III. MAGYAR SZÁMÍTÓGÉPES GRAFIKA ÉS GEOMETRIA KONFERENCIA

BUDAPEST
2005. NOVEMBER 17-18.

Szerkesztette:
Szirmay-Kalos László
Renner Gábor

NJSZT-GRAFGEO

Előszó

Lectori Salutem. Ez a kiadvány a Neumann János Számítógép-tudományi Társaság Számítógépi Grafika és Geometria Szakosztálya (NJSZT-GRAFGEO) által szervezett Harmadik Magyar Számítógépes Grafika és Geometria Konferencia cikkeit tartalmazza. A konferencia a számítógépes grafika hazai és magyar kötődésű külföldi művelőinek seregszemléje, amely a 2002. évi első és a 2003. évi második konferenciát követi a sorban. 2004.-ben úgy döntöttünk, hogy a továbbiakban nem konkurálunk a képfeldolgozók és alakfelismerők rendezvényével, hanem a KÉPAF konferenciával évenkénti váltással, a Grafika és Geometria konferenciát is kétévenként rendezzük meg. Így a számítógépes grafika, képfeldolgozás, alakfelismerés és geometria határterületeire is ráirányíthatjuk a közönség figyelmét.

A benyújtott cikkeket a szereplő témák elismert képviselői bírálták el. Az elfogadott cikkek a számítógépes grafika sokféle területét érintik, mint például a természeti jelenségek szimulációját, a gépi látás eljárásokat, a számítógépes grafika orvosi felhasználását, a valós idejű képszintézis problémáit, valamint a számítógépes geometriát és rekonstrukciót. Ez a szerteágazó gyűjtemény a bizonyítéka annak, hogy a számítógépes grafikának és a geometriai modellezésnek Magyarországon is rangos műhelyei alakultak ki.

A konferenciát a Neumann János Számítógép-tudományi Társaság, a BME Irányítástechnika és Informatika Tanszéke és a Mecenatúra Alapítvány anyagilag támogatta. A kiadvány nyomdai előkészítése Umenhoffer Tamás munkája. A címlapot Szécsi László Maya modellező program felhasználásával és a saját, DirectX alapú képszintézis rendszerével alkotta meg. A konferencia honlapját Czuczor Szabolcs készítette el. A konferencia szervezésében még Lazányi István és Tóth Balázs vettek részt.

Ajánljuk ezt a kiadványt mindenkinek, aki a számítógépes grafika és geometria jelenlegi kérdéseivel és a magyar műhelyek tevékenységével meg szeretne ismerkedni.

Budapest, 2005. október 24.

Szirmay-Kalos László és Renner Gábor

Résztvevő intézmények

- ArchiData Kft.
- BME Geometria Tanszék
- BME Irányítástechnika és Informatika Tanszék,
- BME Méréstechnika és Információs Rendszerek Tanszék
- Debreceni Egyetem Matematikai Intézet
- Debreceni Egyetem Informatikai Kar
- Digic Pictures
- Digital Elite/Waveband, USA
- Egri Eszterházy Károly Főiskola Matematikai és Informatikai Intézete
- ELTE Analízis Tanszék
- ELTE Informatikai Kar
- Geomagic Hungary Kft.
- Miskolci Egyetem Ábrázoló Geometria Tanszék
- MTA SZTAKI
- Pázmány Péter Katolikus Egyetem Informatika Kar
- Rovira i Virgili University, Tarragona, Spanyolország
- Semmelweis Egyetem Radiológiai Intézete
- Szegedi Egyetem
- University of Girona, Spanyolország
- University of Twente, Hollandia
- VerAnim Kft.
- Veszprémi Egyetem Képfeldolgozás és Neuroszámítógépek Tanszék

ISBN 963-421-593-9

Tartalomjegyzék

Invited talk

Natural phenomena

Machine Vision

Medical applications

Real-time rendering

Geometry

Graphics systems

Számítógép animációs filmek készítése a Digic Pictures-nél, avagy a Warhammer univerzum életre keltése

Marinov Gábor

Digic Pictures
www.digicpictures.com

Abstract

A Digic Pictures stúdió számítógépes animációs filmek és digitális filmtrükkök készítésével foglalkozik, korábbi munkái közé tartozik 60 digitális trükk jelenet a Terminátor 3 mozifilmhez és a díjnyertes Exigo animációk. Az elmúlt években a Digic 3d-s animációs filmek készítésére koncentrált, munkafolyamatait, eszközeit ez irányban fejlesztette. A filmekben szereplő lények, díszletek modelljeinek, textúráinak elkészítését, azok megmozgatását és a filmek képkockáinak előállítását egy jól bővíthető, rugalmas rendszerbe foglalta. Mindez lehetővé teszi, hogy a cég művészei a valódi képi tartalom előállítására koncentráljanak. A Digic előadása bemutatja a 3d-s filmkészítés valódi produkciós körülmények között kialakított munkafolyamatait, technikai hátterét, mindezt a népszerű Warhammer fantasy világhoz kapcsolódó új filmjein keresztül.

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Painterly Rendering of Images and Real Paintings With SVG Support

Levente Kovács,[1†] and Tamás Szirányi[1‡]

[1] Department of Image Processing and Neurocomputing, University of Veszprém, Veszprém, Hungary
[2] Analogical Comp. Lab., Comp. and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary

Abstract

We propose the use of 2D non-photorealistical painterly rendering techniques for transforming real paintings in order to obtain such a coded representation which can preserve the painterly style without inducing compression artifacts. We also propose to store these painterly representations as scalable vector graphics (SVG), providing a lossless and scalable form of storing these painterly rendered images.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation I.3.4 [Image Processing and Computer Vision]: Image Representation

1. Introduction

From the vast amount of 2D non-photorealistic painterly rendering techniques some fields of application have emerged in recent years. Some of these are achieving painterly effects on real images and video, generating sketches, illustrations and simplified representations of ordinary images also with uses in medical imaging, etc. However, the main goal has remained the achievement of the painterly effect for generating painting-like images. In this work we elaborate on an idea suggested in [2], namely the possibility of automatically re-painting images taken from real life paintings for storing these images as the resulting stroke-series representation. The goal is to provide a coding scheme which preserves the painting-like features of such images, does not induce compression artifacts (i.e. blockiness, ringing, etc.) and at the same time provides a multiscale multilayer representation. We also show that these representations can efficiently be stored in SVG (scalable vector graphics) [1] format, providing a platform- and application-agnostic container to store these images.

As we will show, the painterly rendering method we use is not lossless from the point of view of pixel-level correct reconstruction - it is a non-photorealistic rendering method after all. But it shows its benefits in preservation of the painting style. The very nature of the painting method used makes sure the artificial stroke orientations follow the directions of the main contours in the model image, thus not breaking the continuity of the painting. It also gives us the opportunity of recreating the model images into our own unique version, since the stroke templates can be various user defined shapes and sizes, thus different styles can be simulated.

Haeberli introduced a simple interactive painterly rendering [3]. In Litwinowitcz's work [4] strokes with a given center, length, radius and orientation are generated, with applications for still and motion picture. Hertzmann presented an algorithm [5] which painted an image with a series of multilayer B-spline modeled strokes on a grid over the canvas. Szirányi et al. introduced the so-called Paintbrush Image Transformation [6] which was a simple random painting method using rectangular stroke templates, up to ten different stroke scales in a coarse-to-fine multilayer way also for segmentation and classification purposes. The image was described by the parameters of the consecutive paintbrush strokes. Santella and DeCarlo presented a method [7] which combines aspects from the approaches of Haeberli, Litwinowicz and Hertzmann. They extended their work with a system collecting eye-tracking data from a user [8]. Kovács et al. presented preliminary work on cartoon style transformation of image sequences [9] based on keyframe stochastic paintbrush transformation and optical flow based interframe post-painting.

† kla@vision.vein.hu
‡ sziranyi@sztaki.hu

This work was further refined [10] and the viability of lossless compression on painted images and image sequences was shown. Also, an effective storage and compression of real life paintings processed with stochastic paintbrush transformation was suggested [2], which is the lead we follow in the course of this paper. We suggest the use of painterly transformations to store images of real paintings in a visually pleasant representation by preserving the basic style of the images, and also making room for subjective transformations of these images by using different stroke patterns.

2. Discussion

In achieving our goals outlined above, we need two steps:

- transforming the images of the real paintings with painterly rendering.
- translate the generated painterly representation into SVG syntax.

In the generation of the painterly representation we use a variant of the approach in [2] and we will present it in the next Section. The painterly transformation produces such an output which can be quite easily translated into SVG syntax, as we will show later. As a result of these steps we will obtain a painterly representation which has some important properties:

- retains the painterly effects of the original images.
- provides a representation which can be easily scaled without re-generating the painterly outputs.
- can be easily translated into SVG.

Yet, unavoidably, we will also have some drawbacks. The basic building block of the painterly transformation algorithm (as we will show in the next Section) is the so called (artificial) brush stroke, which in our case is a grayscale template. Thus, it has a defined class of available stroke sizes, meaning that there can be details on the model images which cannot be reproduced if the size of these details is below the availabe smallest stroke size. Secondly, the used stroke templates can have colour variance inside a single stroke, but no texture content. That also means that details below a limit cannot be reproduced. Still, in spite of these limitations, we can produce painting-like transformations, which, when looked upon from a distance - just as in the case of real paintings - give the impression of a real painting, but when zooming in over a threshold one will not see any more details. This is analogous to the real world situation where, when going closer to an oil painting, after a while one cannot see more details of the image, only the structure of the painted strokes. In the basic mode, we can get a representation of the model image, which tries to mimic the impressions of the model. If we use different stroke templates, we can get different versions/styles of the same model.

2.1. The Painting Method

In the painterly transformation methods [2, 10] our goal was to produce painting-like images in a fully automated way that would resemble painting in overall but also be suitable for other purposes like image decomposition, representation, storage, coding, indexing and retrieval. When an image is being represented by a sequence of over- or non-overlapping brush-strokes, other aspects of possible utilization may surface, e.g. alternative ways of compression, storage and retrieval. For instance a special sequence describing an image area can be used for indexing. At the same time compactly designed stroke-parameter series can be effectively compressed in a lossless way.

In our painting method artificial brush strokes having the shape of a given template are placed in a specific order on a blank image (called canvas), trying to construct a painting-like result which resembles the model image. Strokes have main parameters like position, orientation, color and size. The result of the painting process will be a sequence of such stroke parameters which we can store, and later view the painted image from this representation.

The painterly rendering technique we use here is a variation of the one used in [2]. It is a template-based, multilayer, multiscale painting method with edge-oriented stroke directions. The main features of the method are as follows:

- painting is done in three layers: a coarse layer with large strokes, a middle one with somewhat smaller strokes, and a fine layer with small strokes,
- the orientation of a stroke at a given position is determined from the direction of the closest edge lines,
- the color/texture of a stroke depends on the template map (which range from binary to grayscale) and on the color below the target stroke position.

Layers Our previous experiments have shown that an average of three layers can give an optimal rendering considering computation time and result quality. Thus the painting process here uses a coarse-to-fine series of three stroke scales to generate the painting. Each layer builds upon the previous one, and on each layer painting is done until the improvement caused by painting with a specific stroke scale falls below an exponentially decreasing dynamic threshold. At a final step, those placed strokes which get refined at a later step and get fully covered by smaller strokes, are removed to reduce output size.

Orientation To preserve contours and not to disrupt the continuity of the model image, the orientation of the placed strokes follows the direction of the nearby edges. The edge map used is the result of a scale-space weighted edge extraction method used in [11] based on [12]. Thus the contours and main edges of the model image are preserved without the use of antialiasing approaches.

Color The placed strokes are user-specified templates (stroke shapes) given at the beginning of the painting process. The color of a placed stroke is determined by so called majority sampling, which means that the most frequent color of the target stroke area on the model image is taken as stroke color. If the stroke template was a simple binary mask, then this color is explicitly given to the stroke. If the template was grayscale, then the selected color is weighted with the template intensity values to produce a locally varying stroke color. Figure 1 contains an example of painting an image with a template.

Stroke-series representation makes a painted image scale-independent in the sense that differently scaled versions of the painted image can be displayed without re-painting: only the strokes are scaled and the positions adjusted. Figure 2 has an example of a painterly rendered image displayed at different scales not by re-generation, but only by scaling the stroke sizes and adjusting their relative positions to the scale.

2.2. The SVG Representation

The SVG [1] syntax is a fairly simple yet feature-rich meta-language, providing means to describe the contents of a graphical vector image by the shapes it consists of, i.e. their geometry, position, orientation, color. There are some predefined shapes, but free shapes and contour paths can also be constructed. The image description data itself is a text stream, consisting of the lists of the shapes and their parameters. SVG also supports the specification of multiple layers. In the following we will only discuss those features that are needed for the translation of the painterly rendered data into SVG, for the broad SVG feature set see [1].

For presentation purposes let us consider a case where a binary stroke template is used with rectangular shape, with sizes of 60x15, 32x8 and 10x3 pixels. In this case every element of the resulting stroke stream of the painting process will be a rectangle with its own position, orientation and color parameters. These need to be specified for the SVG representation, e.g.:

```
<rect
  width="32"
  height="8"
  transform="translate(52,98)
                  rotate(45.0,16,4)"
  fill="#f4028c"
/>
```

meaning that a 32x8 rectangle needs to be placed at coordinates (52,98), rotated by 45 degrees around the center of the rectangle, having a purpleish colour. Apart from predefined shapes, any shape can be user, e.g.:

```
<path
  d="M 10 10 L 20 10 L 15 20 z"
  fill="#f4028c"
/>
```

The above will produce a purpleish coloured triangle shape (codes mean: M - move to absolute position, L - line to absolute position, z - end of path). Bezier curves and elliptic curves can also be specified, thus stroke shapes can vary in a broad range.

In our case of a series of rectangle strokes, the layers will consist of a series of "rect" specifications, with three layers. Layers are specified with the

```
<g  id="layer1"> ... </g>
```

syntax. Then, we encode the SVG text stream with a lossless coder (gzip, rar, etc.).

