, allowing to zoom in on surfaces indefinitely. We explore the mathematical requirements arising for hatching placement and propose algorithms for the generation of the procedural models and for real-time texturing.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

Photo-realism has been in the focus of rendering systems for decades. Photo-realistic rendering aims at creating images that are indistinguishable from real-world photographs, which is made possible by the precise simulation of physics laws during the rendering process. How accurately physics is applied in the rendering algorithm determines the level of realism of the result.

Computer graphics also tries to mimic artistic expression and illustration styles[7, 16, 18]. Such methods are usually vaguely classified as *non photo-realistic rendering* (NPR). While the fundamentals of photo-realistic rendering are in optics that are well understood, NPR systems simulate artistic behavior that is not mathematically founded and often seems to be unpredictable. Therefore, the first step of NPR is to model the artist establishing a mathematical model describing his style, and then solve this model with the computer. The result will be acceptable if our model is close to the not formally specified artistic behavior. During the history of NPR, many individual styles were addressed. Hatching is one of the basic artistic techniques that is often emulated in stylistic animation.

Hatching strokes should appear hand-drawn, with roughly similar image-space width, dictated by pencil or brush size, but they should also stick to surfaces to provide proper object space shape and motion cues. Both properties must be maintained in an animation, without introducing temporal artifacts. We call these the requirements of image space and world space consistency. The two requirements are in contradiction when the relation of image space surface to object space surface is being altered, i.e. when surface distance or viewing angle is changing. The rendering process should resolve this contradiction while presenting natural randomness inherent in manual work[10, 1].

This paper presents a hatching style NPR rendering algorithm that can be implemented in real-time. The main scientific contributions are

- the introduction of *self-similar tonal art maps*,
- an algorithm for generation of *self-similar seed sets*,
- a single pass, real-time hatching algorithm using the seed sets, with automatic, procedural, continuous level-of-detail (Figure 1) .

The organization of the paper is as follows. In Section 2 we summarize the related previous work on NPR. Section 3 introduces the idea of Self-similar Procedural Tonal Art Maps. In Section 4, we derive the mathematical construct for the placement of hatching strokes that meets the self-similarity requirements. We discuss the interpretation of the model in Section 5, including the level-of-detail scheme and stroke sizing. Tone representation is added in Section 6. A

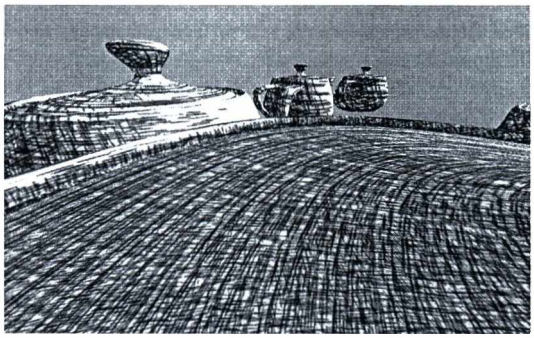


Figure 1: *Teapots rendered with the same shader and settings, featuring different levels of detail.*

detailed description of the final algorithm, and the discussion of results and future work conclude the paper.

2. Previous work

2.1. Density and direction fields

In pencil drawings artists convey the shape and illumination of objects with the density, orientation, length, width and shade of thin hatching strokes[23, 9]. To mimic this, we should find a *density* and a *direction field* in the image plane that is as close as possible to what an artist would use. The density, length, width and shade should be influenced by the current illumination, while the orientation is determined by the underlying geometry. Artists may use several layers of strokes, aligned at different angles to the direction field. If we defined the direction field directly in the image plane, it would be difficult to convey 3D shape or motion. Thus, it is beneficial to determine these directions on object surfaces either from geometric curvatures[6] or from the tone[13]. The *principal curvature directions*[6] of the surface define a field that is an intuitive representation of shape information.

In this paper, we do not address the problem of the curvature field generation, but assume that a proper UV parametrization is already known for surfaces, where isoparametric curves follow desired hatching directions. Proposing an algorithm tailored to specifically for our approach is left for future research.

2.2. Uniformly spaced and random hatching styles

In hand-made artwork, the artist may choose to pay attention to the distance of strokes and keep similar distance between them. The method of Zander[25] integrates the vector field to produce strokes also guaranteeing that they are separated which provides very high quality still images. Unfortunately, this kind of spatial control of strokes is in contradiction with temporal coherence. We have to accept that not all hatching styles are created equal regarding their applicability to animation. Specifically, to reproduce a given shade, we cannot apply strokes that are placed at strictly identical, fixed image space distances, and follow object space motion at the same time. When zooming in or changing the viewing angle, either one of the conditions must be broken. Violating object space consistency will cause the *shower door effect*, meaning the user has the impression that strokes are not fixed to objects but are floating on them. If we allow the regular hatching distance to change, coverage will deviate from the desired shade. The continuous level-of-detail feature of hatching is absent from these techniques.

Counterpointing the above considerations against overly regular hatching, it is also important that strokes are distanced evenly in a statistical sense, and that the distribution of distances between them does not depend on the direction. As strokes make the shading anisotropic, the distribution of stroke locations itself should not only be uniform but also isotropic. Otherwise, superimposed the two will result in direction-dependent patterns, where strokes clump together for unfortunate orientations.

2.3. Image space methods

To guarantee consistency with the image, hatching strokes can be generated directly in image space[11]. In order to avoid the shower door effect, strokes can be moved along with an optical flow or image space velocity field, but placing new strokes on emerging, previously non-visible surfaces still poses problems. Especially if strokes are long, following curvature or feature curves of object surfaces, they should maintain this even when only tiny fractions have become visible. This cannot be assured when only using image space information.

2.4. Seed-based methods

Several works[14, 20] proposed the application of *particles* or *seeds* attached to objects, but extruding them to hatching strokes in image space. Strokes are obtained by integrating the direction vector field started at seed points or particles[25, 15]. The key problem in these methods is the generation of the world-space seed distribution corresponding to the desired image-space hatching density. This either means seed killing and fissioning[24]—even using mesh subdivision and simplification[4]—, or rejection sampling[20]. These techniques are mostly real-time, but require multiple passes and considerable resources. Compositing hatching strokes with three-dimensional geometry is not straightforward: depth testing of extruded hatch curves against triangle mesh objects must be using heavy bias and smooth rejection to avoid flickering. While this allows for modeling some human inconsistencies in performing the same hidden line removal task, it is also extremely ponderous to eliminate them should they not be desired.

In our work, we use the notion of seeds to discuss hatching stroke position and distribution patterns. However, we are not concerned by seed positions in object space, rather in parameter space, and we do not extrude seeds to triangle strip geometry.

2.5. Texture-based methods

In order to make the pencil strokes consistent with the 3D objects, hatches can be generated into textures and mapped onto animated objects[12]. This requires surface parametrization. Unfortunately, this approach does not provide natural pencil art, where each stroke is drawn in image space. There, every stroke has roughly similar width which is determined by the pencil of the artist and is independent of the distance or the viewing angle of the depicted surface. From another point of view, the fact that pencil art is naturally produced in image space implicitly assumes a level-of-detail mechanism, which renders objects with fewer pencil strokes if they are farther away and thus cover just a smaller portion of the image[5].

The most characteristic limitation of texturing-based hatching approaches is limited level-of-detail support. Simple static textures perform extremely poorly, as the width of hatching strokes is fixed in UV space, and—through the UV mapping—also in object space. Note that changing the UV mapping depending on viewing distance is not only insufficient to address projection distortion, but also would result in the hatching pattern floating on the surface.

Figure 2: *A Tonal Art Map with the nesting property. Strokes in one image appear in all the images to the right and down from it. From Praun et. al*[16].

Thus, simple texturing does not allow for hatching that is uniform in screen space. A level-of-detail mechanism called *Tonal Art Maps* has been proposed to alleviate this problem[16]. Using this technique, several texture images are pre-drawn, representing different tones and hatching scales. Figure 2, taken from the referred paper, shows such a set of maps. When rendering surfaces, the appropriate texture in every pixel can be selected based on the desired tone and on-screen hatching stroke width. In order to avoid sharply clipped hatching strokes at boundaries of discrete zones, the patterns fade into each other using interpolation.

In an animation, as the required hatching density is changing, it is important that strokes stay at their on-surface positions. It is allowed for new hatching strokes to appear when the density increases, and for existing strokes to vanish, should the density decrease. However, strokes should not be appearing and vanishing in the same vicinity at the same time. Therefore, denser hatching textures should always contain the strokes of sparser textures as a subset. This is called the *nesting property*, also observable in Figure 2.

Tonal Art Maps, however, only support a range of hatching scales as defined by the most detailed and least detailed map levels. Thus, when zooming in onto a surface, we cannot have finer hatching than what texturing with the most detailed map level would produce, resulting in classic texture magnification artifacts, and huge and sparse hatching strokes in image space. Also, there is a trade-off between the number of detail levels used and the quality of transitions between those levels. With too few textures, a large number of strokes fade in at the same time, resulting in an image with non-uniform stroke weights. While this is acceptable in most cases, as weaker pencil strokes can possibly be used by artists, it is a limitation to the degree of screen-space uniformity we can achieve.

3. Self-similar Tonal Art Maps

Our idea is to make Tonal Art Maps infinitely loopable, by making the nesting property recursive. Hatching strokes are positioned at *seeds*. The texture is broken into four tiles, forming a 2×2 grid. In all four quarters, the seeds must be the subset of the complete seed set scaled down to fit the quarter. That way the seeds are nested in the pattern we get by repeating them twice along both axes (see Figure 3). Thus, when we zoom in to any of the quarters, it is possible to add new strokes re-creating the original most detailed hatching pattern, where the process can be restarted (see Figure 4). This is the *recursive nesting property* of the seed set, which we will define more formally in Section 4. Following the classic Tonal Art Map scheme, this would require four sequences of images, describing how the individual quarters evolve into the full hatching pattern. However, we will never actually create these images, but describe them procedurally, and use this extremely compact representation for rendering.

First, in Section 4, we deal with the problem of placing the hatching strokes to assure the recursive nesting property. We start by considering seed point placement only, addressing hatching stroke length and width in Section 5.

4. Self-similar seed generation

We need a set of seed points to place strokes at. Let the set of all seed points be $S = \{\mathbf{s}_0, \ldots, \mathbf{s}_i, \ldots, \mathbf{s}_{N-1}\}$, where $\mathbf{s}_i = (s_{iu}, s_{iv})$. These positions are defined in the *seed space*, the relation of which to the UV space we explore in Section 5.

Let us define the operator $\mathfrak{D}$ as follows:

$$\mathfrak{D}\mathbf{s} = (\mathrm{frac}\,(2s_u), \mathrm{frac}\,(2s_v)),$$

where the frac function yields the fractional part of a number.

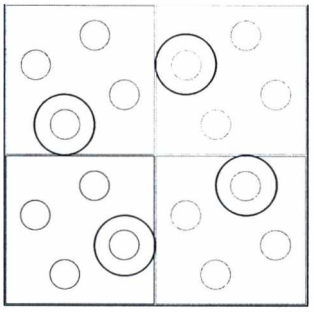


Figure 3: *The smallest possible seed set with the recursive nesting property. The four large circles indicate seeds, small circles are the seed pattern repeated on a 2×2 grid.*

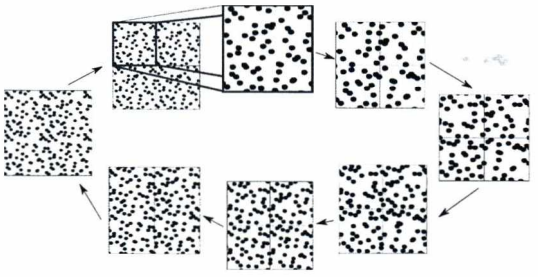


Figure 4: *The recursive nesting property ensures that by adding seeds to any of the quarters we can reproduce the original seed pattern. Seed markers decrease in size, but seed positions are unchanged.*

Geometrically, this operation is a scaling by the factor of two, using the nearest corner of the unit square as the pivot point (Figure 5). Numerically, if we consider the binary radix fraction form of the seed coordinates, the operation removes the first binary digit after the binary radix point, shifting the rest to the left (Figure 6). Note that in the unit square, there is always only a zero on the left side of the radix point.

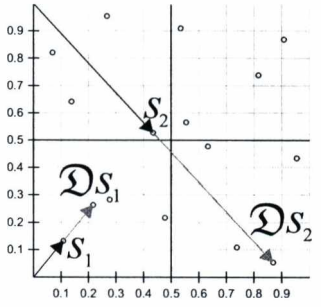


Figure 5: *Geometric interpretation of operator $\mathfrak{D}$. Seeds are projected to double their distance from the nearest corner.*

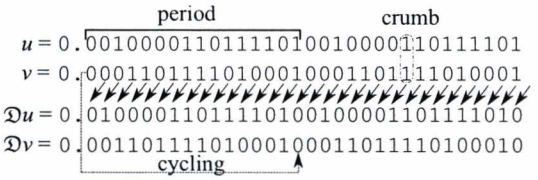


Figure 6: *Interpretation of operator $\mathfrak{D}$ on binary fractions. For periodic fractions, the bits can be cycled.*

The operator can be extended to sets as:

$$\mathfrak{D}S = \{\mathfrak{D}\mathbf{s} \mid \mathbf{s} \in S\}.$$

Set S has the recursive nesting property if

$$\mathfrak{D}S \subseteq S.$$

This means that if there is a seed $\mathbf{s}_i$, then its image $\mathfrak{D}\mathbf{s}_i$ must also be a seed. A sequence of N seeds then must be found as

$$\mathbf{s}_{i+1} = \mathfrak{D}\mathbf{s}_i, \text{ if } 0 \leq i < N,$$

because all seeds will trivially be mapped to another seed. However, $\mathfrak{D}\mathbf{s}_{N-1}$ must also be in S. This can be true if

$$\mathbf{s}_0 = \mathfrak{D}\mathbf{s}_{N-1}.$$

Considering the interpretation of $\mathfrak{D}$ on binary fractions, we can conclude that seeds must be generated by shifting the bit patterns of coordinates s_{0u} and s_{0v}, and after N steps we need to arrive back at $\mathbf{s}_0$. This is possible if s_{0u} and s_{0v} are periodic binary fractions of period N (or a divisor of N). Such periodic binary fractions are easy to generate e.g. using the Bernoulli(1/2) process, finding individual bits as independent coin flips. Even though such a process generates points that are evenly distributed in the statistical sense, a concrete small set of seeds generated in such a way could be not filling the space evenly.

In an infinite random sequence, we expect any fixed-length subsequence to appear with exactly the same probability. Extending that to finite sequences, we expect possible fixed-length subsequences to appear with frequencies as uniform as possible when cycling through the sequence. Bit sequences with such a property are called *uniform cycles*[17]. Although no proof for the existence of arbitrary-length uniform cycles is known, all possible uniform cycles can be enumerated for modest lengths. The bit sequence for the u and v coordinates could be found independently, but then coincidentally identical substrings could appear. Instead, we can combine their respective bits to form *crumbs* (quaternary equivalent of binary bits or decimal digits), forming a quaternary periodic fraction. To distribute points evenly, the crumbs of one period must form a quaternary uniform cycle.

In order to understand what uniformity means in the geometrical sense, let us find the intuitive meanings of the crumb values. Recall that the first crumb in the quaternary representation of a seed point is the combination of the first bits of

the binary fractions for its coordinates. Thus, the value of the first crumb indicates in which quarter of the unit square the seed point is. Then the second crumb indicates in which $\frac{1}{4} \times \frac{1}{4}$ square it is within the quarter, and so on. As we generate our seeds by cycling the crumb pattern, the number of times a subpattern of some length shows up is exactly the number of seeds in the corresponding squarelet. If $N = K^4$ with some K integer, then exactly one seed will fall in every cell of a $K^2 \times K^2$ grid (see Figure 7). This is very similar to the elemental interval property of the low discrepancy Halton sequence[8, 19].

Another obvious connection is that with *iterated function systems* (IFS)[2], and the *chaos game* method of generating their attractors[3]. Just like our construct, the chaos game transforms an initial point repeatedly. The transformation is randomly picked from a set each time. However, if the transformations map the attractor to disjunct areas, then it can be unambiguously determined for a point which the last transformation was. Then, starting with a point, the sequence can be traced back deterministically. In fact, the randomness is all encoded into the choice of the initial point. In our case, we are playing this deterministic version of the chaos game with four transformations, each mapping the unit square to one of its quarters. The attractor of this system is the unit square itself. We only take care that the sequence returns to itself, and thus we can work with a finite number of seeds. As for the crumb pattern, every crumb value there indicates one of the four transformations picked in the chaos game. If we use a uniform cycle, than all transformations appear the same number of times, and this is true for all sequences of transformations of given length, too.

Unfortunately, no polynomial-time algorithm is known for generating binary or quaternary uniform cycles of arbitrary length. In fact, we know of no proof that those exist for any N. However, in practice it is possible to find cycles of modest length by enumerating a set of required *snippets* and performing a brute force search over their permutations until a valid uniform cycle is found[17]. For $N = 16$, this can even be done manually, arranging the snippets 00, 01, 02, 03, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33, yielding e.g. the quaternary sequence 0012202332131103. Note that all crumbs appear four times, and all snippets of two crumbs only once.

However, manual arranging for greater $N = K^4$ would be extremely ponderous, so it is required to define a suitable algorithm, based on the following considerations. Each snippet is a sequence of length K:

$$p_i = \{\pi_{i1}, ..., \pi_{iK}\}$$

Let $G = (P, E)$ be a directed graph, where $P = \{p_1, ..., p_N\}$ is the set of snippets, and $E = \{(p_i, p_j) \mid (\pi_{i2} = \pi_{j1}), ..., (\pi_{iK} = \pi_{j(K-1)})\}$. Thus, an edge connects two snippets if we get one from the other by dropping the first crumb and appending another one. A valid uniform cycle can be found by searching for a proper Hamiltonian cycle in the graph G. Starting with any vertex, the brute force method picks an edge randomly to an available position (see Figure 8), marking the current vertex as expended, and proceeding to the vertex along the selected edge. If there is no directed edge to an available vertex, the algorithm steps back to a previous state. Otherwise, we proceed similarly until a Hamiltonian cycle is found.

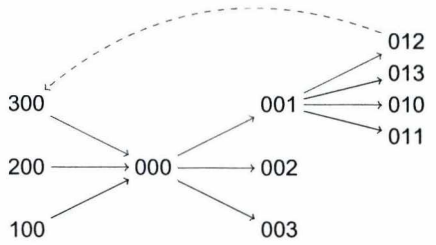


Figure 8: *A part of the graph for $N = 64$. We are looking for a Hamiltonian cycle.*

Algorithm 1 Quaternary uniform cycle generation

```
1:  function UNIFORM(set of snippets P, sequence  )
2:      if |  | ≠ N then                          ▷ sequence incomplete
3:          ρ ← random crumb from (0,1,3,4)
4:          for δ ← ρ, ρ+3 (mod 4) do            ▷ all continuations
5:              p ← (q_{1−K}, q_{2−K}, ..., q_{−1}, δ)   ▷ form snippet
6:              if p ∈ P then                    ▷ snippet available
7:                  P' ← P \ p                   ▷ expend snippet
8:                   ' ←  ‖ δ                    ▷ append to
9:                   ˇ ← UNIFORM(P',  ')         ▷ continue
10:                 if ˇ ≠ ∅ then
11:                     return ˇ                 ▷ success
12:                 end if
13:             end if
14:             return ∅          ▷ nothing worked, fail branch
15:         end for
16:     else                                     ▷ sequence complete
17:         for i ← 0, K − 1 do                  ▷ check wrapping snippets
18:             p ← (q_{i−(K−1)}, q_{i−(K−2)}, ..., q_i)
19:             if p ∉ P then
20:                 return ∅                     ▷ mismatch, reject
21:             end if
22:             P ← P \ p
23:         end for
24:         return                               ▷ accept
25:     end if
26: end function
```

The formal algorithm (Algorithm 1) builds a growing sequence $= (q_0, q_1, \ldots, q_{-2}, q_{-1})$ of crumbs ultimately forming a quaternary uniform cycle. Note that the indices in are understood (mod| |), where | | is the length of . The algorithm tests, for all possible continuations of ,

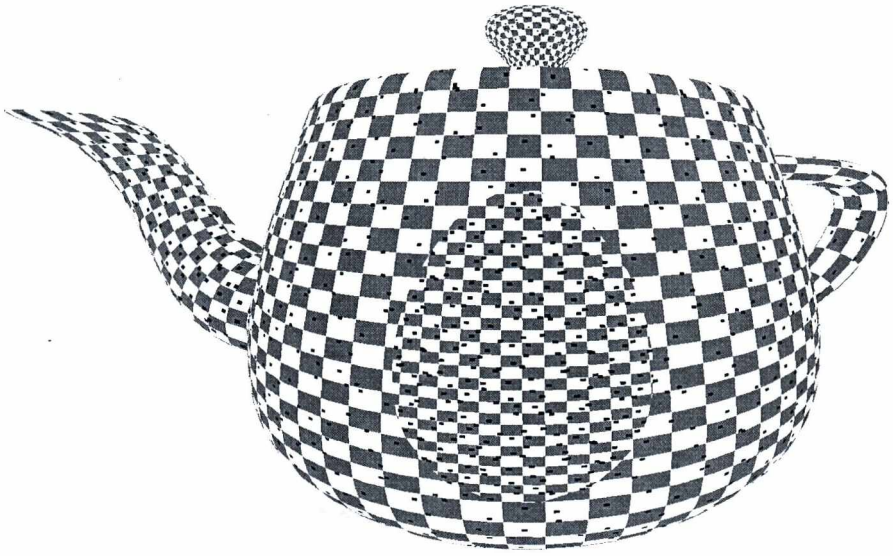


Figure 7: *Seeds generated by a uniform cycle are uniformly distributed in the sense that one seed falls in every grid cell.*

whether the resulting snippet is still available in set P. If it is, we expend the snippet from P and append the continuation to . When is complete, it is verified that the additional snippets generated by cycling the sequence are identical to those remaining in P. If they are, we have found a cycle containing all snippets once, and only once.

We can obtain a uniform cycle specifying the position for an initial seed by calling UNIFORM with an initial sequence of a random snippet (e.g. 000) and a snippet set P with all possible snippets, save for the initial one in . Having obtained the crumbs of the initial seed, we get further seeds by repeatedly applying operator $\mathfrak{D}$ on these numbers, cycling the crumbs within the period.

5. Seed and stroke scale

Although, as evidenced by Algorithm 2 in Section 7, scaling seeds and strokes to match the required level of detail is fairly simple in practice, its rigorous discussion is quite involved. The process is analogous to that of *mipmapping*[22], the difference being that in our case all levels are identical, but scaled by powers of two. Thus some additional considerations for recursion and scaling between detail levels are needed.

As the scale of mapping UV texture coordinates to the viewport is changing, to preserve the same on-screen density, the hatching detail must be smoothly increased or decreased. Shall more detail be required, the original seed pattern should be repeated in all four quarters, continuing recursively until the desired seed density is achieved. To get lower density, the square should be regarded as a quarter of a bigger square, and only the relevant subset of the seeds used, also repeated recursively. Thus, from the mapping scale, a *nesting level* $\lfloor M \rfloor$ needs to be found, so that we know the scale at which the generated seeds can be interpreted as meaningful UV space positions.

Let $\mathcal{L}$ be the mapping of seeds to texture coordinates.

$$\mathbf{u} = \mathcal{L}\mathbf{s},$$

where $\mathbf{s}$ is a seed space position and $\mathbf{u}$ the texture coordinates. Locally, $\mathcal{L}$ must be an isotropic scaling. Globally, the scaling factor will vary for different points on the object surface, but it may change only by powers of two, to allow the self-similar nesting to work. For simplicity, we postulate that the scaling factor must always be a power of two. We show later in this section that this is without the loss of generality. Our objective is to find the scaling factor for any given surface point. In this discussion, we are looking for an arbitrary scaling factor first, and then select a power of two that approximates it.

Let $\mathcal{T}$ be the inverse of the texture mapping operator, defined by the model parametrization. Thus,

$$\mathbf{x} = \mathcal{T}\mathbf{u},$$

where $\mathbf{x}$ is the model space position.

Let $\mathcal{G}$ be the complete model-to-viewport-space mapping of the image synthesis pipeline, including the model, view, projection, and viewport transformations, all defined by ob-

ject and camera setup. Thus,

$$\mathbf{v} = \mathcal{G}\mathbf{x},$$

where $\mathbf{v}$ is the viewport space position.

With these, the transformation from seed space to the viewport can be written as

$$\mathbf{v} = \mathcal{G}\mathcal{T}\mathcal{L}\mathbf{s}. \tag{1}$$

Note that all operations are dependent on the surface point, and some may be non-linear.

Let $\mathbf{h}$ be the *detail direction*, a differential direction vector in the seed space. If strokes are isotropic, e.g. only dots, then the choice is arbitrary. Typically, however, strokes have a dominant direction. While scaling in this dominant direction only influences stroke length, scaling perpendicularly has significant impact on hatching density. The detail vector should align with this perpendicular direction.

What we are interested in is how the detail direction is scaled by the mappings $\mathcal{G}$, $\mathcal{T}$, and $\mathcal{L}$. Therefore, let us introduce the notation

$$\text{stretch}(\mathcal{O}, \mathbf{d}) = \frac{|J_{\mathcal{O}}\mathbf{d}|}{|\mathbf{d}|},$$

where $\mathcal{O}$ is a mapping, $\mathbf{d}$ is a direction vector, and $J_{\mathcal{O}}$ the Jacobian matrix of mapping $\mathcal{O}$ at the surface point in question. Then the scaling factors exercised on direction $\mathbf{h}$ by the mappings can be written as:

$$L = \text{stretch}(\mathcal{L}, \mathbf{h}),$$

$$T = \text{stretch}(\mathcal{T}, J_{\mathcal{L}}\mathbf{h}),$$

$$G = \text{stretch}(\mathcal{G}, J_{\mathcal{T}}J_{\mathcal{L}}\mathbf{h}).$$

With these, Equation 1 can be linearized and applied to differentials, yielding the formula for the scaling of the detail direction as

$$|\mathbf{h}_{\text{vp}}| = GTL|\mathbf{h}|,$$

where $|\mathbf{h}_{\text{vp}}|$ is the length of the detail direction vector as it appears in the viewport. The geometry factor G and texture distortion factor T can be computed easily, and L is the scaling that should be introduced by the choice of the detail level.

The ratio $F = |\mathbf{h}|/|\mathbf{h}_{\text{vp}}|$ captures how densely seeds appear in the viewport. This is a free artistic parameter, and a global constant, as we do not consider density modulation for tone yet. As with regular texture mapping, choosing a lower value of F results in more detail—more seeds, thus more hatching strokes per unit area—, but also more repetition as the texture coordinates wrap around. With this, the desired detail factor L^{-1} can be expressed as

$$L^{-1} = GTF,$$

where all factors are known. Note that if L contained an additional constant scaling factor, it would have the same effect as F here, so introducing another free parameter would be superfluous. Our postulation that $\mathcal{L}$ is a scaling with a power of two was without the loss of generality.

To make use of the nesting property, $\mathcal{L}$ should be a scaling with a power of two, thus $L \approx 2^{\lfloor M \rfloor}$, where we call integer $\lfloor M \rfloor$ the *nesting level*. We first compute the real number M form L^{-1} as

$$M = \log_2 L = -\log_2 L^{-1} = -\log_2 GTF,$$

then take the integer part to get the nesting level $\lfloor M \rfloor$. This gives us the scaling factor between seed space and texture coordinates as

$$\mathbf{u} = 2^{\lfloor M \rfloor}\mathbf{s}.$$

In practice, it is the texture coordinates of a surface point that are known when shading is performed, and the seed space coordinates need to be computed as

$$\mathbf{s} = 2^{-\lfloor M \rfloor}\mathbf{u}.$$

The solution is exact if M is an integer, and integer values define *detail levels*. For non-integer values of M, the *dense level* $\lfloor M \rfloor$ has to transition into the *sparse level* $\lfloor M \rfloor + 1$ smoothly. Therefore, we need to scale strokes appropriately and fade out those that are not visible in the sparse level (Figures 9 and 10).

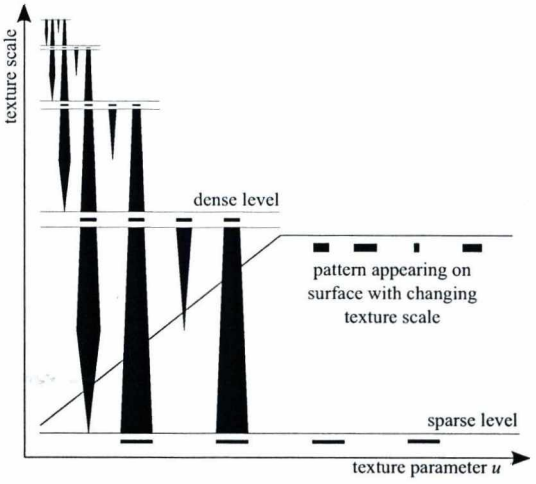


Figure 9: *2D depiction of smooth transition between detail levels by stroke width modulation.*

The stroke width and length in viewport space are artistic parameters, expressed as the two-dimensional vector $\mathbf{e}$. By the definition of F, we know that the seed space stroke size should be $F\mathbf{e}$, if L were not quantized to powers of two. In

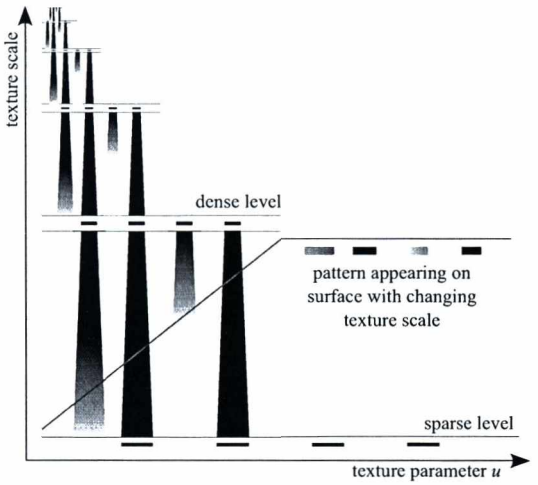


Figure 10: *2D depiction of smooth transition between detail levels by stroke opacity modulation.*

order to compensate for the quantization, we need to scale seed space stroke sizes by

$$\frac{2^M}{2^{\lfloor M \rfloor}} = 2^{M-\lfloor M \rfloor} = 2^{\mathrm{frac}(M)} = 2^m,$$

where m can be seen as an interpolation factor between nesting levels, going from 0 at the dense level to 1 at the sparse level. Intuitively, if the dense and sparse levels are identical but for a factor of two, the strokes need to grow to twice their size as the nesting level increases.

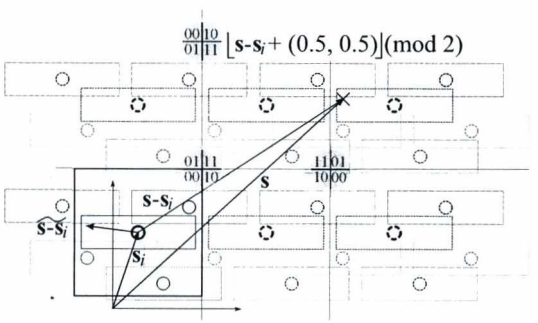


Figure 11: *Computation of stroke space position (without scaling or rotation) and sparse level quarter indicator bits. X marks the shaded point. Its seed space position is* $\mathbf{s}$*, the seed processed is* $\mathbf{s}_i$*, the position relative to the seed is* $\mathbf{s}-\mathbf{s}_i$*, which is mapped to the unit square surrounding the seed to get* $\widetilde{\mathbf{s}-\mathbf{s}_i}$*.*

For every seed $\mathbf{s}_i$, we need to find *stroke space coordinates* $\mathbf{z}_i$ corresponding to seed space position $\mathbf{s}$. To cover the complete seed space, the seed pattern is repeated indefinitely in unit tiles. Strokes extend beyond tile boundaries (see Figure 11). Stroke extents in seed space must to be less than one, so that they do not overlap with themselves. The position of shaded point $\mathbf{s}$ relative to seed position is $\mathbf{s}-\mathbf{s}_i$, but this contains the offset of the tile. We introduce the following notation

$$\widetilde{(s_u, s_v)} = (\mathrm{frac}(s_u + 0.5) - 0.5, \mathrm{frac}(s_v + 0.5) - 0.5)$$

for wrapping the space to the origin-centered unit square. With this, the stroke coordinates are

$$\mathbf{z}_i = \mathbf{R}\widetilde{\mathbf{s}-\mathbf{s}_i} \oslash (2^m F\mathbf{e}),$$

with $\oslash$ standing for the elementwise division, and $\mathbf{R}$ is a rotation matrix in two dimensions for cross-hatching stroke alignment. Stroke coordinates can be used to access the stroke texture. Contributions for all seeds must be composited.

To be able to tell whether a dense seed is present on the sparse level, we need to know which quarter of the sparse level we are in, and whether the seed appears in that quarter. The parity bits of the tile's row and column indices indicate the quarter. Which tile we are is thus exactly identified by the integer part discarded with $\widetilde{\mathbf{s}-\mathbf{s}_i}$. This can be computed as

$$\mathbf{w} = \lfloor \mathbf{s} - \mathbf{s}_i + (0.5, 0.5) \rfloor.$$

Recall that the sparse level has the same seeds as the dense level, scaled up by a factor of two. A dense level seed that also appears in the sparse level must therefore be a scaled-up image of a seed in the dense level. This is true for every dense level seed that was generated by the $\mathfrak{D}$ operator from a seed in the respective quarter. To see what the predecessor of a seed was, we need to check the final bits in the cycled bit pattern. If those are identical to the parity bits of $\mathbf{w}$, then the seeds exits on both levels. Others have to be faded out as m increases.

Fading strokes out can be accomplished in several ways. We can use alpha blending or stroke width modulation, and we can fade out all strokes simultaneously or one after the other, as the detail decreases. Modulating all strokes simultaneously allows them to blend smoothly, without abruptly appearing or disappearing strokes, but a large number of strokes will be semi-transparent or intermediate-sized, not achieving uniform hatching. If strokes appear one after the other, the method of modulation hardly matters, as they appear more abruptly, but the image space consistency is better. Note that because of the self similar property, hatching strokes only appear or disappear only when hatching needs to grow denser or sparser, and no flickering is present.

6. Tone

In order to convey illumination, we need to be able to modulate hatching density depending on the locally desired shade

or tone. The challenge is to preserve the quality of hatching. In our scheme, seeds cannot be removed without breaking the recursive nesting property. The number of seeds could be decreased by decreasing the length of the quaternary uniform cycle, with the overall pattern remaining consistent and most seeds changing positions only slightly, but there would inevitably be seeds that have very different positions. Thus, the only option remains to use several sets of self-similar seeds, and overlay them. Fortunately, this is in perfect agreement with artistic practice[9], where about four distinct tones are rendered by overlaying strokes, usually at an angle, which is known as *cross-hatching*.

Thus, four seed sets are generated, and the contributions of a seed set are added if the desired local tone is darker than an associated threshold. However, to avoid clipping strokes at tone segment boundaries, this transition also has to be smooth. Again, strokes can be faded out together, producing a smoother animation, but with a lot of semi-transparent strokes, or one after the other, producing more abruptly appearing strokes of more consistent appearance.

7. Implementation

Once proper seed sets have been generated, the algorithm can be implemented in a single shader (Algorithm 2). The quaternary uniform cycles $\{b_1,\dots,b_4\}$ defining the seed sets, rotation matrices for cross-hatching alignment $\{\mathbf{R}_1,\dots,\mathbf{R}_4\}$, the global, non-modulated seed density F, and stroke size $\mathbf{e}$ are uniform global inputs. The smoothstep function $\text{sstep}_{[\nu,\mu]}(\lambda)$ clamps and normalizes λ to $[\nu,\mu]$, then performs a Hermite interpolation. The shader psuedocode presented here uses four hatching layers for tone. It fades strokes one after the other, using opacity modulation, both for tone and detail interpolation. The contributions of strokes are composited with the alpha blending logic, but within the shader. The stroke texture sampler must return zero alpha for out-of-range texture coordinates.

It has to be noted that this implementation uses $4N$ texture samples. With a seed set of 64 elements, this is 256 samples per pixel. Although these are samples from the same, presumably small stroke texture, this is still a brute force approach that can be improved if we filter strokes by proximity.