The benefit of the SVG representation over the binary stroke series storage is that we do not require a specific viewer application to view the stored paintings since there are many applications on many platforms that can handle SVG data. E.g. even some internet browsers have native SVG handling support. Other than that, it also provides a humanly readable content description instead of binary data. Also, in many cases the SVG representation is smaller in size than the binary stroke series.

2.3. Samples, Compression

Here we provide a few quick samples of painterly rendered images. Figures 3 and 4 show some model-painted image pairs where the models were images of real paintings. The bottom images are close comparisons of the original and the painted versions. We can generally say that the painterly transformation preserves the painterly style of the model images and at the same time produces an easy to handle, scalable representation.

As we stated above, the nature of the multilayer coarse-to-fine stroke-based painting makes possible the easy scaling of the painted images. Figure 5 contains such an example of a painterly rendered image translated into SVG and then displayed at different resolutions.

Last but not least we present some numerical data showing the compactness of the painterly generated output, be that a binary stroke stream or an SVG translation. Figure 6 shows data comparing the compression results of a painted images by general image coders, our earlier painterly rendering method [2] and SVG [1] representations. Here the point is to find out how efficiently the painterly rendered images can be losslessly coded, thus the output of the painterly transformation process is encoded with different coders. The graph shows that the binary stroke-series based representation is always the smallest, the SVG translation falling somewhat behind.

Still, on many different types of images the SVG representation of the painterly rendered outputs can be more efficiently encoded as the binary stroke series. Figure 7 contains comparison data of some of such examples (the images producing these results are on Figure 8).

3. Conclusions

In this paper we suggest to use painterly rendering/transformation methods to generate such representations of real paintings which retain the style of the model, yet provide an easy to handle and scalable representation and at the same time give freedom to manipulate the process and generate different stylized versions of the model image. We also show that the picked painterly transformation that we used provides a straightforward way to translate the generated stroke series representation into a broadly supported SVG format, without the need of special viewers for the painterly rendered imagery.

References

1. Scalable Vector Graphics, 1.1 Specification. http://www.w3.org/TR/SVG/, January 2003. 1, 3
2. L. Kovács and T. Szirányi. Efficient Coding of Stroke-Rendered Paintings. *Proceeedings of the 17th ICPR*, IAPR&IEEE, pp. 835-839, 2004. 1, 2, 3
3. P. Haeberli. Paint By Numbers: Abstract Image Representation. *Computer Graphics*, **24**,(4):207–214, Aug. 1990. 1
4. P. Litwinowicz. Processing Images and Video for An Impressionist Effect. *Proceedings of ACM SIGGRAPH 97*, pp. 407–414, 1997. 1
5. A. Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. *Proceedings of ACM SIGGRAPH 98*, pp. 453–460, 1998. 1
6. T. Szirányi, Z. Tóth and I. Kopilovic. Paintbrush Image Transformation. *Proceedings of Fundamental Structural Properties in Image and Pattern Analysis*, IAPR, pp. 157–168, 1999. 1
7. D. DeCarlo and A. Santella. Stylization and Abstraction of Photographs. *Proceedings of ACM SIGGRAPH 2002*, pp. 769–776, 2002. 1
8. A. Santella and D. DeCarlo. Abstracted Painterly Rendering Using Eye-Tracking Data. *Proceedings of NPAR 2002*, pp. 75-82, 2002. 1
9. L. Kovács and T. Szirányi. Creating Animations Combining Stochastic Paintbrush Transformation and Motion Detection. *Proceeedings of the 16th ICPR*, IAPR&IEEE. (2):1090–1093, 2002. 1
10. L. Kovács and T. Szirányi. Coding of Stroke-Based Animations. *Posters Papers Proceeedings of WSCG 2004*, pp. 81–84. 2
11. L. Kovács and T. Szirányi. Painterly Rendering Controlled by Multiscale Image Features. *Proceeedings of SCCG 2004*, pp. 183-190, 2004. 2
12. T. Lindeberg. Edge Detection and Ridge Detection With Automatic Scale Selection. *International Journal of Computer Vision*, **30**(2):117–154, 1998. 2

Figure 1: *An example of generating a painterly rendering from the model image (top) with a stroke template (top left). Middle: first step (coarse scale), bottom: final step.*

Figure 2: *A painterly rendered image displayed in larger sizes without re-generating the painting.*

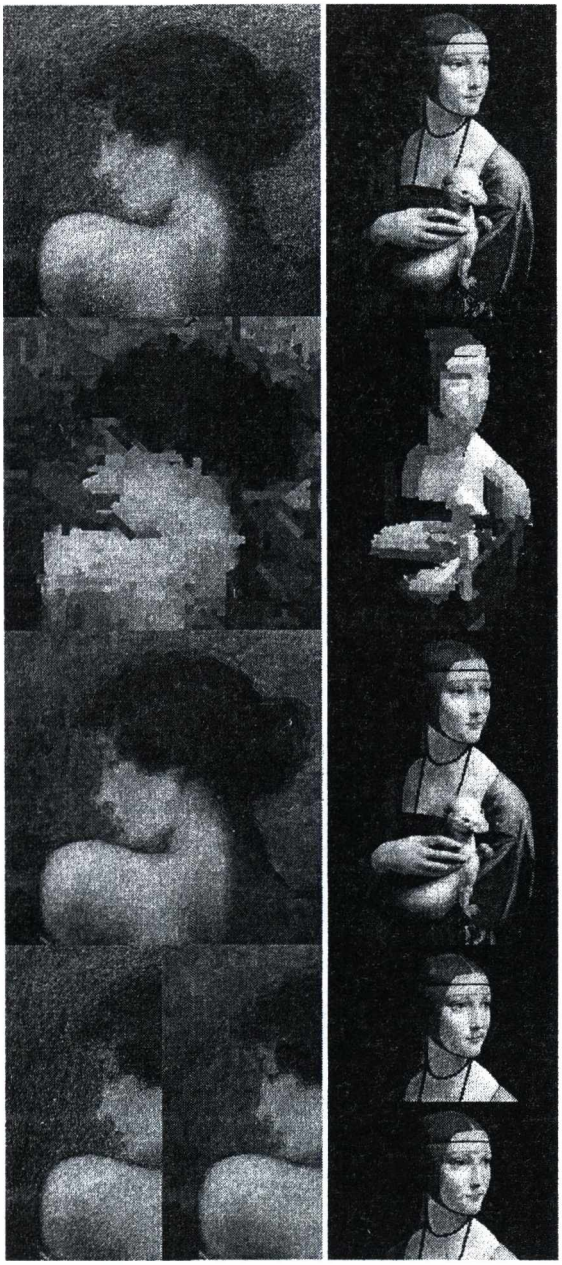


Figure 3: *Examples of painterly rendered images (from top to bottom, top: model image, bottom: final and comparison).*

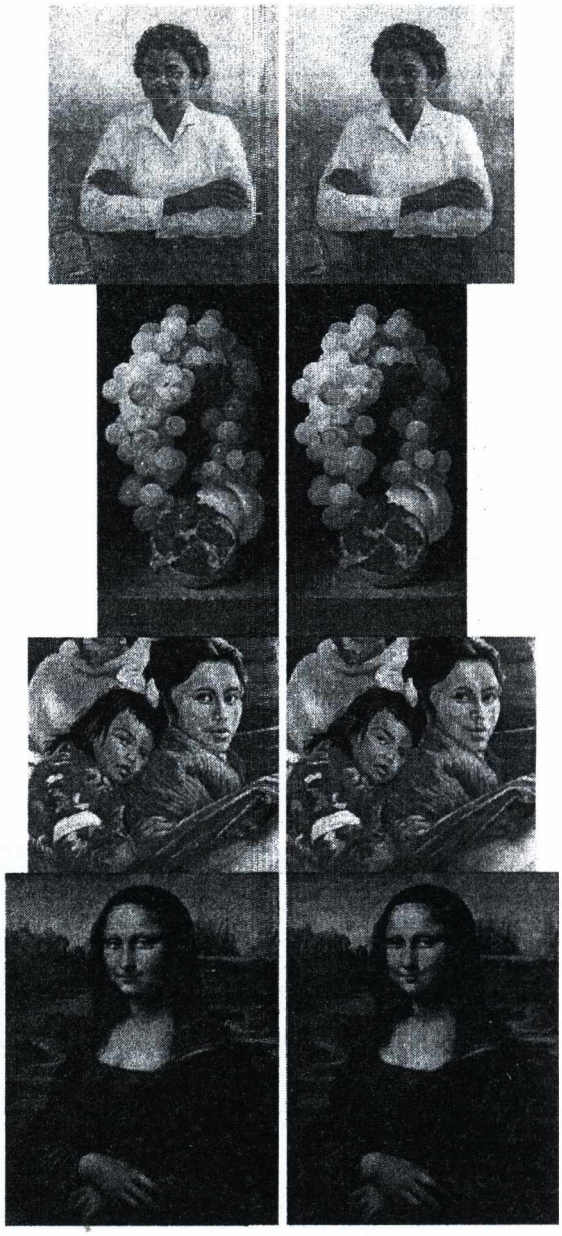


Figure 4: *Examples of painterly rendered images (from top to bottom, top: model image, bottom: final) of real oil paintings.*

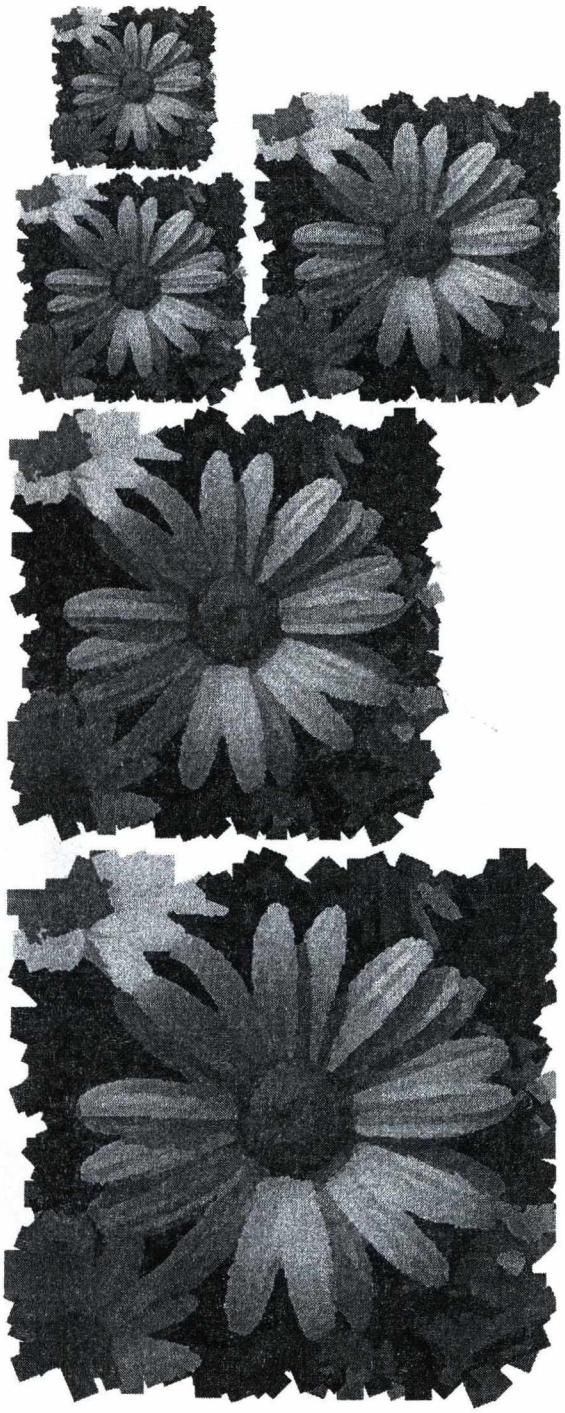


Figure 5: *Painterly rendered output as SVG, displayed in different sizes.*

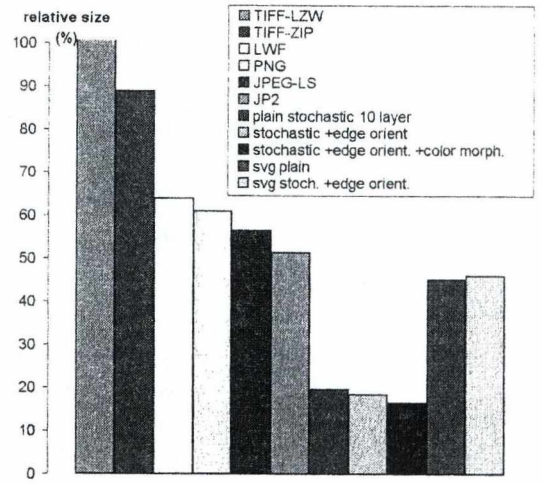


Figure 6: *Storing the painted images as stroke series is more efficient than coding them with image coders.*

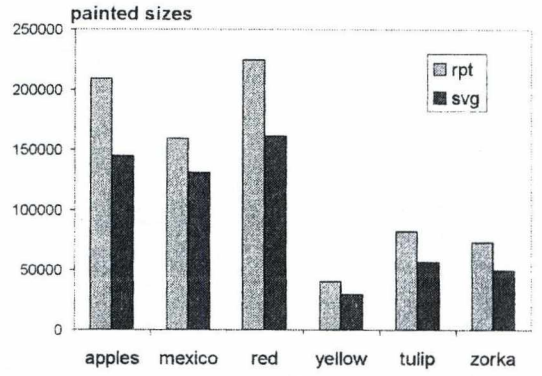


Figure 7: *In many cases the SVG representation is shorter than the binary stroke sequence.*

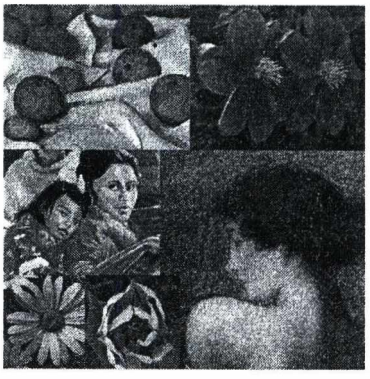


Figure 8: *Some images used in the evaluation.*

Tree Rendering with Billboard Clouds

Ismael Garcia, Mateu Sbert, and László Szirmay-Kalos

University of Girona and Budapest University of Technology
Emails: igarcia@ima.udg.es, mateu@ima.udg.es, szirmay@iit.bme.hu

Abstract

This paper presents a method to render complex trees on high frame rates while providing accurate occlusion and parallax effects. Based on the recognition that a planar billboard is accurate if the represented polygon is in its plane, we find a billboard for each of those groups of tree leaves that lie approximately in the same plane. The tree is thus represented by a set of billboards, called billboard cloud. The billboards are built automatically by a clustering algorithm. Unlike classical billboards, the billboards of a billboard cloud are not rotated when the camera moves, thus the expected occlusion and parallax effects are provided. On the other hand, this approach allows the replacement of a large number of leaves by a single semi-transparent quadrilateral, which considerably improves the rendering performance. A billboard cloud well represents the tree from any direction and provides accurate depth values, thus the method is also good for shadow, obscurances, and indirect illumination computation. In order to provide high quality results even if the observer gets close to the tree, we also propose a novel approach to encode textures representing the billboards. This approach is called indirect texturing and generates very high resolution textures on the fly while requiring just moderate amount of texture memory.

1. Introduction

One of the major challenges in rendering of vegetation is that the large number of polygons needed to model a tree or a forest exceeds the limits posed by the current rendering hardware [3]. Rendering methods should therefore apply simplifications. To classify these simplification approaches, we can define specific scales of simulation at which rendering should provide the required level of realism [15]:

- *Insect scale*: A consistent, realistic depiction of individual branches and leaves is expected. The images of individual leaves should exhibit parallax effects when the viewer moves, including the perspective shortening of the leaf shape and its texture, and view dependent occlusions.
- *Human scale*: Scenes must look realistic through distances ranging from an arm's reach to some tens of meters. Consistency is desired but not required.
- *Vehicle scale*: Individual trees are almost never focused upon and consistency is not required. Viewing distance may exceed several hundred meters.

There are two general approaches for realistic tree rendering: geometry-, and image-based methods. As it takes roughly hundred thousand triangles to build a convincing model of a single tree, geometry-based approaches should apply some form of Level of Detail technique [6].

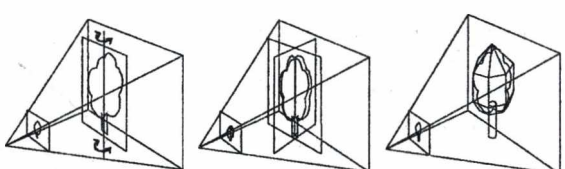


Figure 1: *A tree representation with images*

Image-based methods [12, 1] represent a trade-off of consistency and physical precision in favor of more photorealistic visuals (figure 1). *Billboard rendering* is analogous to using cardboard cutouts. In order to avoid the shortening of the visible image when the user looks at it from grazing angles, the billboard plane is always turned towards the camera. Although this simple trick solves the shortening problem, it is also responsible for the main drawback of the method. The tree always looks the same no matter from where we look at it. This missing parallax effect makes the replacement too

easy to recognize. The user expects that the size and texture of the leaves, as well as the distance and relative occlusions of leaf groups change as he moves. In order to handle this problem, the *view-dependent sprite* method pre-renders a finite set from a few views, and presents the one closest in alignment with the actual viewing direction. A popping artifact is visible when there is an alignment change. The *complex cutout* approach uses texture transparency and blending to render more than one views at the same time onto properly aligned surfaces. Both methods yield surprisingly acceptable results in vehicle scale simulations, but fail to deliver quality in close-up views. In order to handle occlusions correctly, billboards can also be augmented with per-pixel depth information [13, 9, 14]. If the tree is decomposed to parts and each part is represented by such a *nailboard* or *2.5 dimensional impostor*[16], the parts can be properly composited. On the other hand, rendering different parts with different billboards a limited parallax is also provided.

An advanced approach that has been actually implemented in commercial entertainment software is the basic *free-form textured tree model*, where the images are imposed on the approximated geometry. Resulting visuals are satisfactory, although due to the simple geometric model used, close-up views usually look artificial.

Finally, the *billboard cloud*[2] approach decomposes the original object into subsets of patches and replaces each subset by a billboard. These billboards are fixed and the final image is the composition of their images. Unlike nailboards, the billboards of a billboard cloud do not have per-pixel depth, but the depth is calculated from the plane of the billboard.

The method of this paper falls into the class of billboard clouds, but it prepares the billboards automatically and so carefully that the results are satisfactory not only on vehicle scale, but also on human scale, and with some compromises on insect scale as well. Since the billboard cloud model has been proposed to model conventional man-made 3D objects, such as teapots, helicopters, etc., we have to alter their construction to take into account the special geometry of natural phenomena. Our method is also good to generate shadows [11], to compute indirect illumination, and can be extended to a hierarchical, level of detail approach [8].

2. The new method

In order to reduce the geometric complexity of a tree, we use billboards having transparent textures to represent leaf groups. Since our billboards are not rotated when the camera changes, the expected parallax effects are provided, and the geometric accuracy is preserved.

The key of the method is then the generation of these billboards. We should first observe that a planar billboard is a perfectly accurate representation of a polygon (i.e. leaf) if the plane of billboard is identical to the plane of the polygon. On the other hand, even if the leaf is not on the billboard plane but is not far from that and is roughly parallel to the plane, then the billboard representation results in acceptable errors. Based on this recognition, we form clusters of the leaves in a way that all leaves belonging to a cluster lie approximately on the same billboard plane and replace the whole group by a single billboard. In order to control clustering, we shall introduce an error measure for a plane and a leaf, which includes both the distance of the leaf from the plane and the angle of the plane and leaf normals. The applied *K-means clustering algorithm* [7] starts with a predefined number of random planes, and iterates the following steps:

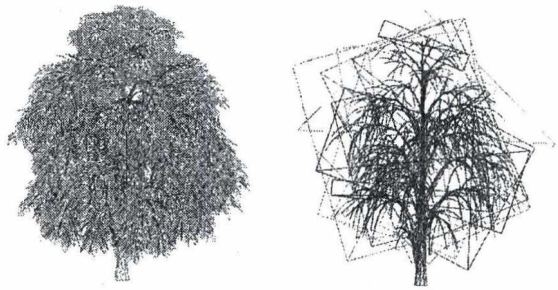


Figure 2: *The basic idea of the proposed method: the original tree defined by a hundred thousand polygons is approximated by a few fixed billboards, where a single billboard itself approximates many leaves.*

1. For each leaf we find that plane that minimizes the error. This step clusters the leaves into groups defined by the planes.
2. In each leaf cluster, the plane is recomputed in order to minimize the total error for the leaves of this cluster with respect to this plane.

Let us examine the definition of the error measure and these steps in details.

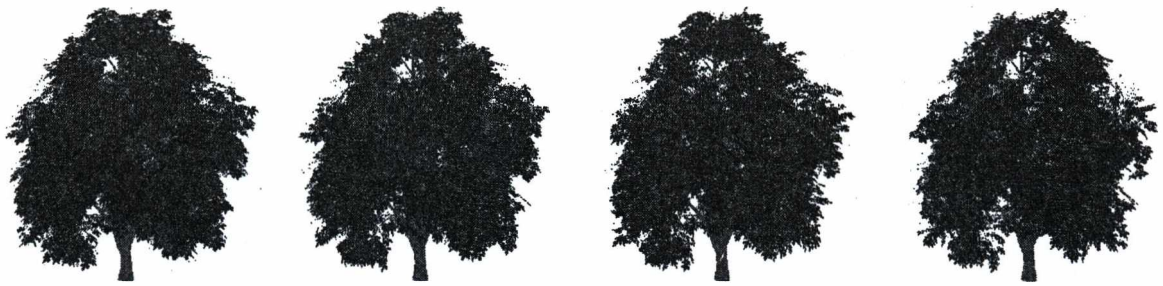
2.1. Error measure for a leaf and a plane

Those $\mathbf{p} = [x, y, z]^T$ points are on a plane that satisfy the following plane equation (T denotes matrix transpose):

$$D_{[\mathbf{n},d]}(\mathbf{p}) = \mathbf{p}^T \cdot \mathbf{n} + d = 0,$$

where $\mathbf{n} = [n_x, n_y, n_z]^T$ is the normal vector of the plane, and d is a distance parameter. We assume that $\mathbf{n}$ is a unit vector and is oriented such that d is non-negative. In this case $D_{[\mathbf{n},d]}(\mathbf{p})$ returns the signed distance of point $(\mathbf{p})$ from the plane.

From the point of view of clustering, leaf i is defined by its unit length average normal $\mathbf{n}_i$ and its center $\mathbf{p}_i$, that is by pair $(\mathbf{n}_i, \mathbf{p}_i)$.



polygonal tree | 107 billboards, 163 FPS | 54 billboards, 256 FPS | 29 billboards, 292 FPS

Figure 3: *Visual and speed comparisons of the original polygonal tree and the tree rendered with different number of billboards.*

Leaf i approximately lies in plane $[\mathbf{n},d]$ if $D^2_{[\mathbf{n},d]}(\mathbf{p}_i)$ is small, and its normal is approximately parallel with the normal of the plane, which is the case when $1-(\mathbf{n}_i^T\cdot\mathbf{n})^2$ is also small. Thus an appropriate error measure for plane $[\mathbf{n},d]$ and leaf $(\mathbf{n}_i,\mathbf{p}_i)$ is:

$$E([\mathbf{n},d]\leftrightarrow(\mathbf{n}_i,\mathbf{p}_i))=D^2_{[\mathbf{n},d]}(\mathbf{p}_i)+\alpha\cdot(1-(\mathbf{n}_i^T\cdot\mathbf{n})^2), \quad (1)$$

where α expresses the relative importance of the orientation similarity with respect to the distance between the leaf center and the plane.

2.2. Finding a good billboard plane for a leaf cluster

To find an optimal plane for a leaf group, we have to compute plane parameters $[\mathbf{n},d]$ in a way that they minimize total error $\sum_{i=1}^{N}E([\mathbf{n},d]\leftrightarrow(\mathbf{n}_i,\mathbf{p}_i))$ with respect to the constraint that the plane normal is a unit vector, i.e. $\mathbf{n}^T\cdot\mathbf{n}=1$. Using the Lagrange-multiplier method to satisfy the constraint, we have to compute the partial derivatives of

$$\sum_{i=1}^{N}E([\mathbf{n},d]\leftrightarrow(\mathbf{n}_i,\mathbf{p}_i))-\lambda\cdot(\mathbf{n}^T\cdot\mathbf{n}-1)=$$

$$\sum_{i=1}^{N}(\mathbf{p}_i^T\cdot\mathbf{n}+d)^2+\alpha\cdot\sum_{i=1}^{N}(1-(\mathbf{n}_i^T\cdot\mathbf{n})^2)-\lambda\cdot(\mathbf{n}^T\cdot\mathbf{n}-1) \quad (2)$$

according to n_x,n_y,n_z,d,λ, and have to make these derivatives equal to zero. Computing first the derivative according to d, we obtain:

$$d=-\mathbf{c}^T\cdot\mathbf{n}. \quad (3)$$

where

$$\mathbf{c}=\frac{1}{N}\cdot\sum_{i=1}^{N}\mathbf{p}_i$$

is the *centroid* of leaf points. Computing the derivatives according to n_x,n_y,n_z, and substituting formula 3 into the resulting equation, we get:

$$\mathbf{A}\cdot\mathbf{n}=\lambda\cdot\mathbf{n}, \quad (4)$$

where matrix $\mathbf{A}$ is

$$\mathbf{A}=\sum_{i=1}^{N}(\mathbf{p}_i-\mathbf{c})\cdot(\mathbf{p}_i-\mathbf{c})^T-\alpha\cdot\sum_{i=1}^{N}\mathbf{n}_i\cdot\mathbf{n}_i^T. \quad (5)$$

Note that the extremal normal vector is the eigenvector of symmetric matrix $\mathbf{A}$. In order to guarantee that this extremum is a minimum, we have to select that eigenvector which corresponds to the smallest eigenvalue. We could calculate the eigenvalues and eigenvectors, which requires the solution of the third order characteristic equation. On the other hand, we can take advantage that if $\mathbf{B}$ is a symmetric 3×3 matrix, then iteration $\mathbf{B}^n\cdot\mathbf{x}_0$ converges to a vector corresponding to the largest eigenvalue from an arbitrary, non-zero vector $\mathbf{x}_0$ [17]. Thus if we select $\mathbf{B}=\mathbf{A}^{-1}$, then the iteration will result in the eigenvector of the smallest eigenvalue.

Figure 4: *Two from the 32 leaf clusters representing the tree*

2.3. Smooth level of detail

The accuracy of the impostor representation can be controlled by the number clusters. If clusters are organized hierarchically, then the accuracy can be dynamically set by specifying the used level of the hierarchy. The level is selected according to the viewing distance, which results in a level of detail approach. To make the changes smooth, we can interpolate the plane parameters of the two enclosing levels.

3. Indirect texturing

The proposed method uses a single texture for the impostor that includes many leaves. It means that the impostor texture should have high resolution in order to accurately represent the textures of individual leaves. Note that lower resolution textures lead to poor results even if sophisticated texture filtering is applied (figure 6).

The resolution of the impostor texture can be reduced without sampling artifacts if the impostor stores only the position and orientation of the leaves, while the leaves rotated at different angles are put in a separate texture (figure 5). Let us identify the rectangles of the projections of the leaves on the impostor. Each rectangle is defined by its lower left coordinate, the orientation angle of the leaf inside this rectangle, and a global scale parameter (due to clustering the impostor planes are roughly parallel to the leaves thus leaf sizes are approximately kept constant by the projections). If we put the lower left coordinate of the enclosing rectangle and the orientation angle into the impostor texture, and fill up the texels that are not covered by leaves by a constant value outside [0,1], then this texture has constant color rectangles, which can be down-sampled. In fact, this down-sampling could replace a rectangle of the leaf projection by a single texel. We call this low resolution texture as the *leaf distribution impostor*. During rendering, we address this leaf distribution impostor with texture coordinates u, v. If the first color coordinate of the looked up texel value is outside the [0,1] interval, then this texture coordinate does not address a leaf, and thus should be regarded as transparent. However, if the first coordinate is in [0,1], then first two color coordinates r, g are considered as the texture coordinates of the lower left corner of the leaf rectangle, and color coordinate b is regarded as the rotation angle of the leaf. The rotation angle selects the texture that represents this rotation, and u, v texture coordinates are translated by r, g and scaled by size s of the leaf rectangle, i.e. $u' = (u-r)/s, v' = (v-g)/s$ will be the texture coordinates of the single leaf texture selected by component b.