Algorithm 2 Shading a surface point

```
1: function SHADE(texture coords u, position x)
2:     {b_1,...,b_4} ← uniform crumb cycles
3:     F ← global seed density
4:     e ← stroke size
5:     {R_1,...,R_4} ← cross-hatching rotations
6:     a ← tone from illumination in [0,5]
7:     c ← 1                                          ▷ paper color
8:     for j ← 1,4 do                                 ▷ for all tone layers
9:         T ← texture distortion at x
10:        G ← geometry factor at x
11:        M ← −log_2 GTF                             ▷ detail factor
12:        s ← 2^{−⌊M⌋} u                             ▷ seed space
13:        m ← M − ⌊M⌋
14:        for i ← 0,N−1 do                           ▷ for all seeds
15:            s_i ← 0. ‖ b_j ‖ b_j ‖ ...             ▷ seed from crumbs
16:            α ← sstep_{[i/(N+1), (i+1)/(N+1)]}(a − j)   ▷ tone fade
17:            w ← ⌊s − s_i + (0.5,0.5)⌋              ▷ sparse quarter
18:            if w ≢ b_j (mod 2) then                ▷ not in sparse
19:                α ← α sstep_{[i/(N+1), (i+1)/(N+1)]}(m)   ▷ detail fade
20:            end if
21:            z_i ← R(s − s_i) ⊘ (2^m F e)           ▷ stroke space
22:            y ← strokeTex[z_i]                     ▷ sample from texture
23:            α ← α y_α                              ▷ texture alpha
24:            c ← (1 − α)c + α y                     ▷ alpha blending
25:            b_j ← b_j ↻ 1                          ▷ circular shift
26:        end for
27:    end for
28:    return c
29: end function
```

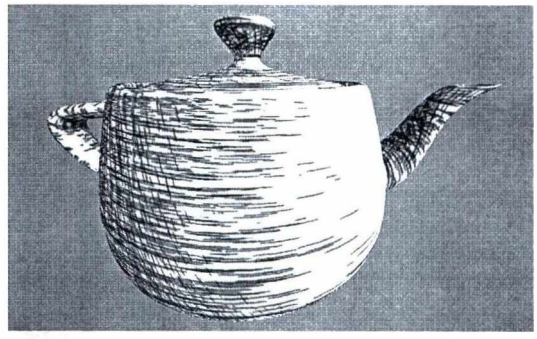


Figure 12: *Teapot rendered with* 4 *tone layers,* 64 *seeds per layer, at 20 FPS,* 1920×1200.

8. Results

We have tested the algorithm on an NVIDIA GeForce GTX 780, with 1920×1200 full-screen resolution. With four tone layers and 64 seeds per layer (Figure 12), we achieved an interactive performance of 20 frames per second. With 16 seeds (Figure 13), the performance went up to 80 FPS, confirming our expectation that rendering time is linearly proportional to the number of seeds.

We examined the options of fading in strokes between levels and tone layers simultaneously of one after the other. Simultaneous fading (Figure 14) always resulted in thin or semi-transparent strokes appearing in conspicuous patterns. Fading strokes individually (Figure 15) resulted in just the occasional stroke being in a transient state briefly, acceptably simulating the stroke appearing as a result of an artistic process. The overall image consistency was in this case perfect. With strokes fading in individually, whether we used

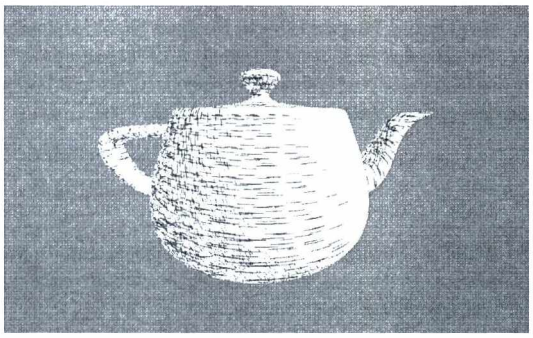


Figure 13: *Teapot rendered with* 4 *tone layers,* 16 *seeds per layer, at 80 FPS,* 1920×1200.

Figure 14: *Teapot rendered with simultaneous stroke fading.*

opacity or line width weighting for fading did not cause a significant visual difference. This was due to the fact that in any method based on local shading like ours, detail level can change strongly along a stroke. Some parts of the stroke will therefore be completely visible and others completely invisible, and the transition is both spatially and temporally confined.

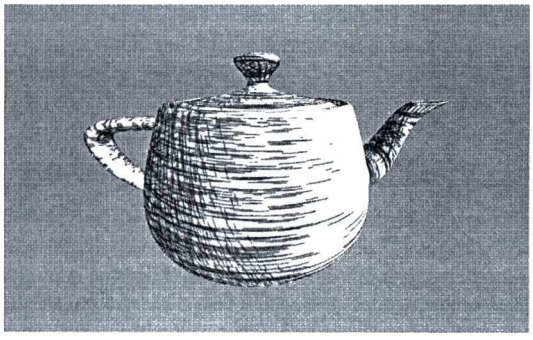


Figure 15: *Teapot rendered with individual stroke fading.*

In animation, the object space coherence, as expected with a parameter space approach, was impeccable. Flickering was not observed. The pattern in which the strokes appear and disappear when zooming in or out is conceivably random.

9. Future work

Our method could be significantly accelerated by not considering all seeds in every pixel. For this, we need to create a texture representing the unit square in seed space, where every texel contains a list of those strokes that may overlap with the texel. This can simply be pre-generated for any seed be rendering all strokes at their maximum size into an S-buffer[21]. We expect that with this addition the proposed method will be barely slower that simple texturing.

When a surface is visible at an integer detail factor, all strokes are completely visible. Otherwise, some may be in a transient state fading in. This difference is unnoticeable when strokes fade individually, but we conjecture that this is the reason why simultaneous fading does not produce unacceptable results. Therefore, we would like to introduce theory for mixed-weight detail levels, where integer level patterns are indistinguishable from interpolated ones. We also believe this will lead us to the implementation of a new artistic parameter that allows intermediate strategies between simultaneous and individual fading. In this overlapped fading model, a customizable, but fixed percentage of strokes will be in transitional state at any time and detail level.

We also plan to examine interaction with outline rendering approaches, and investigate whether we can simulate overdraw with screen space filtering.

10. Acknowledgements

This work has been supported by OTKA PD-104710 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

1. Zainab AlMeraj, Brian Wyvill, Tobias Isenberg, Amy A Gooch, and Richard Guy. Automatically mimicking unique hand-drawn pencil lines. *Computers & Graphics*, 33(4):496–508, 2009.

2. Michael F Barnsley and Stephen Demko. Iterated function systems and the global construction of fractals. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 399(1817):243–275, 1985.

3. Michael F Barnsley and Andrew Vince. The chaos game on a general iterated function system. *Ergodic Theory and Dynamical Systems*, 31(04):1073–1079, 2011.

4. Derek Cornish, Andrea Rowan, and David Luebke. View-dependent particles for interactive non-photorealistic rendering. In *Graphics interface*, volume 1, pages 151–158, 2001.

5. Oliver Deussen, Stefan Hiller, Cornelius Van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. In *Computer Graphics Forum*, volume 19, pages 41–50. Wiley Online Library, 2000.

6. Ahna Girshick, Victoria Interrante, Steven Haker, and Todd Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 43–52. ACM, 2000.

7. Paul Haeberli. Paint by numbers: Abstract image representations. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 207–214. ACM, 1990.

8. John H Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

9. A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

10. Pierre-Marc Jodoin, Emric Epstein, Martin Granger-Piché, and Victor Ostromoukhov. Hatching by example: a statistical approach. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 29–36. ACM, 2002.

11. Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. Line-art illustration of dynamic and specular surfaces. In *ACM Transactions on Graphics (TOG)*, volume 27, page 156. ACM, 2008.

12. Hyunjun Lee, Sungtae Kwon, and Seungyong Lee. Real-time pencil rendering. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 37–45. ACM, 2006.

13. Yunjin Lee, Lee Markosian, Seungyong Lee, and John F Hughes. Line drawings via abstracted shading. In *ACM Transactions on Graphics (TOG)*, volume 26, page 18. ACM, 2007.

14. Barbara J Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484. ACM, 1996.

15. Afonso Paiva, Emilio Vital Brazil, Fabiano Petronetto, and Mario Costa Sousa. Fluid-based hatching for tone mapping in line illustrations. *The Visual Computer*, 25(5-7):519–527, 2009.

16. Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581. ACM, 2001.

17. D.J. Spitzner. Searchable randomness. Technical Report 09-01, Department of Statistics, University of Virginia, January 2009.

18. Thomas Strothotte and Stefan Schlechtweg. *Non-photorealistic computer graphics: modeling, rendering, and animation*. Elsevier, 2002.

19. L. Szirmay-Kalos. *Monte-Carlo Methods in Global Illumination — Photo-realistic Rendering with Randomization*. VDM, Verlag Dr. Müller, Saarbrücken, 2008.

20. Tamás Umenhoffer, László Szécsi, and László Szirmay-Kalos. Hatching for motion picture production. In *Computer Graphics Forum*, volume 30, pages 533–542. Wiley Online Library, 2011.

21. Andreas A Vasilakis and Ioannis Fudos. S-buffer: Sparsity-aware multi-fragment rendering. In *Eurographics 2012-Short Papers*, pages 101–104. The Eurographics Association, 2012.

22. Lance Williams. Pyramidal parametrics. In *ACM Siggraph Computer Graphics*, volume 17, pages 1–11. ACM, 1983.

23. Georges Winkenbach and David H Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM, 1994.

24. Andrew P Witkin and Paul S Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 269–277. ACM, 1994.

25. Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High quality hatching. In *Computer Graphics Forum*, volume 23, pages 421–430. Wiley Online Library, 2004.

Mass-Spring Models for Anisotropic Diffusion

Márton J. Tóth and Balázs Csébfalvi

Budapest University of Technology and Economics, Department of Control Engineering and Information Technology

Abstract

This paper introduces a new mechanical model for image processing. A mass-spring model with a special load method is applied for noise reduction and edge preserving smoothing. The energy gained from the load is distributed in the system in an anisotropic way through springs. The achieved results are illustrated with several examples and a comparison is made with the anisotropic diffusion in some images.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: noise reduction, edge preserving smoothing

1. Introduction

Noise reduction is a very common and important task in the field of image processing. This task occurs not just in medical imaging [1] [13] but practically in any task where images are handled. To solve this problem several algorithms were developed and it is still an active field of research. Probably one of the most efficient methods is the anisotropic diffusion [7]. It has a very good capability in noise reduction and edge preserving smoothing. This has motivated our research. The objective of this paper is to introduce a mechanical model for edge preserving smoothing and noise reduction. A mass-spring model is created form the image and a load method is proposed that enables the identification of the large scale structures in the image while the small noisy regions are blurred. This work is preliminary and there are a lot of open questions, but the the method has a clear potential as illustrated in this paper.

2. Previous Work

The idea of the nonlinear anisotropic diffusion was originally introduced by Perona and Malik in [7] and [8]. Their technique aims to remove noise from the image without removing important parts of its content. To interpret an image usually lines and edges are important. Therefore they have suggested three criteria that a smoothing algorithm has to fulfill:

- *Causality* - which means no spurious detail should be added to the image while processing it
- *Immediate localization* - the region boundaries should be sharp and coincide with the semantically meaningful borders of the image
- *Piecewise smoothing* - which means that intra-region smoothing should be preferred over inter-region smoothing

In order to achieve the desired filtering, a non-linear approach should be used. This means that the filtering kernel must vary from pixel to pixel depending on its neighborhood qualities. This requires to define the edge property somehow. Usually the gradient of the brightness function is used for this purpose.

$$E(x,y) = \nabla I(x,y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \tag{1}$$

Equation (1) defines the edge property, E, of a pixel and defines it as the gradient of the image brightness.

To control the smoothing, Perona and Malik have created a function w, that controls blurring intensity according to $\|E\|$. This w function has to be monotonously decreasing as lower edge membership value requires higher blurring rate and higher edge membership requires less blurring. In this way it can be guaranteed that intra-region smoothing is more powerful than inter-region smoothing. They have introduced

two types of w functions:

$$w(\|E\|) = e^{-((\|E\|/K)^2)}, \tag{2}$$

$$w(\|E\|) = \frac{1}{1+(\frac{\|E\|}{K})^2}. \tag{3}$$

The results of these two functions are different. The first one (2) prefers high-contrast edges over low-contrast ones while the second one (3) prefers wide regions over smaller ones.

However the above described smoothing can achieve good results but the edges usually stay rough. This is because of the way as the smoothing is done. Controlling only the intensity of the blurring makes the smoothing isotropic. To achieve an anisotropic smoothing the shape of the smoothing kernel should be modified. This can be done by taking into consideration the direction of local image gradient (1). A dynamic kernel can be contracted along the direction of the normal ending in an elliptical kernel.

The idea of anisotropic diffusion has inspired many researchers. Gui et al. [5] have used it for segmentation of synthetic-aperture radar (SAR) images. Fernández et al. [3] and Frangakis et al. [4] have used anisotropic diffusion for noise removal in medical images. Márta and Szirmay-Kalos [6] have applied it to enhance the accuracy of the reconstruction process of positron emission tomography. The behavior of the anisotropic diffusion has been examined by You et al. in [14] and an extension to multivalued images has been created by Sapiro et al. in [11].

Mass-spring models are also widely used in image processing, in computer graphics and in visualization tasks. Szirmay-Kalos in [12] has used it to display dynamic graphs, Dornheim et al. [2] have used mass-spring models to segment neck lymph nodes in CT datasets and it has been used to simulate cloth behavior by Provot in [10].

The idea to connect mass-spring models with anisotropic diffusion has been introduced by Pollak et al. in [9]. They have used their model to enhance edges in SAR images and to segment different regions in the images. They have considered the pixels of the image as mass points and have connected them with springs. During the simulation of the movements of the mass points, if two points started to move together (they kept within a small distance) the spring between them was replaced with a rigid connection. In this way the two points could not separate anymore and formed a new small segment. This principle was applied several times during the simulation and points were connected into small segments and small segments were joined into larger segments. By applying this method they have successfully segmented SAR images and they have proved the noise reduction ability of their algorithm.

3. Our Method

As we indicated in the introduction, the aim of this paper is to present a mass-spring model for noise reduction and edge preserving smoothing. To define such model we have to introduce some notations first. Let $\mathbf{u}$ refer to a discretized signal, i.e an N-point discrete sequence $\mathbf{u} = (u_1 \dots u_N) \in \mathbb{R}^N$ or an N-by-N image where $\mathbf{u} \in \mathbb{R}^{N^2}$. For the sake of simplicity we consider now $\mathbf{u}$ as a N-dimensional vector. To each u_i element of $\mathbf{u}$ we assign a mass point p_i with the position $\vec{r}_i$ and with the mass m_i. From these $\vec{r}_i$ and m_i values we construct the $\mathbf{r} = (\vec{r}_1 \dots \vec{r}_N)$ and $\mathbf{m} = (m_1 \dots m_N) \in \mathbb{R}^N$ vectors. To connect the points springs are used. The spring s_i connects the points p_i and p_{i+1}. The layout of the mass points and their connections are illustrated in Figure 1. Each point is connected to the previous one and to the next one, except the first one which is connected only to the second one and the last one which is connected only to the last but one.

The force exerted by spring s_i can be defined as:

$$\vec{F}_{s_i} = d \cdot \Delta\vec{l}, \tag{4}$$

where d is the stiffness of the spring and $\Delta\vec{l}$ is the extension. The correct setting of d is really important to achieve the expected result. High value of d means that the connection is rigid and small difference in the positions of the neighboring points results in a relatively large force exertion. This effect can be useful to maintain the integrity of homogeneous regions and to maximize intra-region smoothing. Low value of d means that the connection is soft and even a large extension of the spring results only in small exerted force. This setting for d is useful at the borders of the regions as it allows a larger difference in the positions of the neighboring points. This grants a small inter-region smoothing effect.

To find a way to determine the exact value of d is out of the scope of this paper, but it can be said that the value of d should be in inverse ratio to $|u_i - u_{i+1}|$. The exact value depends on the image, on its intensity values and on the desired smoothing effect. In this work the user could define the value by using a graphical function editor tool.

The movement of the particles are restricted to N vertical lines as it is illustrated in Figure 1.

To simulate the movements of the defined system, the following equations should be evaluated:

$$\vec{r}_i(t) = \vec{r}_i(t-1) + \vec{v}_i(t) \cdot \Delta t, \tag{5}$$

$$\vec{v}_i(t) = \vec{v}_i(t-1) + \vec{a}_i(t) \cdot \Delta t, \tag{6}$$

$$\vec{a}_i(t) = \vec{F}_i(t)/m_i, \tag{7}$$

where $\vec{r}_i(0) = 0$, $m_i \in \mathbf{m}$ and $\mathbf{m} = \mathbf{u}$. The definition of $\vec{F}_i(t)$ is the following:

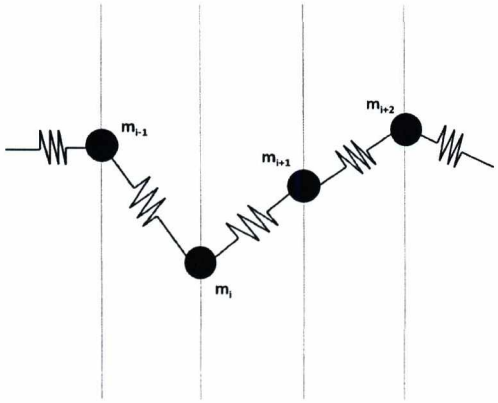


Figure 1: *Connections of mass points.*

$$\vec{F}_i(t) = \vec{F}_{s_{i-1}}(t) + \vec{F}_{s_i}(t) + m_i \cdot \vec{g} - \\ -c \cdot |\vec{v}_i(t-1)| \cdot \vec{v}_i(t-1), \tag{8}$$

where $\vec{F}_{s_{i-1}}$ is the force exerted by the spring s_{i-1}, $\vec{F}_{s_i}$ is the force exerted by the spring s_i and $\vec{g}$ is the acceleration of gravity. The last term of the equation is the drag and c is the drag coefficient. The gravity generates energy in the system, the drag dissipates it and forces exerted by the springs distribute the energy among the neighboring particles. To make this distribution anisotropic the following considerations are done. If two particles have the same mass, which means that the original u_i and u_{i+1} values are similar, the particles move together and there will not be any significant difference in their positions during the simulation. This means that the spring between them transfers no energy as its length does not change. If the particles do not have the same mass, their positions will show a significant difference thanks to the drag and the spring will transfer some energy. To achieve noise reduction and edge preserve smoothing the energy transfer should be limited somehow. There are multiple possibilities to do this. The maximal energy transfer capabilities of the springs can be limited according to the mass difference of the particles or one can define rules to break the springs between the particles. We have tested both options. To limit the energy transfer a spring maximal transfer capability is limited. If this limit is reached the spring transfers the maximal allowed amount of energy and no more. This limit can be configured by the user. The other alternative is cutting the energy transfer completely. During the simulation the springs with the highest energy transfer are marked and after a simulation step the marked springs get broken. The broken springs cannot transfer energy anymore. In this way the energy transfer is progressively terminated between the particles.

4. Results

The above described system distributes energy between the particles in an anisotropic way as the connections between the particles are getting terminated during the simulation or at least the maximal energy transfer is limited. To demonstrate the usability of our system several tests were run. Table 1 shows the results using an axial slice of a brain MRI image. During the simulation, the connections between the particles were progressively terminated. The first image is the original one and the following images shows the excursion values of the mass points during the simulation at different time stamps. The images are derived from the positions of the particles. As one can see at $t = 30$ there are no big differences in the positions of the particles as each $\vec{r}_i(0)$ was initialized to 0. As the simulation continued the differences were getting larger and thanks to the termination of the connections the edges were getting sharper. At the end of the simulation a smoothed image was obtained. As one can observe the intra-region smoothing was dominant while the inter-region smoothing was unremarkable. Table 2 shows the same image and simulation but in this case the connections were not terminated only the maximal energy transfer was limited. The achieved results are similar to the previous case but the inter-region smoothing effect is more significant. This is the result of the continuous energy transfer. The connections between the particles were not terminated only the maximum amount of the transfered energy was limited, so the blurring effect was working during the simulation, only its results were limited.

Table 3 compares the results of the anisotropic diffusion and the results of the proposed method with two different settings. The first row shows a picture of an integrated circuit and the second row shows a CT image. The first column is the original image, the second is the output image of the anisotropic diffusion, the third one is the result of the proposed method with progressively terminated connections and the last column shows the results with limited energy transfer.

5. Conclusions

As this work is only preliminary, there are a lot of open questions. What kind of material model should be used? What are the optimal parameters for noise reduction? Do these parameters depend on the image? These questions should be answered but it can be seen that the proposed method has a clear potential for noise reduction and edge preserving smoothing.

Acknowledgements

This project was supported by the OTKA K-101527 project.

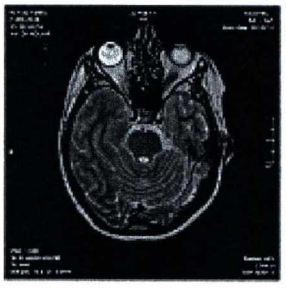
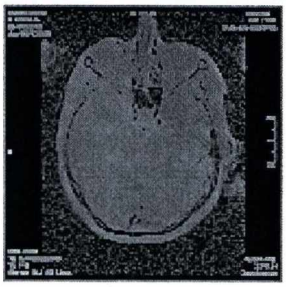
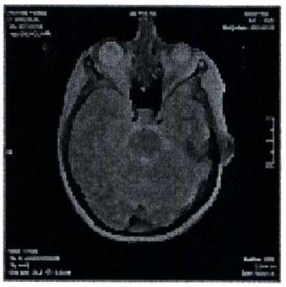
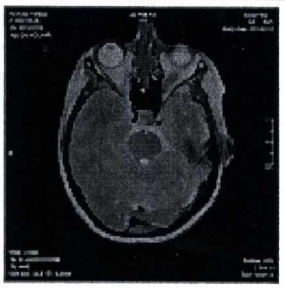
Original	$t = 30$	$t = 200$	$t = 400$
			

Table 1: *Image Generation by Terminating Connections*

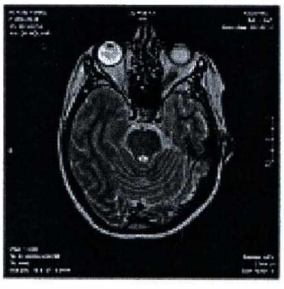
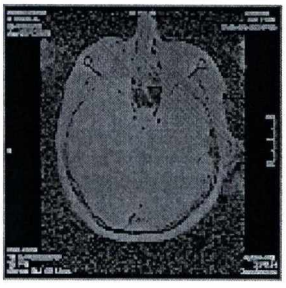
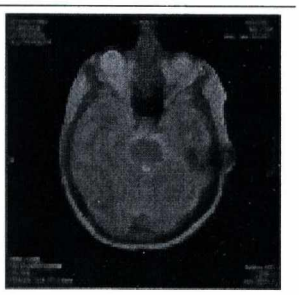
Original	$t = 30$	$t = 200$	$t = 400$
			

Table 2: *Image Generation by Limited Energy Transfer*

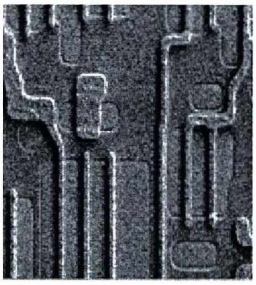
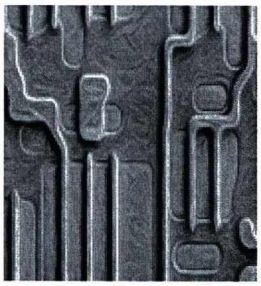
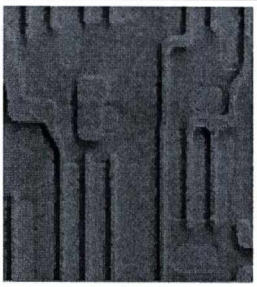
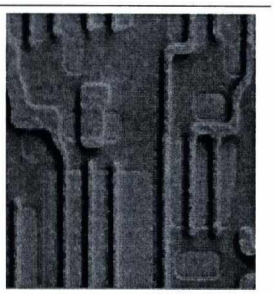
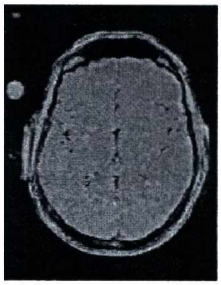
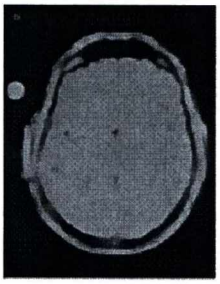
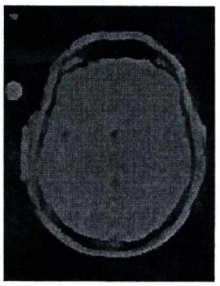
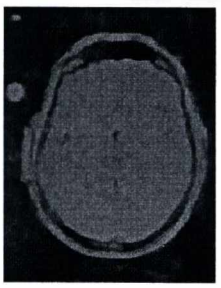
Original	Anisotropic Diffusion	Terminated Connections	Limited Energy Transfer
			
			

Table 3: *Results Compared to Anisotropic Diffusion*

References

1. Paul Bao and Lei Zhang. Noise reduction for magnetic resonance images via adaptive multiscale products thresholding. *Medical Imaging, IEEE Transactions on*, 22(9):1089–1099, 2003.
2. Jana Dornheim, Heiko Seim, Bernhard Preim, Ilka Hertel, and Gero Strauss. Segmentation of neck lymph nodes in ct datasets with stable 3d mass-spring models. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2006*, pages 904–911. Springer, 2006.
3. José-Jesús Fernández and Sam Li. An improved algorithm for anisotropic nonlinear diffusion for denoising cryo-tomograms. *Journal of structural biology*, 144(1):152–161, 2003.
4. Achilleas S Frangakis and Reiner Hegerl. Noise reduction in electron tomographic reconstructions using nonlinear anisotropic diffusion. *Journal of structural biology*, 135(3):239–250, 2001.
5. Gui Gao, Lingjun Zhao, Jun Zhang, Diefei Zhou, and Jijun Huang. A segmentation algorithm for sar images based on the anisotropic heat diffusion equation. *Pattern Recognition*, 41(10):3035–3043, 2008.
6. Zsolt Márta and László Szirmay-Kalos. Partial volume effect correction using anisotropic backward diffusion. In *KÉPAF*, pages 144–157, 2013.
7. Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(7):629–639, 1990.
8. Pietro Perona, Takahiro Shiota, and Jitendra Malik. Anisotropic diffusion. In *Geometry-driven diffusion in computer vision*, pages 73–92. Springer, 1994.
9. Ilya Pollak, Alan S Willsky, and Hamid Krim. Image segmentation and edge enhancement with stabilized inverse diffusion equations. *Image Processing, IEEE Transactions on*, 9(2):256–266, 2000.
10. Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface*, pages 147–147. Canadian Information Processing Society, 1995.
11. Guillermo Sapiro and Dario L Ringach. Anisotropic diffusion of multivalued images with applications to color filtering. *Image Processing, IEEE Transactions on*, 5(11):1582–1586, 1996.
12. László Szirmay-Kalos. Dynamic layout algorithm to display general graphs. *Graphics gems IV*, pages 505–517, 1994.
13. Abdullah Toprak and İnan Güler. Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter. *Digital Signal Processing*, 17(4):711–723, 2007.
14. Yu-Li You, Wenyuan Xu, Allen Tannenbaum, and Mostafa Kaveh. Behavioral analysis of anisotropic diffusion in image processing. *Image Processing, IEEE Transactions on*, 5(11):1539–1553, 1996.

Analysis of Image Descriptors for Zebrafish Toxicity Testing

László Szirmay-Kalos[1], Bence Parajdi[1], and Zsolt Csenki[2]

[1]: Dept. of Control Engineering and Information Technology, Budapest University of Technology and Economics
[2]: Dept. of Aquaculture, Institute of Environmental and Landscape Management, Szent István University

Abstract

This paper evaluates image descriptors for the determination whether zebrafish embryos are dead or alive. The image processing pipeline includes image stitching, illumination compensation, transformation to polar coordinates to emphasize the circular geometry of embryos, extraction of darker chorion and yolk boundary and yolk darkness features, and finally classification with a support vector machine. Individual descriptors should be strongly discriminative and preferably independent to allow the combination of their strengths by the support vector machine.

1. Introduction

The zebrafish (Danio rerio in Latin), also called the *Zebra Danio* is a tropical freshwater fish belonging to the minnow family (Cyprinidae) of order Cypriniformes. It is an important vertebrate model organism in scientific research because of its regenerative abilities and its transparent body which allows in vivo optical inspection. Another advantages for research include the facts that the generation of identical Zebrafish twins is an easy and inexpensive process, and animal protection laws do not apply to fish embryos before the fish first starts eating. Zebrafish embryo toxicity testing is used for routine sewage surveillance in many countries[2].

This paper proposes an image processing system that automatically determines whether 0–16 hour old zebrafish embryos are dead or alive. In this phase, embryos are still unhatched and have a circular like shape (Fig. 1). The system does not require manual intervention and is prepared for arbitrary embryo position and orientation. High resolution images of living and dead embryos are shown by Fig. 2.

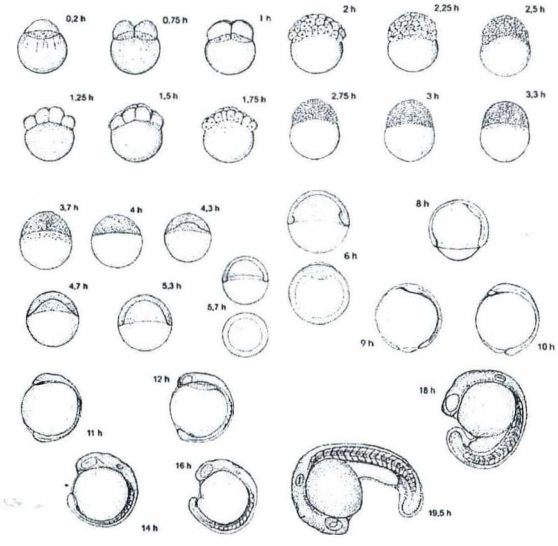


Figure 1: *Early development of healthy zebrafish embryos*[2].

2. Problem analysis

Examining a larger collection of zebrafish images, we can make the following observations:

1. Zebrafish is photographed by trans-illuminating the body and the plate by area light sources of not uniform but slowly changing intensity. The observed illumination intensity is not white and is different in every image.
2. Both healthy and dead embryos have well defined outer dark boundary surrounding the chorion, which has roughly circular shape.
3. The size of the embryos is well defined by biological constraints and has small variation. The diameters of the yolk

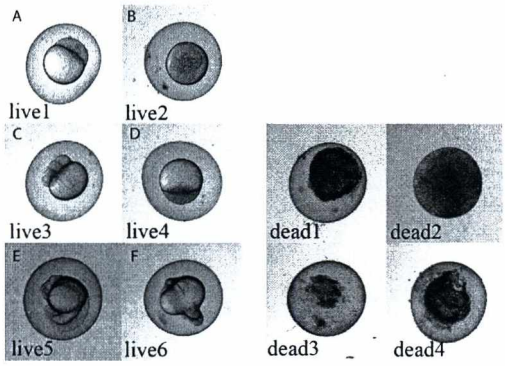


Figure 2: *High resolution images or living (left) and dead (right) zebrafish embryos.*

and the chorion are typically 0.6 mm and 1.2 mm, respectively.

4. The chorion includes the yolk that is also roughly circular in early stages, but gets less circular later. The yolk has also a well defined darker boundary curve in healthy embryos, but the boundary is not darker than the yolk itself in case of mortality.
5. The chorion can be of low noise both in alive and dead embryos.
6. The yolk is a little darker and noisier in dead embryos. The shape of the alive and dead embryos may be similar. The yolk of dead embryos has high variation, including fungi and darker spot like features.

An image processing system is usually built of three main phases, *image enhancement* that eliminates noise and additional effects corrupting the information of our interest and prepares the image for information extraction, *descriptor generation* that reduces the dimension of the information, i.e. extracts the most distinctive features, and finally *classification* that makes the decision based on the descriptors. In our case, image enhancement should eliminate the effects of non-constant and unknown background lighting and of the plate and convert the image into a form that is appropriate for recognizing circular features. For classification we use *Support Vector Machines* (SVM). The key part of the method is the definition of distinctive descriptors.

Our descriptors should be *translation and rotation invariant* since zebrafish can be anywhere and with any orientation in its well of the plate. The descriptor should also be *resolution independent* but *not scale invariant*, since images are made with different magnification and resolution parameters but these are known and the size of the embryo is also well defined (the diameter of the chorion and the yolk are about 1.2 mm and 0.6 mm, respectively). The size can be exploited to increase the robustness of recognition. To make the algorithms physical size dependent but resolution independent, all descriptors are computed in physical space where the unit is [mm]. Whenever a pixel space integral, e.g. convolution is computed, the size of the pixel in physical space is taken into account.

To find good descriptors, we can note that the chorion and the background may be similar for both healthy and dead embryos, so the descriptors should concentrate on the yolk region only. The shape of the chorion is not distinctive since again both healthy and dead embryos can have similar shape. We can check whether it is darker or of higher variation than expected in healthy embryos and also whether its boundary is darker than the yolk itself. In order to improve the discrimination power of image descriptors, the background should be identified and descriptors should be computed only for the foreground characterizing the yolk, which requires the identification of the embryo and the yolk. The circular like shape is emphasized by converting the image from Cartesian to polar coordinates.

A robust and accurate classification requires the identification of features that are significantly different in alive and dead embryos. We have decided to use the features that the yolk boundary of living embryos is a dark curve that separates the lighter chorion and yolk while the boundary is not significantly darker than the yolk in dead embryos. These boundaries are found by *ridge/valley detection* algorithms. Additionally, we examine descriptors that focus on the texture of the yolk, focusing on the darkness and the noise.

3. Previous work

A comprehensive review of Zebrafish image processing is [3], addressing applications like tracking of cells during embryogenesis, heartbeat detection, identification of dead embryos, recognition of tissues and anatomical landmarks, and quantification of behavioral patterns. The identification of dead embryos falls into the category of *phenotype recognition.*

In the method of [10], the image is binarized using an adaptive thresholding, in which a local threshold is set to the mean value of its local neighbors. The binary image is eroded to remove small spurious features and then dilated to connect broken segments. Of the connected objects, the one with the maximum area is recognized as the chorion. The second largest object in the image is the cytoplasm, the boundary of which is represented by a chain code contour. The boundary of the cytoplasm is often not fully connected, thus, a convex hull of the contour is constructed and used as initial positions for subsequent snake tracking. The centroid of the contour is recognized as the cytoplasm center. In order to distinguish the yolk from the cell portion, the cytoplasm contour after snake tracking is fitted into an ellipse using a least squares method, and intercepted into two parts by the minor axis of the fitted ellipse. Based on the fact that the cell portion always has greater convex deficiency, the cell and yolk portions are distinguished.

Liu et al.[6] classified three zebrafish phenotypes: *hatched*, *unhatched* and *dead* by evaluating six intensity and texture descriptors, including Local Edge Histogram Descriptor (LEHD), Color Layout Descriptor (CLD), Scalable Color Descriptor (SCD), Global and Semi-global Edge Histogram Descriptor (GSEHD), Representative Color Descriptor (RCD), and Color Histogram Descriptor (CHD). The LEHD descriptor provides texture information in terms of the spatial distribution of five types of edges, i.e., vertical, horizontal, forward diagonal, backward diagonal, and undirectional edge. LEHD comprises $16 \times 5 = 80$ histogram bins corresponding to the distribution of the five different edge types over 4×4 non-overlapping image blocks of equal size. Global and Semi-global Edge Histogram Descriptors (GSEHD) are constructed by aggregating the block histograms of the entire image and five sub-image groups comprised by 4 blocks. The CLD descriptor captures the local spatial distribution of intensity by using the coefficients of the 8×8 Discrete Cosine Transformation (DCT). The Representative Color Descriptor (RCD) comprised of 64 representative colors. SCD is a Haar transform encoded intensity histogram, which characterizes an image by the global color distribution. The GSEHD, SCD, and CHD are global descriptors capturing overall information about the images.

Jeanray et al.[4] computed pixel-based image descriptors and extremely randomized trees to recognize basic phenotypes and more subtle ones such as pericardial edema and curved tails.

Alshut[1] transformed the images into a 2D feature space where the y-axis gives values for the center of mass of the gray value histogram, and the x-axis indicates the mean intensity value in the chorion center.

4. The proposed system

We should identify first the embryo, then the yolk. The process contains a sequence of operations forming a pipeline. In the following, we discuss the elemental steps one by one.

4.1. Initial stitching

A zebrafish image should be first obtained from a collection of images of the plate that are neither stitched nor positioned on individual wells (Fig. 3). Fig. 4 shows how significant the error is when images are just placed one after the other. To reduce this error, we implemented a stitching algorithm that translates individual images along the border to minimize the difference, i.e. the L_2 error between the two edges.

4.2. Image enhancement

4.2.1. Illumination compensation

The objective of this step is to make the image independent of the actual lighting, camera setup and plate/water effects

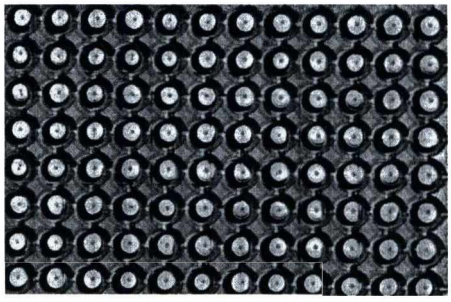


Figure 3: *Collection of images made by stepping the camera over a plate.*

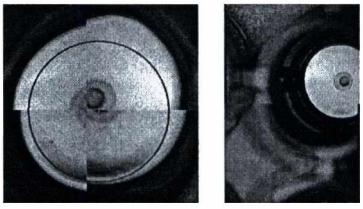


Figure 4: *Neighboring images before stitching and having executed the stitching operation.*

in order to make the image characterize solely the examined object. The compensation is done according to an illumination model that assumes that the embryo is trans-illuminated by an area light source having bi-linearly varying but unknown wavelength dependent intensity $L_0(x,y,\lambda)$ where x and y are pixel coordinates and λ is the wavelength corresponding to the additive RGB primaries. Intensity $L_0(x,y,\lambda)$ depends on light source and also on the glass container as well as the water which may modify the light intensity.