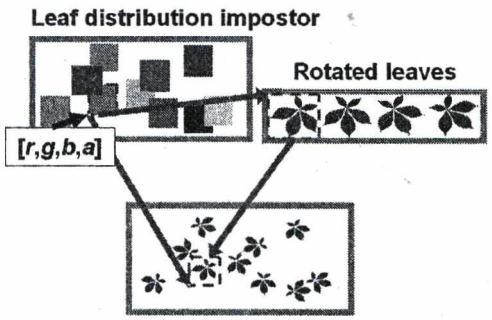


Figure 5: *Indirect texturing. The billboard is replaced by a leaf distribution impostor and the rotated images of the leaves.*

A texel of the leaf distribution impostor stores the position and the orientation of at most a single leaf whose bounding rectangle overlaps with this texel. As can be seen in figure 5, it may happen that multiple bounding rectangles overlap. Since only one of the overlapping leaf can be stored in a single pixel, this simplification may result in noticeable artifacts in form of clipped leaves if a fully transparent part of the leaf texture occludes another leaf. Such artifacts can be mostly eliminated by using not just one but two leaf distribution impostors. The first impostor is created by rendering leafs in ascending, while the second with descending order. Thus if two leaves overlap, the first leaf is visible in the first impostor, while the second is visible in the second impostor. During the rendering of the indirect texture the fragment shader reads both impostors. If a texel contains no leaf, then the fragment is ignored. If both impostors refer to the same leaf, then its leaf texture is scaled and is applied. However, if the two impostors refer to two different leaves, then the first leaf texture is looked up and it is checked whether its corresponding texel is transparent. In case of a non-transparent texel, the first leaf texture is used. In case of a transparent texel, we look up the second impostor and use the leaf selected by this.

Figure 6: *Conventional filtered texturing. The image gets poor when the observer is close.*

Note that indirect texturing exploits the fact that the position of the leaves are not as important as their color pattern, thus stores these two kinds of information separately. The final texture is obtained run time without any storage overhead.

4. Shading

Billboard clouds approximately preserve the original geometry, thus they can also replace the original tree when shad-

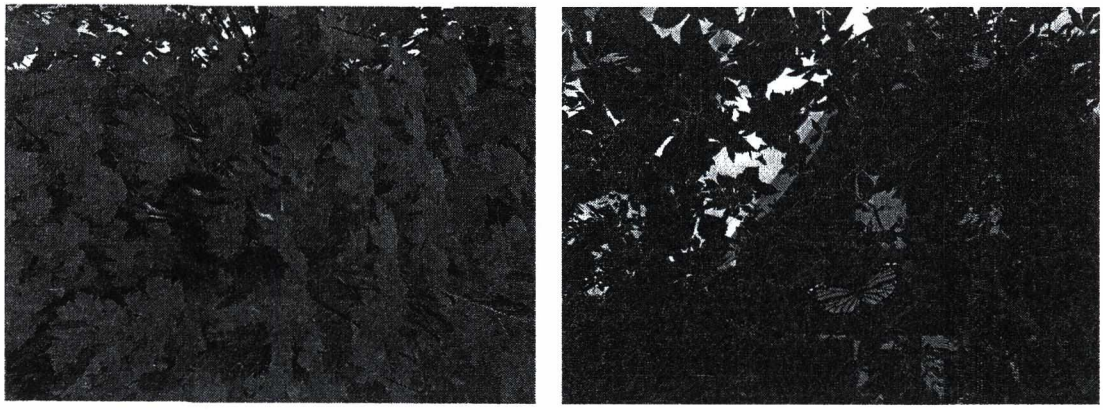


Figure 7: *Indirect texturing. Note the high apparent texture resolution.*

ows or global illumination effects are computed. However, when direct illumination is calculated, the application of the same normal for all leaves belonging to a cluster would make the planar substitution too obvious for the observer. Thus for direct illumination computation, the original normals of the leaves are used, which are stored in a normal map associated with the impostor plane. If no indirect texturing is used, then the normal map may store normals in object space. However, in the case of indirect texturing the normal map should have a similar structure as the rotated leaves texture. The values in this texture store the modulation with respect to the main projection direction, and should be transformed to tangent space during rendering.

Figure 8: *Trees with illumination and shadow computation.*

To compute shadows, i.e. the visibility from point and directional lights (the direction of the sun in particular), the classical z-buffer shadow algorithm has been implemented with the modification that our implementation skips those fragments which corresponds to transparent texels of the impostor planes.

Assuming a single directional light source representing the sun, and sky light illumination, the reflected radiance of a leaf is computed by the following approximation of the rendering equation:

$$L^r = L^{sun} \cdot (k_d \cdot (\vec{N} \cdot \vec{S}) + k_s \cdot (\vec{N} \cdot \vec{H})^n) \cdot v + L^{env} \cdot a,$$

where L^{sun} and $\vec{S}$ are the radiance and direction of the sun, respectively, k_d is the wavelength dependent diffuse reflectance of the leaf, k_s is the specular reflectance, n is the shininess, $\vec{N}$ is the unit normal vector, $\vec{H}$ is the unit halfway vector between the illumination and viewing directions, v is the visibility indicator obtained with shadow mapping, L^{env} is the ambient radiance of the leaf's local environment, and a is the albedo of the leaf. The albedo and the diffuse/specular reflection parameters are not independent, but the albedo can be expressed from them. We use the following approximation [5]:

$$a \approx k_d \cdot \pi + k_s \cdot \frac{2\pi}{n+1}.$$

In order to obtain the ambient radiance of a leaf, that is to simulate indirect illumination and sky lighting, an obscurances approach has been used [10]. During preprocessing, random global directions are sampled for a single billboard cloud tree. The global direction is used as a projection direction. The tree is rendered with the graphics hardware applying a depth peeling algorithm to find not only the closest visible point but all pairs of points that are visible from each other in the given direction. This information is used then to obtain a global occlusion factor o approximating the portion of the illuminating hemisphere in which the sky is not visible from the leaf. Simultaneously the average color of the occluding leaves o_c is also calculated. Since both faces of the leaves can be lit, we compute two obscurance maps, the first represents leaf sides having positive z coordinate in their

normal vector, the second represents leaf sides having negative values. Using these values, the ambient radiance around a leaf can be approximated as:

$$L^{env} \approx L^{sky}(\vec{N}) \cdot (1-o) + o_c \cdot o,$$

where $L^{sky}(\vec{N})$ is the average radiance of the sky around the direction of the normal vector of the leaf, and o and o_c are the obscurance values depending on the actual leaf side.

The simplified pixel shader code evaluating the color of a leaf point is:

```
float3 N = tex2D(normalmap, uv);
float4 obs = tex2d(obsurancemap, uv);
float  o_c = obs.a;  // ratio occlusion
float  o   = obs.rgb;// occluder color
float3 Lenv = Lsky * (1-o) + o_c * o;
float3 diff = kd * max(dot(N,L),0);
float  spec = ks * pow(max(dot(H,N),0), n);
return Lsun * (diff + spec) * v + a * Lenv;
```

This shader gets the normal vector (`N`) from the normal map and the obscurance value (`obs`) from the obscurance map, and approximates the ambient lighting (`Lenv`) around the leaf. The illumination formula then uses material properties like diffuse reflection factor `kd`, specular reflection factor `ks`, and albedo `a`. Parameter `Lsun` is the intensity of the directional or positional light source and `v` is the indicator of its visibility that is determined by the shadow algorithm.

5. Results

The proposed algorithm has been implemented in OpenGL/Cg environment, integrated into the Ogre3D game engine, and run on an NV6800GT graphics card. The tree used in the experiments is a European chestnut (Castanae Sativa) having 11291 leaves defined by 112910 faces and 338731 vertices. The trunk of the tree has 46174 faces. The leaves have been converted to 32 billboards using the proposed algorithm. When we did not apply indirect texturing, the billboard resolution was 512×512. Setting the screen resolution to 1024×768, leaf rendering is speeded up from 40 FPS to 278 FPS when the leaf polygons are replaced by the billboard cloud. The comparison of the images of the original polygonal tree and the billboard cloud tree is shown in figure 3. Note that this level of similarity is maintained for all directions, and the impostor tree also provides realistic parallax effects.

Figure 9 shows a terrain of 4960 polygons with 2000 trees rendered on 30 FPS without instancing (the rendering is CPU limited). We used the proposed level of detail techniques to dynamically reduce the number of leaf cluster impostors per tree for distant trees from 32 to 16 and even to 8.

6. Conclusions

This paper presented a tree rendering algorithm where clusters of leaves are represented by semi-transparent billboards. The automatic clustering algorithm finds an impostor in a way that the represented leaves lie approximately in its plane. This approach is equivalent to moving and rotating the leaves a little toward their common planes, thus this representation keeps much of the original geometry information. Since the impostors are not rotated with the camera, the proposed representation can provide parallax effects and view dependent occlusions for any direction. We also discussed how leaf textures can be visualized without posing high resolution requirements for the impostors, and the possibility of including smooth level of detail techniques.

7. Acknowledgement

This work has been supported by OTKA (ref. No.: T042735), GameTools FP6 (IST-2-004363) project, TIN2004-07451-C03-01 project from the Spanish Government, and by the Spanish-Hungarian Fund (E-26/04).

References

1. C. Andujar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and J. Vinacua. Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum*, 23(3):401–410, 2004.

2. X. Décoret, F. Durand, F. Sillion, and J. Dorsey. Billboard clouds for extreme model simplification. In *SIGGRAPH '2003 Proceedings*, pages 689–696, 2003.

3. O. Deussen, P. Hanrahan, R. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz. Interactive modeling and rendering of plant ecosystems. In *SIGGRAPH '98 Proceedings*, pages 275–286, 1998.

4. I. Garcia, M. Sbert, and L. Szirmay-Kalos. Leaf cluster impostors for tree rendering with parallax. In *Eurographics Conference. Short papers.*, 2005.

5. E. Lafortune and Y. D. Willems. Using the modified Phong reflectance model for physically based rendering. Technical Report RP-CW-197, Department of Computing Science, K.U. Leuven, 1994.

6. D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002.

7. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

8. N. Max, O. Deussen, and B. Keating. Hierarchical image-based rendering using texture mapping. In *Eurographics Workshop on Rendering*, pages 57–62, 1999.

9. N. Max and K. Ohsaki. Rendering trees from precomputed z-buffer views. In *Eurographics Workshop on Rendering*, pages 74–81, 1995.

Figure 9: *Two snapshots from a 30 FPS animation showing 2000 trees without shadow and illumination computation*

10. Alex Mendez, Mateu Sbert, Jordi Cata, Nico Sunyer, and Sergi Funtane. Real-time obscurances with color bleeding. In Wolfgang Engel, editor, *ShaderX4: Advanced Rendering Techniques*. Charles River Media, 2005.

11. A. Meyer, F. Neyeret, and Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*, 2001.

12. A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics*, 23(3), 2004.

13. G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In *Eurographics Workshop on Rendering*, pages 151–162, 1997.

14. G. Szijártó. 2.5 dimensional impostors for realistic trees and forests. In Kim Pallister, editor, *Game Programming Gems 5*, pages 527–538. Charles River Media, 2005.

15. G. Szijártó and J. Koloszár. Hardware accelerated rendering of foliage for real-time applications. In *Spring Conference on Computer Graphics*, 2003.

16. G. Szijártó and J. Koloszár. Real-time hardware accelerated rendering of forests at human scale. *Journal of WSCG*, 2004.

17. E. Weisstein. World of mathematics. 2003. http://mathworld.wolfram.com/Eigenvector.html.

Hungarian Talking Head according to MPEG-4

Zs. Ruttkay,[1,2] A. Fazekas,[3] P. Rigó[3]

[1]Information Technology Faculty, PPKE, Budapest, Hungary
[2]Dept. of Computer Science, University of Twente, The Netherlands
[3]Faculty of Informatics, University of Debrecen, Hungary

Abstract

In this paper we introduce a framework for Hungarian Talking Heads, to utter written text in synthetic speech, accompanied by appropriate moving lips and facial expressions. The faces to be animated, as well as the visemes are given in terms of the MPEG-4 standard; level of detail of the mouth may be different. The CharToon system is used to generate the movements of the mouth and the face, according to the timing information provided by the text to speech engine. Facial expressions may be scripted on a low level, or on a high level, by using the GESTYLE markup language to annotate the text to be spoken.

Categories and Subject Descriptors (according to ACM CCS): 1.3.3 [Computer Graphics]: Animation

1. Introduction

An *Embodied Conversational Agent* (ECA) is a 2d or 3d computer graphics model residing on the screen, resembling to a real human in its embodiment and its communicational skills [3]. In the past 10 years, ECA research has been established as a specific field in the intersection of computer graphics, artificial intelligence, human-computer interaction, and inspired by such non-CS disciplines as psychology and cultural anthropology and even classical animation and fine arts.

The boom in research had two reasons. On the one hand, it is per se a challenging task to model complex and subtle phenomena of language processing, gesturing and facial expressions, emotional and cognitive processes reminiscent of humans, and integrate these computational models into a coherent virtual human. On the other hand, the ECA technology opens a series of applications: first of all, an ECA can serve as a 'most natural user interface' for traditional complex computer systems, making such services accessible for a broad public. Besides, ECAs can take the cast of some traditional human professions, like tutor [12], psychotherapist [13], salesperson [1], weather forecast presenter [15] or newsreader [22]. It has been proven that people react to ECAs principally the same way as they do to real humans [23]. It has been also shown that people are very critical concerning the quality of speech and nonverbal signals of an ECA – the subjective judgment and objective effect of an ECA depends on such subtle aspects like the eyebrow-usage [11] or body posture [14] of the ECA.

A *Talking Head* (TH) is a specific ECA, having only a synthetic face as embodiment, which is capable to talk. As a minimum requirement, a TH should have his mouth moving according to speech. The base-line criterion for lip motion is believability and life-likeness: it would look odd to have a puppet with mouth, which does not move during speech, so the mouth movement should be in sync with the spoken text. The other extreme is lip motion quality good enough to understand speech by lip-reading e.g. for hearing impaired users, or to teach correct pronunciation for patients with speech problems or foreigners learning a language. Hence the subtlety of the lip motion (and accordingly, the head and mouth model) should depend on the application domain.

Besides the mouse moving, there appear additional signals on the human face during speech. Eyebrows and gaze are used for e.g. punctuating speech, indicating syntactical structures or emphasis; emotional content is reflected by facial expressions, some aspects of the

semantics talked about is illustrated (e.g. 'big' may be illustrated by raised eyebrow, open eyelids), turn taking management is regulated by gaze, and cognitive state may be indicated too (e.g. when remembering or thinking, one looks up right) [21]. Hence a TH should also exhibit eye-gaze, facial expressions and head movements next to lip sync, according to the content of the speech, state of the speaker, stage of discourse, etc.

In this paper we present a framework for a Hungarian Talking Head. That is, a head which, taking typed text as input, speaks it out in synthetic speech with the lip moving accordingly. Besides, it is possible to prescribe facial expressions and gaze behavior for the head. As we assume a head which can be animated according to the MPEG-4 standard [10], the framework can be used to drive different faces, as long as they comply with the standard.

In the rest, we first discuss related work on Talking Heads. Then in 3.1 we introduce the MPEG-4 standard for facial animation, and in 3.2 explain how the visemes for Hungarian have been defined using this standard. In 3.3 we discuss the architecture of the system, then we tell about scripting of facial expressions on two levels, and finally in 3.5 talk about the implementation details. In section 4 we sum up our system, outline further work and possible applications.

2. Related work

The generation of lip sync in itself has been a research issue for distinct languages [20, 5]; and nowadays the first commercial systems are available providing lip sync for several languages [19]. The basic idea behind lip sync is to identify characteristic lip shapes, so-called *visemes,* and to use these as key frames for the animation of the lips. For generation of lip movement, the following decisions need to be taken:

(1) mapping of acoustic units of the spoken language, the so-called *phonemes*, to visemes;
(2) precise placement of the viseme during the duration of the phoneme;
(3) interpolation and coarticulation between visemes.

The 2nd and 3rd tasks involve subtle timing and articulation. When talking, it depends on the phoneme if the associated mouth shape is to be produced at the beginning of the duration of pronunciation of the sound (e.g. for p, b) or the mouths shape is to be kept (more or less) during the 'middle' of the duration of the sound (like in case of long vowels). Also, the mouth shape is context sensitive: the viseme associated with a sound may be influenced by the 2nd, 3rd or even further away sound to be pronounced.

This phenomenon of *coarticulation* has been tackled by different approaches: using coarticulation rules [4], time profile functions of addition of effect of individual visemes [5], machine learning techniques to generate visual speech from audio [2]. For Hungarian, L. Czap recently provided a Hungarian viseme set, consisting of 17 characteristic mouth shapes (+ 1 for closed mouth), based on descriptive works of the Hungarian sounds and analysis of video images of real person's lip movements [6]. He characterized the visemes in terms of 3 parameters of the mouth, mouth opening, width and intensity which is proportional to the area surrounded by the inner mouth contour. For each viseme, he classified each parameter as dominant, flexible or free, depending on how resistant the parameter is to the effects of the surrounding visemes. He used these characteristics for a multi-pass generation process to determine the dynamism of the mouth. He gave example of a Hungarian talking head which is driven by acoustic signal. In his work, the visemes were designed for a single virtual head, and the mouth was animated by analyzing the acoustic signal of real speech.

Our work on a Hungarian Talking Head is built upon our previous work for Talking Heads in English and Dutch [28]. In defining the viseme set, we rely on L. Czap's analytical results. Our Hungarian Talking Head, on the other hand, is different to his work in the following aspects:

(1) The input for our talking head is the typed text the head is supposed to utter. The acoustic signal is generated by a TTS engine. The mouth is driven not by the acoustic signal, but by the timed phoneme sequence generated by the TTS engine.
(2) The visemes are given in terms of a number of MPEG-4 mouth (and tongue and jaw) parameters. Hence a viseme in our work is defined in an explicit way by parameters in all detail.
(3) The approach is independent of the head model, as the visemes can be used to drive the animation of any facial model which can be animated according to MPEG-4 parameters.

3. Framework for Hungarian Talking Heads

3.1 Facial animation in MPEG-4

According to the MPEG-4 standard, deformation of facial features – eyebrows, eyelids, mouth, cheek – contributing to facial expressions and speech are given as displacement (in x or y direction, for some cases in z

direction) of well-defined points of these features [10, 18], each corresponding to a *Facial Action Parameter* (FAP). The displacements are given in terms of certain normalized distances on the face, assuring that a FAPs sequence has similar effects on faces with different geometry. The shape of the mouth is to be given by displacement of points shown in Fig. 1, corresponding to FAPs.

In order to animate an MPEG-4 compliant face, for each frame the value of all the FAPs is to be given. It is the facial animation engine's responsibility to assure that the whole face gets rendered accordingly.

3.2 Hungarian phonemes and visemes

In our framework we use the Profivox Hungarian TTS engine [17], and thus rely on the *Phoneme Set* of this system. The TTS system produces, next to the audio file with the speech, a timed sequence of the phonemes corresponding to the generated speech.

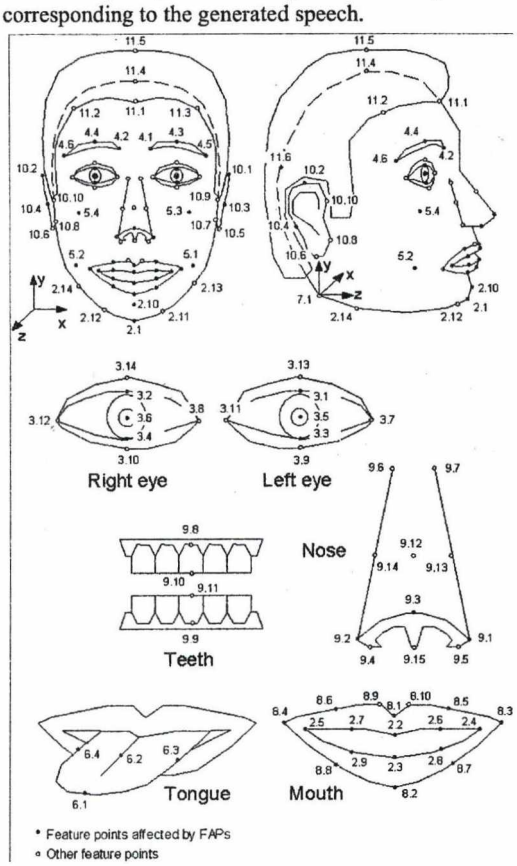


Figure 1: The points of the face (in black) corresponding to MPEG-4 FAPs.

The mouth shapes to accompany the speech are chosen from a *Viseme Set*, and the correspondence is given by a *Phoneme to Viseme Mapping Table*. For Hungarian, we took as a starting point the visemes suggested by L. Czap [6]. For each viseme to be used, the corresponding mouth shape has to be designed once, in terms of (head independent) mouth shape parameters. As we already had at our disposal a Viseme Set used for English [26], we re-used some of the mouth shapes for the Hungarian visemes. Only for sounds ü, ö and é we developed new, specific mouth shapes. The mouth shape for each viseme is expressed in terms of MPEG-4 parameters; hence the visemes can be used for any MPEG-4 compliant face.

Moreover, for 2D faces, we developed earlier a set of mouth designs as components to be used when creating faces with different level of detail and realism [26]. These mouths vary from the most detailed one, with tongue and teeth, to a simple ellipsoid mouth, requiring less and less MPEG-4 parameters to define the shape, see Fig. 2. The mouths were designed such, that (subsets of) the very same parameters result in similar mouth shapes, albeit with less detail. Hence the same viseme set and lip sync production mechanism can be used for different 2d faces, without any adjustment of the Mapping Table or the controlling sequence of FAPs. Simply FAPs not supported by the given mouth will be discarded, and the remaining ones will assure that the right viseme is produced for the given mouth.

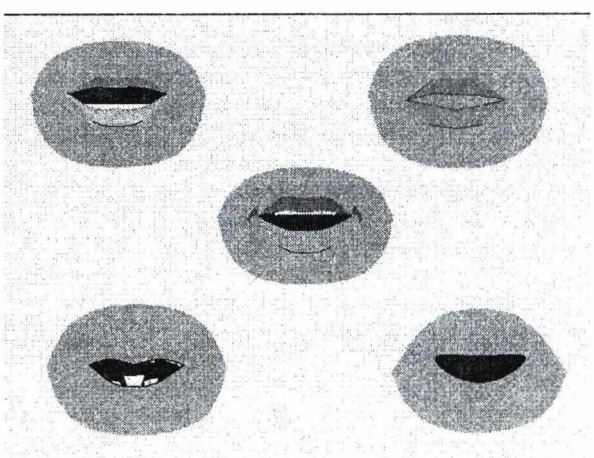


Figure 2: MPEG-4 compliant mouth shapes for 2d CharToon faces of different level of detail.

Finally, different Viseme Sets and/or Mapping Tables can be used. Hence it is possible to test and refine the Viseme Sets to be used in an incremental way.

3.3 The architecture

The architecture of our system is given in Fig. 3. The animation of the face takes place by an adapted version of the CharToon system [25]. The generation of visual speech takes place in the following steps:

Phoneme sequence generation

The Hungarian TTS system Profivox generates a time sequence of phonemes, and the corresponding audio file.

Viseme sequence generation

Based on the timing of the phonemes and the adjustment principle for the corresponding viseme, for each phoneme a viseme is prescribed at a given time. In our current setting, we use the two types of *adjustment principles* (similar to ones we used for Dutch talking heads earlier).

snapshot articulation: a peak-shape articulation function is placed for the duration of the phoneme, the peek being on the first third of the duration for plosives, and on the half of the duration for other phonemes;

held articulation: a trapezoid-shape articulation function is used; the characteristic viseme shape is to be 'held' for the time interval of the inner third of the entire duration of the phoneme for long vowels, unless the duration is too short.

The prescribed times are rounded to nearest time on a dense discrete time scale, due to the (tunable) discrete internal representation of time of the animation system.

FAPs sequence generation

As each viseme is given in terms MPEG-4 FAPs, the viseme sequence is transformed into FAP values prescribed for the articulation times, which are either a snapshot, or a duration, if a trapezoid-shape articulation function is to be used.

FAPs interpolation

For each FAP, a continuous function, based on the FAP values prescribed by the viseme sequence, is generated by interpolation. At present, we use linear interpolation, but in principle different interpolation or approximating functions can be used (see more in discussion).

Drawing the animated face

For each frame at the rate specified by the user (at least 15 fps) the FAPs at the given time are sampled, and the head model (which can be some 2d CharToon head, or a 3d realistic head) is deformed accordingly. The images can be dumped and concatenated to a movie, which can be played with the accompanying synthetic speech sound file.

3.4 Scripting facial expressions

Besides the mouth moving according to speech articulation, other features of the face – eyebrows, eyelids, eyes – may move too, to express emotions, punctuate speech, blink, etc. Moreover, the articulation of the mouth may be also influenced by facial expressions (e.g. speaking while smiling). How to prescribe such additional facial motions? And how to take care of possible conflicting parameters for each FAP?

Different facial expressions are also given in terms of linear FAP functions for the duration of the expression. These expressions, designed once, form the *repertoire* of the face [9]. Any expression from the repertoire may be prescribed for a certain *duration* and with an *intensity* at a *starting time*. A facial animation may be given by a simple low-level script, containing a series of prescribed facial expression with the above three parameters. The CharToon system can generate an animation from such a script, by taking the (time and intensity-scaled) sample from the facial expression repertoire. As a first input in the script, an animation produced as described earlier, corresponding to the lip motion may be given. In such a case, for each following expression in the script its *display preference* may also be given, which regulates how the final FAP value for lip parameters is computed from the one from the viseme and from the one prescribed for an expression.

The above low-level script may be produced by hand, or may be the output of the GESTYLE language processing module. GESTYLE [16] is a text markup language, where the markup tags embrace parts of the text which need to be uttered e.g. with emphasis, or with a smile. In course of processing the marked up text, timing information is taken from the TTS engine, and the facial expressions, defined in a library, are inserted at the right moments, according to possible parameters given (intensity, symmetry). GESTYLE can be used on *meaning level*, that is when not (only) facial expressions are prescribed by tags, but also *content* to be expressed, and the habits of the character of expressing certain meanings by facial signals are given in a *style*

dictionary with possible alternatives. In this way non-repetitive and individual facial animations can be generated for talking heads. By using Profivox as source of the timing information for Hungarian phonemes, the whole richness of the GESTYLE control can be exploited for Hungarian Talking Heads.

3.5 Implementation issues

The facial animation modules form part of the CharToon facial animation system, developed by the first author and her colleagues, implemented in Java 1.1.

This system has been coupled with the Profivox TTS engine developed at BME [17], which was programmed in C++. We developed an integrated environment to generate a movie of speaking talking head based on CharToon and Profivox modules. The integration of GESTYLE and Profivox is in progress.