The trans-illuminated embryo is assumed to be absorbing participating medium of wavelength independent absorbing cross section $\sigma_a(\vec{s})$. We assume that emission and scattering are negligible. Thus, the radiance perceived by the camera is

$$L(x,y,\lambda) = L_0(x,y,\lambda) \cdot a(x,y) = L_0(x,y,\lambda) \cdot \exp\left(-\int \sigma_a(\vec{s})\mathrm{d}s\right).$$

where $a(x,y)$ is the total attenuation, which is the negative exponential of the optical depth, which in turn, is the line integral of the absorption cross section[8]. The embryo is characterized by the total attenuation

$$a(x,y) = \frac{L(x,y,\lambda)}{L_0(x,y,\lambda)}$$

which is independent of the illumination conditions and results in a gray-scale image.

The key part of this step is the determination of unknown

illumination intensity $L_0(x,y,\lambda)$, which is seen directly by the camera where no embryo part is visible. The edges of the individual wells cause strongly non-linear variation, so we consider those points that are outside of the embryo but inside its own well. Such regions are identified by fitting circles on the well and the embryo by Hough transform and considering only those pixels that are in between these circles. For the selected pixels, we fit a bi-linear radiance function $L_0(x,y,\lambda)$ for each representative wavelength λ, i.e. for red, green and blue. Let us consider just the red component, in the implementation the same method should be repeated three times. The samples taken outside the embryo are $\bar{x}_i, \bar{y}_i, R_i$ $(i = 1,\ldots,n)$ where $\bar{x}_i, \bar{y}_i$ are the normalized coordinates of the sample i that are zero at the left bottom corner of the image and are 1 at the right upper corner, and R_i is the red intensity of the sample pixel, and n is the number of samples.

The distribution of the red intensity $R(\bar{x},\bar{y})$ is searched in a bi-linear form:

$$R(\bar{x},\bar{y}) = R_{00}(1-\bar{x})(1-\bar{y}) + R_{01}(1-\bar{x})\bar{y} + R_{10}\bar{x}(1-\bar{y}) + R_{11}\bar{x}\bar{y},$$

where $R_{00}, R_{01}, R_{10}, R_{11}$ are the unknown intensities at the four corners of the image. These unknown values are determined with least-square fitting of the samples, i.e. we solve the following overdetermined system with minimizing the quadratic error:

$$R_i = R_{00}(1-\bar{x}_i)(1-\bar{y}_i) + R_{01}(1-\bar{x}_i)\bar{y}_i + R_{10}\bar{x}_i(1-\bar{y}_i) + R_{11}\bar{x}_i\bar{y}_i,$$

or in matrix form

$$\mathbf{S}_n = \mathbf{P}_{n\times 4} \cdot \mathbf{R}_4$$

where $\mathbf{S}_n$ is the n element vector of intensity samples R_i, $\mathbf{R}_4$ is the four element vector of unknown parameters $R_{00}, R_{01}, R_{10}, R_{11}$, and $\mathbf{P}_{n\times 4}$ is the $n \times 4$ element matrix of sample coordinates

$$\mathbf{P}_{n\times 4} = \begin{bmatrix} (1-\bar{x}_1)(1-\bar{y}_1) & (1-\bar{x}_1)\bar{y}_1 & \bar{x}_1(1-\bar{y}_1) & \bar{x}_1\bar{y}_1 \\ \vdots & & & \\ (1-\bar{x}_n)(1-\bar{y}_n) & (1-\bar{x}_n)\bar{y}_n & \bar{x}_n(1-\bar{y}_n) & \bar{x}_n\bar{y}_n \end{bmatrix}.$$

The least-square solution can be obtained with the Moore-Penrose pseudo inverse:

$$\mathbf{R}_4 = \left(\mathbf{P}_{4\times n}^T \cdot \mathbf{P}_{n\times 4}\right)^{-1} \cdot \mathbf{P}_{4\times n}^T \cdot \mathbf{S}_n.$$

4.2.2. Localization of the embryo

Having executed the illumination compensation, the resulting gray scale image can be imagined as a height field where intensity defines the height of a terrain. The boundary of the chorion is roughly a circle of approximately known radius and is darker than its neighborhood. Circles can be extracted by Hough transform. Making it more robust, the search space is restricted to biologically valid sizes of the embryo. Additionally, we enforce Hough transform to consider only valleys and not arbitrary edges as potential circle points, thus first run a ridge/valley detection algorithm on the gray scale image and apply Hough transform on its result. Ridges and valleys are locally ellipse like features where one axis is significantly larger than the other. A valley point is also local minimum in the maximum curvature direction.

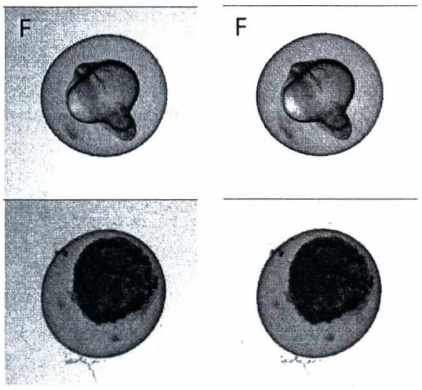


Figure 5: *Results of illumination compensation.*

Ellipses are obtained from the second order approximation of the image function $a(x,y)$:

$$a(x,y) \approx a(x_0,y_0) + \mathbf{g} \cdot [x-x_0, y-y_0]^T + \frac{1}{2}[x-x_0, y-y_0] \cdot \mathbf{H} \cdot [x-x_0, y-y_0]^T,$$

where

$$\mathbf{g} = [\mathbf{g}_x, \mathbf{g}_y], \quad \mathbf{g}_{c_1} = \frac{\partial a(x,y)}{\partial c_1}$$

is the gradient, and

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{xx} & \mathbf{H}_{xy} \\ \mathbf{H}_{xy} & \mathbf{H}_{yy} \end{bmatrix}, \quad \mathbf{H}_{c_1c_2}(x,y) = \frac{\partial^2 a(x,y)}{\partial c_1 \partial c_2}$$

is the Hessian matrix comprising of the second derivatives. In these equations c_1 and c_2 can stand for any of coordinates x and y.

Before computing the derivatives, the noise should be reduced either by regression[7] by convolving with e.g. a Gaussian filter

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right),$$

thus we compute the derivative of filtered image

$$\tilde{a}(x,y) = a(x,y) * G(x,y).$$

Variance σ of the Gaussian controls the scale of the fea-

tures that should be selected to correspond to the width of the boundary line[5].

The filtered derivative can be obtained by convolving the unfiltered function with the derivatives of the Gaussian.

$$\mathbf{g}_{c_1} = \frac{\partial \tilde{a}(x,y)}{\partial c_1} = a(x,y) * \frac{\partial G(x,y)}{\partial c_1},$$

$$\mathbf{H}_{c_1 c_2}(x,y) = \frac{\partial^2 \tilde{a}(x,y)}{\partial c_1 \partial c_2} = a(x,y) * \frac{\partial^2 G(x,y)}{\partial c_1 \partial c_2}.$$

The axis directions of the ellipse approximation are determined by the eigenvectors of the Hessian and the lengths of the axes by the eigenvalues. A pixel is classified as a valley point if one of the eigenvalues (defining the curvature) is a large positive value while the other eigenvalue is close to zero. Additionally, a valley is a local minimum with respect to the direction of the steepest hillside, thus the derivative in the direction of the eigenvector corresponding to the high curvature must be close to zero, i.e we should find a zero crossing between the current pixel and its neighboring pixels.

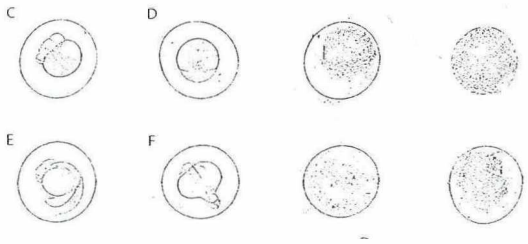


Figure 6: *Results of valley detection.*

The valley image is a binary image showing valley points (Fig. 6). The circle resulting from circle identification in the valley image is drawn over the attenuation image in Fig. 7.

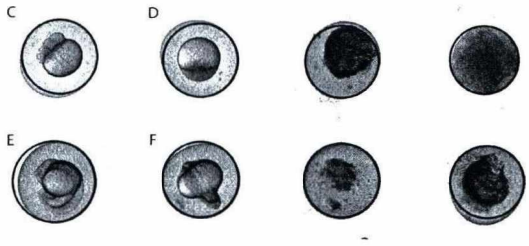


Figure 7: *Circles computed by the Hough transformation of the valley image drawn on the illumination compensated original images. We keep only the strongest circle from candidates where the radius is approximately similar to the expected embryo radius.*

4.2.3. Transformation to polar coordinates

The boundaries of the chorion and the yolk are roughly concentric circles, thus the algorithm extracting these curves should prefer such geometric arrangements. The arrangement of two circles is complicated, thus we transform the images into a space where ideal concentric circles become two horizontal lines. In the transformed space, curve extraction prefers horizontal lines. To set up a polar coordinate system where circular geometry is easier to handle, we have to find the center of the embryo.

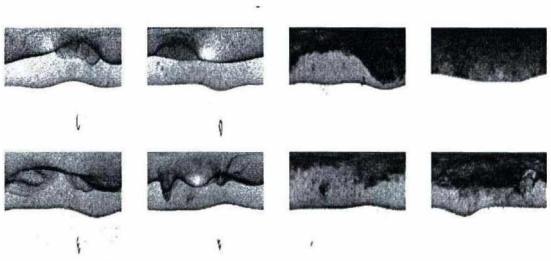


Figure 8: *Embryos of Fig. 2 in polar coordinates.*

Having identified the circle of center o_x, o_y in physical space approximating the boundary of the embryo, the polar map $p(\theta, r)$ is obtained as

$$p(\theta, r) = a(o_x + (r_{\max} - r)\cos(\theta), o_y + (r_{\max} - r)\sin(\theta))$$

where $\theta \in [0, 2\pi]$ and $r \in [0, r_{\max}]$, $r_{\max} = 2$ [mm]. The Jacobi determinant of this mapping is $(r_{\max} - r)^2$, thus it expands regions in the center, i.e. the yolk region, which is particularly useful when the yolk is our region of interest. The polar map is generated at $N = 256$ resolution.

4.3. Image descriptors based on the intensity map

Descriptors can be computed on the original attenuation map as well as its on the polar map. We expect polar map descriptors to be more distinctive since here the yolk represents a larger part of the image.

In polar coordinates the top of the figure corresponds to the center of the embryo where the yolk is located (Fig. 8). Here, the texture of dead embryos is darker and noisier.

To compute mechanical parameters like the mass and moment of inertia, a mass density parameter is needed, which is defined as the intensity of the inverted image:

$$\rho(x,y) = 1 - I(x,y).$$

In this way darker regions are taken into account in the mechanical oriented descriptors.

4.3.1. Mass

The mass is the integral of the mass density over the image, which can be computed for the attenuation image and also for the polar map. As the polar map is a distorted image, the two values will be different.

4.3.2. Moment of inertia

For the original image the moment of inertia is computed for an axis that is perpendicular to the image plane and crosses it in the center of the embryo:

$$\Theta = \int_x \int_y \rho(x,y)((x-o_x)^2 + (y-o_y)^2)\mathrm{d}x\mathrm{d}y.$$

Based on the recognition that dead embryos have darker yolk, we compute the inertia of the image emphasizing the contribution more toward the top. An appropriate measure is the moment of inertia of the inverted image with respect to the axis of the lower edge of the image:

$$\Theta_{\text{polar}} = \int_\theta \int_r \rho(x,y) r^2 \mathrm{d}r\mathrm{d}\theta.$$

4.3.3. Intensity histogram based descriptors

The intensity histogram is computed for pixels inside the circle corresponding to the chorion boundary. The histograms of living and dead embryos and also their average histograms are shown by Fig. 9.

Note that the histograms of dead embryos peak around darker colors while the histograms of living embryos around lighter colors. Based on the Alshut's approach[1], we calculate the *center of mass of the intensity histogram* and use it as an image descriptor.

Another descriptor can be obtained by calculating the "distance" between the current histogram and the histograms of average-living and average-dead. The average-living and average-dead histograms are obtained during the training of the system. There are different possibilities to characterize the difference of two distributions or functions. We could use, for example, the sum of the absolute differences (L_1 distance) or the square root of the sum of the squared differences (L_2 distance), but these metrics do not exploit the fact that we compare two distributions.

For two distributions, the *Kullback–Leibler (K-L) divergence* can be used, which is a measure of the information lost when distribution q_i is used to approximate distribution h_i. By definition, the K-L divergence of distribution q_i from h_i is

$$D_{KL}(h||q) = \sum_i \log\left(\frac{h_i}{q_i}\right) h_i.$$

K-L divergence is not a real metric since it is not symmetric and does not satisfy the triangle inequality. We select the average histogram to be q_i and require all bins to have a non-zero value. The current embryo is characterized by h_i. If h_i is zero, then $0\log 0$ is replaced by zero.

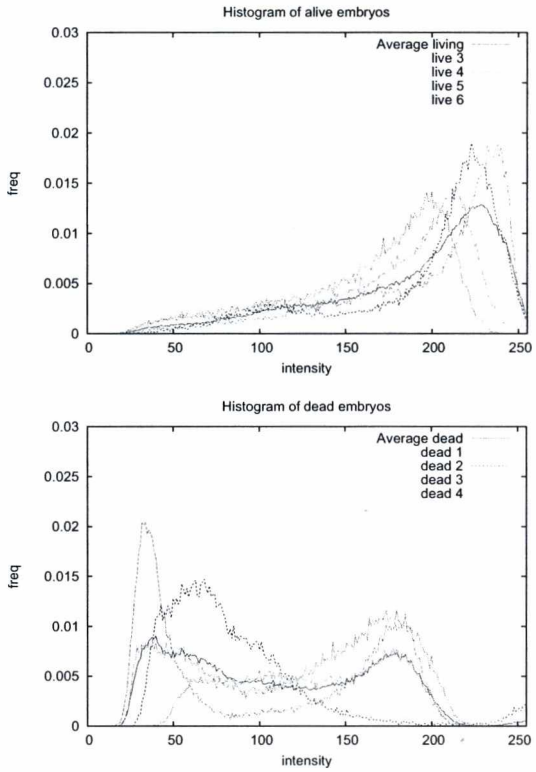


Figure 9: *Intensity histograms and average histograms of living and dead embryos, respectively.*

4.3.4. Intensity in the center of the chorion

Having identified the circle approximating the chorion boundary, the intensity value averaged in a small neighborhood of the center is computed.

4.3.5. Entropy of the gradient histogram

The yolk texture seems to be noisier in dead embryos, thus the noise level needs to be characterized. There are several options for this. We can use the Fourier descriptors or examine histograms. We examined both gray level histograms and gradient direction histograms. Histograms can be used directly as image descriptors in classification, or we can further compress information by calculating a single value from each histogram.

We used the entropy to characterize a gradient direction histogram with a single value:

$$S = -\sum_i p_i \log p_i,$$

where p_i is the probability of bin i that is calculated as the

relative frequency. Entropy is large when the distribution is more uniform, i.e. the image is noisier[9].

4.4. Image descriptors based on valley detection

Having transformed the image into polar coordinates, the boundary of the chorion and the healthy yolk can be imagined as horizontal valleys of the depth field where intensity $p(x,y)$ corresponds to height. Based on the first and second derivatives, a binary image is generated where a pixel is 1 if the first derivatives in this and its upper neighbor have different signs (there is a zero crossing) and the second derivative is larger than an appropriate threshold (Fig. 10).

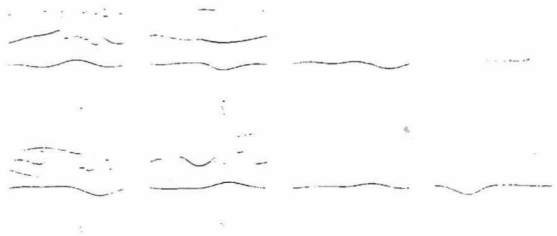


Figure 10: *Binary image of valley detection in the polar map.*

The classification of dead and healthy embryos is based on whether the valley around the yolk can be clearly identified. Note that the boundary of the chorion is always quite clear independently of the state of the embryo, and in addition to the boundaries there are shorter curve segments due to noise. Having identified the pixels that correspond to horizontal valleys, we check whether these pixels form two long mainly horizontal curves.

4.4.1. Lengths of yolk boundary segments

The binary image of valley detection is processed by a modified *flood filling* algorithm that prefers horizontally expanding curves.

This algorithm moves horizontally on white pixels until it can. When the next horizontal neighbor is black, it checks the upper right and lower right pixels and continues if one of them is white. If they are also black, the same check is repeated by only limited number of times, which defines the maximum gap this algorithm can fill. The algorithm stops when it reaches the right side or when the current curve cannot be continued. During flood filling, we report the lengths of the identified curves and keep the five largest values (Tab. 1).

We note that the largest value is close to 256, the resolution of the valley map, when the boundary of the chorion can be clearly identified. The remaining segments may belong to the boundary of the yolk. If they are longer or their sum is about 256, the yolk boundary is strong, which indicates a healthy embryo. To further compress information, we can find the segment length sufficit or deficit after removing the length of the chorion boundary which is expected to be 256 in both healthy and dead embryos. For living embryos, we expect a positive result, which is equal to 256 in the ideal case, and a negative or zero result for dead embryos.

Embryo	Segment lengths
Live1	199, 181, 79, 26, 13
Live2	255, 120, 111, 24, 12
Live3	241, 114, 45, 18, 17
Live4	238, 153, 39, 20, 17
Live5	232, 81, 34, 29, 25
Live6	249, 69, 26, 24, 12
Dead1	249, 0, 0, 0, 0
Dead2	20, 15, 5, 5, 2
Dead3	189, 34, 4, 3, 0
Dead4	53, 50, 48, 28, 21

Table 1: *Valley segment lengths for the embryos of Fig. 2. The polar map has 256 pixel resolution.*

5. Results

Tab. 2 summarizes the discussed descriptor values for the embryos of Fig. 2, including the *mass* of the inverted intensity image, *inertia* of both the inverted intensity image and of the polar map, the *intensity in the center of the chorion* (Int in Cent), the *center of mass of the intensity histogram* (Center Hist), the *Kullback-Leibler divergence* from the average alive and average dead, respectively, the *entropy of the gradient histogram of the polar map* and the *sufficit/deficit of the valley lengths in the polar map*. We can conclude that all these descriptors are able to deterministically separate the given 10 test cases. However, their performance is different on larger test sets.

The mass and inertia of the inverted intensity map is robust to the detection of the chorion but is less discriminative than the inertia of the polar map. The transformation to the polar map distorts the image emphasizing pixels in the center where the yolk is, which automatically improves the discriminative power. The intensity of the center and the center of mass of the intensity histogram together form a robust discriminator. Similarly, the Kullback-Leibler distances from the average dead and average living histograms should be considered together and classify an embryo based on their

Embryo	Mass	Inertia	Polar Inertia	Int in Cent	Center Hist	KL Live	KL Dead	Entropy	Valley length
Live1	1838	28	48	239	210	0.263	2.805	5.07	242
Live2	3103	46	116	115	186	0.134	2.36	5.11	266
Live3	2878	47	85	209	196	0.051	2.276	5.05	179
Live4	2727	41	92	159	194	0.05	2.35	5.12	211
Live5	4387	75	109	212	160	0.371	0.609	5.11	145
Live6	3776	61	103	200	172	0.138	1.114	5.12	124
Dead1	6571	103	220	50	103	1.507	0.228	5.16	-7
Dead2	6643	94	225	40	88	1.649	0.417	5.17	-209
Dead3	4806	78	169	82	142	0.677	0.282	5.13	-26
Dead4	6336	95	212	71	109	1.037	0.023	5.13	-56

Table 2: *Image descriptors for the embryos of Fig. 2.*

relation. The entropy of the gradient histogram is not too strong, so we plan to use other measures for the characterization of the gradient histogram. Finally, the valley length sufficit/deficit is also good, but it is sensitive to the threshold setting determining the minimum curvature for valleys.

6. Conclusions

In this paper we discussed and analyzed several image descriptors to classify alive and dead zebrafish embryos. The power of individual descriptors is improved further by applying all of them in a support vector machine if descriptors are "orthogonal" in the sense that they focus on different image features. This is the main reason, we proposed descriptors falling into three categories, characterizing the darkness, nosiness and the boundary of the yolk.

Acknowledgement

This work was supported by OTKA K–104476 and the Hungarian Government, managed by the National Development Agency, and financed by the Research and Technology Innovation Fund (grant no. KMR 12-1-2012-0221).

References

1. R. Alshut, J. Legradi, R. Mikut, U. Strahle, and M. Reischl. Robust identification of coagulated zebrafish eggs using image processing and classification techniques. In *19th Workshop on Computational Intelligence*, pages 9–21, 2009.
2. Thomas Braunbeck and Eva Lammer. Fish embryo toxicity assays. Technical Report UBA Contract Number 203 85 422, Aquatic Ecology and Toxicology, Department of Zoology, University of Heidelberg, 2006.
3. Ralf Mikut et al. Automated processing of zebrafish imaging data: a survey. *Zebrafish*, 10(3):401–421, 2013.
4. N. Jeanray, R. Maree, B. Pruvot, O. Geurts P. Stern, L Wehenkel, and et al. Phenotype classification of zebrafish embryos by supervised learning. *Toxicol Lett*, pages 211–215, 2012.
5. József Koloszár, László Szirmay-Kalos, Zsolt Tarján, and Dávid Jocha. Shape based computer aided diagnosis and automated navigation in virtual colonoscopy. In *Spring Conference on Computer Graphics*, pages 113–119, Budmerice, 2006.
6. R. Liu, S. Lin, R. Rallo, Y. Zhao, R. Damoiseaux, and et al. Automated phenotype recognition for zebrafish embryo based in vivo high throughput toxicity screening of engineered nanomaterials. *PLoS ONE*, 7(4):e35014, 2012.
7. L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4D linear regression. In *Computer Graphics Forum*, volume 19(3), pages 351–358, 2000.
8. L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.
9. Balázs Teréki, Pirkko Oittinen, and László Szirmay-Kalos. Informational aesthetic measure for 3d stereoscopic imaging. In *KÉPAF*, pages 57–66, 2013.
10. W. Wang, X. Liu, D. Gelinas, B. Ciruna, and Y. Sun. A fully automated robotic system for microinjection of zebrafish embryos. *PLoS ONE*, 2(9):e862, 2007.

Seventh Hungarian Conference on Computer Graphics and Geometry, Budapest, 2014

American Hand Sign Recognition in Video Streams

Domonkos Varga[1],[2]

[1]MTA SZTAKI, H-1111 Budapest, Kende u. 13-17.
[2]BME VIK Department of Control Engineering and Information Technology, H-1117 Budapest, Magyar tudósok krt. 2.

Abstract

We introduce in this paper a hand gesture recognition program based on computer vision. We detect and interpret the signs of the American Hand Sign Language. After reviewing related work a method is described for hand sign recognition in video streams. Our program follows the steps of a typical gesture-recognition algorithm - segmentation, feature extraction and classification. Then we outlined in detail the worked-out and applied algorithms. The segmentation in our case contains two steps - the spatial and the temporal segmentation. We extracted the so-called Fourier descriptors from the segmented video frames. On the grounds of Fourier descriptors we solved the classification. Finally we described our experimental results which appertain to the speed, robustness and running time of the program. CUDA is used for accelerating the running time of our application.

Categories and Subject Descriptors (according to ACM CCS): I.4 [Image Processing and Computer Vision]: Gesture recognition, segmentation, Fourier descriptor

1. Introduction

Hand gesture recognition is a new challenging means of interactions with computers. Device-based techniques such as electronic gloves can also be used for communicating with computers but our approach is a vision-based one using only a simple camera. This allows to communicate with the computer directly but recognition of shape of hand is a complex tasks which needs many algorithms.

Gestures to be recognized must be simple but different enough from each other. That is why we used the American Sign Language which satisfies these requirements.

This paper presents a hand gesture recognition system using a contour based approach. The first step is the spatial segmentation then the temporal segmentation comes. In the spatial segmentation a skin-detection algorithm will be used to determine the pixels which belongs to the hand. In the temporal segmentation we detect a moving or unmoved hand. We realize the temporal segmentation with the help of a final state machine. With morphological operations the hand contour can be extracted so the image is converted into boundary image. The first 50 coefficients of the Fourier transform are used to characterize the the hand gesture. For classification distance metric and SVM techniques are used.

We also outline the possibility of enhancing the speed of computation by using CUDA thrust. This time CUDA thrust is used for the determining the geometric properties.

The method has certain limitations. The user must wear a single colored shirt or pullover and he must hold his hand in the middle of the frame. No flashing signs are allowed in the background. In my experience these mean no severe limitations.

The paper is organized as follows. In section 2 the related work is summarized. Section 3 gives an overview of the algorithms and methods used while results and analysis comes in section 4.

2. Related Work

In hand gesture recognition studies two main approaches can be distinguished: hand model-based and appearance-based techniques[1].

Hand model-based methods detect the exact 3D pose of the hand and use a device. Device can be electronic or colored gloves which have and interface to the computer. They area excellent methods but not the topic of this paper.

For view-dependent techniques a simple built-in camera in laptops is sufficient, they are efficient in computation time

but they can not distinguish fine movements as device-based methods.

Ravikiran et. al proposed a method of recognizing American Sign Language using number of fingers open in a hand gesture[2]. Mapari et. al developed a method where they crop and resize the image and count the number of peaks and valleys on the hand contour[3]. After dividing the image into sixteen parts the number of peaks and valleys are added to the counters of the image parts. Neural network is used to train the program for classifying the hand signs.

The appearance-based methods mostly differ in methods extracting content representing features of images and in classifying techniques. Region-based descriptors are for example the Hu[4] moments while Fourier descriptors[5] are widely used as contour descriptors.

For classification in simple cases multidimensional Euclidean or Manhattan distance is used while for complex cases either neural network or SVM (Support Vector Machine) techniques are applied. In the latter cases a machine learning process is involved where the number of training hand sign are important factors.

3. Overview of the methodology

The program works with a laptop built-in camera or with a simple camera but some constraints have to be made. The user should held his hand in the middle of the image, he must wear long-sleeved single color shirt or pullover. Cropping of images was not necessary of enhancing the reliability of recognition. The outline of the algorithm can be seen in Figure 1. Spatial segmentation is made by skin color detection. In spatial segmentation we detected the pixels which belongs to the hand of the user. Temporal segmentation is achieved using a finite state machine model. Hand gesture is only recognized when the hand remains still for a predefined period of time. Then we extracted from the segmented image the so-called Fourier descriptors. On the grounds of Fourier descriptors we solve the problem of classification.

3.1. Spatial segmentation

For spatial segmentation we used a skin-color-based detection of the hand. First, the image in RGB was converted to HSI color space. We applied the algorithm of histogram equalization to the "I" component of HSI color space. After this we reconverted the preprocessed image in HSI to RGB. Applying the histogram equalization on the Red, Green, and Blue components of an RGB image may yield dramatic changes in the color balance of the image. That is why we did histogram equalization in HSI.

After the preprocessing we converted to HSV. The HSV color space is more related to human color perception[6]. The skin in channel H is characterized by values between 6 and 38, in the channel S from 0.23 to 0.68 for Caucasian ethnics[7].

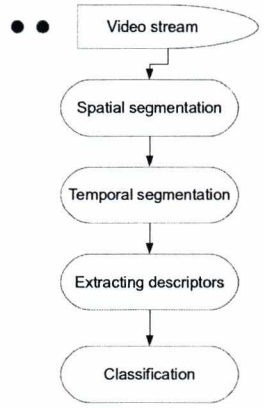


Figure 1: *Outline of the gesture recognition algorithm.*

Pixels classified as skin were set to value 255, and non-skin pixels were set to value 0.

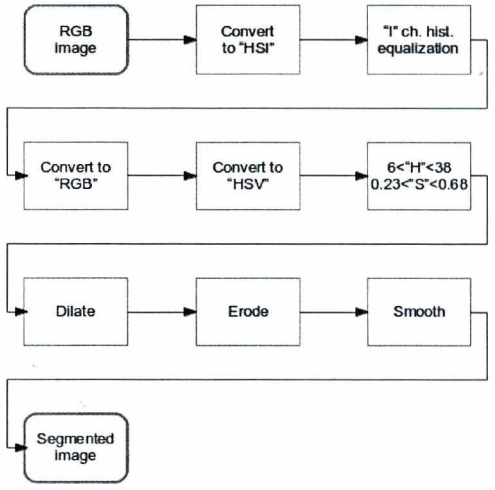


Figure 2: *The process of spatial segmentation.*

Next step minimizes noises, using a 5×5 structuring element in morphological filters. First, we used the structuring element with a dilatation filter. After that the same structuring element was used to erode the image. Then, a 3×3 median filter was used to eliminate the small noises.

The whole process of spatial segmentation can be seen in Figure 2.

3.2. Temporal segmentation

We realize the temporal segmentation with a finite-state machine. The possible states are start, hand movement, hand

unmoved, hand recognized and finish. The state diagram can be seen in Figure 3. The start state is shown drawn with an arrow pointing at it from any where. The accept state is represented by thickened line.

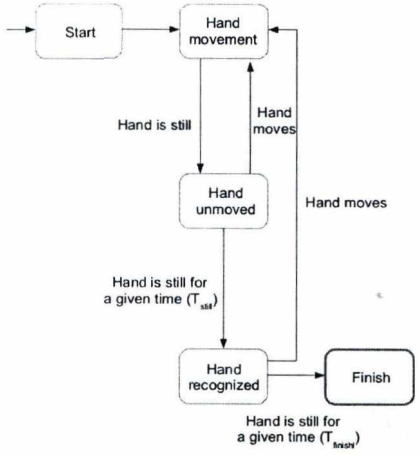


Figure 3: *The state diagram of temporal segmentation.*

The system only recognizes a gesture as a hand sign if the hand is still for a predefined period (T_{still}). The performed sign is recognized in the hand-recognized state. The next hand sign is only detected when the state of the recognition turns into hand movement by the detection of the hand movement. The input mechanism can be stopped when the recognition is in the hand-recognized state but the user's hand is still for a given period (T_{finish}).

Our method continuously analysis hand movement by function Move(t)[8]:

$$\text{Move(t)} = \begin{cases} 1 & \text{if } X_t > \Omega \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$X_t = \max\{\text{median}_{i=0}^{N}(|v_{t-i} - v_{t-i-1}|), \\ \text{median}_{i=0}^{N}(|h_{t-i} - h_{t-i-1}|)\}. \tag{2}$$

The hand moves in frame t if Move(t) = 1. The value N denotes the time window's length of the temporal analysis, v_t and h_t give the vertical and horizontal position of the center of the spatial segmented hand in frame t, and Ω is a threshold value to detect movement.

In our experiments, we tested different parameters: $T_{still} =$ 1.5 seconds and $T_{finish} = 2$ seconds. These intervals offered enough time for users to perform gestures. We set the threshold $\Omega = 6$. We set the size of the temporal windows as $N = T_{motion} * FPS$, where T_{motion}=0.5 seconds and FPS is frames per second and gives the actual processing speed of the system.

If the motion is continuous and we cannot detect static stages, HMM based state estimation can be used to find the gesture state positions.

3.3. Fourier descriptors

Points of the hand contour can be represented with the following signatures:

- complex coordinates
- central distance
- curvature
- cumulative angular function.

Using complex coordinates each point M_i of the hand contour is represented by a complex number z_i where N is the number of the contour points:

$$\forall i \in [1,N], \qquad M_i(x_i,y_i) \Leftrightarrow z_i = x_i + jy_i \tag{3}$$

Calculating the Fourier transform with FFT leads to the N Fourier coefficients C_k:

$$C_k = \sum_{i=0}^{N-1} z_i \exp\left(\frac{-2\pi jik}{N}\right), \qquad k = 0,1,...N-1 \tag{4}$$

C_0 is discarded because it represents the translation of the shape. In order to achieve scale invariance the Fourier coefficients are divided by the magnitude of the second coefficient[9]:

$$I_k = \frac{|C_k|}{|C_1|}, \qquad k = 2,...N-1 \tag{5}$$

The computation of Fourier descriptors is not invariant to the zero point. That is why we determine the inertia tensor and the mass center of the segmented hand image. Then we rotate the image into the direction of eigenvectors. The contour of the segmented and rotated image is extracted with morphological operations. The whole process of extraction of Fourier descriptors can be seen in Figure 4.

Low frequency coefficients give information about the overall shape of the hand while high frequency coefficients represent fine details and noise. We used the first 50 descriptors in the classification process so very high frequencies are removed to decrease noise sensitivity.

3.4. CUDA thrust

CUDA thrust is a development in CUDA with the following advantages:

- incorporation STL-like algorithm
- high-level interface to GPU
- hiding the details of parallel processing
- fast switching between GPU, CPU and OpenMP.

These new characteristics make CUDA thrust especially suitable for solving image analysis problems because the

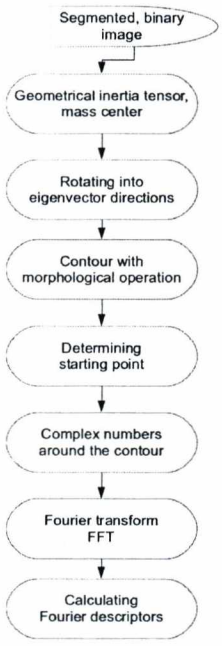


Figure 4: *Extracting the Fourier descriptors.*

users do not need to bother with memory management functions and by writing a C-like code can concentrate on the core issues.

3.5. Classification

First we use for the classification different types of distance metric. Euclidean D_e and Manhattan D_M distances in multidimensional space:

$$D_e = \sqrt{(\hat{I}_0 - I_0)^2 + (\hat{I}_1 - I_1)^2 + ... + (\hat{I}_{49} - I_{49})^2} \quad (6)$$

$$D_M = |\hat{I}_0 - I_0| + |\hat{I}_1 - I_1| + ... + |\hat{I}_{49} - I_{49}| \quad (7)$$

Where $\hat{I}_i$ variables represent reference values of Fourier descriptors. We determine these constants with the analysis of the user's hand while the user performed the hand signs.

Then we use a support vector machine (SVM) for classification. SVM is a machine learning technique which uses hyperplanes to separate the characteristics of the images in multidimensional space. Properly choosing the kernel function nonlinear cases can also be solved by transforming them to linear cases.

A SVM carries out the classification by the selected feature vectors[10]. The classifier has 50 input parameters and returns the identifier of the recognized hand gesture. We used LIBSVM to build up our support vector machine[11]. The SVM kernel is a radial basis function.

For American Sign Language recognition and under stable lighting conditions there were no difference in the three classification techniques. Nevertheless with unexperienced users it is advisable to employ the SVM method.

4. Experimental results

The codes were first developed in a MATLAB environment using the official toolbox functions. Cropping of images was not necessary for enhancing the reliability of recognition. Optimizing the code and using C++/OpenCV the running times could be decreased see table 1.

Operation	Running time
Spatial segmentation	88.2 msec
Center of mass	28.2 msec
Temporal segmentation	13.4 msec
Extracting FD	352.4 msec
Classification (Euc.)	0.1 msec
Classification (Man.)	0.1 msec
Classification SVM	56.7 msec
Total with distance metrics	480.3 msec
Total with SVM	536.9 msec

Table 1: *Running time. Genius FaceCam 312 (640 × 480).*

Using CUDA thrust the running times of the spatial segmentation, computation of center of mass, temporal segmentation, and extracting of Fourier descriptors can be decreased significant see table 2. We can establish that we can reduce the running time using CUDA thrust with 378.5 msec.

Operation	Running time
Spatial segmentation	9.8 msec
Center of mass	2.8 msec
Temporal segmentation	1.7 msec
Extracting FD	89.3 msec

Table 2: *Running time using CUDA thrust. Genius FaceCam 312 (640 × 480).*

We tested the usability and the performance of our system up to the present with 3 users. These people were selected from our friends. The tests were performed with unexperienced users. The hand silhouette depends on the hand physiology and the length of the sleeve could also modify the contour. That is why preliminary training is necessary. In our first experiments we collected training samples from the users by making gesture snapshots.

We noticed that recognition rates are dependent on the users. Some unexperienced users have difficulties to realize one of the gestures. In our experiments, each user trained the system with 90 samples of hand signs. This means 9 samples

per hand sign. The average recognition rates of hand signs are summarized in a so-called confusion matrix.

	A	B	C	D	G	H	I	L	V	Y
A	**88.8**	1.4	0	0	2.0	2.7	0	0	1.7	3.5
B	1.1	**91.6**	0	0	0	3.5	0	0	1.8	2.0
C	0.8	0	**91.7**	7.5	0	0	0	0	0	0
D	0.2	0	8.7	**87.9**	0	3.1	0	0	0	0
G	1.0	0	0	0	**83.3**	10.1	1.0	0	4.6	0
H	0.2	0	0	2.9	3.4	**81.2**	10.1	0	0	0
I	1.7	0.2	0	0	0	0	**89.1**	0	0	9.1
L	3.1	2.1	0	0	0	2.1	0	**83.3**	1.0	8.3
V	5.6	2.1	0	0	0	0	5.0	12.2	**75.1**	0
Y	8.5	0	0	0	0.3	0	0	0	10.4	**80.7**

Figure 5: *Confusion matrix with the classification method of SVM.*

In Figure 5 average recognition rates (%) are summarized in a confusion matrix. Rows indicate signs to be performed, while columns depict the distribution of recognition results.