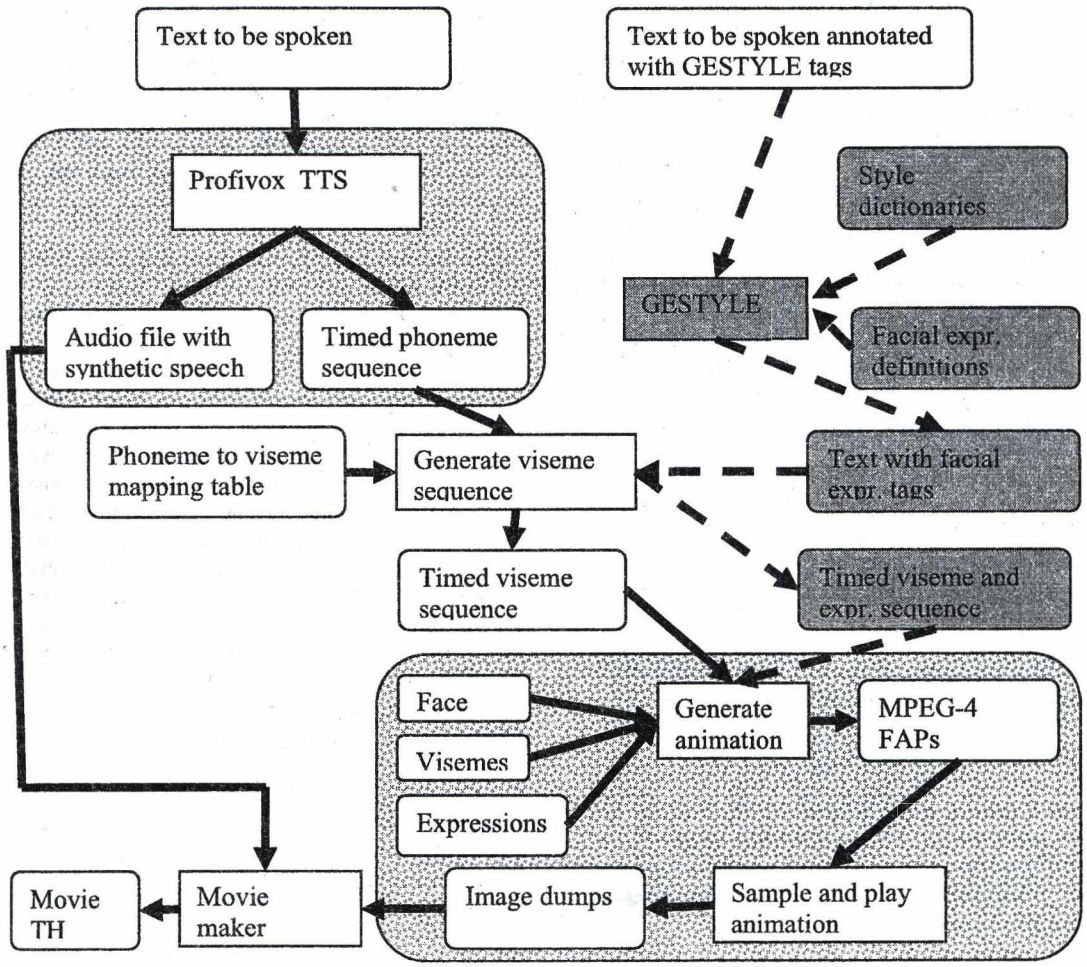


Figure 3: Architecture of a TH with the Profivox and CharToon components and the possible GESTYLE extension.

4 Discussion

4.1 Summary of our work

We have developed a Hungarian Talking Head system with the following characteristics:

Lip-sync is produced automatically, based on text to be uttered, and the timing information of phonemes made available by the TTS engine.

The visemes are expressed in terms of MPEG-4 parameters; hence the framework can be used for any facial model which can be animated according to the MPEG-4 standard.

The number and types of visemes to be used, their mapping, as well as articulation timing with respect to the duration of the sound are given as explicit and easy to change data, so the framework allows experimentation with modified data sets.

For 2d CharToon faces, a set of mouth variants as plug-in components for a face are available, which correspond to different level of details, and can be controlled without any change in the viseme set and mapping

4.2 Experimenting with lip-sync

Our framework can be used easily to refine some visemes, phoneme-viseme mapping and viseme adjustment principle, and hence, experiment with and test different settings in order to study the phenomena of Hungarian lip-sync and improve the quality of the visual speech.

As the CharToon facial animation system also supports the scaling of pieces of animation to be used, so the phenomena of reducing mouth motion along some FAPs, e.g. in case of fast speech as suggested by L. Czap, could be achieved.

As of coarticulation, in the first implementation it is only the interpolation which assures the blending of on- and offset durations of visemes. However, the CharToon facial animation system has an extension FaceInc [24], where constraints may be assigned to certain subset of parameters in pieces of animations, and thus in visemes. Such *constrained animations* then can be re-used and inserted, and a constrain solving mechanism will assure that the total animation is readjusted such that the constraints remain valid. In our case, this would mean that visemes are not defined as strict mouth shapes, but as a set of possible mouth shapes (variations) expressed in terms of constraints on single or multiple FAPs. By this mechanism, the for the production of the sound essential configurations could be assured, and flexibility could be introduced.

As GESTYLE also handles emotional speech [27], it would be worth experimenting with the defining and controlling expressive speech in Profivox, which is a task for the TTS experts.

4.3 Real-time performance

Ultimately, we would like to use the presented framework to control a responsive Hungarian TH, in applications like a information provider or Hungarian language tutor. That is, an ECA which can generate in near real time the speech and the facial animation for a response in Hungarian. As of the technical requirement, we shall optimize CharToon's Face Player engine (originally not designed for such a usage). Particularly, no movie will be generated, by saving images for each frame rendered, but a new player will be used to render the face directly based on its vector-based graphics definition, and play the speech audio. Also, we will explore the possibility of driving real-tine 3d MPEG-4 compatible models other than the currently used single one.

4.4 Perception-driven control

Finally, we plan to extend the framework with an image-based perception module, which would provide information about the user in front of the screen: if somebody is present, satisfied or frustrated, is about to leave, etc. For this purpose, we plan to couple robust facial recognition [7] with high-level behavioral control of the TH. Based on face detection [8] the user's authorities can be checked, too. This information, extended with possible other sources influencing the ECA's emotional state (e.g. status of a game being played with the user) could be used to prescribe the facial expressions on the fly to accompany utterances.

4.5 Emotional visual speech

The GESTYLE language is prepared to define and use also *speech habits and styles* of a virtual character [27]. Hence it would be a natural interface for emotional speech for Hungarian Talking Heads, where the emotional characteristics are to be given in Profivox parameters. The is interest in using such an emotional Hungarian Talking Head as a medium to test the perception of emotional expressions by healthy and psychiatrically ill people. Also it would be interesting to explore possible cultural differences, e.g. by repeating our earlier Dutch perception experiments with Hungarian subjects. In itself, the exploration of Hungarian visual speech, beyond viseme coarticulation, is an open terrain.

Acknowledgement

We thank Géza Németh for making Profivox available to generate speech for the Hungarian Talking Head, and for Géza Kiss for providing technical help to use the TTS software.

The first author's contribution was made possible due to the Szent-Györgyi Albert Fellowship at the Pázmány Péter Catholic University in 2005.

Bibliography

1. E. Andre, T. Rist: Presenting through performing: On the use of multiple lifelike characters in knowledge-based presentation systems. In: H. Lieberman (ed.): International Conference on Intelligent User Interfaces 2000. pp.1-8.
2. M.E. Brand: Voice puppetry, Proc. of ACM SIGGRAPH 1999. pp 21-28 ,
3. J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.): Embodied Conversational Agents, MIT Press, Cambridge, MA. 2000.
4. J. Cassell, C. Pelachaud, N.I. Badler, M. Steedman, B. Achorn, T. Becket, B. Douville, S. Prevost, M. Stone, Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents, Proc. of SIGGRAPH'94, 1994. pp. 413-420.
5. M. M., Cohen, D. Massaro: Modeling coarticulation in synthetic visual speech. In N. M. Thalmann & D. Thalmann (Eds.) Models and Techniques in Computer Animation. Tokyo: Springer-Verlag, 1993.
6. Czap, L.: Audiovizuális beszédfelismerés és beszédszintézis, PhD értekezées, 2004, BME Távközlési és Médiainformatikai Tanszék
7. A. Fazekas, I. Sánta: Recognition of facial gestures based on support vector machines, Lectures Notes in Computer Science **3522**. 2005. pp. 469-475.
8. A. Fazekas, C. Kotropoulos, I. Buciu, I. Pitas: Support vector machines on the space of Walsh functions and their properties, Proc. of 2nd International Symposium on Image and Signal Processing and Analysis, 19-21 June, 2001. Pula, Croatia, pp. 43-48.
9. J. Hendrix, Zs. Ruttkay, P. ten Hagen, H. Noot, A Lelievre, B.de Ruiter: A facial repertoire for avatars, Proceedings of the Workshop "Interacting Agents", Enschede, The Netherlands, pp. 27-46. 2000.
10. ISO, Text for ISO/IEC FDIS 14 496-1,2 ISO/IEC JTC1/SC29/WG11, N2502, Nov. 1998.
11. E. Krahmer, Zs. Ruttkay, M. Swerts, W. Wesselink: Audiovisual cues to prominence, Proceedings International Conference Spoken Language Processing, Denver, CO, 2002, pp. 1933-1936.
12. J. Lester, S. Converse, S. Kahler, S. Barlow, B. Stone, R Bhogal: The Persona effect: affective impact of animated pedagogical agents, Proc. of CHI'97, 1997, pp. 359-366.
13. S. Marsella: Interactive Pedagogical drama: Carmen's bright ideas assessed. IVA 2003: pp. 1-4
14. C. Nass, K. Isbister, E. Lee: Truth is beauty: researching embodied conversational agents. In: J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.): Embodied Conversational Agents, MIT Press, Cambridge, MA. 2000.
15. T. Noma, L.Zhao, N. Badler. Design of a virtual human presenter, IEEE Computer Graphics and Applications 20(4), July/August 2000, pp. 79-85.
16. H. Noot, Zs.Ruttkay: Variations in gesturing and speech by GESTYLE, International Journal of Human-Computer Studies, Special Issue on 'Subtle Expressivity for Characters and Robots', to appear in 2005.
17. G. Olaszy, G. Németh, P. Olaszi, G. Kiss, Cs. Zainkó, G. Gordos: Profivox – a Hungarian TTS System for Telecommunications Applications. International Journal of Speech Technology. Vol 3-4. Kluwer Academic Publishers. 2000. pp. 201-215.
18. I. S. Pandzic, R. Forchheimer (editors): MPEG-4 Facial Animation - The Standard, Implementations and Applications, John Wiley & Sons, 2002.
19. I. S. Pandzic, J. Ahlberg, M. Wzorek, P. Rudol , M. Mosmondor: Faces everywhere: Towards ubiquitous production and delivery of face animation, Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia, Norrkoping, Sweden, 2003.
20. C. Pelachaud, E. Magno-Caldognetto, C. Zmarich, P. Cosi, Modelling an Italian Talking Head, Audio-Visual Speech Processing,2001, pp, 7-9
21. I. Poggi, C. Pelachaud, Performative facial Expressions in Animated Faces, In J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.), Embodied Conversational Agents, Cambridge (Mass.): MIT Press, 2000. pp.155-188.
22. Reana, http://jerry.zavod.tel.fer.hr/humanoid/usluge/index.html
23. B. Reeves, C. Nass: The media equation. How people treat computers, television, and new media like real people and places. Stanford, CA: CSLI Publications, Cambridge University Press, 1996.
24. Ruttkay, Zs.: Constraint-based facial animation, Int. Journal of Constraints, Vol. 6. 2001. pp 85-113.
25. Zs. Ruttkay, H. Noot, Animated CharToon Faces, Proceedings of NPAR 2000 – First International

Symposium on Non Photorealistic Animation and Rendering, 2000. pp. 91-100.
26. Zs. Ruttkay, A. Lelievre: CharToon 2.1 extensions: Expression repertoire and lip sync, CWI Report INS-R0016, Amsterdam, 2000.
27. Zs. Ruttkay, V. van Moppes, H. Noot: The jovial, the reserved and the robot, Proc. of the AAMAS03 Ws on "Embodied Conversational Characters as Individuals", 15th July, 2003, Melbourne, Australia
28. Zs. Ruttkay, H. Noot: Cartoon Talking Heads, ,Proc. of the 1st Hungarian Computer Graphics Conference, 2002. Budapest, pp. 2-9.

Rendering Fire and Smoke with Spherical Billboards

Tamás Umenhoffer, László Szirmay-Kalos

Department of Control Engineering and Information Technology,
Budapest University of Technology, Hungary
Email: umitomi@freemail.hu, szirmay@iit.bme.hu

Abstract

This paper proposes an improved billboard rendering method, which renders particles as aligned quadrilaterals similarly to previous techniques, but takes into account the spherical geometry of the particles during fragment processing. The new method can eliminate billboard clipping and popping artifacts of previous techniques, happening when the participating medium contains objects, or the camera flies into the volume. This paper also describes how to use this new technique to render high detail fire and smoke on high frame rates.

1. Introduction

Participating media [1] are often represented by particle systems [5]. In case of a particle system a particle point represents its spherical neighborhood where the volume is locally homogeneous. Particle system rendering methods usually splat particles onto the screen, which substitute them with a semi-transparent, camera-aligned rectangles, called *billboards* [6].

2. Billboard clipping and popping artifacts

The main problem with billboard type particle systems is that billboards are planes, thus they have no extension along one dimension.

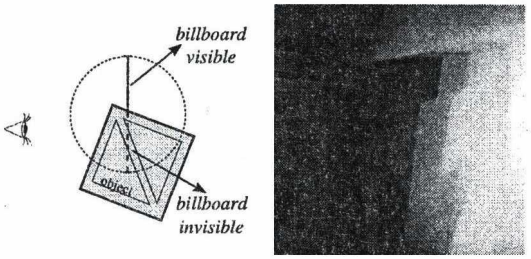


Figure 1: *Billboard clipping artifact. When the billboard plane intersects the object, transparency becomes spatially discontinuous.*

This can cause artifacts when billboards intersect objects making the intersection of the billboard plane and the object clearly noticeable (figure 1). The core of this problem is that a semi transparent billboard fades those objects that are behind it according to its transparency as if the object were fully behind the sphere of the particle. However, those objects that are in front of the billboard plane are not faded at all, thus transparency changes abruptly at the object billboard intersection.

When the camera moves into the media, billboards also cause popping artifacts. In this case, the billboard is either behind or in front of the front clipping plane and the transition between the two stages is instantaneous. The former case corresponds to a fully visible, while the latter to a fully invisible particle, which results in an abrupt change during animation (figure 2).

Solutions to solve billboard clipping artifacts in case of solid objects have already been proposed, but non of them deals with volumetric media [9, 8]. In this paper we propose a novel solution for including objects into volumetric media without billboard clipping artifacts, and also to eliminate billboard popping during animation.

3. Spherical billboards

Billboard clipping artifacts are solved by calculating the real path length a light ray travels inside a given particle, as this length controls the opacity value to be used during rendering.

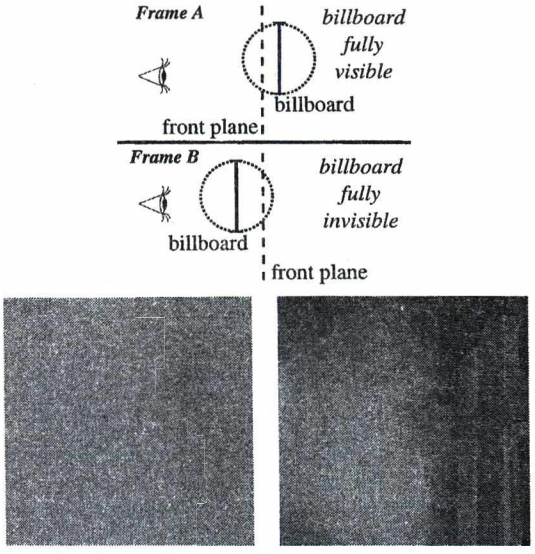


Figure 2: *Billboard popping artifact. Where the billboard gets to the other side of the front clipping plane, the transparency is discontinuous in time (the figure shows two adjacent frames in an animation).*

This calculation is done with dealing with the spherical geometry of particles instead of assuming that a particle can be represented by a planar billboard. However, in order to keep the implementation simple and fast, we still send the particles through the rendering pipeline as quadrilateral primitives, and take into account the spherical shape only during fragment processing.

Let us review the individual rendering steps. We assume that a preliminary rendering pass have been made to store the scene depth values as camera space z coordinates in a texture.

The particles are rendered as quads perpendicular to axis z, placed at the farthest point of the particle sphere from the camera to avoid unwanted front plane clipping. Disabling depth test is also needed to eliminate incorrect object-quad clipping.

When rendering a fragment of the particle, we compute the interval the ray travels inside the particle sphere in camera space. This interval is obtained with considering the scene depths and the camera's front clipping plane distance. With the use of the segment length we can compute the opacity for each fragment in such a way that both fully occluded or fully visible and partially occluded particles will be displayed correctly, giving the illusion of a volumetric media. During computation we assume that the density is uniform inside a particle.

Let us use the notations of figure 3 where a particle of

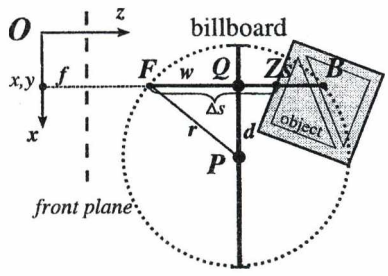


Figure 3: *Computation of the length of the ray segment traveling inside a particle sphere in camera space.*

center $\vec{P} = (x_p, y_p, z_p)$ is rendered as a quad perpendicular to axis z, and a ray is cast through point $\vec{Q} = (x, y, z_q)$ of the quadrilateral. The radius of the particle sphere is r, the distance between the ray and the particle center is $d = \sqrt{(x-x_p)^2 + (y-y_p)^2}$. The closest and farthest points of the particle sphere on the ray from the camera are $\vec{F}$ and $\vec{B}$, respectively. The ray travels inside the particle in interval $[|\vec{F}|, |\vec{B}|]$, where $|\vec{F}| = z_p - \sqrt{r^2 - d^2}$ and $|\vec{B}| = z_p + \sqrt{r^2 - d^2}$.

In order to take into account the front clipping plane and the object depths, these distances must be altered. First to eliminate popping artifacts, we should ensure that $|\vec{F}|$ is never smaller than the front clipping plane distance f, thus the distance the ray travels in the particle before reaching the front plane is not included. Secondly, we should also ensure that $|\vec{F}|$ is never greater than Z_s, which is the stored scene depth at the given pixel, thus the distance travelled inside the objects is not considered.

With these altered distances we can obtain the real length of the ray segment:

$$\Delta s = max(f, |\vec{F}|) - min(Z_s, |\vec{B}|),$$

and also the corresponding opacity value.

The fragment program gets some of its inputs from the vertex shader: the particle position in camera space (*In.P*), the shaded point in camera space (*In.Q*), the particle radius (*In.r*), the screen coordinates of the shaded point (*In.screenCoord*). The fragment program also gets some uniform parameters: the texture containing the scene depth values (*sceneDepth*), the density (*tau*) and the camera's front clipping plane distance (*frontPlane*). The fragment program executes the following operations:

```
float d=length(In.P.xy-In.Q.xy);
if(d>In.r) alpha=0;
else {
 float w=sqrt(In.r*In.r-d*d);
 Zs=tex2D(sceneDepth,In.screenCoord).r;
 Zf=max(frontPlane,Zp-w);
 Zb=min(Zs,zp+w);
```

```
float ds=Zb-Zf;
alpha= 1 - exp(-tau * ds);
}
```

With this simple calculation method we obtained the real ray segment length, thus we can compute the real opacity of the given particle, and eliminate clipping and popping artifacts (see figure 4).

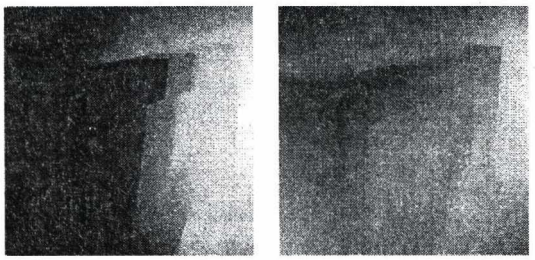


Figure 4: *Particle system with and without clipping artifacts.*

4. Rendering participating media with particle systems

A particle system is a discretization of a continuous volume, which allows us to replace the differentials of the volumetric rendering equation by finite differences. Denoting the length of the ray segment intersecting the sphere of particle j by Δs_j, and the *density*, *albedo* and *phase function* of this particle by τ_j, a_j, P_j, respectively, we obtain the following equation expressing *outgoing radiance* $L(j,\vec{\omega})$ of particle j at direction $\vec{\omega}$:

$$L(j,\vec{\omega}) = I(j,\vec{\omega}) \cdot (1-\alpha_j) + \alpha_j \cdot C_j + E_j(\vec{\omega}), \qquad (1)$$

where $I(j,\vec{\omega})$ is the *incoming radiance*, $\alpha_j = 1 - e^{-\tau_j \Delta s_j}$ is the *opacity* that expresses the decrease of radiance caused by this particle due to *extinction*, $E_j = L_j^e(\vec{\omega}) \cdot \Delta s$ is the emission of particle j in direction $\vec{\omega}$, and

$$C_j = a_j \cdot \int_{\Omega'} I(j,\vec{\omega}') \cdot P_j(\vec{\omega}',\vec{\omega})\, d\omega'$$

is the contribution from *in-scattering*. Note that the travelled path length in a given particle plays an important role in the above equations.

If we know the in-scattering term, the volume can efficiently be rendered from the camera using alpha blending. The in-scattering term is attenuated according to the total opacity of the particles that are between the camera and this particle. This requires the sorting of particles in the view direction before sending them to the frame buffer in back to front order. At a given particle, the evolving image is decreased according to the opacity of the particle and increased by its in-scattering and its emission term (equation 1).

4.1. Rendering dust and smoke

In our test scene we used two smoke-like particle systems. The first is used to display low albedo smoke in the fire, as we separated the fire into a participating and an emitting part. The second system is used to give atmosphere to the scene, representing the swirling dust in the air (figure 5).

When rendering dust and smoke we assume that the smoke particles does not emit radiance so the emission term is zero. Our main task is to calculate the in-scattering term. To do this, we need the travelled ray length, the albedo, the density and the phase function (see equation 1). We use the Henyey-Greenstein phase function [3,2]:

$$P(\vec{\omega}',\vec{\omega}) = \frac{1}{4\pi} \cdot \frac{3(1-g^2)\cdot(1+(\vec{\omega}'\cdot\vec{\omega})^2)}{2(2+g^2)\cdot(1+g^2-2g(\vec{\omega}'\cdot\vec{\omega}))^{3/2}},$$

where $g \in (-1,1)$ is a material property describing how strongly the material scatters forward or backward. To speed up rendering, function values are read from a pre-rendered 2D texture.

To be able to use fewer number of particles, we used a grey scale smoke texture on the billboards to give details in opacity changes within a billboard. To increase variety in the smoke, we took a high resolution texture, and each particle has its own area within this texture according to the particle's id. As we render an animation this texture should be animated too. It is efficient to store the animated image in a 3D texture as inter-frame interpolation and looping can automatically be done by the graphics hardware's texture sampling unit [4].

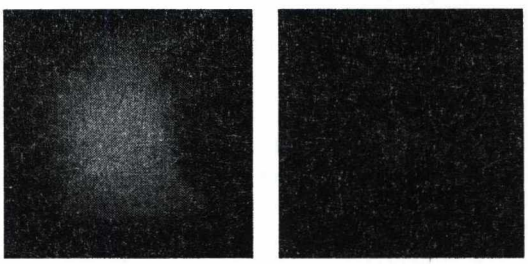


Figure 5: *High albedo dust and low albedo smoke.*

4.2. Rendering fire

We treat fire as a black-body radiator not as a participating medium, so only the emission term is needed. The main problem is to find the suitable colors for the fire. For a black-body we can compute the emitted radiance for a given wavelength using Planck's formula:

$$L_{e,\lambda}(x) = \frac{2C_1}{\lambda^5(e^{C_2/(\lambda T)}-1)}$$

where $C_1 = 3.7418 \cdot 10^{-16}\ Wm^2$, $C_2 = 1.4388 \cdot 10^{-2}\ mK^\circ$ and T is the temperature of the radiator [7]. Figure 6 shows the

spectral radiance of black-body radiators at different temperatures, the higher the temperature is the more blueish the color gets.

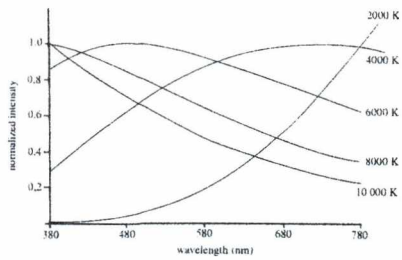


Figure 6: *Black-body radiator spectral distribution*

For different temperature values, we can compute the RGB components by integrating the spectrum. We stored the results of these values within the interval $T \in 2500K^\circ - 3200K^\circ$ in a texture (see figure 7).

Figure 7: *Black-body radiator colors from* $0K^\circ$ *to* $10000K^\circ$. *Our demo used temperature values from* $2500K^\circ$ *to* $3200K^\circ$.

Just like in the case of smoke we applied an animated grey scale image to alter the opacity values of the fire billboards. The intensity values stored in this texture also defines a mapping to the temperature values to be used, namely higher intensity values represent higher temperatures (see figure 8). We used the same method to increase variety as in the case of smoke and dust.

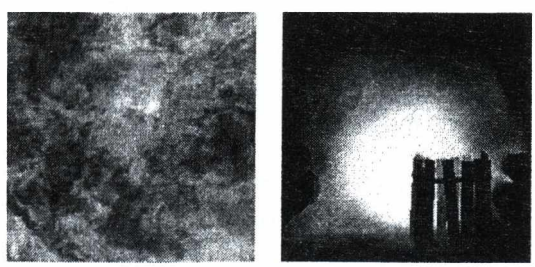


Figure 8: *Fire texture and the final fire color.*

5. Layer composition

To combine the particle systems with the scene, we used a layer composition method because it has several advantages [4]. This way we should render the scene and the systems in separate textures and compose them. This leads to three rendering passes: the first pass renders the scene objects, the second pass renders the dust, fire and smoke, and the final pass composites them together.

A great advantage of rendering the participating medium into a texture is that we can use floating point blending. Another advantage is that this render pass can have smaller resolution than the final display resolution which speeds up rendering especially because blending needs a huge amount of pixel processing power through pixel overdraw (see figure 9).

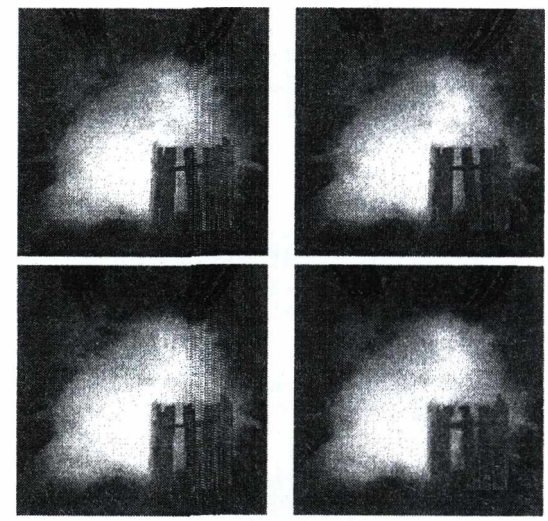


Figure 9: *Particles rendered to render targets with different sizes. Up left: particle render target with screen resolution (30 FPS). Up right, bottom left, bottom right: render target with half (40 FPS), quarter (50 FPS) and eigth (60 FPS) of the screen resolution.*

Another rendering pass should be made for producing scene depth which is needed by the particle systems, but this pass can be encapsulated within the first pass with the use of multiple render targets. This way scene geometry can be processed only once.