Comparing our results to similar solutions, in[8] Table 1 only 7 gestures were detected with 81 - 96% efficiency. Here we demonstrated static signs only. In case of dynamic motions, like the the repeat action in[12], the error rate in Figure 5 can be significantly decreased. In[13] the performances of Fourier descriptors and Hu moments were tested for vision-based hand posture recognition. They have shown that Fourier descriptors outperform Hu moments. Hand posture gestures were detected with 60 - 96% efficiency with the help of Fourier descriptors.

5. Conclusion

The recognition of the static American Sign Language can be performed with a simple built-in camera. Running times make possible the usage in real-time environment.

CUDA thrust is a suitable way of decreasing the running time in parallel processing without the need of bothering with memory management and CPU, GPU hardware specifications.

Robustness, reliability and speed of static recognition is the base to recognize dynamic hand movements and converting them into text or spoken word.

6. Acknowledgement

I would like to express my thanks to my professor Dr. Ferenc Vajda for teaching and mentoring me. I also would like to thank Dr. Tamás Szirányi for his support and advices.

References

1. S. Ong and S. Ranganath. Automatic sign language analysis: A survey and the future beyond lexical meaning. *IEEE Transactions on Pattern Analysis and Machine Intelligence vol. 27*, (6):873-891, 2005.

2. J. Ravikiran, K. Mahesh, S. Mahishi, R. Dheeraj, S. Sudheender, and, N. V. Pujari Finger detection for sign language recognition. *In Proceedings of the International MultiConference of Engineers and Computer Scientists vol. 1*, 18:20, 2009.

3. R. Mapari and G. Kharat Hand gesture recognition using neural network. *International Journal of Computer Science and Network vol. 1, issue 6*, 2012.

4. M. K. Hu. Visual pattern recognition by moment invariants. *IEEE Transactions on Information Theory vol. 8*, pp. 179:187, 1962.

5. E. Persoon and K. Fu. Shape discrimination using Fourier descriptors. *IEEE Transactions on Man and Cybernetics vol. 8*, (3):388-397, 1986.

6. A. Albiol, L. Torres, and E. J. Delp Optimum color spaces for skin detection. *In proceedings of the 2001 International Conference on Image Processing vol. 1*, 122:124, 2001.

7. V. A. Oliveira and A. Conci Skin detection using HSV color space. *Workshops of Sibgraphi*, 1:2, 2009.

8. A. Licsár, T. Szirányi, L. Kovács, and B. Pataki A folk song retrieval system with a gesture-based interface. *IEEE Multimedia vol. 16*, (3):48:59, 2009.

9. D. Toth and T. Aach Detection and recognition of moving objects using statistical motion detection and Fourier descriptors. *In proceedings of the 12th International Conference on Image Analysis and Processing vol. 1*, 430:435, 2003.

10. K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks vol. 12*, (2):181:201, 2001.

11. C.-C. Chang and C.-J. Lin LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology vol. 2*, (3):1:27, 2011.

12. A. Licsár and T. Szirányi User-adaptive hand gesture recognition system with interactive training. *Image and Vision Computing vol. 23*, (12):1102:1114, 2005.

13. S. Conseil, S. Bourennane, and L. Martin Comparision of Fourier descriptors and Hu moments for hand posture recognition. *In proceedings of European Signal Processing Conference vol. 1*, 2007.

Digitális képérzékelők egységes paraméterezése információtartalom és fraktálszerkezet alapján

Berke József

Gábor Dénes Főiskola, 1139, Budapest, Mérnök út. 39, Hungary
University of Hertfordshire, Hatfield, Hertfordshire, AL10 9AB, United Kingdom

Abstract

A digitális képalkotó berendezések világméretű elterjedésével, valamint az alkalmazott kutatások során használt speciális képalkotó berendezések rohamos fejlődésével (multi- és hiperspektrális képalkotás, multitemporális képalkotás) számos eltérő elektronikai felépítésű digitális képérzékelő kerül alkalmazásra. Az egyes képalkotó érzékelők azonban minden esetben az elektromágneses hullámokat érzékelik, azaz működésük azonos fizikai törvényekre épülnek. A keletkezett digitális képek jellemzésére is számos paraméter került bevezetésre (geometriai felbontás, csatornaszám, rétegszám, intenzitásértékek, színmélység), amelyek a képi adatfeldolgozás szempontjából általában függetlennek tekinthetők, azonban a gyakorlatban a képérzékelők jellemzői miatt erősen korrelálhatnak. A feldolgozások információ tartalom vagy szerkezet alapúk, esetenként ezek kevert módszereire épülnek. Figyelembe véve a képi adatfeldolgozás tartalom és szerkezet alapú módszereit, a szerző javaslatot tesz digitális képérzékelők egységes paraméterezhetőségére, az érzékelők által előállított képi adatok jelenleg ismert és használt paramétereinek felhasználása alapján.

Categories and Subject Descriptors (according to ACM CCS): I.4.7 [Image Processing and Computer Vision]: Feature Measurement

1. Bevezetés

A 80-as években nagy reményeket fűztek olyan matematikai eljárások gyakorlati alkalmazásához, amelyek elsősorban törtdimenziós matematikai konstrukciókra épültek. Kezdetben számos területen - anyagszerkezet vizsgálat, kaotikus jelenségek (földrengés, tornádó, turbulens áramlás) szimulációja, valós folyamatok modellezése informatikai eszközökkel, folyók, partszakaszok hosszának meghatározása - születtek eredmények [1, 3, 16, 17, 24, 25, 26].

Később az alkalmazott informatika területein, az adattömörítés [2, 9, 28, 32], a számítógépes osztályozás [22, 23, 22] során értek el jelentősnek mondható eredményeket a gyakorlati alkalmazásokban. Napjaink informatikai alapú kutatás-fejlesztései programjaiban egyre gyakrabban találkozhatunk fraktálokra visszavezethető eljárásokkal, fraktál alapú algoritmusokat alkalmazó programokkal, valamint ezek gyakorlati alkalmazásának eredményeivel - iszapkatasztrófa légifelmérése [13, 14], a gépjárművek okozta nehézfém szennyeződések felmérése [21], burgonyaminősítés [11], tőzegvizsgálat [12], stb.

A digitális képalkotó berendezések világméretű elterjedésével számos eltérő elektronikai felépítésű digitális képérzékelő kerül alkalmazásra. Az egyes képalkotó érzékelők minden esetben az elektromágneses hullámokat érzékelik, azaz működésük azonos fizikai törvényeket követ. A keletkezett digitális képek jellemzésére is számos paraméter került bevezetésre (geometriai felbontás, csatornaszám, rétegszám, intenzitásértékek, színmélység), amelyek a képi adatfeldolgozás szempontjából általában függetlennek tekinthetők, azonban a gyakorlatban a képérzékelők jellemzői miatt erősen korrelálhatnak.

A képi adatfeldolgozások információ tartalom vagy szerkezet alapúk, esetenként ezek kevert módszereire épülnek. Célszerű lenne a digitális képérzékelők egységes paraméterezhetőségének megteremtése, az érzékelők által előállított képi adatok jelenleg ismert és használt paramétereinek felhasználása alapján. Mindez megteremthetné a konzekvens, eltérő típusú, eltérő paraméterű érzékelő adatainak egységes kezelését, feldolgozását, amely a jelenlegi eltérő távérzékelési módszerekkel, érzékelőkkel előállított adatok egységes feldolgozásának alapját képezheti.

2. Az entrópia

Az entrópia napjainkban használt információelméleti fogalmát 1948-ban Claude E. Shannon [29, 30] vezette be, majd gyakorlati példán keresztül szemléltette [31], melyet Neumann János javaslatára nevezett el entrópia függvénynek. Ezek szerint az üzenetek átlagos információ tartalma (független üzenetek esetén) – entrópiája, az alábbiak szerint határozható meg:

$$H = \sum_{i=1}^{m} p_i \, ld\left(\frac{1}{p_i}\right)$$

ahol H - az információelméleti entrópia

p_i - az i-edik üzenet előfordulási valószínűsége (gyakorlatban relatív gyakoriság)

Az entrópia matematikai értelemben vett általános definícióját Rényi Alfréd adta 1961-ben [27], amely szerint

$$H_\alpha(X) = \frac{1}{1-\alpha}\log\left(\sum_{i=1}^{n} p_i^\alpha\right)$$

ahol $\alpha \geq 0$ és $\alpha \neq 1$

Az entrópia gyakorlati esetekben történő számítása során célszerű figyelembe venni még az alábbiakat:

- Egy zárt rendszer információelméleti entrópiája az alábbi értékeket veheti fel:

$$0 \leq H \leq \log_2 n$$

ahol n a lehetséges üzenetek száma.

- A entrópia akkor a legkisebb, ha a forrás mindig ugyanazt az üzenetet küldi azaz a képen egyetlen szín vagy intenzitásérték szerepel.
- A entrópia akkor veszi fel a legnagyobb értéket, ha az összes üzenet valószínűsége egyenlő – ($p_i = -\log_2 n$) amely

$$H_{max} = log_2 n$$

Képek esetén ez azt jelenti, hogy minden egyes lehetséges intenzitásérték egyenlő számban fordul elő a képen. Azonban a gyakorlatban előforduló képalkotó eszközök legtöbb esetben egyetlen lapkát tartalmaznak, amelyek 12, 14, vagy 16 bit felbontásúak. Az érzékelők geometriai felbontása általában jelentősen meghaladja az intenzitásfelbontást – pl. Canon EOS 5D MII esetén a lapka minden egyes pixele 14 bit információt tartalmaz, ugyanakkor a lapkán 21 MPixel érzékelő található. A 21 MPixel közel $2^{24,33}$, míg 14 biten 2^{14} azaz 16 384 különböző érték ábrázolható. Vagyis a lapka maximális entrópiája a két érték közül a kisebb azaz 14 bit.

Digitális kamerák képérzékelő eszközei azonban három csatornát rögzítenek (RGB) a Bayer-mintázat alapú érzékelő adatai alapján, melyet interpolációval állítanak elő. Az interpoláció után –a fenti példában szereplő kamera esetén- 3x14 bit kerül rögzítésre, ami elvben 42 bit maximális információtartalom rögzítését teszi lehetővé. Ugyanakkor egy ilyen érzékelőt tartalmazó kamerával készített felvétel maximális információtartalma csak 24,33 bit, hiszen csak ennyi eltérő érték fordulhat elő, mivel ennyi a lapkában található érzékelők száma.

3. Spektrális fraktáldimenzió

A fraktáldimenzió a törtdimenziók közé tartozó matematikai fogalom. Önhasonló alakzatok matematikai leírására az elsők között találjuk (1904 körül) a von Koch-féle hópehely görbékre adott leírásokat [17]. A fraktáldimenzió segítségével meghatározható, mennyire szabálytalan egy fraktál görbe. Általában a vonalakat egydimenziósnak, a felületeket kétdimenziósnak, a testeket pedig háromdimenziósnak nevezzük. Tekintsünk azonban egy nagyon szabálytalan görbét, amely ide-oda vándorol egy felületen (például egy papírlapon) vagy a háromdimenziós térben. Gyakorlatban számos ilyen tekervényes görbét ismerünk: például a növények gyökérzete, a fák ágai, az emberi test érhálózata, nyirokrendszere, egy úthálózat, stb. Így a szabálytalanságra úgy is tekinthetünk, mint a dimenzió fogalmának a kiterjesztésére. Vagyis egy szabálytalan görbe dimenziója 1 és 2 között lesz, míg egy szabálytalan felületé 2 és 3 közé esik.

Egy fraktálgörbe dimenziója olyan szám, amely azt jellemzi, hogy a görbe két kiválasztott pontja között hogyan nő a távolság, midőn növeljük a felbontást. Tehát amíg a vonal és a felület topológiai dimenziója mindig 1, illetve 2, addig a fraktáldimenzió lehet egy ezek közti érték is. A valós világban előforduló görbék, illetve felületek nem valódi fraktálok, olyan folyamatok hozták létre ő ket, amelyek csak egy meghatározott mérettartományban található alakzatokat képesek kialakítani. Így D változhat a felbontással. A változás segíthet abban, hogy jellemezhessük a létrehozásban közreműködő folyamatokat.

Mandelbrot az alábbiak szerint definiálta a fraktál fogalmát: "*A fractal is by definition a set for which the Hausdorff-Besicovitch dimension strictly exceeds the topological dimension*" [25] azaz fraktálnak tekinhető minden olyan halmaz, amelynek a Hausdorff-Besicovitch dimenziója nagyobb a topológiai dimenziónál. A gyakorlatban - elsősorban a digitálisan rögzített halmazok, adatok (pl. képek, hangok, videók) esetén - szinte mindig teljesül a fenti definíció.

A fraktáldimenzió elméleti leírása [1]:

Legyen (X,d) egy metrikus tér, valamint $A \in H(X)$. Legyen $N(\varepsilon)$ a minimális ε sugarú gömbök száma, amely lefedi A halmazt. Ha

$$FD = \lim_{\varepsilon \to 0}\left\{ Sup\left\{ \frac{LnN(\bar{\varepsilon})}{Ln(1/\bar{\varepsilon})} : \bar{\varepsilon} \in (0,\varepsilon) \right\}\right\}$$

létezik, akkor FD-t az A halmaz fraktáldimenziójának nevezzük. A fraktáldimenzió (FD) általános definíciója a következő:

$$FD = \frac{\log \frac{L_2}{L_1}}{\log \frac{S_1}{S_2}}$$

ahol L_1 és L_2 a (fraktál) görbén mért hosszúságok, S_1 és S_2 pedig a használt (tetszőleges) mérték nagysága (pl. digitális képek esetén a felbontás).

Számos olyan módszer került kifejlesztésre, amely a fraktáldimenzió számítására is alkalmas [10, 32] – 1. táblázat

Módszer	Legfontosabb jellemző
Legkisebb négyzetek módszer	Elméleti megközelítésekre
Walking-osztó	Gyakorlatban hosszúság mérésére
Boksz-módszer	Legelterjedtebb módszer
Prizma-módszer	Egydimenziós jelekre
Epszilon-formula	Görbék mérésére
Kerület-terület alapú kapcsolat	Eltérő típusú képek osztályozására
Fraktál alapú Brown-mozgás	Boksz-módszerhez hasonló
Energia eloszlás	Digitális, fraktál jellegű jelekre
Hibrid módszerek	1D módszerek felhasználásával 2D fraktálok számításához

1. táblázat *Fraktáldimenzió számítására alkalmas módszerek*

Az SFD egy, az általános fraktáldimenzióból [25] származtatott szerkezetvizsgálati eljárás, amely a fraktálok egy újszerű alkalmazását jelenti. Az SFD [4, 5, 6, 7] a térbeli szerkezeten kívül a spektrális sávok színszerkezetének mérésére is alkalmas, és elegendő információt nyújt a színek, árnyalatok fraktál tulajdonságaira vonatkozóan is. Az SFD értékek számításához (két vagy több képsáv esetén, azonos spektrális felbontás esetén) a spektrális fraktáldimenzió alábbi definíciója alkalmazható a mért adatokra, mint függvényre (értékes spektrális dobozok száma az összes spektrális doboz függvényében) egyszerű matematikai átlagolással számítva az alábbiak szerint [7]:

$$SFD_{ESR} = \frac{n \times \sum_{j=1}^{S-1} \frac{\log(BM_j)}{\log((2^S)^n)}}{S-1}$$

ahol

n – a képrétegek vagy képcsatornák száma

S – a spektrális felbontás bitben

BM_j - értékes képpontot tartalmazó spektrális dobozok száma j-bit esetén

BT_j – összes lehetséges spektrális dobozok száma j-bit esetén

A lehetséges spektrális dobozok száma j-bit esetén az alábbiak szerint számítható:

$$BT_j = (2^S)^n$$

A fentiekben definiált SFD_{ESR} metrika, azaz kielégíti az alábbi feltételeket:

nemnegatív definit, azaz

$$\rho(P_1, P_2) \geq 0$$

$$\rho(P_1, P_2) = 0 \qquad \text{ha} \qquad P_1 = P_2$$

szimmetrikus, azaz

$$\rho(P_1, P_2) = \rho(P_2, P_1)$$

teljesíti a háromszög egyenlőtlenséget, azaz

$$\rho(P_1, P_3) \leq \rho(P_1, P_2) + \rho(P_2, P_3)$$

A metrika teljesülésének további feltétele a regularitás feltételének teljesülése is. Azaz, a diszkrét képsík pontjai egyenletes sűrűségűek legyenek. A gyakorlatban az A/D átalakító előtt a képfüggvényt nemlineáris transzformációnak vetik alá, aminek hatására a képfüggvény sűrűségfüggvénye állandó lesz. Így digitális képek esetén általában teljesül vagy annak tekinthető a regularitás feltétele.

Mivel az SFD_{ESR} összefüggés metrika [7], a kiértékelések során mind hiper- mind multispektrális felvételek esetén mérésekre egzaktul használható.

4. Véges felbontású digitális képek

Az alábbiakban gyakorlati összefüggést adunk, véges felbontású (finite spatial resolution) digitális képek esetén alkalmazható SFD számításokhoz (a CCD és CMOS érzékelők által adott képek mind ilyenek) [8]. Az előző fejezetben tárgyalt összefüggések alapján közvetlenül megállapítható, hogy

$$0 \leq SFD \leq n$$

azaz SFD értéke 0 és a számításokba szereplő csatornák/rétegek száma közötti értéket vehet fel.

A további becsléshez használjuk ki azon tényt, hogy a digitális képet alkotó pixelek száma ismert, legyen ez K,

$$K = X * Y$$

ahol

K – a képet alkotó pixelek száma

X - a kép szélességének mérete pixelben

Y - a kép hosszúságának mérete pixelben

Amennyiben

$$K \geq BT_j$$

akkor

$$SFD_{\max} = n$$

ha viszont

$$K < BT_j$$

akkor maximálisan annyi különböző spektrális képpontom lehet, amennyi a képpontok száma, ekkor

$$SFD_{ESR-MAX} = \frac{n \times \left(\sum_{j=Z}^{S-1} \frac{\log(BM_j)}{\log((2^S)^n)} + (Z-1) \right)}{S-1}$$

ahol igaz, hogy

$$1 \leq Z \leq S-1$$

és

Z-t úgy választom, hogy Z-1 esetén

$$K \geq BT_j$$

teljesüljön.

A ténylegesen megépített és használt képérzékelők esetén általában a képpontok száma kisebb, mint a lehetséges spektrális képpontok száma, így az $SFD_{ESR\text{-}MAX}$ összefüggés alkalmazandó.

5. Eredmények

Nézzük meg, hogy néhány gyakorlati esetben mit jelent a fentiekben vázolt $SFD_{ESR\text{-}MAX}$ összefüggés (2. táblázat):

	Emberi szem csapok	Phase One P65+	Nikon D3X	EOS 1Ds MIII
x	-	8984	6048	5616
y	-	6732	4032	3744
S	21bit	16 bit	14 bit	14 bit
K	2000000	60480288	24385536	21026304
n	3	3	3	3
SFD	2,0486	2,5710	2,6498	2,6416
SSRR	40,9720	41,1352	34,4471	34,3408

2. táblázat *Geometriai és spektrális felbontást szemléltető táblázat*

Az $SFD_{ESR\text{-}MAX}$ értéke az S, n és K függvénye, így értéke a CCD/CMOS érzékelők esetén, azok hasonló paraméterei (K – érzékelő tényleges vagy generált pixelszáma, S – az érzékelő csatornánkénti intenzitás felbontása bitben, n - (spektrális) csatornaszám) alapján változik. Digitális kamerák érzékelői esetén n=3, így csak két paraméter (S és K) függvénye lesz SFD értéke. Rögzített képpontú kép esetén *(K_1=18 MP – Canon EOS-M, K_2=21 MP - Canon 1Ds MIII, K_3=24 MP – Nikon D3X)* az S értékének növelésével az SFD értéke csökken (3. táblázat).

S /bit/	16	14	12	10	8
max (SFD) 18 MP	2,4982	2,6328	2,7737	2,9087	3,0000
max (SFD) 21 MP	2,5078	2,6416	2,7812	2,9135	3,0000
max (SFD) 24 MP	2,5167	2,6498	2,7880	2,9181	3,0000

3. táblázat *Az SFD maximumának változása rögzített K esetén*

Önmagában csak az SFD érték nem jellemző paraméter az érzékelőre. A K vagy csak az S értéke sem jellemző mint egyedüli paraméter.

Amennyiben az $SFD_{ESR\text{-}MAX}$ összefüggést megszorozzuk (S-1)-el, az alábbi értéket kapjuk, melyet SSRR-nek nevezünk (Spatial and Spectral Resolution Range):

$$SSRR_{CCD/CMOS} = (S-1)\times\left(SFD_{ESR\text{-}MAX}\right)$$

azaz

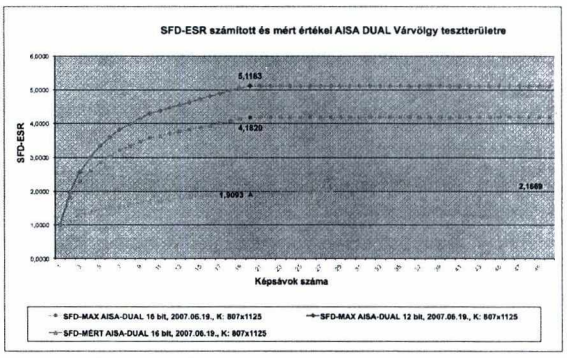
$$SSRR_{CCD/CMOS} = n\times\left(\sum_{j=Z}^{S-1}\frac{\log(BM_j)}{\log((2^S)^n)}+(Z-1)\right)$$

Ezen mennyiség értéke monoton nő, amennyiben az S, K és n értéke közül bármelyik kettő rögzített és a harmadik értéke nő. Tartalmazza mindhárom digitális érzékelőkre jellemző paramétert (K, S, n), így önmagában jellemző értéke lehet tetszőleges digitális képérzékelőnek.

Az 1. ábrán egy AISA Dual hiperspektrális érzékelő által készített, 359 csatornás légifelvétel első 50 csatornájának (400-620 nm), 1 m/pixel terepi felbontású, 807x1125 pixelxpixel méretű, Várvölgy teszterületre vonatkozó SFD_{ESR} értékeit tüntettük fel. Az érzékelő sávonként 12 bit mélységű adatokat készít, melyet megfelelő (szoftveres) korrekció után 16 bit mélységben tárolunk. A felső két görbe a 12 és 16 biten elméletileg elérhető maximális SFD_{ESR} értéket mutatja, melyet mindkét esetben a 20 csatorna esetén már elérünk. Az legalsó görbe a tényleges felvétel adatai alapján mért értékeket mutatja. A görbe a sávok számának emelésével monoton nő, azonban jelentősen elmarad az elméletileg elérhető maximális értékektől. Az első 150 sáv esetén is csak 2,5267 értéket kapunk úgy, hogy az első 50 sáv esetén 2,1669 volt, míg a maximális érték 3 körüli.

Az érzékelő bitben mért spektrális felbontása, a csatornák száma és a feldolgozásra kerülő képrészlet geometriai mérete alapján, az SFD_{ESR} görbe maximumpontjai közül a minimális darabszám megadja, hogy a teljes kép hány darab optimális adatszerkezetű csatornával lenne leírható – esetünkben ez 20 volt. Hiperspektrális képosztályozás esetén mindez nagyon jó egyezésben volt a legnagyobb találati pontosságot adó csatornák számával, amely esetünkben 21 volt [20, 22, 23].

Vagyis az elméletileg számított SFD_{ESR} görbe maximuma a gyakorlatban közvetlenül felhasználható adatokkal szolgálhat a légifelvételezés tervezéséhez, valamint az adatszelekciós eljárások optimális megválasztásához.



1. ábra *SFD_{ESR} értékek az összevont képsávok (dimenziók) függvényében*

6. Javaslatok

Javasoljuk, hogy digitális képérzékelő berendezések esetén kerüljön bevezetésre a

$$SSRR_{CCD/CMOS} = n\times\left(\sum_{j=Z}^{S-1}\frac{\log(BM_j)}{\log((2^S)^n)}+(Z-1)\right)$$

mennyiség, mint Spatial and Spectral Resolution Range - SSRR, amely mindhárom fontos jellemzőt (érzékelők képpontjainak száma, spektrális felbontás, csatornaszám) tükrözi. Fraktálszerkezetre és maximális információtartalomra épül.

Multispektrális vagy hiperspektrális felvételek esetén sávonként/rétegenként képezhető az SFD értéke. Ezen értékek grafikus ábrázolásával az eredeti képre vagy azon levő objektumokra kaphatunk egyedi görbét/görbéket. Ezen görbék önállóan is felhasználhatók, pl. növények,

ásványok esetén könyvtárak hozhatók létre a reflektanciára épülő spektrumkönyvtárakhoz hasonlóan [10, 20, 23]. Előnye, hogy közvetlenül a detektor által rögzített képről ad információt, valamint a görbék elemzésével információt kaphatunk a távérzékelő légi- és űreszközök esetén az atmoszféra vagy az érzékelő által okozott képi zajokra [20, 22].

A hiperspektrális érzékelő és a célterület alapadatai alapján az elméletileg számított SFD_{ESR} görbe maximuma a gyakorlatban közvetlenül felhasználható adatokkal szolgálhat a légifelvételezés tervezéséhez, valamint az adatszelekciós eljárások optimális megválasztásához.

Hivatkozások

1. Barnsley, M. F., Fractals everywhere, Academic Press, 1998.
2. Barnsley, M. F. and Hurd, L. P., Fractal image compression, AK Peters, Ltd., Wellesley, Massachusetts, 1993.
3. Batty, M. and Longley, P. Fractal cities, Academic Press, 1994.
4. Berke, J., Fractal dimension on image processing, 4th KEPAF Conference on Image Analysis and Pattern Recognition, Vol.4, 2004, pp.20.
5. Berke, J., The Structure of dimensions: A revolution of dimensions (classical and fractal) in education and science, 5th International Conference for History of Science in Science Education, July 12 – 16, 2004.
6. Berke, J., Measuring of Spectral Fractal Dimension, Advances in Systems, Computing Sciences and Software Engineering, Springer pp. 397-402., ISBN 10 1-4020-5262-6, 2006.
7. Berke, J., Measuring of Spectral Fractal Dimension, Journal of New Mathematics and Natural Computation, ISSN: 1793-0057, 3/3: 409-418, 2007.
8. Berke, J. (2008): Using Spectral Fractal Dimension in Image Classification, Computing Sciences and Software Engineering, SCSS'2008, Ref. Nr. 81.
9. Berke, J. and Busznyák, J., Psychovisual Comparison of Image Compressing Methods for Multifunctional Development under Laboratory Circumstances, WSEAS Transactions on Communications, Vol.3, 2004, pp.161-166.
10. Berke, J., Spectral fractal dimension, Proceedings of the 7th WSEAS Telecommunications and Informatics (TELE-INFO '05), Prague, 2005, pp.23-26, ISBN 960 8457 11 4.
11. Berke, J. – Wolf, I. – Polgar, Zs., Development of an image processing method for the evaluation of detached leaf tests, Eucablight Annual General Meeting, 24-28 October, 2004.
12. Berke, J. – Kozma-Bognár V., Fernerkundung und Feldmessungen im Gebiet des Kis-Balaton I., Moorschutz im Wald / Renaturierung von Braunmoosmooren, Lübben, 2008.
13. Berke, J. – Bíró, T. - Burai, P. - Kováts, L.D. – Kozma-Bognár, V. - Nagy, T. - Tomor, T. – Németh, T. (2013): Application of Remote Sensing in the Red Mud Environmental Disaster in Hungary, Carpathian Journal of Earth and Environmental Sciences, Vol. 8, No. 2., pp. 49-54., ISSN 1844-489X.
14. Berke, J. - Kozma-Bognár, V. - Burai, P. - Kováts, L.D. - Tomor, T. - Németh, T., Remote Sensing Investigation of Red Mud Catastrophe and Results of Image Processing Assessment, Lecture Notes in Electrical Engineering 152, Innovations and Advances in Computer, Information, Systems Sciences, and Engineering Part I, Chapter 12., pp. 149-156., ISSN: 1876-1100, 2013.
15. Burrough, P.A., Fractal dimensions of landscapes and other environmental data, Nature, Vol.294, 1981, pp. 240-242.
16. Buttenfield, B., Treatment of the cartographic line, Cartographica, Vol. 22, 1985, pp.1-26.
17. Encarnacao, J. L. – Peitgen, H.-O. – Sakas, G. – Englert, G. eds. Fractal geometry and computer graphics, Springer-Verlag, Berlin Heidelberg 1992.
18. Kozma-Bognár, V. – Hegedűs, G. – Berke, J., Fractal texture based image classification on hyperspectral data, AVA 3 International Conference on Agricultural Economics, Rural Development and Informatics, Debrecen, 20-21 March, 2007.
19. Kozma-Bognár, V. – Berke, J., New Applied Techniques in Evaluation of Hyperspectral Data, Georgikon for Agriculture, a multidisciplinary journal in agricultural sciences, Vol. 12./2., 2008.
20. Kozma-Bognár, V. - Berke, J., Determination of Optimal Hyper- and Multispectral Image Channels by Spectral Fractal Structure, International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, December 7-9, 2012.
21. Kozma-Bognár, V. - Berke, J. - Martin, G., Application possibilities of aerial and terrain data evaluation in particulate pollution effects, European Geosciences Union General Assembly, EGU2012-3063, 22-27 April, 2012, Wien.
22. Kozma-Bognár, V., Berke, J. 2013. Entropy and fractal structure based analysis in impact assessement of black carbon pollutions, Georgikon for Agriculture. 17: (2). pp. 53-68. ISSN: 0239-1260.
23. Kozma-Bognár, V., Investigation of Hyperspectral Image Processing and Application in Agriculture, Ph.D. dissertation, University of Pannonia, 2012.
24. Lovejoy, S., Area-perimeter relation for rain and cloud areas, Science, Vol.216, 1982, pp.185-187.
25. Mandelbrot, B. B., The fractal geometry of nature, W.H. Freeman and Company, New York, 1983.
26. Peitgen, H-O. and Saupe, D. eds. The Science of fractal images, Springer-Verlag, New York, 1988.
27. Rényi, A., Onmeasures of information and entropy, Proceedings of the 4th Berkeley Symposiumon Mathematics, Statistics and Probability, 1960:547–561.
28. Sayood, K. Introduction to Data Compression, Elsevier, 2012.
29. Shannon, C. E., A Mathematical Theory of Communication, The Bell System Technical Journal, 27:379–423, 1948.
30. Shannon, C. E., A Mathematical Theory of Communication, The Bell System Technical Journal, 28:623–656, 1948.
31. Shannon, C. E., Prediction and entropy of printed English, The Bell System Technical Journal, 30:50–64, 1951.
32. Turner, M. T., - Blackledge, J. M. – Andrews, P. R., Fractal Geometry in Digital Imaging, Academic Press, 1998.

Emberi jelenlét érzékelése képfeldolgozási módszerekkel

Róbert Kovács, Ambrus Palotai és Attila Fazekas

Debreceni Egyetem, Informatikai Kar

Abstract

Ebben a cikkben ismertetett rendszer egy kis költségvetésű, a szoba egyik sarkában, mennyezethez közeli magasságban elhelyezett kamera képeit elemzi, hogy felismerje a veszélyes szituációkat, azaz az idős ember elesését és az azt követő mozdulatlanságot, valamint a tartós ideig fennálló mozdulatlanságot. A kifejlesztett rendszer az első kísérleteken jól bizonyított, így a fejlesztés és a tesztelés tovább folytatódik.

1. Bevezetés

Az időseket segítő rendszerek célja az, hogy az idős, egyedül élő emberek számára biztonságosabbá tegyék a mindennapi életüket. Az idős emberek számára nagy veszélyt jelentenek az elesések, és azok következményei. Felügyelet és segítség hiányában akár hosszabb idő is eltelhet ebben a testhelyzetben. Gyakran a baleset közbeni sérülések – törés, zúzódás – is megnehezítik a saját erőből történő felállást. Az esést követő mozdulatlanság és az önerőből történő felállás képtelensége miatt az emberre télen legrosszabb esetben akár kihűlés is várhat. Az esések felismerésére különböző technikák léteznek. Az egyik csoportosítási elv szerint megkülönböztethetünk videó, illetve nem videó alapú rendszereket.

A nem videó alapú rendszerek általában valamilyen mechanikus érzékelőn alapulnak, mint például a higanykapcsoló, vagy rezgésérzékelő. Az utóbbi megvalósításáról részletesebben olvashatunk például Y. Zigel és szerzőtársainak cikkében[1]. A testen viselhető érzékelőkön alapuló rendszerek már régebben is megtalálhatók voltak az irodalomban[2] [3] [4]. Napjainkban ezen elvek mentén a leggyakoribb megvalósítás a gyorsulásmérők alkalmazásán alapul[5]. Ezen eszközök viselése sokszor kényelmetlen és megnehezítik a mindennapi életet az idősebb korosztály számára.

A videó alapú rendszerek alkalmazásával ezek a problémák kiküszöbölhetőek, a legnagyobb előnyük, hogy a személynek nem kell semmilyen bonyolult technikai eszközt magukon viselni. Az egyes rendszerek használhatnak egyszerű web, infra, vagy akár Kinect kamerákat is. A kamerák képeinek felhasználásával bizonyos feltételek mellett lehetőség van a valóság 3D-s modellálására. Ennek segítségével jobban megfigyelhető az élettér, és az esések érzékelésére is merőben más módszert kell alkalmazni, mint a két dimenziós képek esetén. Ilyen esetekben leggyakrabban Kinect kamerákat alkalmaznak[6]. Több, hagyományos kamera egyidejű alkalmazásával is mód van a valóság 3-dimenziós rekonstrukciójára [7] [8] [9].

A legegyszerűbb esetben egyetlen kamera is elég lehet a feladat megoldásához, amihez számos ötlet található az irodalomban. Például az egyik lehetőség az árnyék detektálása, mint esés megállapítására szolgáló jelenség[10]. Más szerzők az emberi alak deformitását vizsgálták, amely olyan döntő információval tud szolgálni, ami alapján elkülöníthető az esés a hétköznapi, normál cselekedettől[11]. Az emberi alak szegmentálása után az esést, a befoglaló téglalap szélesség/magasság aránya alapján is számos cikkben vizsgálták [12] [13]. Más publikációban befoglaló téglalap helyett ellipszist illesztettek a képen megjelenő emberre [14].

Az egyes módszerek gyakran kombinálódnak az irodalomban. Például a sebesség alapján történő esésdetektálás körvonalelemzéssel kombinálható a hatékonyabb módszer érdekében [15].

Ebben a cikkben egy hatékony és egyszerű egy kamerán alapuló módszer kifejlesztését tűztük ki célul. A rendszer olcsó hardveren megfelelő hatékonysággal működik, amit az elsődleges kísérletek is alátámasztottak.

2. Rendszerleírás

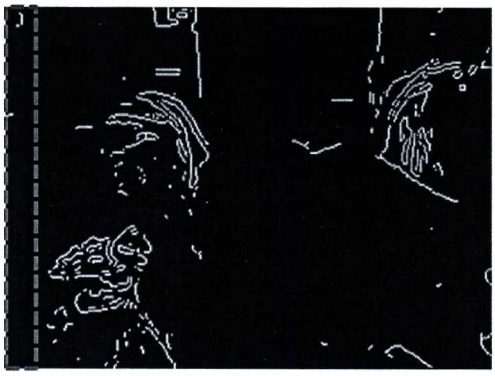
Az esés detektálására kétfajta módszert fogunk alkalmazni. Az egyik az ember képének befoglaló téglalap középpontjának (súlypontjának) nagy mértékű mozgására, míg a másik

ugyanennek a pontnak a sebességére fókuszál. Ezen módszerek által szolgáltatott jelzések kombinálásával hatékonyabban tudjuk felismerni az esés megtörténését. Rendszerünk működését a kulcsfontosságú modulok mentén kívánjuk ismertetni.

2.1. Megfigyelt padlóterület detektálása

A padló területének meghatározása kulcsfontosságú a rendszerünk szempontjából, ugyanis az itt történt esésekkel fogunk csak foglalkozni, így elkerülhetjük a hamis detektálásokat. Másrészt a feldolgozandó adatok mennyiségének csökkentésével a végrehajtás sebessége növelhető.

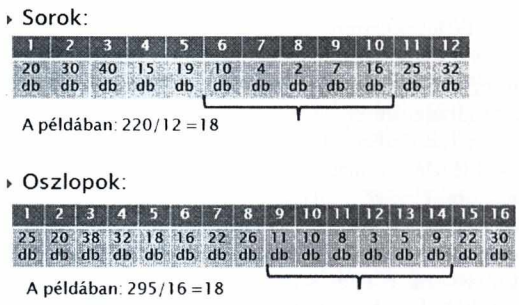
Rendszerünk első lépésként készít egy felvételt a megfigyelt helyiségről. Ezen kép Sobel-féle élképét alapul véve az élben szegény terület egy része lesz a padló. Ennek érdekében az élképet 20 pixel szélességű sávokra bontva (vízszintesen és függőlegesen is) az ott található élpontok számát határozzuk meg. Az 1. ábra egy ilyen, 20 pixel széles, függőleges sávot mutat be.



1. ábra. Egy 20 pixel széles sáv.
A képen egy megfigyelt helyiség éltérképe látható. A fehér képpontok reprezentálják a detektált éleket. A pirossal bekeretezett rész jelképez egy 20 pixel széles sávot.

A sávokban lévő élpontok számát összeadjuk. Ezeket az összegeket egy tömbben eltároljuk a sávok sorrendjében. Ami úgy néz ki, hogy a tömb első értéke az első sávban található pontok összege, a másodikban a második sávé, és így tovább, az utolsó sávig. Két ilyen tömb lesz, az egyik a sor, a másik az oszlop sávok értékeinek tárolására szolgál, ahogy az a 2. ábrán látható.

Ebben a két tömbben meg kell keresni azt a legnagyobb összefüggő sávhalmazt, amiben az átlagosnál kevesebb él szerepel. Miután megvannak az összefüggő sávok, utána kiszámolja a program a rá vonatkozó terület értékeket egy egyszerű területszámítással. Így végül két értéket kapunk, az egyik a vízszintesen legnagyobb összefüggő sávhalmaz területét, míg a másik a függőlegesen legnagyobb összefüggő



2. ábra. Élszegény terület meghatározása.
Sorok és oszlopok szerint is sávokra bontottuk a képet. Ebben a példában átlagosan 18 db élpont található a sávokban. A kapcsok azt a legnagyobb összefüggő sávhalmazt jelölik, ahol az átlagosnál kevesebb éltpont található.

sávhalmaz területét fogja tartalmazni. Ezek közül a nagyobbat kiválasztjuk, és a későbbiekben ez a rész lesz az ajánlott padlóterület. A program lehetőséget biztosít arra, hogy kézi beállítással finomítható legyen a padlóterület. Ezt a finomhangolást csak a rendszer első indulásánál kell elvégezni.

2.2. Háttérfrissítés

A megfigyelt területen megjelenő tárgyakat egy idő után be kell építeni a háttérbe, azaz új háttért kell meghatározni. A háttér elkülönítése az előtértől a kulcs lépés az ember szegmentálásához. A tárgyak háttérbe építésének alapja a sebességük. A nyugalomban lévő tárgyaknak nagyon alacsony a sebességük. Gyakran előfordul, hogy az ember bevisz egy tárgyat a megfigyelt térbe, leteszi, és ő maga tovább mozog a kamera előtt, de már a tárgytól elkülönülve.

Ehhez elsőnek meg kell vizsgálni, hogy az aktuális kameraképen van-e eltérés a háttérmodellhez képest. Ennek érdekében létrehozunk egy különbségképet, amiben megkeressük azt a maximális összefüggő területet, ahol különbség került detektálásra. Az így meghatározott területeket kell további vizsgálatok alá vonni annak érdekében, hogy dönthessünk arról, hogy ezek a részek beépülésre kerülhetnek-e a háttérbe.

A beépítés alapja a tárgyak sebessége. Ha egy tárgy sebessége egy adott korlátnál alacsonyabb, akkor mozdulatlannak tekintjük. Hosszabb ideig mozdulatlan tárgyakat be lehet építeni a háttérbe.

A tárgy beépítése alapesetben úgy történhetne, hogy az aktuális képet tekintjük az új háttérmodellnek a régi helyett. Itt problémát jelenthet, hogy a képen szereplő személy is beépülne a háttérbe. Ennek megoldására azt a módszert választottuk, hogy a háttérmodellnek csak azt a részét frissíti a program, ahol a mozdulatlan tárgy található.

A tesztelések során felmerült egy probléma két objektum érzékelése esetén. Az objektumokat befoglaló téglalapok egyesültek, majd újra szétváltak. Főként azokban az esetekben fordult ez elő, amikor az ember túl közel volt a tárgyhoz, vagy a fényviszonyok nem voltak megfelelőek. Ennek a problémának a megoldására egy logikai változót használunk, amivel kezelni tudjuk a bekövetkezett egyesüléseket. Azt kell figyelnünk, hogy két tárgyat feltételezve, az azokat befoglaló téglalapok egyesülnek-e, majd szétválnak-e. Ha ez bekövetkezik, és újra két objektumként látjuk egy adott időn keresztül, akkor biztosan két objektum van a képen, ellenben hibának tekintjük a jelenséget.

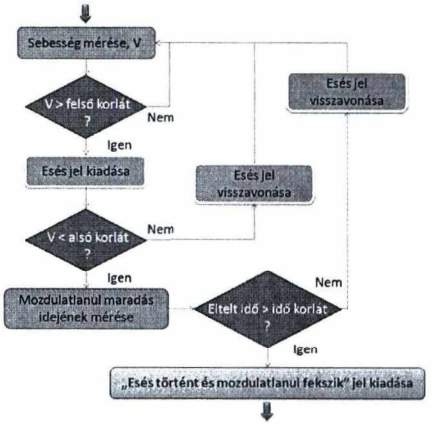
3. Elesés érzékelése sebesség alapján

A sebesség alapján történő esésdetektáláshoz a test pillanatnyi sebességét vesszük alapul. Minden érzékelt testnek van pillanatnyi sebessége, amivel a mozgásukat lehet jellemezni. A szobába belépő emberek általában egyenletes sebességgel mozognak, de ez a sebesség változhat a tevékenységeik során. Ha például megállnak és leülnek, akkor a sebességük nagymértékben lecsökken, lényegében nulla lesz. Ugyanez megfigyelhető a tárgyak esetében is. Ha beviszünk egy tárgyat a helyiségbe, akkor, amíg a kezünkben van, addig a sebessége megegyezik a miénkkel, de amikor letesszük, magára hagyjuk, akkor a sebessége hirtelen nullára csökken. A sebesség ezen változásait használjuk esésdetektálásnál is.

Esésnél az ember sebessége drámaian megváltozik. Feltételezzük, hogy esés előtt egyenletes sebességgel mozgott és az esés hirtelen következett be. El kell különíteni az esést, mint folyamatot, és az esést, mint állapotot. Az előbbi az magát az esés pillanatát jelenti. Itt mérhető a hirtelen és nagymértékben bekövetkező sebességnövekedés. Az utóbbi pedig egy esés utáni állapot, ami azt jelenti, hogy az esés már bekövetkezett. Ez lényegében egy nyugalmi állapot. Ezt az állapotot a sebesség drámai lecsökkenése eredményezi. Ha felismertük az esést az még semmit sem jelent, hiszen esés után az ember általában fel tud állni. Ha egy esés után hosszabb ideig fekve marad az ember, akkor nagy valószínűséggel valami komolyabb baj történt. Ezért az esés után bekövetkező mozdulatlanság időtartamát is vizsgálni kell.

A gyakorlati megvalósítás során egy meghatározott küszöbszám segítségével vizsgáljuk meg, hogy a sebesség kiugró sebesség-e. A másik fontos tényező az esés utáni pillanat. Ha a sebesség egy másik, adott küszöbszám alá kerül, akkor nagy valószínűséggel elesés történt.

A program a működése során folyamatosan figyeli az előtérben lévő elemek sebességét. Ha ez a sebesség meghaladja a felső korlátot, akkor kiad egy jelet, amivel jelzi, hogy talán esés következett be. A jel kiadása előtt meg kell vizsgálni, hogy az észlelt mozgás milyen irányú. Ugyanis esés szempontjából csak a lefelé irányuló mozgások a lényegesek. Majd meg kell vizsgálni, hogy a sebesség drámaian visszaesett-e a következő képkockákon, tehát végbement-e



3. ábra. Sebesség alapján történő elesés érzékelés folyamata.

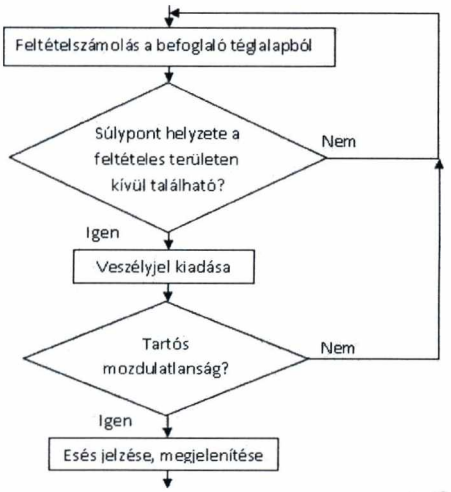
az esés. Ha a sebesség az alsó korlát alá esik, és ott is marad huzamosabb ideig, akkor arra lehet következtetni, hogy esés történt. Ha az alany az esés után csak rövid ideig maradt a földön mozdulatlanul, vagyis fel tudott kelni, akkor a kiadott jelet visszavonja a program. Ennek az a jelentősége, hogy mivel a személy az esést követően fel tudott kelni, már nem szükséges tovább mérni a mozdulatlanságát. Így a kiadott esésjel is visszavonható. A program pedig visszatér az alap sebességméréshez, és újra a kiugró sebességnövekedéseket fogja keresni. Az algoritmus működését a 3. ábra szemlélteti.

A tesztelés során fény derült egy problémára. Ugyanis előfordulhat, hogy az esés végén a megfigyelt személy egyszer vagy akár többször is „visszapattan" a padlóról. Itt persze nem nagymértékű „visszapattanásokra" kell gondolni, hanem kisebbekre, amik nehezen észrevehetőek, de észlelhető sebességnövekedést okoznak. Az esésjel kiadása után a program automatikusan azt kezdi el vizsgálni, hogy az esést követő nyugalmi állapot bekövetkezett-e, és hogy meddig áll fenn. Ha „visszapattanás" történik, akkor a sebesség meghaladja a nyugalmi állapot sebességkorlátját, ezért az esésjel visszavonásra kerül. Ebből az okból az esésjel kiadása után a program egy rövid várakozás után kezdi el vizsgálni, hogy teljesül-e a nyugalmi állapotba kerülés feltétele. Ezzel lényegében az esés utáni „visszapattanások" sebességadatait hagyjuk figyelmen kívül.

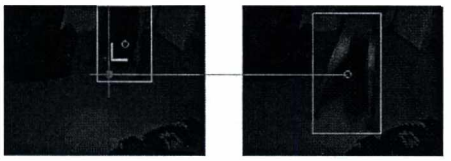
3.1. Esésfigyelés súlypont helyének változása alapján

A befoglaló téglalap méretéhez képest arányos súlypontmozgás jellemzi a tárgy mozgását.

Ha súlypont pozíciója egyik képkockáról a másikra a befoglaló téglalap szélességének negyedével változik, akkor esésjel kiadása történik meg. Egy konkrét esési szituáció kapcsán a súlypont helyváltozására vonatkozó feltételt az 5. ábra szemlélteti.



4. ábra. Esésfigyelés súlypont helyének változása alapján.



5. ábra. Esésjel kiadás feltételének szemléltetése. *Bal oldali kép az esés kezdeti frame-t mutatja, jobb oldali pedig ennek a rákövetkezőjét. A bal oldali képen narancssárga színnel jelzett területen kívül kell lennie a rákövetkező súlypontnak, hogy jelkiadás történjen.*

Az esést háromfajta esemény követheti: azonnali felállás, huzamosabb ideig történő mozdulatlanság, felállásra irányuló próbálkozások.

Az első esetben a személy saját erejéből képes segítséget kérni, így a rendszer szempontjából a detektálás befejeződött. Az utolsó két esetben azonban nagy a veszélye a komolyabb sérülésnek, így azonnali beavatkozás is szükséges lehet. Mindkét esetben a mozdulatlanságot kell figyelni. Kismértékű mozgás az ember mozgási aktivitásától függetlenül is bekövetkezhet. Esésből való felállási próbálkozás hiányában a középpont mozgására/kilengéseire tapasztalati úton meghatározott minimális mozgáskörzet engedélyezett.

Az esésjel kiadása utáni állapotban - az illető elesett, és már fekvő helyzetben van – az ember megpróbálhat felállni. Ilyen esetekben az álló helyzet elérésére irányuló törekvések - súlypont koordinátáinak változását is magában foglaló mozgások - nagyobbak lesznek, mint a „mozdulatlanságnál" megengedett minimális képpont mozgás engedmény. A helyzet kezelését a kisméretű középpont mozgására való kitétel nem oldaná meg, hisz ekkor a legelső próbálkozásnál (pl. kézzel igyekszik kinyomni magát az illető a földtől) a rendszer azt jelezné, hogy ez az említettnél nagyobb a mozgás, tehát az illető már nem mozdulatlan a továbbiakban. Ennek a problémának a megoldására két egymást követő képkockán meghatározásra kerül a befoglaló téglalapok aránya, ami, ha nagyobb, mint 0,75 és a mozdulatlanság vizsgálata már elkezdődött, akkor a súlypont változásra nagyobb (egy másik) korlát lesz érvényben. Ennek az engedménynek célja az, hogy a fekvő helyzetben lévő illető az esés után, ha megpróbál felállni, de eközben visszazuhan, akkor a rendszer ne vonja vissza a veszély fennállásának tényét. Az esés után a leggyakoribb felállásra irányuló mozdulat a kézzel való elnyomás a földtől, ekkor a felsőtest felső része fog mozogni, ami az egész testnek körülbelül 25%-a.

4. Tesztelés

A szoftver tesztelése egy HP Compaq Pressario CQ 56 típusú laptopon történt, amely 4 GB memóriával és AMD Dual-Core 2,10 GHz-es processzorral rendelkezik. A program fejlesztése ugyanezen a gépen, MATLAB 2013a szoftverben történt, Windows 7 64-bites operációs rendszer alatt.

Az esések többféleképpen kerültek kivitelezésre: kamerához közeli és távolabbi elhelyezkedésben, valamint előre, hátra és két oldal valamelyikére történő esésként. A szakirodalomban az esés detektálásával foglalkozó rendszerek esetében ez a négy fajta tartozik a leggyakoribb, esés helyes elfogadását tesztelő események közé [16] [17] [8] [11] [9]. Természetesen részletesebben olvashatunk további lehetőségekről az irodalomban [17] [11].

A tesztelés során a 20 darab előre esésből 19-et helyesen detektált a rendszer. A két oldal valamelyikére bekövetkező esés során nem tapasztalható különbség az adott irányok között, mindkét esetben ugyanolyan eredmények születtek, mint az előre esés esetén. A hátrafele történő eséseknél tapasztalható a többihez képest alacsonyabb ráta, ez általában 70-80% körüli értéket jelent. Abban az esetben, ha az illető nincs jelen a képen, akkor is történhet olyan szituáció, hogy „beesik" a képbe. Ekkor nincs előzetes információ a befoglaló téglalapról, illetve a súlypontról, de a személy például a küszöbben megbotlik, és így a képsorokon már az a jelenet látható, hogy esési fázisban van. A rendszer ekkor 15 szituációból 13-szor helyes döntést hozott. Az esést követő feltételes mozdulatlanság tesztelésénél a rendszer 20 darab eseményből 19-szer azt a döntést hozta helyesen, hogy az illető még mindig veszélyes helyzetben van, és a mozgása elhanyagolható.

Nagyon gyors leguggolás szimulálása esetén a rendszer 18 alkalomból 6-szor jelezte az esést, és 12-szer pedig azt, hogy ez hétköznapi tevékenység. Az előbbitől lassabb (kontrolláltabb) leguggolásoknál azonban 10 alkalomból egyetlen esetben sem minősült esésnek. A teszthelyiségekben való sétálgatás sem eredményez hibás döntést. Lehajlás esetén 15-ből 0 esetben volt téves detektálás, az ágy alá való benézés esetében pedig 10-ből 2 esetben. A leülésre elvégzett 10 kísérlet alatt 3 darab hibás döntés született.

A tesztelések során négy ember által szimulált jeleneteket elemeztettünk a szoftverrel. A kísérletek során így nem csak a mi mozgásformánkat elemezte a rendszer, hanem további két fiatal hölgyét is, akik esést szimuláltak, valamint hétköznapi tevékenységeket is végeztek az adott teszthelyiségekben. A rendszer tesztelése során elért eredményeket az alábbi két táblázatban összegeztük.

Esések detektálása irányok alapján	
Esés (előre, oldalra, hátra) helyes megítélése	45/41
Beesés a képbe helyes megítélése	15/13
Esés helyes megítélése (előző kettő együttvéve)	60/54

1. táblázat.

Esések detektálása különböző mozdulatok esetén	
Guggolás esésnek való hibás érzékelése	28/6
Sétálgatás esésként való hibás megítélése	25/0
Ágy alá benézés, ebből esésnek érzékelve	10/2
Lehajlás cipőt kötni, ebből esésnek érzékelve	15/0
Leülés fotelba, ebből esésnek érzékelve	10/3
Hétköznapi aktivitás helytelen megítélése (előző öt együttvéve)	88/11

2. táblázat.

5. Összefoglalás

Az elsődleges tesztelések alapján a kifejlesztett módszer hatékony esés, illetve a huzamosabb ideig tartó mozdulatlanság felismerésére képes. Az esések megállapítására két módszert alkalmazunk, amelyek egymást segítik. Az egyik módszer a súlypont sebességének változásán, a másik pedig ugyanezen pont koordinátáinak változásán alapul. A rendszer minimális beállítást igényel az indításkor, ami az ajánlott padlóterület esetleges finomhangolását jelenti, majd aztán emberi beavatkozás nélkül automatikusan végzi az otthoni környezetben egyedül élő személy tevékenységeinek elemzését, a szoba felső sarkában elhelyezett kamera által biztosított képkockákon.

További tervek között elsősorban a feldolgozás sebességének gyorsítása szerepel. A jelenlegi feldolgozási idő (200 ms/frame) redukálását bizonyos fokú párhuzamosítással, illetve speciális eszköz igénybevételével is lehet biztosítani. Az irodalomban található FPGA-s rendszer összes számítását egy ilyen kártya végezte, így a mi rendszerünket is érdemes lenne átültetni ilyen eszközre [18]. Gyorsabb feldolgozással több emberes környezetben való használatra is képes lehet a szoftver, ami akár az időseket ellátó szociális otthonban való elhelyezést is lehetővé tenné.

6. Köszönetnyilvánítás

A publikáció elkészítését a TÁMOP-4.2.2.C-11/1/KONV-2012-0001 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

References

1. Y. Zigel, D. Litvak and I. Gannot A Method for Automatic Fall Detection of Elderly People Using Floor Vibrations and Sound—Proof of Concept on Human Mimicking Doll Falls. *IEEE Transactions on Biomedical Engineering*, **56**(12):2858–2867, 2009.
2. C. J. Lord and D. P. Colvin. Falls in the elderly: Detection and Assessment. *in Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Orlando, Florida*, **1**938–1939, 1991.
3. U. Lindemann, A. Hock, M. Stuber, W. Keck and C. Becker. Evaluation of a fall detector based on accelerometers: a pilot study. *Medical and Biological Engineering and Computing*, **43**(5):548–551, 2005.
4. N. Noury, T. Herve, V. Rialle and G. Virone. Monitoring behavior in home using a smart fall sensor and position sensors. *in 1st Annual International, Conference On Microtechnologies in Medicine and Biology, Lyon*, **6**07–610, 2000.
5. J. Cheng, X. Chen and M. Shen. A Framework for Daily Activity Monitoring and Fall Detection Based on Surface Electromyography and Accelerometer Signals. *IEEE Journal of Biomedical and Health Informatics*, **17**(1):38–45, 2013.
6. C. Rougier, E. Auvinet, J. Rousseau, M. Mignotte and J. Meunier. "Fall Detection from Depth Map Video Sequences. *Lecture Notes in Computer Science*, **6719**:121–128, 2011.
7. R. Cucchiara, A. Prati and R. Vezzani. A Multi-Camera Vision System for Fall Detection and Alarm Generation. *Expert Systems*, **24**(5):334–345, 2007.
8. E. Auvinet, L. Reveret, A. St-Arnaud, J. Rousseau and J. Meunier. "Fall detection using multiple cameras. *in 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vancouver, BC*, **2**554–2557, 2008., 2011.

9. E. Auvinet, F. Multon, A. Saint-Arnaud, J. Rousseau and J. Meunier. Fall Detection With Multiple Cameras: An Occlusion-Resistant Method Based on 3-D Silhouette Vertical Distribution. *IEEE Transactions on Information Technology in Biomedicine*, **15**(2):290–299, 2011.

10. Y.-T. Chen, Y.-R. Lin and W.-H. Fang. Detection of Human Fall in Video Using Shadow Information. *Iin IEEE Biomedical Circuits and Systems Conference (BioCAS), Hsinchu*, **2**84-287, 2012.

11. C. Rougier, J. Meunier, A. St-Arnaud and J. Rousseau. Procrustes Shape Analysis for Fall Detection. *Marseille: The Eighth International Workshop on Visual Surveillance - VS2008*, **2**008.

12. J. Tao, M. Turjo, M.-F. Wong, M. Wang and Y.-P. Tan. Fall Incidents Detection for Intelligent Video Surveillance. *in Fifth International Conference on Information, Communications and Signal Processing, Bangkok*, **1**590-1594, 2005.

13. D. Anderson, J. M. Keller, M. Skubic, X. Chen and Z. He. Recognizing Falls from Silhouettes. *in 28th IEEE EMBS Annual International Conference of Engineering in Medicine and Biology Society, New York, NY*, **6**388-6391, 2006.

14. M. Yu, A. Rhuma, S. M. Naqvi, L. Wang and J. Chambers. A Posture Recognition-Based Fall Detection System for Monitoring an Elderly Person in a Smart Home Environment. *IEEE Transactions on Information Technology in Biomedicine*, **16**(6):1274–1286, 2012.

15. B. Mirmahboub, S. Samavi, N. Karimi, S. Shirani. Automatic Monocular System for Human Fall Detection Based on Variations in Silhouette Area. *IEEE Transactions on Biomedical Engineering*, **60**(2):427–436, 2013.

16. Z. Fu, T. Delbruck, P. Lichtsteiner and E. Culurciello. An Address-Event Fall Detector for Assisted Living Applications. *IEEE Transactions on Biomedical Circuits and Systems*, **2**(2):86–96, 2006.

17. N. Noury, A. Fleury, P. Rumeau, A. K. Bourke, G. Ó. Laighin, V. Rialle and J. E. Lundy. Fall detection – Principles and Methods. *in 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Lyon*, **1**663–1666, 2007.

18. M. Humenberger, S. Schraml, C. Sulzbachner, A. N. Belbachir, Á. Srp and F. Vajda. Embedded Fall Detection with a Neural Network and Bio-Inspired Stereo Vision. *in IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Providence, RI*, **2**012.

Contrast Enhancement of Volume Rendered Images

Balázs Csébfalvi, Balázs Tóth

Department of Control Engineering and Information Technology,
Budapest University of Technology and Economics

Abstract

Using classical volume visualization, typically a couple of isosurface layers are rendered semi-transparently to show the internal structures contained in the data. However, the opacity transfer function is often difficult to specify such that all the isosurfaces are of high contrast and sufficiently perceivable. In this paper, we propose a volume-rendering technique which ensures that the different layers contribute to fairly different regions of the image space. Since the overlapping between the effected regions is reduced, an outer translucent isosurface does not decrease significantly the contrast of a partially hidden inner isosurface. Therefore, the layers of the data become visually well separated. Traditional transfer functions assign color and opacity values to the voxels depending on the density and the gradient. In contrast, we assign also different illumination directions to different materials, and modulate the opacities view-dependently based on the surface normals and the directions of the light sources, which are fixed to the viewing angle. We will demonstrate that this model allows an expressive visualization of volumetric data.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and RealismColor, shading, shadowing, and texture.

1. Introduction

The major goal of direct volume rendering is to represent the internal structures of the input 3D data by 2D images. This is a challenging task, since the volume is mapped to a one dimension lower representation, potentially leading to a loss of information. In order to avoid that important features get hidden, the isosurface layers of the data are rendered semi-transparently. If the opacities assigned to the different materials are relatively low then, theoretically, all the voxels contribute to the image. On the other hand, an isosurface of a low opacity is hardly perceivable due to its foggy appearance. The contrast can be increased by increasing the opacity, but then the partially hidden isosurfaces become less apparent. The different layers are difficult to distinguish especially if they contribute to the same regions of pixels. To remedy this problem, we propose a volume-rendering technique that well distributes the visual contributions of the isosurfaces in the image space. Therefore, the visual separation of the different structures becomes easier. The key idea is to illuminate the different layers from different directions and modulate their opacities depending on the illumination direction. To the best of our knowledge, this approach has not been considered for improving volume rendering so far.

2. Related Work

Due to the inherent complexity of volume data, resolving the problem of occlusion has been an important research topic in the area of volume visualization. A simple approach for removing unwanted occluders is clipping. Weiskopf et al. [14] presented techniques for interactively applying arbitrary convex and concave clipping objects. Konrad-Verse et al. [8] proposed the use of a mesh which can be flexibly deformed by the user with an adjustable sphere of influence. Zhou et al. [15] applied distance-based opacity modulation to emphasize and deemphasize different regions. The work of Viola et al. [13] introduced a method for automatically adapting the representation of contextual structures based on occlusion relationships resulting in visualizations similar to common cutaway views.

Another way of resolving occlusion is to employ a sparser representation which selectively emphasizes features. The seminal work of Levoy [9] introduced the idea of modulating

the opacity at a sample position using the magnitude of the local gradient. As homogeneous regions are suppressed, this effectively enhances boundaries in the volume. Rheingans and Ebert [12] presented several illustrative techniques which enhance features and add depth and orientation cues. Csébfalvi et al. [6] enhanced object contours based on the magnitude of local gradients as well as on the angle between the viewing direction and the gradient vector using depth-shaded maximum intensity projection. The concept of two-level volume rendering, proposed by Hauser et al. [7], allows focus+context visualization of volume data. This approach combines different rendering styles, such as direct volume rendering and maximum intensity projection.

A way for defining transfer functions based on occlusion information was introduced by Correa and Ma [4]. The occlusion spectrum enables volume classification based on the ambient occlusion of voxels. By assigning different opacities to certain degrees of occlusion, hidden structures can be uncovered. In further work [5], they also presented the use of visibility histograms which allow semi-automatic generation of transfer functions which maximize the visibility of important structures.

Our work makes use of illumination information to enable the uncluttered simultaneous depiction of multiple structures of interest. Lum and Ma [11] proposed to assign colors and opacities as well as parameters of the illumination model through a transfer function lookup. They applied a two-dimensional transfer function to emphasize material boundaries using illumination. Their work only allows the specification of varying ambient, diffuse, and specular reflection properties for different structures in the volume. Bruckner and Gröller [1] parameterized the shading model to allow simple generation of illustrative effects such as metallic shading. They also introduced style transfer functions [2] which combine color and shading in a single transfer function defined by multiple sphere maps.

Context-preserving volume rendering, as introduced by Bruckner et al. [3], employed the idea of using illumination as an input to the opacity function. While our work is based on a similar notion, we use selective inconsistent illumination in order to maximize the visibility of different features in the volume. One inspiration for our approach is the work of Lee at al. [10]. Their system optimizes the placement of local light sources to enhance the curvature of a polygonal mesh. In this paper, we demonstrate how inconsistent lighting can be used to enhance the visibility of multiple structures in a volume data set.

3. Illumination-Driven Opacity Modulation

Although our new volume-rendering technique is not physically plausible, for the sake of clarity, we still explain it through a fictional physical model. Afterwards, we discuss how to use this model with an appropriate lighting design for expressive volume visualization.

3.1. Fictional Physical Model

We assume that n isosurfaces need to be rendered. We apply the same number of light sources emitting invisible light of different frequencies f_i, where $i \in \{1,2,...,n\}$. None of the isosurfaces attenuate the invisible light rays; thus, the incoming invisible light is completely transmitted without changing its direction, frequency, or intensity. However, each isosurface is sensitive to exactly one frequency. More concretely, if an isosurface s_i is hit by a light ray of frequency f_i then the incoming invisible light is not just completely transmitted, but additionally reflected and its frequency is changed such that it becomes visible. The reflection direction depends on the Bidirectional Reflectance Distribution Function (BRDF) assigned to s_i. Consequently, the shaded color of the intersection point is calculated by evaluating a local shading model according to the BRDF taking the surface normal, the direction of the incoming light, and the viewing direction into account. Although the isosurfaces fully transmit the incoming invisible light, they do attenuate the reflected visible light. Therefore, we assign opacity values to each intersection point depending on how much the given point is illuminated by the corresponding light source. Without a loss of generality, assume that the isosurfaces are diffuse. In this case, based on the Lambertian shading model, the illumination intensity is calculated as $\mathbf{max}(\mathbf{N} \cdot \mathbf{L}, 0)$, where $\mathbf{N}$ is the surface normal and $\mathbf{L}$ is the direction of the light source. The opacity values assigned to the isosurfaces are then modulated by this illumination intensity.

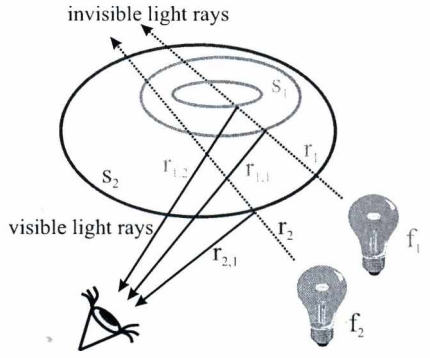


Figure 1: *Illustration of our fictional optical model.*

Figure 1 shows the illustration of our model for two light sources ($n = 2$). Invisible light rays r_1 and r_2 are of frequencies f_1 and f_2, respectively. Both of them are transmitted by the isosurfaces s_1 and s_2. As isosurface s_1 is sensitive to frequency f_1, it reflects r_1. The reflected rays originated from the first and second intersection points are denoted by $r_{1,1}$ and $r_{1,2}$, respectively. Since these reflected rays are visible, they are attenuated by isosurface s_2. Because of the intersec-

tion with the outer layer of s_1, reflected ray $r_{1,2}$ is attenuated by isosurface s_1 as well. The attenuation factors depend on the normal vectors at the corresponding intersection points and the direction of the light source the given isosurface interacts with. Isosurface s_2 is sensitive only to frequency f_2. Therefore, it reflects only the invisible light ray r_2. The reflected ray $r_{2,1}$ is visible and reaches the eye position without attenuation.

For an arbitrary number of isosurfaces, compositing of the visible rays is implemented by the algorithm shown in Listing 1.

```
COLOR RayCompositing(VOLUME volume, RAY ray)
{
  COLOR output = BLACK;
  VECTOR position = ray.origin;
  REAL transparency = 1.0;
  REAL density = 0.0;

  // front-to-back compositing along the ray
  while(IsInBoundingBox(position))
  {
    REAL prevDensity = density;
    position += ray.direction;
    REAL density = volume.Resample(position);

    COLOR color;
    REAL opacity;

    // for each isosurface
    for(int i = 0; i < n; i++)
    {
      REAL threshold = surface[i].threshold;

      // the ith isosurface is intersected
      if((prevDensity - threshold) *
         (density - threshold) < 0)
      {
        color = surface[i].color;
        VECTOR normal = volume.NormalResample(position);
        VECTOR lightDir = lightSource[i].direction;
        REAL lighting = max(dot(normal, lightDir), 0);
        color *= lighting;

        // illumination-driven opacity modulation
        opacity = lighting * surface[i].opacity;
      }
    }

    color *= transparency * opacity;
    transparency *= 1.0 - opacity;
    oputput += color;
  }

  return output;
}
```

Listing 1: *Ray compositing with illumination-driven opacity modulation.*

The outer `while` loop goes through the samples along the ray in front-to-back order. The densities of the current and the previous samples are represented by variables `density` and `prevDensity`, respectively. An intersection point with an isosurface is detected if `density` is greater than the isosurface threshold and `prevDensity` is lower, or vice versa. The inner `for` loop goes through all the isosurfaces and checks the potential intersection points. At each intersection point, a normal is evaluated by gradient estimation, and a lighting coefficient is calculated depending on the directions of the normal and the corresponding light source. The color of an intersection point is the diffuse color assigned to the given isosurface modulated by the lighting coefficient. The opacity is calculated similarly, i.e., the opacity value assigned to the given isosurface is modulated by the lighting coefficient as well. After having the color and opacity values defined for each intersection point, the standard front-to-back compositing is performed.

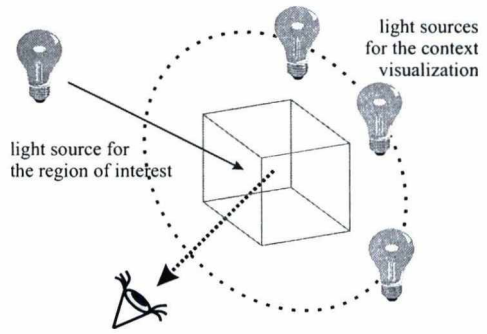
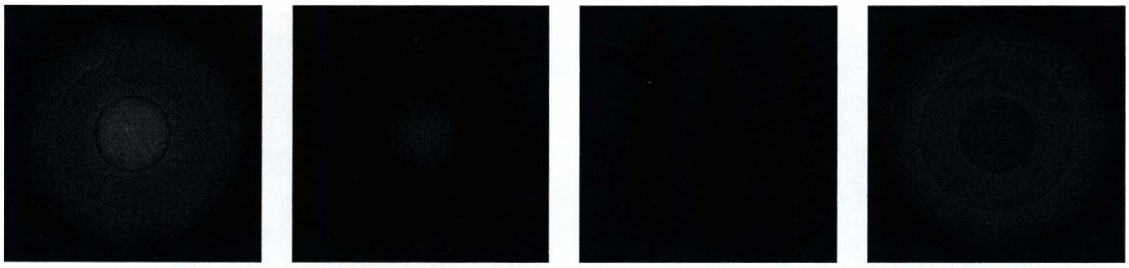


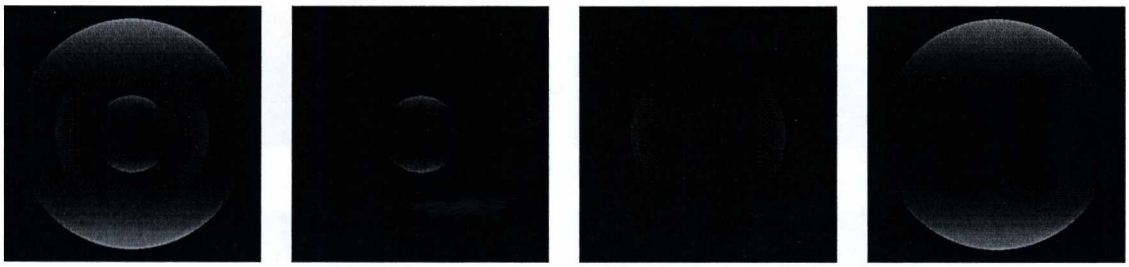
Figure 3: *Arrangement of the light sources around the volume.*

3.2. Illumination Design

The goal of the illumination design is to enhance the region of interest and to preserve the context information at the same time. To render the context, we propose to locate the corresponding light sources along a circle, which is perpendicular to the major viewing direction and has its center in the middle of the volume (see Figure 3). This arrangement of light sources combined with our illumination-driven opacity modulation guarantees that each outer isosurface appears with high contrast and does not significantly suppress the inner isosurfaces. This is demonstrated by Figure 2 that shows three concentric spherical isosurfaces. Using traditional volume rendering, the different layers are of low contrast and their 3D nature is hardly perceivable. Practically, only the inner most red layer seems to be a well-defined spherical surface, while the two outer layers look like fog around the red sphere rather than sharp boundary surfaces. In contrast, applying illumination-driven opacity modulation, the 3D structure of each isosurface is clearly comprehensible. The red and green layers are illuminated from a vertical direction, whereas the blue layer is illuminated from a horizontal direction. Therefore, the different isosurfaces contribute to fairly different regions of the image space. This makes the visual separation of the layers much easier than in case of traditional volume rendering. There the visual separation is supposed to be guaranteed only through the color and opacity information. Even though the red, green, and blue color channels are assigned separately to the three isosurfaces, none of them can fully exploit the range of its own



Traditional volume rendering.



Illumination-driven opacity modulation.

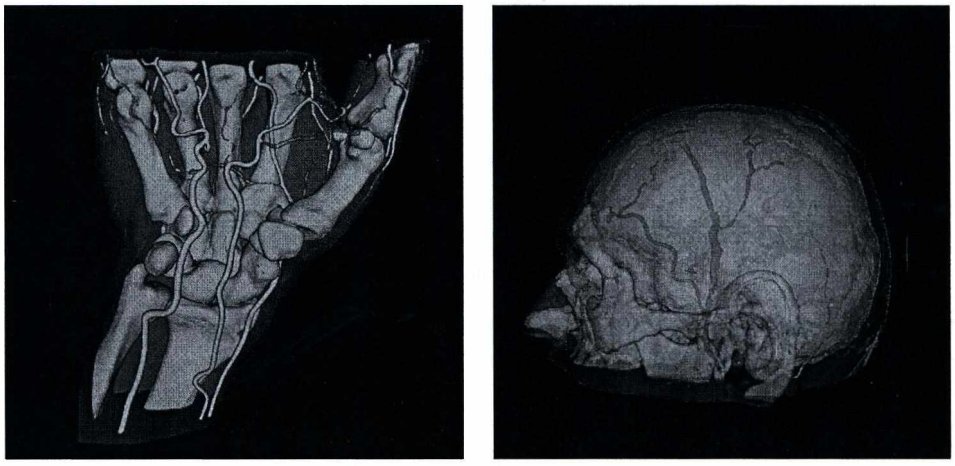
Figure 2: *Comparison of traditional volume rendering to illumination-driven opacity modulation. The contributions of the different isosurfaces are also shown separately. Note that our illumination-driven opacity modulation approach significantly enhances the contrast of each isosurface.*

color channel because the constant opacity modulation drastically limits the highest possible brightness.

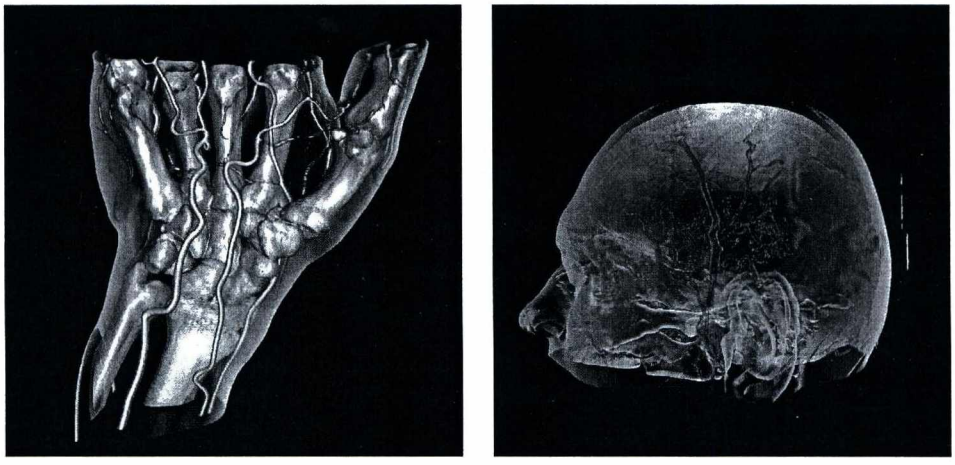
The isosurfaces belonging to the region of interest should not necessarily be lit perpendicularly to the viewing direction. In this case, the illumination direction can be interactively controlled on the fly. In Section 5, we will demonstrate how this additional degree of freedom can be utilized for volumetric data exploration. In fact, the light source assigned to the region of interest is used as a "magic lamp", which can enhance different subsurfaces of the internal structures depending on the illumination direction.

4. Experimental Results

We tested our volume-rendering model on real-world CT data. We intentionally chose test data sets, in which the density thresholds of two important isosurfaces are relatively close to each other. Therefore, their visual separation is a difficult task. One of the test data sets is a CT scan of a human hand, while the other one is the CT scan of a human head. In both data sets, the blood vessels contain contrast agent. Since the density of the contrast agent is nearly the same as that of the bone, it is not easy to separate the blood vessels from the bone structures. The upper two images in Figure 4 show the results generated by classical volume rendering. It is clearly apparent that the blood vessels can hardly be distinguished from the bones based on only the color information assigned according to the densities. However, it is still worthwhile to show the lower-density membrane around the bones and the blood vessels with red color. Note that, using classical alpha-blending with constant opacities, this red membrane is visible only around the thinner vessels, so the meaning of the color information becomes incoherent. Instead, we propose to illuminate the two layers from different directions (see the lower two images in Figure 4). To visualize the hand data set, we illuminate the white layer from the viewing direction and the red layer from a horizontal direction. The yellowish isosurface of the skin, is also illuminated horizontally. Here the interpretation of the red regions is quite clear, as they visually well represent the lower-density membranes. It is also important to mention that the horizontally illuminated skin layer, which represents the context information, does not significantly hide the internal structures (in other words, the region of interest) but its shape is still perceivable. To visualize the head data set, we used almost the same illumination settings as for the hand, but the skull is illuminated vertically and not from the viewing direction, to avoid that it hides the internal blood vessels. It is clearly visible that, compared to classical volume rendering, our visualization model results in images richer in detail. In Figure 6, the contributions of the three different layers are shown separately. Note that, using illumination-driven opacity modulation, the contrast of each isosurface is very well preserved for both test data sets. Such a high contrast is not at all provided by



Traditional volume rendering.



Illumination-driven opacity modulation.

Figure 4: *Comparison of traditional volume rendering to illumination-driven opacity modulation on real-world CT data. Note that, unlike traditional volume rendering, our illumination-driven opacity modulation ensures high contrast for each isosurface.*

classical volume rendering, which significantly reduces the range of colors because of the constant opacity modulation.

5. Interactive Illumination Control

Due to the efficient GPU implementation, the illumination directions can be interactively modified (rendering images of resolution 512×512, we measured frame rates of 8-10 fps on an NVIDIA GeForce GTX 480 graphics card). Especially the lighting of the region of interest is exciting to modify. This is illustrated in Figure 5, where the skin and the blood vessels are lit horizontally, while the illumination of the skull is varying between the vertical and viewing directions. Note that, changing the directions of the light sources, the different features of the data can be enhanced or suppressed in a flexible way. We think that this kind of volume exploration is more intuitive than a simple modification of the transfer function parameters having the illumination directions fixed. Although we have not done a thorough user study yet, the physicians we consulted with found this interaction scheme useful.

6. Conclusion

In this paper, we have introduced illumination-driven opacity modulation for expressive volume rendering. We have shown that lighting the different isosurfaces from different directions guarantees that the layers are visually well separated, as they contribute to fairly different regions in the generated image. Moreover, our method well preserves the

Figure 5: *Illustration of the interactive illumination control. The illumination of the skin and the blood vessels is fixed, while the illumination of the skull is varying between the vertical and viewing directions.*

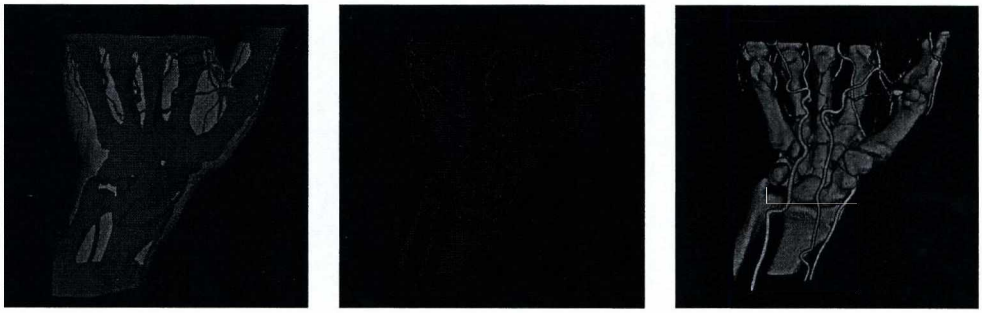
contrast of all the isosurfaces, so their visual interpretation becomes easier. Last but not least, using an efficient GPU implementation, the illumination directions can be interactively controlled, which is especially useful for enhancing the region of interest.

Acknowledgements

This work has been supported by projects TÁMOP-4.2.2.B-10/1–2010-0009 and OTKA K-101527.

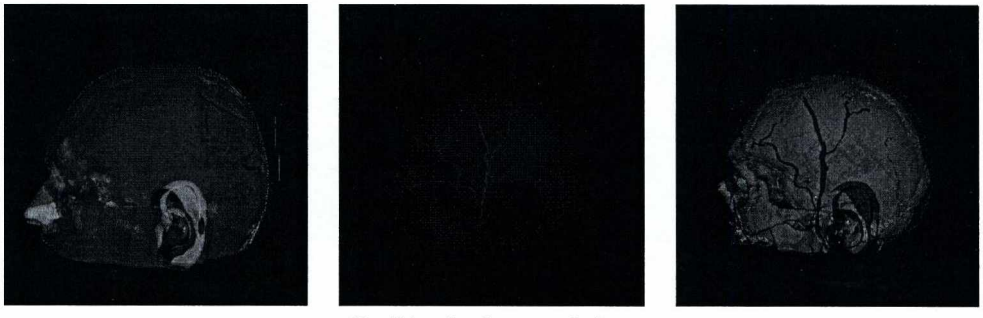
References

1. BRUCKNER S., GRÖLLER M. E.: VolumeShop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization 2005* (2005), pp. 671–678.
2. BRUCKNER S., GRÖLLER M. E.: Style transfer functions for illustrative volume rendering. *Computer Graphics Forum 26*, 3 (2007), 715–724.
3. BRUCKNER S., GRIMM S., KANITSAR A., GRÖLLER M. E.: Illustrative context-preserving exploration of volume data. *IEEE Transactions on Visualization and Computer Graphics 12*, 6 (2006), 1559–1569.
4. CORREA C. D., MA K.-L.: The occlusion spectrum for volume classification and visualization. *IEEE Transactions on Visualization and Computer Graphics 15* (2009), 1465–1472.
5. CORREA C. D., MA K.-L.: Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics 17*, 2 (2011), 192–204.
6. CSÉBFALVI B., MROZ L., HAUSER H., KÖNIG A., GRÖLLER M. E.: Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum 20*, 3 (2001), 452–460.
7. HAUSER H., MROZ L., BISCHI G. I., GRÖLLER M. E.: Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (2001), 242–252.
8. KONRAD-VERSE O., PREIM B., LITTMANN A.: Virtual resection with a deformable cutting plane. In *Proceedings of Simulation und Visualisierung 2004* (2004), pp. 203–214.
9. LEVOY M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37.
10. LEE C. H., HAO X., VARSHNEY A.: Geometry-dependent lighting. *IEEE Transactions on Visualization and Computer Graphics 12*, 2 (2006), 197–207.
11. LUM E. B., MA K.-L.: Lighting transfer functions using gradient aligned sampling. In *Proceedings of IEEE Visualization 2004* (2004), pp. 289–296.
12. RHEINGANS P., EBERT D. S.: Volume illustration: Nonphotorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (2001), 253–264.
13. VIOLA I., KANITSAR A., GRÖLLER M. E.: Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 408–418.
14. WEISKOPF D., ENGEL K., ERTL T.: Interactive clipping techniques for texture-based volume visualization and volume shading. *Visualization and Computer Graphics, IEEE Transactions on 9*, 3 (2003), 298–312.
15. ZHOU J., DÖRING A., TÖNNIES K. D.: Distance based enhancement for focal region based volume rendering. In *Proceedings of Bildverarbeitung für die Medizin 2004* (2004), pp. 199–203.

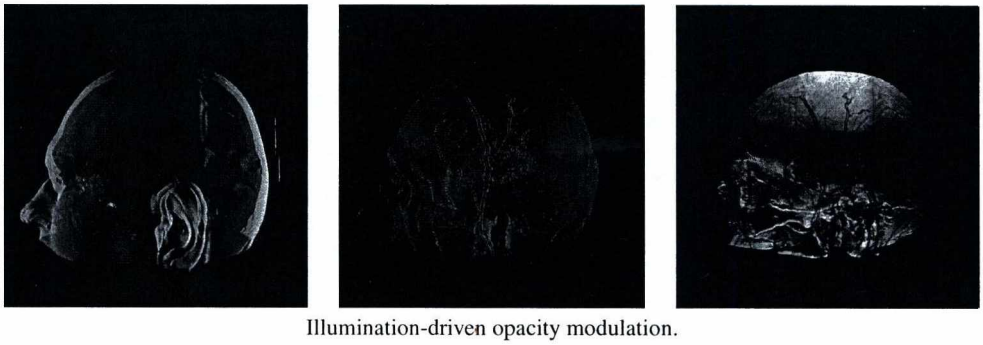


Traditional volume rendering.

Illumination-driven opacity modulation.



Traditional volume rendering.



Illumination-driven opacity modulation.

Figure 6: *Contributions of the different isosurfaces using traditional volume rendering and illumination-driven opacity modulation. Note that, in case of illumination-driven opacity modulation, the three different layers contribute to fairly different regions of the image space. Therefore, their visual separation becomes easier than in case of traditional volume rendering. Furthermore, our model well maintains the contrast of each isosurface.*

Efficient Monte Carlo Methods for Emission Tomography

László Szirmay-Kalos, Balázs Tóth, Gábor Jakab, Péter Gudics, and Milán Magdics

Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Hungary
http://cg.iit.bme.hu

Abstract

Iterative Positron Emission Tomography (PET) reconstruction computes projections between the voxel space and the LOR space, which are mathematically equivalent to the evaluation of multi-dimensional integrals. These integrals are elements of the System Matrix (SM) and can be obtained either by deterministic quadrature or Monte Carlo (MC) methods. Due to the enormous size of the SM, it cannot be stored but integral estimation should be repeated whenever matrix elements are needed. In this paper we show that it is worth using random SM estimates, because this way errors made in projections can compensate each other and do not accumulate to unacceptable values which can happen in case of deterministic approximation.

1. Introduction

In Positron Emission Tomography (PET) we need to find the spatial positron emission density. At a positron–electron annihilation, two oppositely directed photons are generated. Assuming that the electron and the positron are "not moving" before collision, the energy E of the photons can be obtained from the rest mass m_e of the colliding particles and the speed of light c, $E = m_e c^2 = 511$ keV. As these photons fly in the medium, they might collide with the electrons of the material. The probability that collision happens in unit distance is the *cross section* σ. During such collision the photon may get scattered, absorbed according to the *photoelectric effect* and new photon pair may be generated, but in our energy range and in human body only incoherent, i.e. Compton scattering is relevant. When scattering happens, there is a unique correspondence between the relative scattered energy and the cosine of the scattering angle, as defined by the *Compton formula*:

$$\varepsilon = \frac{1}{1+\varepsilon_0(1-\cos\theta)} \quad \Longrightarrow \quad \cos\theta = 1 - \frac{1-\varepsilon}{\varepsilon_0 \varepsilon},$$

where $\varepsilon = E_1/E_0$ expresses the ratio of the scattered energy E_1 and the incident energy E_0, and $\varepsilon_0 = E_0/(m_e c^2)$ is the incident energy relative to the energy of the electron.

The differential of the *scattering cross section*, i.e. the probability density that the photon is scattered from direction $\vec{\omega}$ to $\vec{\omega}'$, is given by the *Klein-Nishina formula*[12]:

$$\frac{d\sigma}{d\omega} \propto \varepsilon + \varepsilon^3 - \varepsilon^2 \sin^2\theta$$

where the proportionality ratio includes the classical electron radius and the electron density of the material. Instead of using these physical parameters explicitly, we use the measured cross section of Compton scattering on energy level 511 keV, i.e. $\varepsilon_0 = 1$ for the representation of the material. From this, the phase function which is supposed to be normalized can be found as:

$$P_{KN}(\cos\theta) = \frac{\varepsilon + \varepsilon^3 - \varepsilon^2 \sin^2\theta}{\int_\Omega \varepsilon + \varepsilon^3 - \varepsilon^2 \sin^2\theta d\omega}.$$

The energy dependence of the Compton scattering cross section can be computed from the scaling factor in the Klein-Nishina formula:

$$\sigma(\varepsilon_0) = \sigma(1) \cdot \frac{\int_\Omega \varepsilon(\varepsilon_0) + \varepsilon^3(\varepsilon_0) - \varepsilon^2(\varepsilon_0)\sin^2\theta d\omega}{\int_\Omega \varepsilon(1) + \varepsilon^3(1) - \varepsilon^2(1)\sin^2\theta d\omega}.$$

As photons travel in the considered volume, they may get scattered several times before they leave the volume or are captured by a detector.

A PET/CT collects the numbers $\mathbf{y} = (y_1, y_2, \ldots, y_{N_{LOR}})$ of simultaneous photon incidents in detector pairs, also called *Lines Of Responses* or *LORs*, and obtains the *material map* of the examined object by a CT scan. The output of the reconstruction method is the *tracer density* function $x(\vec{v})$,

which is approximated in a *finite function series* form:

$$x(\vec{v}) = \sum_{V=1}^{N_{\text{voxel}}} x_V b_V(\vec{v}), \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_{N_{\text{voxel}}})$ are unknown coefficients and $b_V(\vec{v})$ $(V = 1, \ldots, N_{\text{voxel}})$ are *basis functions*, which are typically defined on a *voxel grid*. The correspondence between the coefficients of the tracer density function (voxel values) and the LOR hits is established by the *system sensitivity* $\mathcal{T}(\vec{v} \to L)$ defining the probability that a radioactive decay happened in $\vec{v}$ is detected by LOR L.

The Maximum Likelihood – Expectation Maximization (ML-EM) scheme searches tracer density coefficients $x_1, \ldots, x_{N_{voxel}}$ that maximize the probability of measurement results $y_1, \ldots, y_{N_{LOR}}$ by an iterative algorithm[5], which alternates simulations, called forward projections, and corrective steps based on the computed and measured values.

Forward projection computes the expectation value of the number of hits in each LOR L $\tilde{\mathbf{y}} = (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_{N_{\text{LOR}}})$:

$$\tilde{y}_L = \int_{\mathcal{V}} x(\vec{v}) \mathcal{T}(\vec{v} \to L) \mathrm{d}v = \sum_{V=1}^{N_{\text{voxel}}} \mathbf{A}_{LV} x_V \tag{2}$$

where $\mathcal{V}$ is the domain of the reconstruction, i.e. the field of view of the tomograph, and $\mathbf{A}_{LV}$ is the *System Matrix* (*SM*):

$$\mathbf{A}_{LV} = \int_{\mathcal{V}} b_V(\vec{v}) \mathcal{T}(\vec{v} \to L) \mathrm{d}v \tag{3}$$

Taking into account that the measured hits follow a Poisson distribution, after each forward projection, the ML-EM scheme executes a *back projection* correcting the voxel estimates based on the ratios of measured and computed LOR values:

$$x_V^{(n+1)} = x_V^{(n)} \cdot \frac{\sum_L \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L^{(n)}}}{\sum_L \mathbf{A}_{LV}}, \tag{4}$$

where

$$\tilde{y}_L^{(n)} = \sum_{V'=1}^{N_{\text{voxel}}} \mathbf{A}_{LV'} x_{V'}^{(n)}$$

is the result of forward projecting the current estimate. With a more compact matrix notation, we can also write

$$\mathbf{x}^{(n+1)} = \langle x_V^{(n)} \rangle \cdot \bar{\mathbf{A}}^T \cdot \frac{\mathbf{y}}{\mathbf{A} \cdot \mathbf{x}^{(n)}} \tag{5}$$

where $\langle x_V^{(n)} \rangle$ is an N_{voxel}^2 element diagonal matrix of current voxel values,

$$\bar{\mathbf{A}}_{LV} = \frac{\mathbf{A}_{LV}}{\sum_{L'} \mathbf{A}_{L'V}}$$

is the *normalized SM*, and vector division is interpreted element-wise manner.

This iteration converges to a fixed point $\mathbf{x}^*$ where voxel values are not modified by this formula, i.e. the iteration solves the following equation:

$$\bar{\mathbf{A}}^T \cdot \frac{\mathbf{y}}{\mathbf{A} \cdot \mathbf{x}^*} = \mathbf{1}.$$

In order to study the convergence properties, let us express the activity estimate in step n as $\mathbf{x}^{(n)} = \mathbf{x}^* + \Delta\mathbf{x}^{(n)}$, i.e. with the difference from the fixed point. Substituting this into the iteration formula and replacing the terms by first order Taylor's approximations, we obtain[4]:

$$\Delta\mathbf{x}^{(n+1)} \approx \left(\mathbf{1} - \langle x_V^* \rangle \cdot \bar{\mathbf{A}}^T \cdot \langle \frac{y_L}{\tilde{y}_L^2} \rangle \cdot \mathbf{A} \right) \cdot \Delta\mathbf{x}^{(n)},$$

where $\langle \frac{y_L}{\tilde{y}_L^2} \rangle$ is a N_{LOR}^2 element diagonal matrix of ratios $\frac{y_L}{\tilde{y}_L^2}$. The iteration is convergent if

$$\mathbf{T} = \mathbf{1} - \langle x_V^* \rangle \cdot \bar{\mathbf{A}}^T \cdot \langle \frac{y_L}{\tilde{y}_L^2} \rangle \cdot \mathbf{A}$$

is a *contraction*.

2. Error analysis

To compute forward and back projections, we should consider all points where positrons can be generated and all possible particle paths that can lead to an event in LOR L. A particle path can be described by a sequence of particle–matter interaction points, thus potential contribution $\mathcal{T}(\vec{v} \to L)$ of positron emission point $\vec{v}$ to LOR L is a high-dimensional integral, so are the expected LOR hits in Eq. 2 and SM elements in Eq. 3.

In tomography the size of the SM is enormous since both N_{voxel} and N_{LOR} may exceed 10^8, thus matrix elements cannot be pre-computed and stored, but must be re-computed each time when a matrix element is needed. The standard ML-EM reconstruction scheme is based on the assumption that SM elements as well as forward projections $\tilde{y}_L$ storing the expected number of hits in LORs can be precisely computed. However, this is not the case since re-computation involves numerical quadrature. Deterministic quadrature results in estimations of deterministic error while Monte Carlo (MC) methods result in random values involving random approximation error. To show why MC methods offer a better solution, we first analyze deterministic approximation.

2.1. Deterministic approximation

Deterministic approximation makes a similar error in each iteration step, and thus errors may accumulate during the iteration sequence. To formally analyze this issue, let us first consider that SM estimations may be different in forward projection and back projection, and due to the numerical errors both differ from the exact matrix $\mathbf{A}$. Let us denote the forward projection SM by $\mathbf{F} = \mathbf{A} + \Delta\mathbf{F}$ and the normalized back projection SM by $\bar{\mathbf{B}} = \bar{\mathbf{A}} + \Delta\bar{\mathbf{B}}$. The ML-EM iteration

scheme using these matrices is

$$\mathbf{x}^{(n+1)} = \langle x_V^{(n)} \rangle \cdot \tilde{\mathbf{B}}^T \cdot \frac{\mathbf{y}}{\mathbf{F} \cdot \mathbf{x}^{(n)}}, \tag{6}$$

where

$$\tilde{\mathbf{B}}_{LV} = \frac{\mathbf{B}_{LV}}{\sum_{L'} \mathbf{B}_{L'V}}$$

is the normalized back projector matrix.

The question is how the application of approximate matrices modifies the fixed point $\mathbf{x}^*$ of the iteration scheme. Let us express the activity estimate in step n as $\mathbf{x}^{(n)} = \mathbf{x}^* + \Delta\mathbf{x}^{(n)}$. Substituting this into the iteration formula and replacing the terms by first order Taylor's approximations, we obtain:

$$\Delta\mathbf{x}^{(n+1)} \approx \mathbf{T} \cdot \Delta\mathbf{x}^{(n)} + \langle x_V^* \rangle \cdot (\Delta\mathbf{b} - \Delta\mathbf{f})$$

where

$$\Delta\mathbf{f} = \tilde{\mathbf{A}}^T \cdot \langle \frac{y_L}{\tilde{y}_L^2} \rangle \cdot \Delta\mathbf{F} \cdot \mathbf{x}$$

is the error due to the forward projection estimation, and

$$\Delta\mathbf{b} = \Delta\tilde{\mathbf{B}}^T \cdot \frac{\mathbf{y}}{\tilde{\mathbf{y}}}$$

is the error due to the back projection estimation.

The limiting value will be different from $\mathbf{x}^*$ due to the errors of forward and back projections:

$$\Delta\mathbf{x}^{(\infty)} = \left(\mathbf{A}^T \cdot \langle \frac{y_L}{\tilde{y}_L^2} \rangle \cdot \mathbf{A} \right)^{-1} \cdot (\Delta\mathbf{b} - \Delta\mathbf{f}). \tag{7}$$

According to this formula, matrix $\left(\mathbf{A}^T \cdot \langle \frac{y_L}{\tilde{y}_L^2} \rangle \cdot \mathbf{A} \right)^{-1}$ expresses *error accumulation*, i.e. how the error made in a single step is scaled up during iteration. If the execution of a forward and then a back projection for a point source, i.e. multiplying with matrix $\mathbf{A}^T \cdot \mathbf{A}$ is far from the identity, or many LORs have small or even zero measured value y_L, then error accumulation can be prohibitively large even if the error of a single step is acceptable.

2.2. Random approximation

The error accumulation problem of deterministic approximations can be attacked by applying Monte Carlo quadrature to re-compute projections, because an unbiased Monte Carlo quadrature causes a random error of zero mean, so errors made in different iteration steps can hopefully compensate each other. In this case, projector matrices $\mathbf{F}^{(n)}$ and $\tilde{\mathbf{B}}^{(n)}$ are realizations of random variables and have a different value in each iteration step. Note that as we have to re-compute the matrix elements anyway, the costs of repeating the previous computation or obtaining a statistically independent new estimation are the same.

If projections are computed with *unbiased estimators*, then the expectations of the random projection matrices will be equal to the exact ones:

$$E[\mathbf{F}] = \mathbf{A}, \quad E[\tilde{\mathbf{B}}] = \tilde{\mathbf{A}}.$$

Using random approximation instead of the deterministic approximation of expectation of LOR value $\tilde{y}_L$, we obtain a random variable $\hat{y}_L$ that only approximates the expected value. This random variable depends on the random numbers used to compute the MC estimate, thus it can change in every iteration step[6]. This random and varying error makes the iteration not convergent but the iterated value will fluctuate around the exact solution. To get an accurate reconstruction, the center of the fluctuation should be identical or close to the real solution and its amplitude should be small when actual estimate $\mathbf{x}$ is close to fixed point $\mathbf{x}^*$.

2.2.1. Center of the fluctuation

The center of the fluctuation is the real solution if iterating from the fixed point, the expectation of executing a forward and a back projection for the real solution is still the real solution, i.e.

$$E\left[\tilde{\mathbf{B}}^T \cdot \frac{\mathbf{y}}{\mathbf{F} \cdot \mathbf{x}} \right] = \tilde{\mathbf{A}}^T \cdot \frac{\mathbf{y}}{\mathbf{A} \cdot \mathbf{x}}.$$

Unfortunately, this requirement is not met even by unbiased projectors.

If the forward projector and the back projector are statistically independent, then the expectation of their product is the product of their expectations:

$$E\left[\tilde{\mathbf{B}}^T \cdot \frac{\mathbf{y}}{\mathbf{F} \cdot \mathbf{x}} \right] = E\left[\tilde{\mathbf{B}}^T \right] \cdot E\left[\frac{\mathbf{y}}{\mathbf{F} \cdot \mathbf{x}} \right] = \tilde{\mathbf{A}}^T \cdot E\left[\frac{\mathbf{y}}{\mathbf{F} \cdot \mathbf{x}} \right]. \tag{8}$$

Note that the second factor is generally not equal to $\frac{\mathbf{y}}{\mathbf{A} \cdot \mathbf{x}}$ since the forward projection result is in the denominator, thus its non-linear, reciprocal function determines the expectation value:

$$E\left[\frac{\mathbf{y}}{\mathbf{F} \cdot \mathbf{x}} \right] \neq \frac{\mathbf{y}}{E[\mathbf{F}] \cdot \mathbf{x}} = \frac{\mathbf{y}}{\mathbf{A} \cdot \mathbf{x}}.$$

To examine this for a single element of the vector, let us consider the expectation of the ratio of measured and computed hits, $y_L / \hat{y}_L$. According to the relation of harmonic and arithmetic means, or equivalently to the Jensen's inequality taking into account that $1/\hat{y}_L$ is a convex function, we obtain:

$$E\left[\frac{y_L}{\hat{y}_L} \right] \geq \frac{y_L}{E[\hat{y}_L]} = \frac{y_L}{\tilde{y}_L}. \tag{9}$$

This inequality states that $y_L / \tilde{y}_L$ has a random estimator of positive bias[9]. An intuitive graphical interpretation of this result is shown by Fig. 1. Here we assume that the iteration is already close to the fixed point, so different estimates are around the expected detector hit corresponding to the maximum likelihood. Note that the division in the back projection may amplify forward projection error causing large fluctuations, especially when $\tilde{y}_L$ is close to zero.

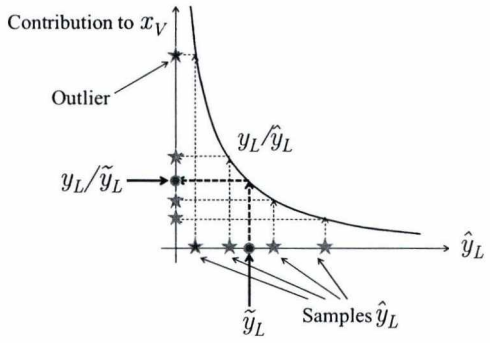


Figure 1: *Expected LOR hit number $\tilde{y}_L$ is approximated by random samples $\hat{y}_L$, which have mean $\tilde{y}_L$. These random samples are shown on the horizontal axis. Back projection computes ratio $y_L/\hat{y}_L$ to obtain voxel updates, which is a non-linear, convex function, resulting in voxel values that may be much higher than the correct value $y_L/\tilde{y}_L$. These overshooting samples are responsible for a positive bias and occasionally cause a large random increase in the voxel value.*

This bias can be tolerated if the forward projector has low variance, thus the generated values $\hat{y}_L$ are in a small interval where the application of a non-linear reciprocal function can be well approximated by a linear one. Another approach is the modification of the ML-EM scheme in order to correct the distorted sample distributions to restore unbiasedness even after the application of the reciprocal function[9].

If the forward projector and back projector are not statistically independent even the factorization of Equ. 8 fails in addition to the problem of non-linear operations. From the point of view of having the random process oscillating around the real solution, we can conclude that the forward projector and the back projector should preferably be independent and the forward projector should have small variance.

2.2.2. Amplitude of the fluctuation

The second requirement of accurate reconstruction in addition to the correct center of the fluctuation is that the fluctuation should have small amplitude, i.e. the variance of applying a complete iteration step is small, especially when the process is close to the fixed point.

To get small variance, the following factors need to be taken into account. The forward projector should be of small variance especially where the LOR value is small, because this LOR value will be the denominator in the back projection formula. The slope of the $1/\tilde{y}_L$ function is $-1/\tilde{y}_L^2$, which scales up the variance of the forward projector especially when $\tilde{y}_L$ gets close to zero.

The variance of the back projector is included in the variance of the result without any amplification. As the back projector matrix elements are in the numerator and the forward projector matrix elements in the denominator, the variance can also be reduced if they are made correlated. When, due to the random approximation of the forward projector a matrix element is overestimated, and thus the corresponding LOR value in the denominator gets greater than needed, the modified voxel value can be made more accurate by simultaneously increasing the matrix element in the numerator, which represents the back projector. So, from the point of view of the oscillation, it seems advantageous to use the same projector for back projection as used for forward projection of the same iteration step. Establishing such a correlation is easy if the same algorithm is used to compute the forward projection and the back projection, only the seed of the random number generation should be set to the same value before back projection as was set before forward projection of this iteration step.

2.2.3. Optimal randomization

Analyzing the mean and the variance of a single ML-EM step involving random projectors, we noted that the accuracy of forward projection is more crucial than that of the back projection, but for the independence or correlation of forward and back projectors, unbiasedness and low variance resulted in different requirements. Unbiasedness requires statistically *independent* forward and back projectors, but low variance due to error compensation needs *correlated* forward and back projectors.

Matrix elements are integrals of Equ. 2 where the integrand is a product of source intensity $x(\vec{v})$ and scanner sensitivity $\mathcal{T}(\vec{v} \to L)$ and integration happens in path space where a "point" corresponds to a path of particles from the emission to the absorption in the detectors. The variance of the MC quadrature depends on the number and distribution of the samples and on the variation of the integrand[10, 7]. There are many possibilities to generate sample paths, which differ in the direction of path building and also in whether roughly the same number of samples is used for each LOR integral, or the sampling process prefers some LORs to others and allocates most of the samples to the preferred ones.

If natural phenomena are directly simulated, then annihilation points are sampled proportionally to their emission density, each sample path is generated with its real probability and can cause a single hit, thus the number of samples in LORs will be proportional to the expected values that are computed. This method is called *voxel driven*[8]. This means that different LORs will be calculated with a similar absolute error. However, when roughly the same number of samples are allocated for each LOR, their error depends just on the variation of their corresponding integral. If the variation is proportional to the integrand, then different LORs are computed with the same relative error. This approach is called *LOR driven*.

Whether it is worth trading more bias for less variance,

i.e. using correlated projectors rather than independent ones, depends on the level of fluctuation. This level can be very high when low contribution LORs are estimated with similar absolute error than high contribution LORs. In the extreme case it can happen that a LOR value is approximated by zero in forward projection, which results in an infinite fluctuation unless the matrix elements corresponding to this LOR are also zero in back projection. Thus, voxel driven methods seem to be better with correlated projectors, but LOR driven approaches prefer lower bias provided by independent projectors.

3. Photon tracing

In *Photon Tracing*, first annihilation point $\vec{v}$ is sampled, then the paths of the two annihilation photons are generated mimicking the free path length and scattering in the real material. If annihilation points are sampled proportional to the activity[3, 2], then we have a voxel driven approach. A voxel driven approach initiates

$$N_V = \frac{x_V N_{\mathrm{PT}}}{\sum_{V'=1}^{N_{\mathrm{voxel}}} x'_V}$$

number of photons from voxel V where N_{PT} is the total number of paths initiated from all photons. When the same

$$N_V = \frac{N_{\mathrm{PT}}}{N_{\mathrm{voxel}}}$$

annihilation sample points are allocated to each voxel, then the method is LOR driven.

The paths of the two annihilation photons are obtained with scanner sensitivity $\mathcal{T}(\vec{v} \to L)$. To do this, an initial direction is drawn from uniform distribution. Two photons are started from the annihilation point and their free paths are sampled to find the photon–material interaction points. At scattering, a new direction is generated mimicking the Klein–Nishina formula, and the photon energy is adjusted according to the Compton law. When one of the photons leaves the detector or its energy drops below the discrimination threshold, the photon pair is lost and no LOR is contributed. If photons hit the detector surfaces of LOR L, the simulation of this path is terminated and the affected SM element A_{LV} is given a contribution equal to $1/N_V$.

Photon tracing, as MC methods in general, results in random SM elements, i.e. projections. The MC simulation is repeated in forward and back projections in each iteration step. The correlation or independence between the forward and back projections can be controlled by whether or not the seed of the pseudo random number generator is reset between these projections. As stated, the accuracy of forward projection is more important, therefore we propose two techniques to increase the accuracy of forward projections without increasing the number of samples, i.e. computation time.

4. Statistical filtering

Recall that the classical ML-EM scheme works with two values in a LOR, the measured value y_L and its mean $\tilde{y}_L$ computed from the actual activity estimate. The expected value is a scalar determined by the activity, the measured value is a realization of a random variable of Poisson distribution having mean $\tilde{y}_L$. Based on the concept of maximum likelihood estimation, the activity estimate is found so that the joint probability of measured values given the expectations obtained from the activity has a maximum.

This classical view should be altered and we should accept the fact that expectation $\tilde{y}_L$ cannot be accurately computed, we can only get random samples $\hat{y}_L$ approximating the expectation. When a LOR is processed we have two random samples, measured value y_L and random estimate $\hat{y}_L$ of its expectation. Fluctuations of the activity can be suppressed if those estimates $\hat{y}_L$ that are unacceptable outliers are replaced by some robust estimate.

To detect whether the pair of measured value y_L and expected value approximation $\hat{y}_L$ is acceptable or an outlier, we should check whether y_L could be a reasonable realization of a Poisson distributed random process of mean $\hat{y}_L$.

Given an observation y_L, the confidence interval for mean $\tilde{y}_L$ with confidence level $1 - \alpha$ is given by the following fairly sharp approximation[1, 11]:

$$F(y_L) \le \tilde{y}_L \le F(y_L + 1)$$

where

$$F(y_L) = y_L \left(1 - \frac{1}{9y_L} - \frac{z_{\alpha/2}}{3\sqrt{y_L}}\right)$$

and $z_{\alpha/2}$ is the standard normal deviate with upper tail integral $\alpha/2$. It means that when our approximation of the expected value satisfies inequality

$$\hat{y}_L < y_L \left(1 - \frac{1}{9y_L} - \frac{z_{\alpha/2}}{3\sqrt{y_L}}\right),$$

then we cannot be confident in this approximation and therefore we correct it. We set $z_{\alpha/2} = 2.1$ because it guarantees that $\hat{y}_L = 0$ is outside the confidence interval if measured value $y_L \ge 1$. This corresponds to 98% percent confidence.

When the current expected value approximation is out of the confidence interval, we should replace it with an acceptable value or completely ignore this LOR in this iteration step. We first try to substitute value $\hat{y}_L$ by the average of the approximations of this LOR in previous iteration cycles. Note that when the algorithm is close to the converged state, the expected LOR hit does not change too much, so averaging the previous estimates helps decrease the variance of the estimator by trading variance to a small bias. We accept the average when it is in the confidence interval. If even the average is out of the confidence interval, then this LOR is skipped during back projection.

5. LOR space blurring

The result of forward projection is the random estimation of LOR hits $\hat{y}_L$, which is usually an unbiased estimate having higher variance. As stated, low variance LOR values are essential especially when the expectation is close to zero. Thus, it is worth trading some bias for reduced variance. We can assume that neighboring LORs get similar number of hits, thus variance can be reduced by 4D spatial blurring, which means that each LOR value is replaced by the average of neighboring LOR values. We used 3×3 size uniform blurring kernel.

6. Results

We examine a simple *flat-land* problem describing a 2D PET where $N_{\text{LOR}} = 2115$ and $N_{\text{voxel}} = 1024$ (Fig. 2). The reason of the application of the flat-land model is that the 2D planar case makes the SM of reasonable size so the proposed methods can be compared to ground truth solutions when the SM is pre-computed and re-used in iteration steps (recall that this is not possible in fully 3D PET because of the prohibiting size). In order to test scattering in the flat-land model, the Compton law and the Klein-Nishina phase function should be converted to be consistent with the planar case. This means that we use only the scattering angle θ and assume that photons remain in the flat-land even after scattering. In 3D, a uniform random rotation around the original direction is also involved, which is now replaced by a random mirroring of 0.5 probability, i.e. by a random decision whether or not θ is multiplied by -1.

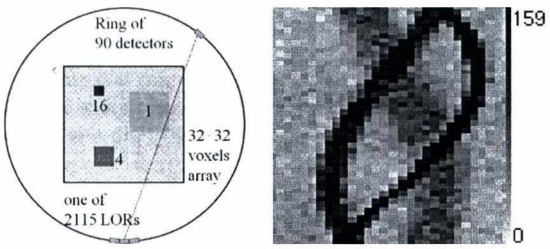


Figure 2: *2D tomograph model: The detector ring contains 90 detector crystals and each of them is of size 2.2 in voxel units and participates in 47 LORs connecting this crystal to crystals being in the opposite half circle, thus the total number of LORs is* $90 \times 47/2 = 2115$. *The voxel array to be reconstructed is in the middle of the ring and has* 32×32 *resolution, i.e. 1024 voxels. The ground truth voxel array has three hot squares of activity densities 1, 4, and 16 and of sizes* 8^2, 4^2, *and* 2^2.

The reference activity is a simple function defined by three hot rectangles of Fig. 2. The measured values are obtained by sampling Poisson distributed random variables setting their means to the product of the SM and the reference activity (right of Fig. 2).

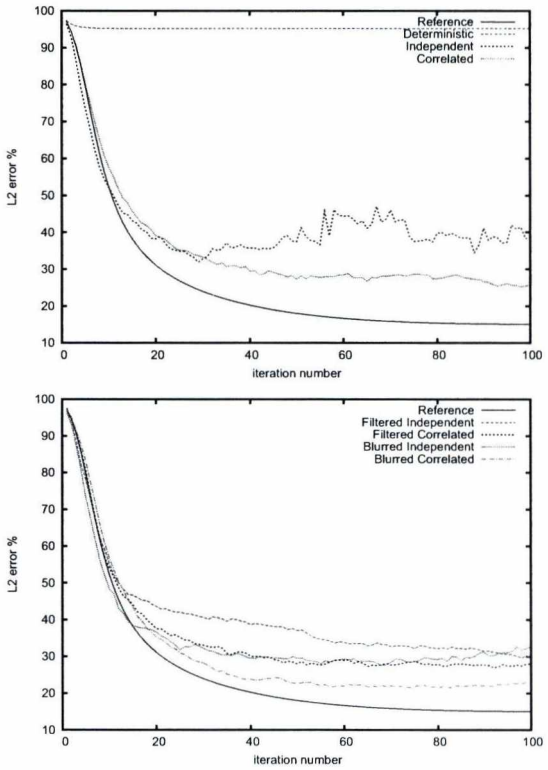


Figure 3: *L_2 errors of a voxel driven direct MC method. We used* 10^4 *samples in all cases.*

The L_2 error curves are shown by Figs. 3 and 4 for voxel driven and LOR driven methods, respectively, and Figs. 5 and 6 show the reconstructed volumes. As in the case of the given tomograph model where all events are captured and of this non uniform phantom the voxel driven method is more efficient, we used 10^4 MC samples in the voxel driven method and $3 \cdot 10^5$ samples for the LOR driven method. The reference matrix is obtained by $2 \cdot 10^6$ LOR driven samples. The measurement is simulated with $5 \cdot 10^4$ samples.

Deterministic iteration, where the same matrix is used in all forward and back projections, has unacceptably poor accuracy if the number of samples is not particularly high. For voxel driven sampling, Correlated projectors provide better solution than Independent projectors. Independent projectors are significantly improved by either statistical filtering or blurring, but only blurring helps Correlated projectors.

In case of LOR driven sampling, Correlated projectors are almost as bad as Deterministic projectors due to the added bias and blurring helps but filtering does not. Independent

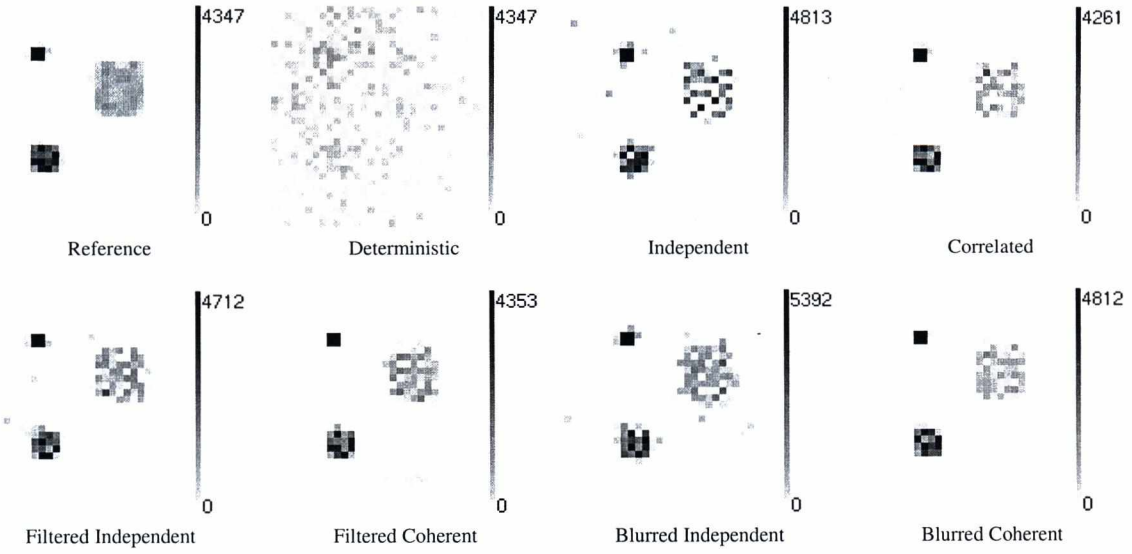


Figure 5: *Reconstructions of the voxel driven method.*

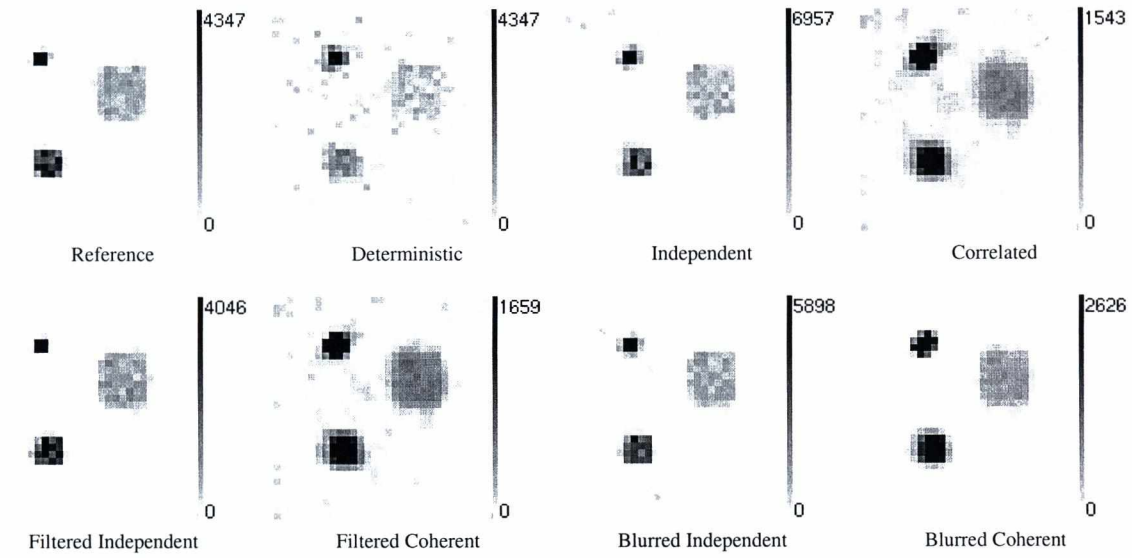


Figure 6: *Reconstructions of the LOR driven method.*

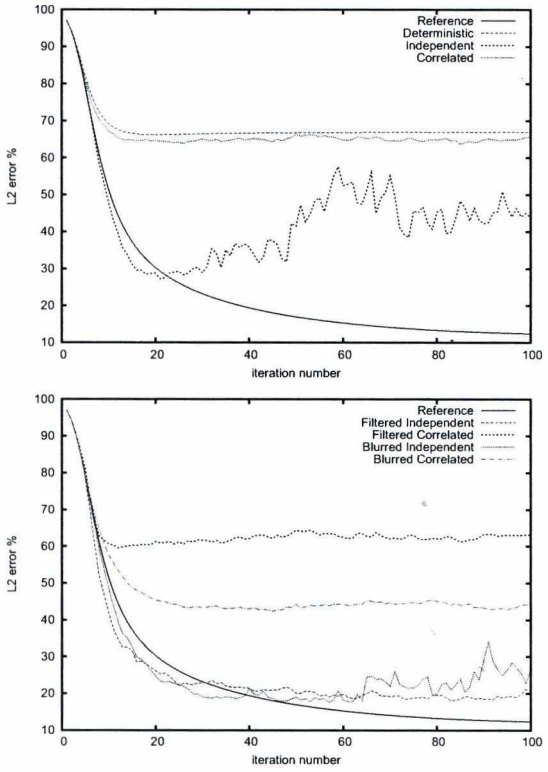


Figure 4: *L_2 errors of a LOR driven direct MC method. We used $3 \cdot 10^5$ samples in all cases.*

projectors are much better here and their high fluctuation is successfully reduced by both filtering and blurring.

7. Conclusions

This paper examined why it is worth using MC estimates to compute forward and back projections in iterative PET reconstruction. We also analyzed the questions whether forward and back projections should be statistically independent or correlated and proposed two techniques to improve the accuracy. These techniques are basically filtering, but statistical filtering operates in the time domain while blurring in the spatial LOR domain.

We concluded that voxel driven methods are worth combining with correlated sampling but LOR driven methods with independent projections. In the future, we examine how the benefits of both approaches can be obtained. Additionally, we also develop more sophisticated LOR blurring methods and instead of applying the same kernel for all contributions, we plan to increase the kernel size depending on the number of scattering events occurred on the photon path.

Acknowledgement

This work has been supported by OTKA K–104476.

References

1. N.E. Breslow and N.E. Day. *Statistical Methods in Cancer Research: Volume 2 – The Design and Analysis of Cohort Studies*. International Agency for Research on Cancer. ISBN 978-92-832-0182-3, 1987.
2. B. Csébfalvi. An evaluation of prefiltered reconstruction schemes for volume rendering. *IEEE Trans. on Vis. and Comp. Graph.*, 14(2):289–301, 2008.
3. B. Csébfalvi and L. Szirmay-Kalos. Monte carlo volume rendering. In *Proceedings of the 14th IEEE Visualization Conference (VIS'03)*, pages 449–456, 2003.
4. M. Magdics, L. Szirmay-Kalos, B. Tóth, and A. Penzov. Analysis and control of the accuracy and convergence of the ML-EM iteration. *LECTURE NOTES IN COMPUTER SCIENCE*, 8353:147–154, 2014.
5. L. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Trans. Med. Imaging*, 1:113–122, 1982.
6. L. Szirmay-Kalos. Stochastic iteration for non-diffuse global illumination. *Computer Graphics Forum*, 18(3):233–244, 1999.
7. L. Szirmay-Kalos. *Monte-Carlo Methods in Global Illumination — Photo-realistic Rendering with Randomization*. VDM, Verlag Dr. Müller, Saarbrücken, 2008.
8. L. Szirmay-Kalos, M. Magdics, and B. Tóth. Multiple importance sampling for PET. *IEEE Trans Med Imaging*, 33, 2014.
9. L. Szirmay-Kalos, M. Magdics, B. Tóth, and T. Bükki. Averaging and metropolis iterations for positron emission tomography. *IEEE Trans Med Imaging*, 32(3):589–600, 2013.
10. L. Szirmay-Kalos and L. Szécsi. Deterministic importance sampling with error diffusion. *Computer Graphics Forum*, 28(4):1056–1064, 2009.
11. Wikipedia. http://en.wikipedia.org/wiki/poisson_distribution, 2013.
12. C. N. Yang. The Klein-Nishina formula & quantum electrodynamics. *Lect. Notes Phys.*, 746:393–397, 2008.

Analysis of Bregman Iteration in PET reconstruction

László Szirmay-Kalos[1] and Gábor Jakab[2]

[1]: Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Hungary
[2]: Mediso Ltd.

Abstract

Positron Emission Tomography reconstruction is ill posed. The result obtained with iterative maximum likelihood estimation is often unrealistic and has noisy behavior. The introduction of additional knowledge in the solution process is called regularization. *Common regularization methods penalize high frequency features or the total variation, thus compromise even valid solutions that have such properties. Another problem is the determination of the strength of regularization, for which no practically useful approach is available in complex problems like PET reconstruction. Bregman iteration offers a better choice enforcing regularization only where needed by the noisy data, thus we incorporate this strategy into our reconstruction algorithm. This paper analyzes Bregman iteration from the point of view of GPU-based PET reconstruction.*

1. Introduction

Tomography reconstruction is the *inverse problem* of particle transport, which requires the iteration of particle transport simulations and corrective back projections [6]. The inputs of the reconstruction are the measured values in *Lines of Responses* or *LORs*: $\mathbf{y} = (y_1, y_2, \ldots, y_{N_{\text{LOR}}})$ The output of the reconstruction method is the *tracer density* function $x(\vec{v})$, which is approximated in a *finite function series* form:

$$x(\vec{v}) = \sum_{V=1}^{N_{\text{voxel}}} x_V b_V(\vec{v}), \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_{N_{\text{voxel}}})$ are the coefficients to be computed, and $b_V(\vec{v})$ $(V = 1, \ldots, N_{\text{voxel}})$ are *basis functions*, which are typically defined on a *voxel grid*. As only non-negative tracer density makes sense, we impose non-negativity requirement $x(\vec{v}) \geq 0$ on the solution. If basis functions $b_V(\vec{v})$ are non-negative, this requirement can be formulated for the coefficients as well: $x_V \geq 0$.

The correspondence between positron density $x(\vec{v})$ and the expected number of hits $\tilde{y}_L$ in LOR L is described by *scanner sensitivity*[8] $\mathcal{T}(\vec{v} \to L)$ that expresses the probability of generating an event in LOR L given that a positron is emitted in point $\vec{v}$ of volume $\mathcal{V}$:

$$\tilde{y}_L = \int_{\vec{v} \in \mathcal{V}} x(\vec{v}) \mathcal{T}(\vec{v} \to L) \mathrm{d}v = \sum_{V=1}^{N_{\text{voxel}}} A_{LV} x_V. \tag{2}$$

where A_{LV} is the *System Matrix* (*SM*):

$$A_{LV} = \int_{\mathcal{V}} b_V(\vec{v}) T(\vec{v} \to L) \mathrm{d}v. \tag{3}$$

A system matrix element is the probability of that a positron is born in $\vec{v}$ with probability density $b_V(\vec{v})$ and generates an event in LOR L. Assuming that photon incidents in different LORs are independent random variables with Poisson distribution, the *Expectation Maximization* (ML-EM) algorithm[7] should maximize the following likelihood function:

$$\log \mathcal{L}(x) = \log \left(\prod_{L=1}^{N_{LOR}} \frac{\tilde{y}_L^{y_L}}{y_L!} e^{-\tilde{y}_L} \right) =$$

$$\sum_{L=1}^{N_{LOR}} (y_L \log \tilde{y}_L - \tilde{y}_L) - \log y_L!.$$

subject to $x_V \geq 0$. Here $\log y_L!$ is independent of the voxel intensities, and thus can be ignored during optimization.

2. Regularization

The ML-EM algorithm, as the solution of inverse problems in general, is known to be ill-conditioned, which means that enforcing the maximization of the likelihood function may result in a solution with drastic oscillations and noisy behavior (Fig. 1). To handle this problem, we have to recognize

ground truth $n = 10$ $n = 30$ $n = 50$ $n = 70$ $n = 90$

Figure 1: *Problem of overfitting to noise in ML-EM iteration when the Three Squares phantom is reconstructed.*

the cases caused by overfitting and exclude them from the solutions.

To recognize when the data is being fitted to noise, we can measure the *quality* of the approximation, i.e. how free the data is from unwanted high frequency characteristics. *Tychonoff regularization*[10] assumes the data set to be smooth and continuous, and thus enforce these properties during reconstruction. However, the typical data in PET reconstruction are different, there are sharp features that should not be smoothed with the regularization method. We need a penalty term that minimizes the unjustified oscillation without blurring sharp features. A better functional is the *Total Variation* (TV) of the solution[5, 3, 4]. In one dimension the total variation measures the length of the path traveled by the function value while its parameter runs over the domain. For differentiable functions, the total variation is the integral of the absolute value of the function's derivative. If the basis functions are piece-wise linear tent-like functions, then the TV in one dimension is

$$TV(x) = \sum_{V=1}^{N_{voxel}} |x_V - x_{V-1}|. \tag{4}$$

In higher dimensions, the total variation can be defined as the integral of the absolute value of the gradient:

$$TV(x) = \int_{\mathcal{V}} |\nabla x(\vec{v})| \mathrm{d}v. \tag{5}$$

There are different possibilities to include regularization information in the reconstruction:

1. *Early termination* stops the iteration when the quality becomes degrading during ML-EM.
2. *Constrained optimization* is based on the recognition that we have two optimization criteria, the likelihood and the quality of the data term, so we can take one of them as an optimization objective while the other as a constraint. However, there are two problems. Firstly, the constraint cannot be well defined since it would require either the likelihood or the quality of the true solution, which are not available. Secondly, constrained optimization is more difficult computationally than unconstrained optimization.
3. Merging the data term and the regularization term into a single objective function where poor quality solutions are penalized by the regularization term.

In this paper, we investigate the third option, the merging of a penalty or regularization term $R(x)$ and the likelihood. The penalty term should be high for unacceptable solutions and small for acceptable ones. In PET reconstruction, under positivity constraint $x(\vec{v}) \geq 0$, we find where the sum of the negative likelihood and a term that is proportional to the total variation has its minimum:

$$E(x) = -\log \mathcal{L}(x) + \lambda R(x). \tag{6}$$

Here, λ is the *regularization parameter* that expresses the strength of the regularizing penalty term.

The objective can be regarded as a functional of tracer density function $x(\vec{v})$, or alternatively, having applied finite element decomposition to $x(\vec{v})$, as a function of voxel values $\mathbf{x} = (x_1, \ldots, x_{N_{voxel}})$:

$$E(\mathbf{x}) = -\log \mathcal{L}(\mathbf{x}) + \lambda R(\mathbf{x}). \tag{7}$$

To minimize the multi-variate objective function with inequality constraints of non-negativity, we can use the Kuhn-Tucker conditions, which lead to:

$$x_V \frac{\partial E(\mathbf{x})}{\partial x_V} = x_V \left(\lambda \frac{\partial R(x(\vec{v}))}{\partial x_V} - \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L} + \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \right) = 0$$

for $V = 1, 2, \ldots, N_{voxel}$. Rearranging the terms, we obtain the following equation for the optimum:

$$x_V \left(\lambda \frac{\partial R(x(\vec{v}))}{\partial x_V} + \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \right) = x_V \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L}.$$

There are many possibilities to establish an iteration scheme that has a fix point satisfying this equation. A provably converging backward scheme would solve the following equation in every step:

$$x_V^{(n+1)} \left(\lambda \frac{\partial R(x^{(n+1)}(\vec{v}))}{\partial x_V} + \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \right) = x_V^{(n)} \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L^{(n)}}$$

where

$$\tilde{y}_L^{(n)} = \sum_{V=1}^{N_{voxel}} A_{LV} x_V^{(n)}$$

is the expected number of hits computed from the activity distribution available in iteration step n.

On the other hand, the forward scheme, also called *One Step Late* (OSL) scheme, does not need to solve any equation in a single iteration step:

$$x_V^{(n+1)} = \frac{x_V^{(n)} \cdot \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L^{(n)}}}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} + \lambda \frac{\partial R(x^{(n)}(\vec{v}))}{\partial x_V}}.$$

However, the convergence of this scheme is proven only if $\lambda \partial R(x^{(n)}(\vec{v}))/\partial x_V$ is constant[7].

In this paper, we use this One Step Late option with Total Variation regularization and demonstrate that it is stable in practical situations. The partial derivatives of the total variation functional are:

$$\frac{\partial TV(x(\vec{v}))}{\partial x_V} = \int_{\mathcal{V}} \frac{\partial \vec{\nabla} x(\vec{v}) / \partial x_V}{\sqrt{|\vec{\nabla} x(\vec{v})|^2}} \mathrm{d}v. \tag{8}$$

The integrand of this formula has a singularity where the gradient is zero, which needs to be addressed.

The singularity can be avoided by re-defining the TV term by adding a small positive constant β:

$$TV(x) = \int_{\mathcal{V}} \sqrt{|\vec{\nabla} x(\vec{v})|^2 + \beta} \mathrm{d}v.$$

Note, however, that this modified TV term will not be invariant to sharp features anymore, and introduces some blurring. Instead of adding a small constant to the gradient, which eventually introduces blurring, primal-dual methods increase the free variables of the search to solve the problem of the singularity of the TV term[3, 4].

3. Analysis of TV regularization

If both the negative likelihood and the regularization term are convex, then the objective function has a global minimum, which moves closer to the minimum of the regularization term if λ is increased. However, this means that the optimum of the likelihood is modified even if the noise level is small and thus no regularization is needed. Classical regularization terms measure the distance between the constant function and the actual estimate. The variation of the true solution is also penalized, so the optimum would be modified. As a result, Tychonoff regularization produces blurred, oversmoothed edges, TV regularization reduced contrast solutions having stair-case artifacts. The optimal weight can be obtained with *Hansen's L-curves*[2], which states that the optimal λ is where the $(-\log \mathcal{L}(x_\lambda)), TV(x_\lambda))$ parametric curve has maximum curvature (note that in the final solution x depends on λ, so both the likelihood and the total variation will be functions of the regularization parameter). However, the algorithm developed to locate the maximum curvature points assumes Tychonoff regularization, and there is no practically feasible generalization to large scale problems based on TV.

To examine these artifacts formally, let us consider the one dimensional case when the TV is defined by Equ. 4. The partial derivatives of the TV functional is

$$\frac{\partial TV(x(\vec{v}))}{\partial x_V} =$$

$$\frac{\partial \sum_{V=1}^{N_{voxel}} |x_V - x_{V-1}|}{\partial x_V} = \begin{cases} 2 & \text{if } x_V > x_{V-1} \text{ and } x_V > x_{V+1}, \\ -2 & \text{if } x_V < x_{V-1} \text{ and } x_V < x_{V+1}, \\ 0 & \text{otherwise.} \end{cases}$$

Thus the iteration formula becomes independent of the regularization when $x_V^{(n)}$ is not a local extremum:

$$x_V^{(n+1)} = \frac{x_V^{(n)} \cdot \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L^{(n)}}}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV}}.$$

When $x_V^{(n)}$ is a local maximum, regularization makes it smaller by increasing the denominator by 2λ:

$$x_V^{(n+1)} = \frac{x_V^{(n)} \cdot \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L^{(n)}}}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} + 2\lambda}.$$

Similarly, when $x_V^{(n)}$ is a local minimum, regularization makes it greater by decreasing the denominator by 2λ.

This behavior is advantageous when local maxima and minima are due to the noise and overfitting. Note that the regularization has a discontinuity when x_V is equal to one of its neighbors so iteration is likely to stop here, resulting in staircase-like reconstructed signals. On the other hand, when the true data really has a local extremum, regularization decreases its amplitude, resulting in contrast reduction. Assume, for example, that the true data is a point source like feature that is non-zero only in a single point V. For this value, the ratio of the reconstruction with and without regularization, i.e. the contrast reduction is

$$C(\lambda) = \frac{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV}}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} + 2\lambda} \approx 1 - \frac{2\lambda}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV}}.$$

Note that contrast reduction is not uniform for different voxels but depends on *sensitivity* $\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV}$, which expresses the probability that a positron born with probability density of basis function b_V is detected by LOR L. The sensitivity is high in the center of a fully 3D PET but can

be very small close to the exits of the gantry, causing over-regularization here. To attack this problem, we propose an *Equalized One Step Late* (EOSL) scheme:

$$x_V^{(n+1)} = \frac{x_V^{(n)} \cdot \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L^{(n)}}}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \left(1 + \lambda \frac{\partial R(x^{(n)}(\vec{v}))}{\partial x_V}\right)}.$$

4. Bregman iteration

An optimal regularization term would have its minimum at the ground truth solution when $x = x_{\text{true}}$. In this case, regularization would not compromise the solution when regularization is not needed, and would become larger when the $R(x)$ is significantly different from $R(x_{\text{true}})$. Thus, an optimal regularization term would measure the "distance" $D(x, x_{\text{true}})$ between x and x_{true}. An appropriate distance function is the *Bregman distance*[1, 12, 11] that can be based on an arbitrary convex penalty term $R(x)$:

$$D(x, x_{\text{true}}) = R(x) - R(x_{\text{true}}) - \langle \mathbf{p}, x - x_{\text{true}} \rangle$$

where $\mathbf{p}$ is the gradient of $R(x)$ at x_{true} if it exists:

$$\mathbf{p}_V = \frac{\partial R(x(\vec{v}))}{\partial x_V}.$$

In practice, we do not know the true solution, so it is replaced by an earlier estimate $x^{(k)}$. Note that if the regularization term is linear between the true solution and $x^{(k)}$, then this approximation is precise since

$$D(x, x_{\text{true}}) = D(x, x^{(k)}).$$

Total variation is based on the absolute value function, which results in piece-wise linear regularization term, making it particularly attractive for Bregman iteration.

Replacing the TV functional by the Bregman distance induced by the TV functional, the goal of the optimization is

$$E(\mathbf{x}) = -\log \mathcal{L}(\mathbf{x}) + \lambda D(\mathbf{x}, \mathbf{x}^{(k)}). \tag{9}$$

Using the result of an arbitrary regularization for the special case of the Bregman distance, we obtain the following forward iteration:

$$x_V^{(n+1)} = \frac{x_V^{(n)} \cdot \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L}}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} + \lambda \frac{\partial D(x^{(n)}(\vec{v}), x^{(k)}(\vec{v}))}{\partial x_V}}.$$

where the partial derivatives of the Bregman distance are:

$$\frac{\partial D(x(\vec{v}), x^{(k)}(\vec{v}))}{\partial x_V} = \frac{\partial TV(x(\vec{v}))}{\partial x_V} - \mathbf{p}_V^{(k)}. \tag{10}$$

When k is incremented, the gradient vector of the total variation, $\mathbf{p}$, should be updated. This can be done directly considering the criterion of optimality. Iteration arrives at the optimum when the derivative of the objective function is zero:

$$0 = \frac{\partial E(\mathbf{x})}{\partial x_V} = \left(\lambda \frac{\partial D(x(\vec{v}))}{\partial x_V} - \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L} + \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV}\right) =$$

$$\lambda \frac{\partial TV(x(\vec{v}))}{\partial x_V} - \lambda \mathbf{p}_V^{(k)} - \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L} + \sum_{L=1}^{N_{LOR}} \mathbf{A}_{LV}.$$

Setting $\mathbf{p}^{(k+1)}$ to the gradient of the TV term and expressing it from the equation, we get

$$\mathbf{p}_V^{(k+1)} = \mathbf{p}_V^{(k)} + \frac{1}{\lambda} \sum_{L=1}^{N_{LOR}} \left(\mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L} - \mathbf{A}_{LV}\right).$$

However, this step size may to too big and may make the iteration unstable, so we scale it and use the following update formula:

$$\mathbf{p}_V^{(k+1)} = \mathbf{p}_V^{(k)} + \delta \sum_{L=1}^{N_{LOR}} \left(\mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L} - \mathbf{A}_{LV}\right),$$

where δ is a controllable step size. The reconstruction algorithm based on Bregman iteration can be summarized as:

for $k = 1$ **to** K **do** *// Bregman iterations*
  for $n = (k-1)K$ **to** $kK - 1$ **do** *// subiterations*
    $x_V^{(n+1)} = x_V^{(n)} \cdot \frac{\Sigma_L \mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L}}{\Sigma_L \mathbf{A}_{LV} + \lambda \left(\frac{\partial TV}{\partial x_V} - \mathbf{p}_V^{(k)}\right)}$.
  endfor
  $\mathbf{p}_V^{(k+1)} = \mathbf{p}_V^{(k)} + \delta \Sigma_L \left(\mathbf{A}_{LV} \frac{y_L}{\tilde{y}_L} - \mathbf{A}_{LV}\right)$.
endfor

Both TV regularization and Bregman iteration require the computation of the derivative of the total variation with respect to each coefficient in the finite element representation of the function to be reconstructed. If the finite element basis functions have local support, then the derivative with respect to a single coefficient depends just on its own and its neighbors' values. Thus, this operation becomes similar to a image filtering or convolution step, which can be very effectively computed on a parallel machine, like the GPU[9].

5. Results

We examine a simple 2D PET model where a SM of dimensions $N_{\text{LOR}} = 2115$ and $N_{\text{voxel}} = 1024$ (Fig. 2).

We considered three phantoms, the *Three Squares* where each square has 64 Bq activity, the *Point Source* of 20 Bq activity, the *Homogeneity* of $2 \cdot 10^4$ Bq activity and using a Monte Carlo particle transport method, we simulated a 5 sec long measurement for all these phantoms (Fig. 3). It means that the Three Squares phantom is projected with 1000 photon pairs, the Point source with 100 photon pairs, and the

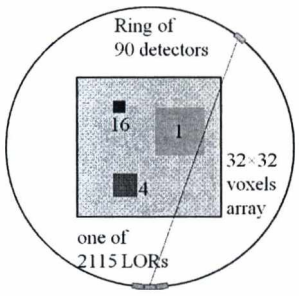


Figure 2: *2D tomograph model: The detector ring contains 90 detector crystals and each of them is of size 2.2 in voxel units and participates in 47 LORs connecting this crystal to crystals being in the opposite half circle, thus the total number of LORs is* $90 \times 47/2 = 2115$. *The voxel array to be reconstructed is in the middle of the ring and has* 32×32 *resolution, i.e. 1024 voxels. The ground truth voxel array of the Three Squares phantom has three hot squares of activity densities 1, 4, and 16 and of sizes* 8^2, 4^2, *and* 2^2.

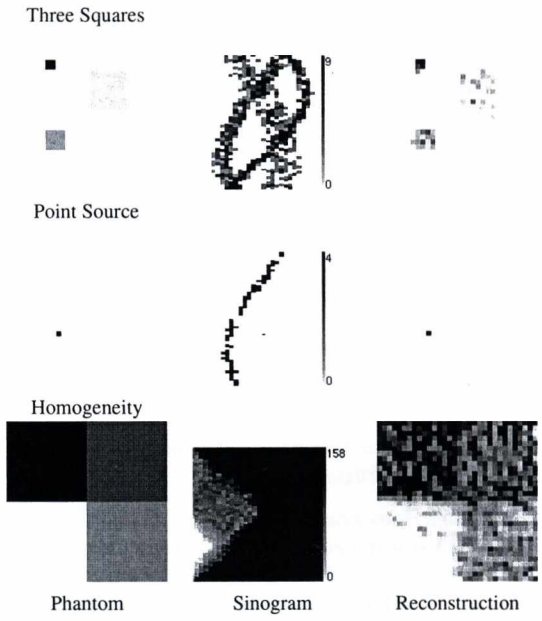


Figure 3: *The three phantoms used in the experiments, their random projections in sinogram parametrization, and the reconstructions without regularization.*

Homogeneity with 10^5 photon pairs, resulting in measured data having 1.21 Signal-to-Noise ratio (SNR) for the Three Squares, 1.07 SNR for the Point source and 1.69 SNR for the Homogeneity. Only geometric effects were simulated, we ignored scattering and absorbtion in the phantom. Three Squares is formed by three active squares of increasing size, and represents a realistic example between the other two extremes. Point Source and the Homogeneity represent two extreme cases. Point Source has a high variation since it has just a single voxel where the activity is non-zero and is well determined by the measurement. Thus, the reconstruction of Point Source would not need regularization, and regularization would just slow down the convergence. Homogeneity is formed by four constant activity squares, so the activity distribution is rather flat and the measurement is quite noisy. Such cases badly need regularization.

First we compared TV regularization strategies including the classical TV, One Step Late (OSL) and Equalized One Step Late (EOSL) for the Three Squares phantom. The L_2 error curves with respect to iteration number n are shown by Fig. 4. Note that there is no significant difference between the classical TV and the OSL options, while the OSL is much easier to compute. The Equalized version seems to be better when overregularization happens and poorer when the regularization is not strong enough, but the fact is that for the 2D tomograph model, the probability that a voxel event is detected is quite uniform thus the sensitivity is already equalized. The apparent difference is due to the fact that in the equalized option, the regularization parameter is scaled down by the sensitivity value, so similar results could be obtained if the global regularization parameter λ is scaled up in the equalized case.

Then, we considered the Bregman schemes and used only the TV-OSL as a reference. The L_2 error curves with respect to iteration number n are depicted by Figs. 5, 6, and 7, the reconstruction results by Figs. 8, 9, and 10. We set Bregman period K to 10 and step size δ to 1 in all tests.

We can make the following observations. Figs. 5 and 8 demonstrate the reconstruction of the Three Squares. This is a low statistics measurement where regularization is necessary. Total variation regularization and Bregman iteration with period 10 result in error level 30%, which remains the same for Bregman iteration even for strong regularization but gets slightly worse for total variation regularization. The One Step Late version of Bregman iteration outperforms all other methods despite the fact that it tends to oscillate when regularization is too strong.

The reconstruction of the Point phantom can be evaluated in Figs. 6 and 9. This measurement is of high statistics, so the L_2 error decreases even if no regularization is applied. On the other hand, the phantom has a high variation, so here regularization slows down the convergence and reduces the contrast. Total variation regularization stops the convergence on error levels 13%, 25% and 60% when the parameter of regularization is $\lambda = 0.01$, $\lambda = 0.02$, and $\lambda = 0.05$, respectively, and there is no difference whether or not the One Step Late option is used. Bregman iteration can help and make

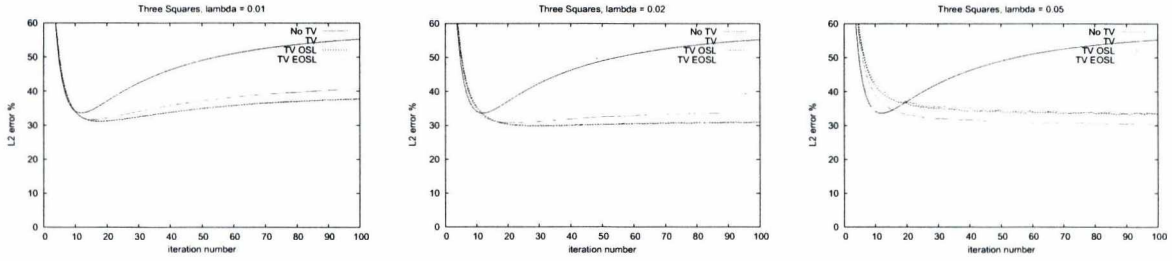


Figure 4: *L_2 error curves of the Three Squares reconstruction with TV regularization options.*

the process still converging, but the convergence is slower than without regularization.

When the Homogeneity is reconstructed (Figs. 7 and 10), regularization is indeed needed since when there is no regularization (No TV), the error grows after an initial reduction due to overfitting. TV-OSL and Bregman iteration perform similarly for this phantom. Regularization parameter $\lambda = 0.01$ seems to be too small since the error slightly increases. When $\lambda = 0.02$ the error converges to 21%, and with very strong regularization set by $\lambda = 0.05$, the error is also stable but at a higher level. The similarity of TV regularization and Bregman iteration in this case is explained by the fact that the phantom itself is flat, so its total variation is modest. The interesting fact is that One Step Late Bregman iteration performs better than both TV and Bregman regularization for lower regularization parameters, but gets the error to oscillate when the regularization is too strong. This oscillation is due to the fact that the step size of the optimization algorithm gets two large, so it steps over the minimum.

6. Conclusions

Based on the analysis and simulation results we can conclude that the One Step Late option works both for TV regularization and Bregman iteration, making their implementation fairly simple. Bregman iteration is a promising alternative to TV regularization and its one step late evaluation not only makes it more efficient to compute but also helps improving the error reduction. The only drawback of One Step Late Bregman iteration with respect to One Step Late TV regularization is that we should maintain another voxel array $\mathbf{p}_V$ in addition to activity values x_V. This not only doubles the storage space but also slows down the GPU implementation where the data transfer is the bottleneck.

There are two issues that need to be addressed in our future work. Firstly, in case of overregularization One Step Late Bregman iteration may have oscillating error curves which means that the process jumps over the minimum in each step. We plan to consider the cases when the derivative

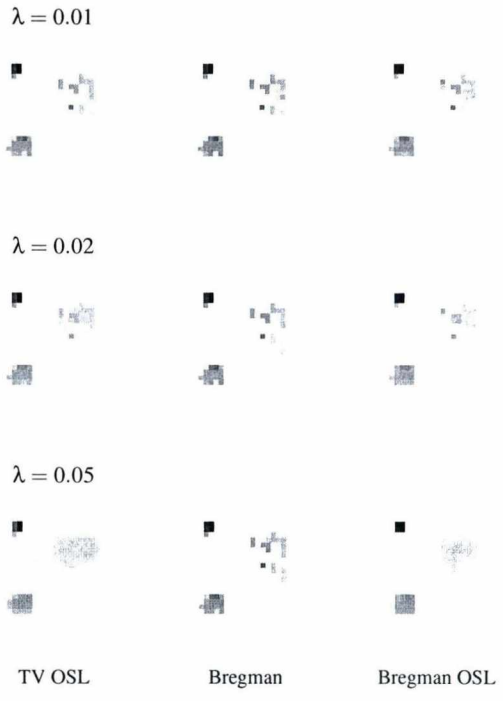
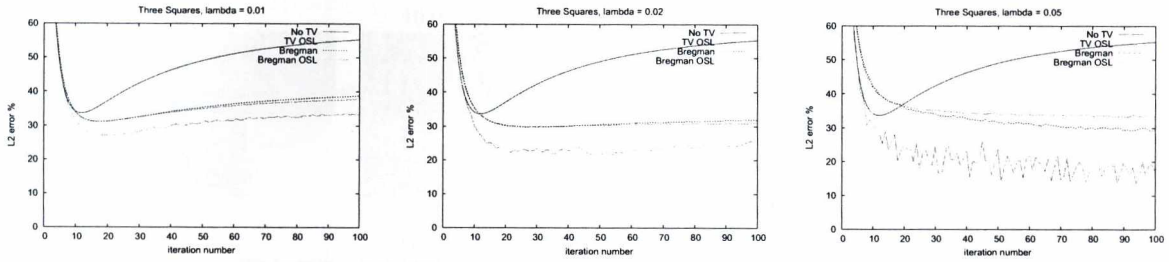


Figure 8: *Reconstructions of the Three Squares phantom.*

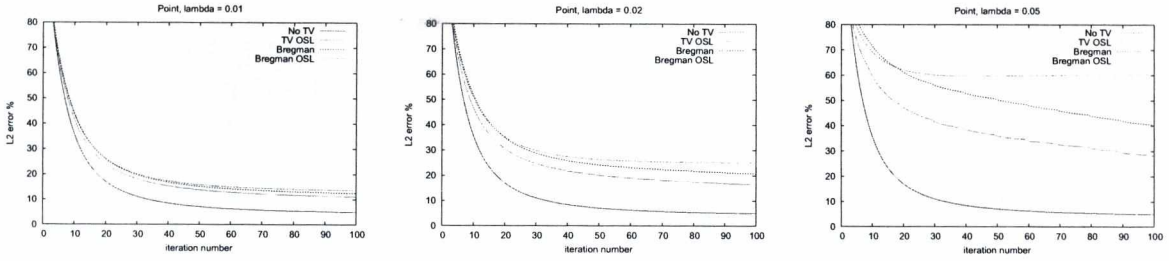
of the regularization term changes its sign and reduce the step size to enforce convergence.

Secondly, when the phantom has very high variation like in the case of Point phantom, even Bregman iteration slows down the convergence. This problem can be addressed by reducing the Bregman period, when the original convergence speed can be restored. However, in other cases, when regularization is badly needed, too low Bregman period cannot prevent the process from divergence. Thus our goal is to find



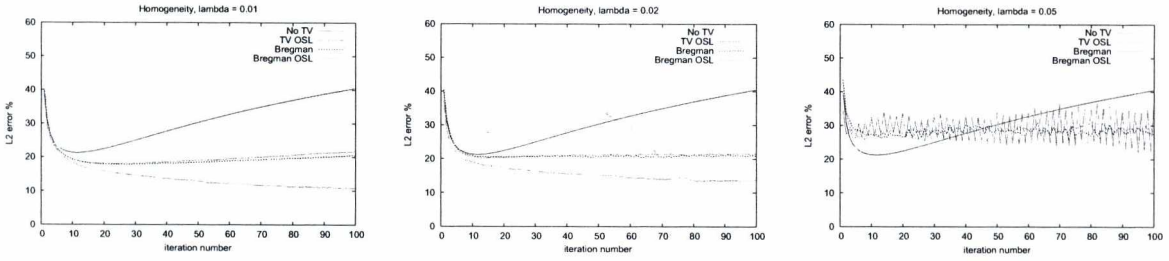
$\lambda = 0.01$ $\lambda = 0.02$ $\lambda = 0.05$

Figure 5: *L_2 error curves of the Three Squares reconstruction.*



$\lambda = 0.01$ $\lambda = 0.02$ $\lambda = 0.05$

Figure 6: *L_2 error curves of the Point phantom*



$\lambda = 0.01$ $\lambda = 0.02$ $\lambda = 0.05$

Figure 7: *L_2 error curves of the Homogeneity phantom*

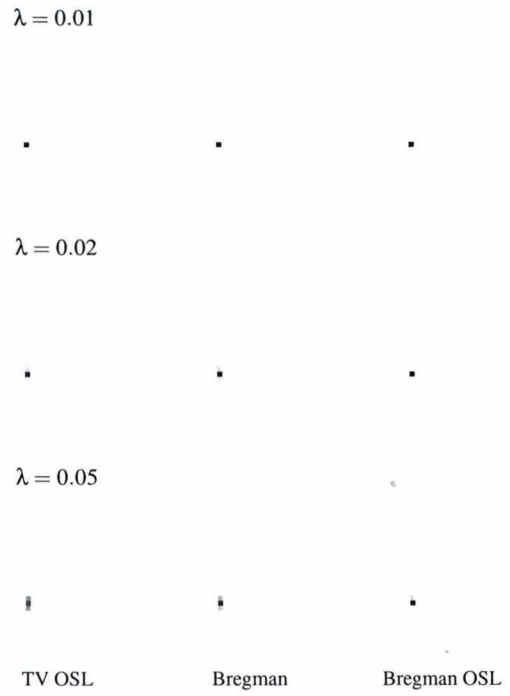


Figure 9: *Reconstructions of the Point phantom*

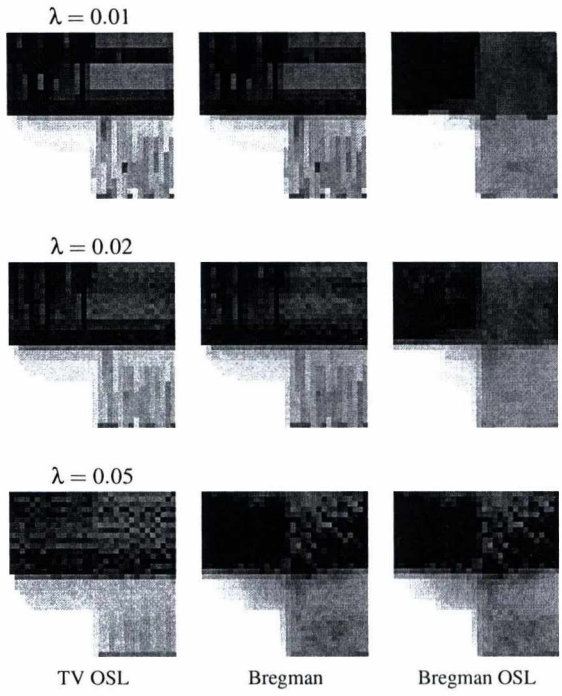


Figure 10: *Reconstructions of the Homogeneity phantom.*

an optimal Bregman period based on the properties of earlier iteration step.

Acknowledgement

This work has been supported by OTKA K–104476.

References

1. L. Bregman. The relaxation method for finding common points of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.
2. P. C. Hansen. The L-curve and its use in the numerical treatment of inverse problems. In *Computational Inverse Problems in Electrocardiology, ed. P. Johnston, Advances in Computational Bioengineering*, pages 119–142. WIT Press, 2000.
3. F. Knoll, M. Unger, C. Diwoky, C. Clason, T. Pock, and R. Stollberger. Fast reduction of undersampling artifacts in radial MR angiography with 3D total variation on graphics hardware. *Magnetic Resonance Materials in Physics, Biology and Medicine*, 23(2):103–114, 2010.
4. Milán Magdics, Balázs Tóth, Balázs Kovács, and László Szirmay-Kalos. Total variation regularization in PET reconstruction. In *KÉPAF*, pages 40–53, 2011.
5. M. Persson, D. Bone, and H. Elmqvist. Total variation norm for threedimensional iterative reconstruction in limited view angle tomography. *Physics in Medicine and Biology*, 46(3):853–866, 2001.
6. A. J. Reader and H. Zaidi. Advances in PET image reconstruction. *PET Clinics*, 2(2):173–190, 2007.
7. L. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Trans. Med. Imaging*, 1:113–122, 1982.
8. L. Szirmay-Kalos, M. Magdics, B. Tóth, and T. Bükki. Averaging and metropolis iterations for positron emission tomography. *IEEE Trans Med Imaging*, 32(3):589–600, 2013.
9. L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.
10. A. N. Tychonoff and V. Y. Arsenin. *Solution of Ill-posed Problems*. Winston & Sons., Washington, 1977.
11. M. Yan, A. Bui, J. Cong, and L.A. Vese. General convergent expectation maximization (EM)-type algorithms for image reconstruction. *Inverse Problems and Imaging*, 7:1007–1029, 2013.
12. W. Yin, S. Osher, J. Darbon, and D. Goldfarb. Bregman iterative algorithms for compressed sensing and related problems. *IAM Journal on Imaging Sciences*, 1(1):143–168, 2008.

Recent Results on Shape-Based Interpolation

Márton József Tóth, Balázs Csébfalvi

Department of Control Engineering and Information Technology,
Budapest University of Technology and Economics

Abstract

In this paper, we overview our recent results [23] on shape transformation of multidimensional density distributions. We extend 1D distribution interpolation to 2D and 3D by using the Radon transform. Our algorithm is fundamentally different from previous shape transformation techniques, since it considers the objects to be interpolated as density distributions rather than level sets of Implicit Functions (IF). First, we perform distribution interpolation on the precalculated Radon transforms of two different density functions, and then an intermediate density function is obtained by an inverse Radon transform. This approach guarantees a smooth transition along all the directions the Radon transform is calculated for. Unlike the IF methods, our technique is able to interpolate between features that do not even overlap and it does not require a one dimension higher object representation. We will demonstrate that these advantageous properties can be well exploited for 3D modeling and metamorphosis.

Categories and Subject Descriptors (according to ACM CCS): I.4.5 [Image Processing and Computer Vision]: Reconstruction–Transform methods

1. INTRODUCTION

Shape-based interpolation is mainly used for (1) modeling or reconstruction of 3D objects from 2D cross sections [20, 12, 11, 24, 26] and (2) morphing [16, 7, 25]. The major application fields of these techniques are Computer Aided Design (CAD), movie industry, and medical image processing and visualization. In CAD systems, 3D geometrical models can be built from contours defined in cross-sectional slices [24, 17]. Surfaces that fit onto the contours are obtained by using a contour-interpolation method between the subsequent slices. In the movie industry, shape transformation is used for making special effects, such as morphing characters. In 3D medical imaging, it is usual that the resolution of a volumetric data set is lower along the z axis than along the x and y axes. Therefore, a shape-based interpolation technique is applied to produce intermediate slices to obtain an isotropic volume representation [20, 12, 11, 24]. The most popular way of automatic shape transformation is based on an Implicit Function (IF) representation of 2D or 3D shapes [4, 13]. An intermediate shape is simply produced as a level set of an IF that is calculated by interpolating between the IFs belonging to the initial and final shapes [20, 12]. This approach is easy to implement and robust in a sense that topologically different shapes can be interpolated without searching for pairs of corresponding points. Nevertheless, in this paper, we show that the previous IF methods are able to make a smooth transition between two features only if they are overlapping, otherwise the features get disconnected. Furthermore, shape-based interpolation of gray-scale images (in other words, density functions) requires a one dimension higher representation than a shape-based interpolation of object boundaries [11]. To remedy these problems, we propose a fundamentally different approach for shape-based interpolation. Our major goal is to guarantee a continuous transition between the lower-dimensional projections of density functions to be interpolated. Therefore, we precalculate the Radon transforms [9] of the density functions, which represent the lower-dimensional projections from different angles, and apply a distribution interpolation [21] between the corresponding projections. The result is then transformed back by the classical Filtered Back-Projection (FBP) algorithm, which implements the inverse Radon transform. The modeling potential of this algorithm is much higher than that of the IF methods, as it is able to connect features that do not overlap. Moreover, our method is efficient to use even for interpolating between 3D density functions, as the alternative representation produced by the Radon transform remains 3D, while the classical shape-based interpolation of gray-scale volumes would require 4D

IFs to calculate [11]. Distribution interpolation and the Radon transform are well-known tools that have been used separately in different application fields, but to the best of our knowledge, we are the first to combine them for shape-based interpolation [23].

2. RELATED WORK

In computer graphics, shape-based interpolation is usually applied for interpolating between the boundary contours of 2D shapes or morphing between the boundary surfaces of 3D objects. However, an automatic morphing between translucent objects [15], which are defined by volumetric density functions rather than explicit geometrical models is still a challenging task. It depends on subjective preferences whether a morphing algorithm should be fully automatic or user-controlled. We think that the advantages of these approaches are complementary and it depends on the given application which one to prefer. As we focus on automatic morphing, warping techniques that require user intervention [3, 16, 7, 10] are out of the scope of our work. Automatic morphing is favorable, for instance, if a shape-based interpolation is required between all pairs of consecutive slices in a huge volumetric data set, or a morphing needs to be performed between 3D objects that are completely different geometrically and corresponding features can hardly be specified. The major expectation from an automatic morphing method is to provide intermediate objects that show features of both objects which we interpolate between, and to guarantee a smooth and gradual transition between the features.

2.1. Shape-Based Interpolation of Boundary Contours

Early shape-based interpolation techniques were proposed for medical imaging applications, where the goal was to reconstruct 3D shapes of different organs from 2D slices of CT or MRI scans [20, 12, 11]. For example, this is a typical application field, where an automatic processing is clearly an advantage, since a huge amount of voxel data is required to be efficiently processed preferably without any user interaction. As in a usual volumetric data set the inter-slice distance is higher than the distance between the pixels of the slices, additional intermediate slices need to be interpolated to produce an isotropic volume. The brute-force method is to directly interpolate between the original slices, and to apply the well-known Marching Cubes algorithm [18] to extract a boundary surface. However, this approach often results in severe staircase artifacts. To reproduce smooth boundary surfaces, shape-based interpolation techniques first detect boundary contours on the slices and then apply a more sophisticated contour-interpolation method. For contour interpolation, a variety of methods have been published that build a triangular mesh which connects the two consecutive contours [2, 6, 24, 17]. Generally, this is a difficult task, since the problem of self-intersection and topologically different contours need to be carefully handled. Unlike the direct contour-interpolation techniques, the IF methods can easily avoid these problems. The basic idea is to interpolate between the IFs that represent the consecutive contours, and extract intermediate contours from interpolated IFs. As a shape-based interpolation method is required to handle the distance information somehow, it is a natural choice to use a Signed Distance Map (SDM) as an IF [4, 13]. The pixels of a 2D SDM represent the distance to the nearest contour point, but inside the contour the sign is positive, while outside the contour it is negative. An intermediate contour is obtained by extracting the zero-crossing level set of the interpolated SDMs [20, 12, 11]. The SDM representation is efficient to calculate using the chamfering method [1].

2.2. Shape-Based Interpolation of Boundary Surfaces

Shape-based contour interpolation is straightforward to extend to surface interpolation [20]. A boundary surface of an object can be represented by a 3D SDM, where the voxels store the distance to the closest surface point. Similarly to the 2D SDMs, the sign is positive inside the object and negative outside the object. The interpolation of the 3D SDMs and the extraction of the intermediate boundary surface are done analogously to the 2D case. Although this method can produce transitions between topologically different objects, the transitions are often not smooth enough due to the discontinuous curvature of the SDMs. In order to achieve smoother transitions, variational interpolation was proposed [25], which constructs IFs of minimal aggregate curvature. The IFs are searched for as a linear combination of Radial Basis Functions (RBF) [5] and the coefficients are determined such that the curvature is minimized. This requires the solution of a large linear equation system, which is time-consuming for complex shapes. Furthermore, in case of such constrained optimization problems, the coefficient matrix is prone to be ill-conditioned [26]; thus, its inversion by the proposed LU decomposition could easily become instable if the number of the unknown variables drastically increase. This is probably the reason why the variational interpolation [25] has not been extended to gray-scale images or volumes.

2.3. Extension to Gray-Scale Images and Volumes

The shape-based interpolation of gray-scale images [11] is computationally much more expensive than the shape-based interpolation of contours as it requires 3D IFs to interpolate rather than only 2D IFs. Each image is considered to be a height field, which can alternatively be represented by a 3D SDM. The intermediate images are obtained as height fields extracted from the interpolated 3D SDMs. Thus, both the chamfering and the extraction of the zero-crossing level set require the processing of 3D volumes for each pair of consecutive images. A shape-based interpolation of gray-scale volumes [11] is even more expensive. Here, the height fields are defined over the 3D space; therefore, the corresponding SDMs are 4D data sets. Consequently, 4D data processing

is required to obtain each single intermediate volume. The variational interpolation scheme [25] has not been adapted to gray-scale images or volumes yet, but using the same extension as for the SDMs, it would also require a one dimension higher object representation.

In contrast, our algorithm represents the gray-scale images and volumes by their Radon transforms, which are of the same dimensionality as the original data. Additionally, we perform efficient processing on the Radon transforms; thus, all the computation can be completed in a reasonable time. Moreover, as the smoothness of the transition is guaranteed by the distribution interpolation on the projections the Radon transform is evaluated for, a computationally expensive constrained optimization [8, 19] is not necessary.

3. MULTIDIMENSIONAL DISTRIBUTION INTERPOLATION

In this section, we describe how to extend 1D distribution interpolation [21] to higher dimensions by using the Radon transform.

3.1. Distribution Interpolation in 1D

Let us assume that we have two 1D density functions $f_0(x)$ and $f_1(x)$, and we want to interpolate between them. For example, Figure 1 shows two Gaussian density functions that are scaled and centered differently. In $f_0(x)$ the same amount of material is concentrated on the left side as in $f_1(x)$ on the right side. Therefore, it is a natural expectation that this mass is gradually moved from left to right in the intermediate interpolated density functions. Note that a simple linear interpolation along the y axis clearly does not fulfill this requirement. A distribution interpolation [21], however, does exactly what is expected from a shape-based interpolation technique. Instead of directly interpolating the density functions along the y axis, this method actually interpolates the Cumulative Distribution Functions (CDF) along the x axis. The CDFs for $f_0(x)$ and $f_1(x)$ are defined as follows:

$$F_0(x) = \int_{-\infty}^{x} f_0(x')dx', \tag{1}$$

$$F_1(x) = \int_{-\infty}^{x} f_1(x')dx'.$$

The first step of the distribution interpolation is to find x_0 and x_1 such that $F_0(x_0) = F_1(x_1) = y$. The interpolated CDF $F_w(x)$ takes the same value y at a linearly interpolated position $x = (1-w)x_0 + wx_1$. Positions x_0 and x_1 are simply obtained by inverting the CDFs:

$$F_w^{-1}(y) = x = (1-w)x_0 + wx_1 \tag{2}$$

$$= (1-w)F_0^{-1}(y) + wF_1^{-1}(y).$$

The interpolated CDF $F_w(x)$ is completely defined by its inverse function $F_w^{-1}(y)$, and the corresponding interpolated density function is obtained by a simple derivation:

$$f_w(x) = \frac{dF_w(x)}{dx}. \tag{3}$$

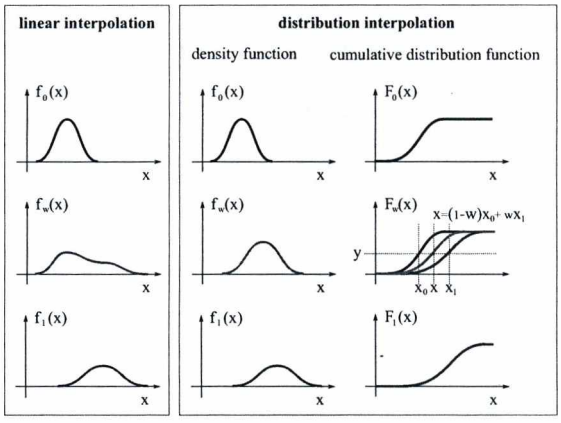


Figure 1: *Distribution interpolation in 1D.*

Figure 1 shows the interpolated density function for $w = 1/2$. Distribution interpolation is usually applied between probability density functions, so their integrals are assumed to be equal to one. However, this scheme can be easily adapted to mass distributions if the distributions are normalized before the interpolation and the interpolated distributions are rescaled such that the a continuous transition of the total mass is guaranteed.

3.2. The Radon Transform

Note that the 1D distribution interpolation cannot be directly extended to higher dimensions. Since we propose an extension scheme that is based on the Radon transform, here, we briefly overview its evaluation and inversion. The Radon transform [9] of a 2D density function $f(x,y)$ is defined by a set of 1D projections $p_\theta(t)$:

$$p_\theta(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(x\cos(\theta) + y\sin(\theta) - t)dxdy, \tag{4}$$

where δ is the Dirac delta and θ is the projection angle.

The Radon transform is invertible, and its inverse can be evaluated by the classical Filtered Back-Projection (FBP) algorithm [14], which consists of the following steps:

1. Fourier transform of the projections:

$$\hat{p}_\theta(\nu) = \int_{-\infty}^{\infty} p_\theta(t)e^{-i2\pi t\nu}dt. \tag{5}$$

2. Filtering in the frequency domain, where the frequency response of the filter is $|\nu|$:

$$\hat{q}_\theta(\nu) = \hat{p}_\theta(\nu)\cdot|\nu|. \tag{6}$$

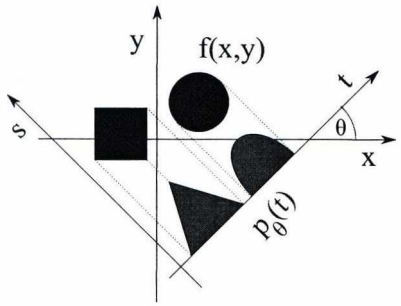


Figure 2: *The Radon transform of a 2D density function is defined as a set of 1D projections* $p_\theta(t)$.

3. Inverse Fourier transform of $\hat{q}_\theta(\nu)$:

$$q_\theta(t) = \int_{-\infty}^{\infty} \hat{q}_\theta(\nu) e^{i2\pi t\nu} d\nu. \tag{7}$$

4. Back-projection of the filtered projections $q_\theta(t)$:

$$f(x,y) = \int_0^{\pi} q_\theta(x\cos(\theta) + y\sin(\theta)) d\theta. \tag{8}$$

3.3. 2D Distribution Interpolation

Now assume that we want to interpolate between two 2D density functions $f_0(x,y)$ and $f_1(x,y)$. In order to guarantee that the projections of the interpolated density functions make a smooth transition between the projections of $f_0(x,y)$ and $f_1(x,y)$, we apply distribution interpolation on the 1D projections rather than a direct interpolation between the 2D density functions. The Radon transforms of $f_0(x,y)$ and $f_1(x,y)$ are denoted by $p^0_\theta(t)$ and $p^1_\theta(t)$, respectively. A distribution interpolation makes sense only if the projection functions are normalized before. Therefore, we need to calculate the integrals of $f_0(x,y)$ and $f_1(x,y)$:

$$s_0 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_0(x,y)dxdy, \tag{9}$$

$$s_1 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_1(x,y)dxdy.$$

By using distribution interpolation between the normalized projections $p^0_\theta(t)/s_0$ and $p^1_\theta(t)/s_1$, we obtain normalized intermediate distributions $p^w_\theta(t)$. To ensure a smooth transition of the total mass between $f_0(x,y)$ and $f_1(x,y)$, the inverse Radon transform is performed on rescaled projections $p^w_\theta(t)((1-w)s_0 + ws_1)$. The result of the inverse Radon transform is the interpolated density function $f_w(x,y)$.

3.4. 3D Distribution Interpolation

In order to interpolate between two 3D density functions $f_0(x,y,z)$ and $f_1(x,y,z)$, 2D projections need to be calculated:

$$p^0_\theta(t,z) = \tag{10}$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_0(x,y,z)\delta(x\cos(\theta) + y\sin(\theta) - t)dxdy,$$

$$p^1_\theta(t,z) =$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_1(x,y,z)\delta(x\cos(\theta) + y\sin(\theta) - t)dxdy.$$

For a fixed angle θ, $p^0_\theta(t,z)$ and $p^1_\theta(t,z)$ are, in fact, 2D density functions. Therefore, a 2D distribution interpolation can be applied between them as described in Section 3.3. Afterwards, an intermediate 3D density function $f_w(x,y,z)$ is reconstructed from the interpolated 2D projections $p^w_\theta(t,z)$ by using the standard FBP algorithm, which is a de facto standard solution for this classical tomography reconstruction problem. In a practical implementation, a discrete approximation of the continuous integrals is applied. In order to avoid a loss of information, it is required that the total number of pixels in all discretized projections is not smaller than the number of the voxels in the discrete volumetric representations of the 3D density functions.

4. APPLICATIONS

4.1. Shape-Based Interpolation of Gray-Scale Images

Since our method is based on a Radon Transform Interpolation, we refer to it as RTI. In contrast, the method by Udupa and Grevera [11] is based on a Distance Transform Interpolation; thus, we refer to it as DTI. We compare RTI to DTI for the following reasons:

1. DTI is a de facto standard for an automatic shape-based interpolation of gray-scale images.
2. DTI is a general solution and not designed for a specific application field.
3. Although for contour interpolation the variational framework of Turk and O'Brien [25] provides smoother transitions than the interpolation of SDMs, its extension to gray-scale images is not trivial, and to the best of our knowledge, has not been published so far.

Note that, using DTI, the chamfering results in just an approximation of the Euclidean distance transform. Although the approximation can be improved by larger chamfering windows, it significantly increases the computational costs. Therefore, to make our comparison independent from the precision of the distance transform, we evaluated the true Euclidean distance maps for analytically defined height fields, which are interpreted as gray-scale images.

Figure 3 shows several examples for shape-based interpolations between different density distributions. We generated 64 intermediate slices, and rendered the resulting volume using direct volume rendering. The first example is the easiest

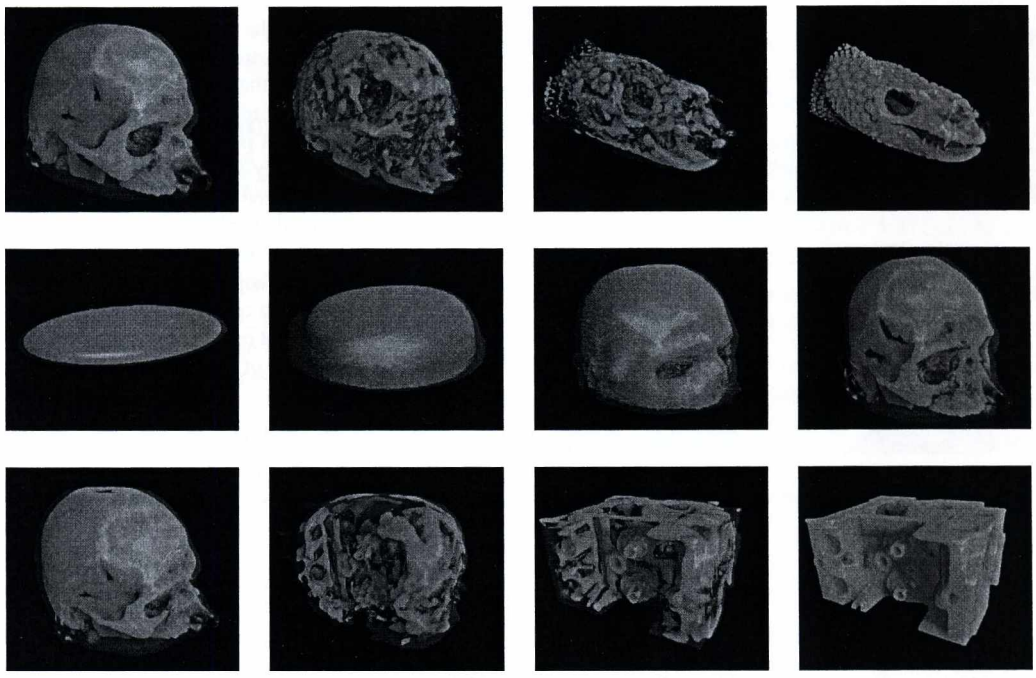


Figure 4: *Automatic morphing between different gray-scale volumes using distribution interpolation on the Radon transforms.*

one, where we interpolate between two overlapping disc-shaped distributions. Although DTI is able to make a connection, it also produces an unexpected curvature. In contrast, RTI results in a perfect tubular connection. In the second example, only the distance between the disc-shaped distributions is increased such that they do not overlap anymore. Note that, in this case, DTI fails to make a connection, while RTI still provides a perfect transition. The third and fourth examples well demonstrate that, unlike DTI, RTI can appropriately handle bifurcations. In the last example, again two density distributions are interpolated, which do not overlap. DTI cannot make a connection for this case either, while RTI is able to connect the ring-shaped and disc-shape density distributions forming a glass shape. These examples clearly show that RTI can be a reasonable alternative of DTI, for instance, in a modeling application.

4.2. Metamorphosis of Gray-Scale Volumes

In order to test our method also on gray-scale volumes, we implemented the entire algorithm on the GPU using CUDA. Note that, all the processing steps, such as the Radon transform, its inversion by FBP, and the distribution interpolation of the 1D projections are easy to map onto the parallel architecture of the GPU. For the frequency-domain ramp filtering, we used the CUDA FFT library. Figure 4 shows a couple of examples for 3D morphing. For each pair of volumes we generated 20 intermediate volumes of resolution 256^3, which took less than an hour on an nVidia Tesla M2070 graphics card. In contrast, we found it unfeasible to efficiently implement DTI for gray-scale volumes on the GPU. Although there exist fast GPU implementations for calculating the distance transform in 2D or 3D [22], applying DTI, the shape-based interpolation of gray-scale volumes would require 4D distance maps to calculate. For example, using floating-point arithmetics, for a volume of resolution 256^3, the corresponding 4D distance map would take $256^4 \times 4 = 16$ Gbytes of memory, which exceeds the capacity of recent graphics cards. Furthermore, the calculation of such a huge 4D distance map would involve a significant computational cost. Variational interpolation [25] is also very difficult to extend to gray-scale volumes. Using the extension scheme of Udupa and Grevera [11], the gray-scale volumes could be treated as height fields defined over the 3D space. Since these height fields can be represented by 4D IFs, the shape transformation between them would require a 4D RBF interpolation. To avoid the loss of fine details, at least to each voxel a 4D RBF needs to be assigned. Thus, for a pair of volumes with resolution 256^3, the number of unknown variables is $256^3 \times 2$. Consequently, the coefficient matrix of the corresponding linear equation system contains $256^6 \times 4$ elements, which makes the inversion practically impossible. Overall, we think that compared to the computational and storage costs of the previous methods, our solution is simple and efficient.

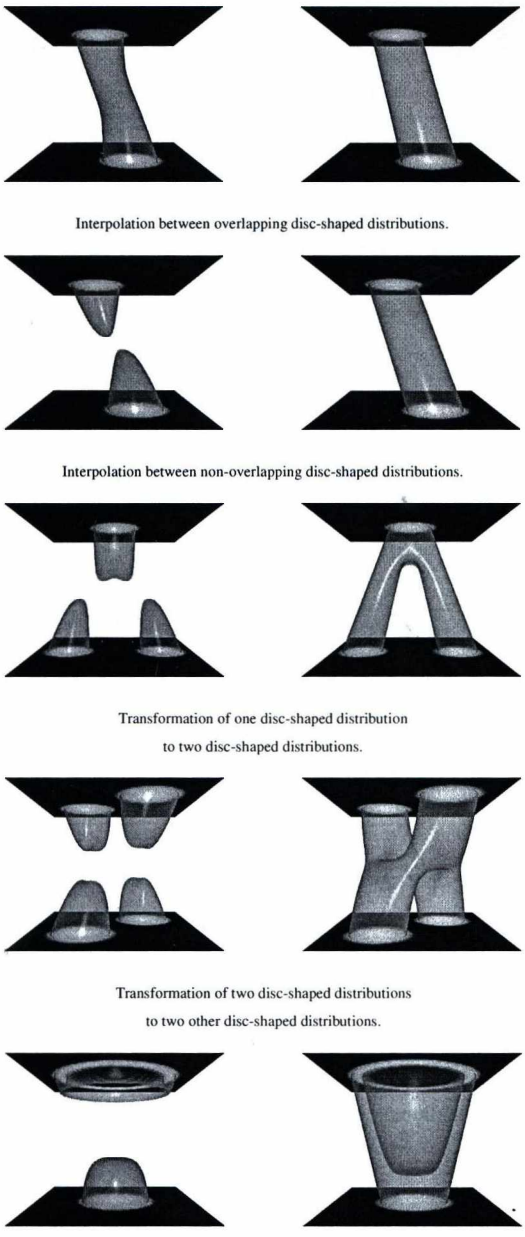


Figure 3: *Shape-based interpolation between different 2D density distributions. Left: Interpolation of the distance transforms. Right: Distribution interpolation of the Radon transforms.*

5. CONCLUSION

In this paper, we presented a novel algorithm for shape-based interpolation of gray-scale images and volumes. As far as we know, our technique is the first to combine distribution interpolation and the Radon transform. The major advantage of this combination is that the Radon transform does not require a one dimension higher representation than the original images and volumes. This is not the case for the previous IF methods, where the gray-scale images and volumes can alternatively be represented by 3D and 4D IFs, respectively. Moreover, we demonstrated that the distribution interpolation of the Radon transforms can make a smooth connection between non-overlapping features that the IF methods are typically not able to connect. Due to this advantageous properties, we think that our method represents a significant contribution to the field of shape-based interpolation.

ACKNOWLEDGEMENTS

This work was supported by projects TÁMOP-4.2.2.B-10/1–2010-0009 and OTKA K-101527. The Heloderma data set is from the Digital Morphology (`http://www.digimorph.org`) data archive. Special thanks to Dr. Jessica A. Maisano for making this data set available to us.

References

1. Akmal Butt, M. and Maragos, P. (1998). Optimum design of chamfer distance transforms. *IEEE Transactions on Image Processing*, 7(10):1477–1484.
2. Bajaj, C. L., Coyle, E. J., and nan Lin, K. (1996). Arbitrary topology shape reconstruction from planar cross sections. In *Graphical Models and Image Processing*, pages 524–543.
3. Beier, T. and Neely, S. (1992). Feature-based image metamorphosis. *SIGGRAPH Computer Graphics*, 26(2):35–42.
4. Borgefors, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371.
5. Buhmann, M. (2009). *Radial Basis Functions: Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press.
6. Cheng, S.-W. and Dey, T. K. (1999). Improved constructions of Delaunay based contour surfaces. In *Proc. ACM Sympos. Solid Modeling and Applications 99*, pages 322–323.
7. Cohen-Or, D., Solomovic, A., and Levin, D. (1998). Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141.
8. Csébfalvi, B., Neumann, L., Kanitsar, A., and Gröller, E. (2002). Smooth shape-based interpolation using the conjugate gradient method. In *Proceedings of Vision, Modeling, and Visualization*, pages 123–130.

9. Deans, S. R. (1983). *The Radon Transform and Some of Its Applications*. Krieger Publishing.

10. Fang, S., Srinivasan, R., Raghavan, R., and Richtsmeier, J. T. (2000). Volume morphing and rendering - an integrated approach. *Computer Aided Geometric Design*, 17(1):59–81.

11. Grevera, G. J. and Udupa, J. K. (1996). Shape-based interpolation of multidimensional grey-level images. *IEEE Transactions on Medical Imaging*, 15(6):881–92.

12. Herman, G. T., Zheng, J., and Bucholtz, C. A. (1992). Shape-based interpolation. *IEEE Computer Graphics and Applications*, 12(3):69–79.

13. Jones, M. W., Baerentzen, J. A., and Sramek, M. (2006). 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12:581–599.

14. Kak, A. C. and Slaney, M. (1988). *Principles of Computerized Tomographic Imaging*. IEEE Press.

15. Kniss, J., Premoze, S., Hansen, C., and Ebert, D. (2002). Interactive Translucent Volume Rendering and Procedural Modeling. In *Proceedings of IEEE Visualization Conference (VIS) 2002*, pages 109–116.

16. Lerios, A., Garfinkle, C. D., and Levoy, M. (1995). Feature-based volume metamorphosis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 449–456.

17. Liu, L., Bajaj, C., Deasy, J., Low, D. A., and Ju, T. (2008). Surface reconstruction from non-parallel curve networks. *Computer Graphics Forum*, 27(2):155–163.

18. Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169.

19. Neumann, L., Csébfalvi, B., Viola, I., Mlejnek, M., and Gröller, E. (2002). Feature-Preserving Volume Filtering . In *VisSym 2002 : Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 105–114.

20. Raya, S. and Udupa, J. (1990). Shape-based interpolation of multidimensional objects. *IEEE Transactions on Medical Imaging*, 9(1):32–42.

21. Read, A. L. (1999). Linear interpolation of histograms. *Nuclear Instruments and Methods*, A425:357–360.

22. Schneider, J., Kraus, M., and Westermann, R. (2009). GPU-based real-time discrete euclidean distance transforms with precise error bounds. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 435–442.

23. Tóth, M. J., and Csébfalvi, B. (2014). Shape Transformation of Multidimensional Density Functions using Distribution Interpolation of the Radon Transforms. *9th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (GRAPP)*, pages 5–12.

24. Treece, G. M., Prager, R. W., Gee, A. H., and Berman, L. H. (2000). Surface interpolation from sparse cross-sections using region correspondence. *IEEE Transactions on Medical Imaging*, 19(11):1106–1114.

25. Turk, G. and O'Brien, J. F. (1999). Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 1999*, pages 335–342.

26. Turk, G. and O'brien, J. F. (2002). Modelling with implicit surfaces that interpolate. *ACM Transanctions on Graphics*, 21(4):855–873.

ISBN 978-615-5036-08-8