To enhance realism, we simulated heat shimmering that distorts the image. This is done with rendering particles with a noisy texture. This noise is used in the final composition as u, v offset values to distort the scene image (figure 10). This pass can also be encapsulated in the rendering of fire particles with multiple texture targets.

The final effect that could be used through composition is motion blur, which can easily be done with blending, letting the new frame fade into previous frames. The complete rendering process is shown in figure 11.

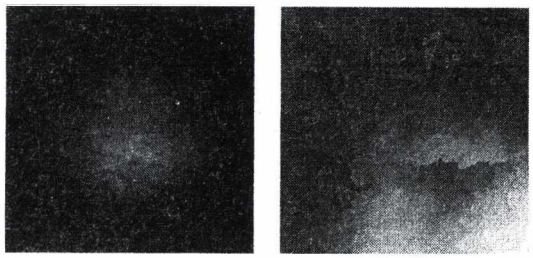


Figure 10: *Heat noise texture and the final distorted image.*

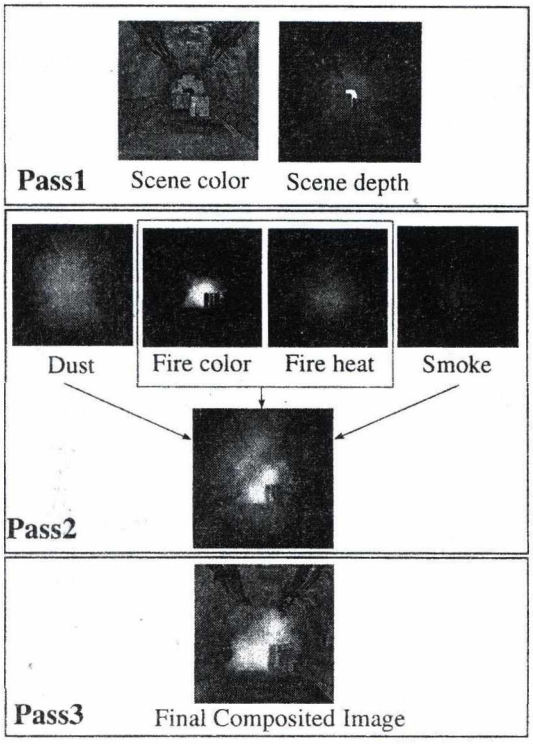


Figure 11: *Rendering algorithm*

6. Results

The presented algorithm has been implemented in OpenGL/Cg environment on an NV6800GT graphics card. The dust, fire and smoke consist of 16,115 and 28 animated particles, respectively, with relatively huge sizes. The modelled scene consists of 16800 triangles. The scene is rendered with per pixel Phong-Blinn shading. Our algorithm still offers realtime rendering speed with high details (see figure 12). The frame rate strongly depends on the number of overridden pixels. It is about 40 FPS. The scene without the particle system is rendered at 70 FPS. We should mention that the classic billboard rendering method would also run at a frame rate about 40 FPS. This means that the performance we loose by the clipping calculation can be regained by decreasing the particle system's render target resolution.

7. Conclusion

This paper proposed to consider particles as spheres rather than planar billboards during rendering while still rendering them as billboards, which eliminated billboard clipping and popping artifacts. The paper also introduced an efficient method to display fire, smoke and dust. It also used post rendering effects and still offers high frame rates, taking advantage of the GPU.

8. Acknowledgement

This work has been supported by OTKA (T042735), GameTools FP6 (IST-2-004363) project, by the Spanish-Hungarian Fund (E-26/04).

References

1. J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82 Proceedings*, pages 21–29, 1982. 1

2. W. Cornette and J. Shanks. Physical reasonable analytic expression for single-scattering phase function. *Applied Optics*, 31(16):31–52, 1992. 3

3. G. Henyey and J. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 88:70–73, 1940. 3

4. H. Nguyen. *GPUGems : Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapter 6 Fire in the "Vulcan" Demo, pages 87–105. Addison-Wesley, 2004. 3, 4

5. W. T. Reeves. Particle systems - techniques for modelling a class of fuzzy objects. In *SIGGRAPH '83 Proceedings*, pages 359–376, 1983. 1

6. G. Schaufler. Dynamically generated impostors. In *1 Workshop - Virtual Worlds - Distributed Graphics*, pages 129–136, 1995. 1

7. R. Siegel and J. R. Howell. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, D.C., 1981. 3

8. G. Szijártó. 2.5 dimensional impostors for realistic trees and forests. In Kim Pallister, editor, *Game Programming Gems 5*, pages 527–538. Charles River Media, 2005. 1

9. G. Szijártó and J. Koloszár. Hardware accelerated rendering of foliage for real-time applications. In *Spring Conference of Computer Graphics '03*, 2003. 1

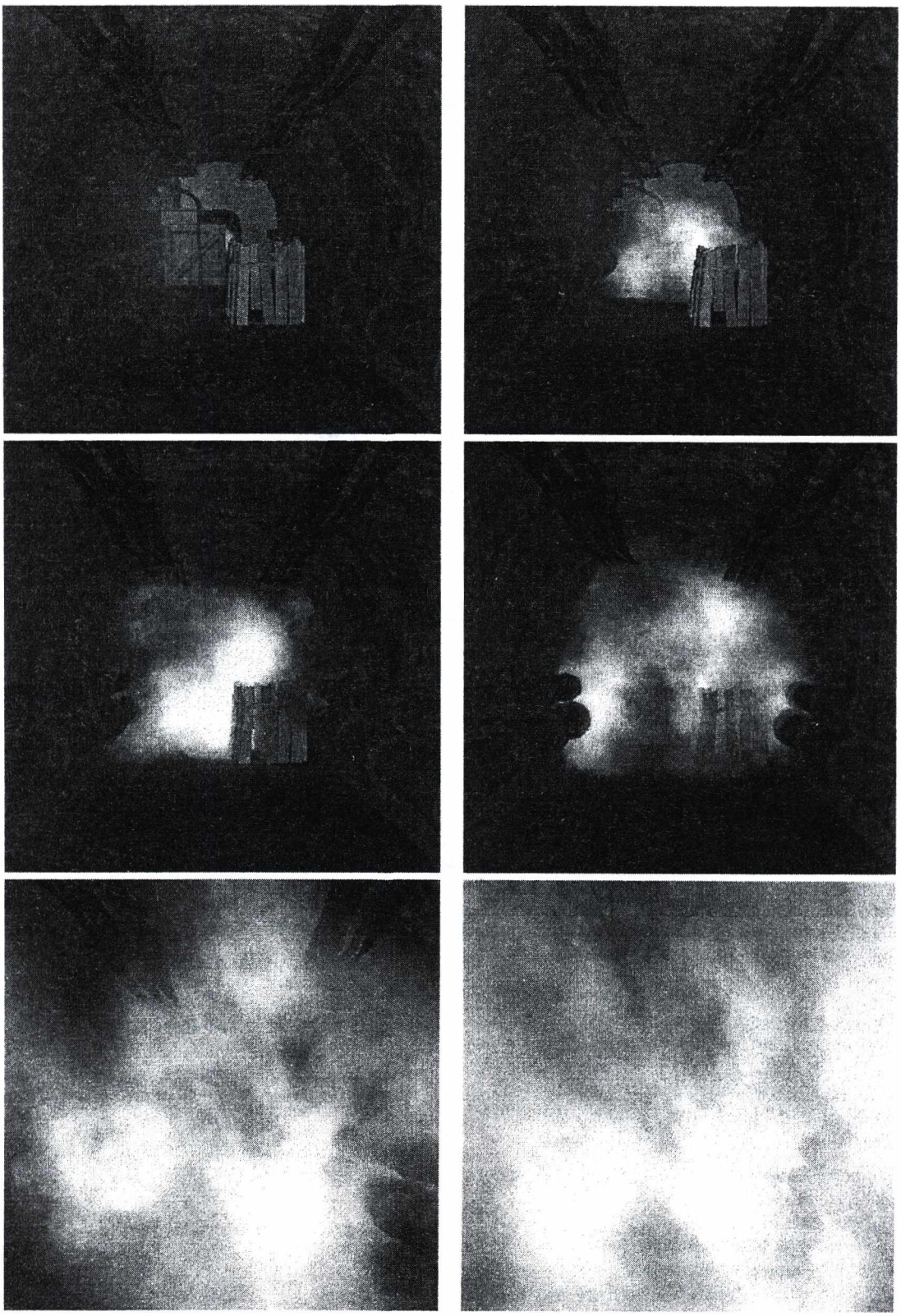


Figure 12: *Rendered frames from the animation sequence.*

An Iterative Improvement of the Tomasi-Kanade Factorization

Levente Hajder[1]

[1] Computer and Automation Institute, Hungarian Academy of Sciences
Budapest, Kende u. 13-17, H-1111 Hungary

Abstract

Reconstruction of moving rigid objects is a widely applicable and challenging computer vision task. In this paper, we give an improvement of the well-known factorization method published by Tomasi and Kanade [11]. The proposed method can deal with the reconstruction of moving nonrigid objects under orthography and weak-perspective. The proposed and the original methods are quantitatively compared on synthetic data in different simulated situations. The new method is applied for an existing outlier rejection method [4] and significantly improves its quality.

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Image Processing and Computer Vision]: Scene Analysis

1. Introduction

The Structure from Motion (SfM) problem (recovering scene geometry and camera motion from a video sequence) has attracted attention of the computer vision community since late eighties. The original factorization method of Tomasi and Kanade [11] can calculate the 3D coordinates of an object from a sequence of tracked feature points of the object. The input of the method is the 2D coordinates of the tracked feature points, while the output is the 3D coordinates of the points and the base vectors of the camera planes in all frames. In the literature, the three-dimensional data is called the structure data, the base vectors are called the motion information.

The original method calculates structural information for a single moving rigid object under orthographic projection. Recent studies [3, 12, 15] try to extend the theory to the nonrigid case; in this article, we only discuss the rigid case. There are also studies that extend the theory to the weak-perspective [14], paraperspective [9] and perspective [10] camera models.

The original factorization method consists of two main steps:

1. SVD-step: The first step is a rank reduction of the so-called measurement matrix containing the 2D data of feature points by a Singular Value Decomposition (SVD). Then the measurement matrix is factorized into pre-structure and pre-motion data.
2. Refinement-step: The pre-motion data is transformed by a matrix according to the constraints on the base vectors.

The drawback of the method is that the SVD-step reduces the space of the measurement matrix, and the refinement-step can only modify the motion and structure data in the previously reduced space. The key idea of this paper is to improve the original factorization by an iterative algorithm, which reduces an error value by fining the structure and the motion data independently.

The contribution of this paper are as follows: The original factorization method [11] and its extension [14] to the weak-perspective case are reviewed first. The proposed method is given in section 3 . The quantitative test on synthetic data is presented in section 4. We demonstrate the efficiency of the improved factorization method by applying it for an outlier filtering algorithm [4]. Finally, section 6 sums up the results.

2. SfM under orthography and weak perspective

Given P feature points of a rigid object tracked across F frames, $x_{fp} = (u_{fp}, v_{fp})^T$, $f = 1,\ldots,F$, $p = 1,\ldots,P$, the goal of SfM is to recover the structure of the object. For orthogonal projection, the 2D coordinates are calculated as

$$x_{fp} = R_f s_p + t_f, \tag{1}$$

where $R_f = [r_{f1}, r_{f2}]^T$ is the orthonormal rotation matrix, s_p the 3D coordinates of the point and t_f the offset. Under the weak perspective model, the equation is

$$x_{fp} = q_f R_f s_p + t_f, \tag{2}$$

where q_f is the nonzero scale factor of weak perspective. The offset vector is eliminated by placing the origin of 2D coordinate system at the centroid of the feature points.

For all points in the f-th image, the above equations can be rewritten as

$$\underbrace{W_f}_{2\times P} = (x_{f1} \dots x_{fP}) = \underbrace{M_f}_{2\times 3} \cdot \underbrace{S}_{3\times P} \tag{3}$$

where M_f is called the motion matrix, $S = (s_1, \dots, s_P)$ the structure matrix. Under orthography $M_f = R_f$, under weak respective $M_f = q_f R_f$.

For all frames, the equations (3) form

$$\underbrace{W}_{2F\times P} = \underbrace{M}_{2F\times 3} \cdot \underbrace{S}_{3\times P}, \tag{4}$$

where $W^T = [W_1^T, W_2^T, \dots, W_F^T]$ and $M^T = [M_1^T, M_2^T, \dots, M_F^T]$.

The task is to factorize the measurement matrix W and obtain the structural information S. This can be done in two steps. In the first step the rank of W is reduced to three by the singular value decomposition (SVD), since the rank of W is at maximum three: $W^{2F\times P} = \hat{M}^{2F\times 3} \cdot \hat{S}^{3\times P}$. This factorization is determined only up to an affine transformation because an arbitrary 3×3 non-singular matrix Q can be inserted so that $W = \hat{M}QQ^{-1}\hat{S}$. Therefore $\hat{M}$ contains the base vectors of the frames deformed by an affine transformation. The matrix Q can be determined by imposing the orthonormality constraint on the frame base vectors. The estimated motion vectors can be written as $R = \hat{M}Q$, where $R = [r_{11}, r_{12}, \dots, r_{F1}, r_{F2}]^T$.

2.1. Metric constraints under orthography

A closed-from solution for Q under orthography has been originally proposed in [8]. For orthographic projection the base vectors in each frame are orthonormal. This provides three constraints per frame:

$$\begin{aligned} r_{f1}^T r_{f1} &= \hat{m}_{f1}^T Q^T Q \hat{m}_{f1} = 1, \\ r_{f2}^T r_{f2} &= \hat{m}_{f2}^T Q^T Q \hat{m}_{f2} = 1, \\ r_{f1}^T r_{f2} &= \hat{m}_{f1}^T Q^T Q \hat{m}_{f2} = 0 \end{aligned} \tag{5}$$

Introduce symmetric matrix L as

$$L = Q^T Q = \begin{bmatrix} l_1 & l_2 & l_3 \\ l_2 & l_4 & l_5 \\ l_3 & l_5 & l_6 \end{bmatrix} \tag{6}$$

One can optimally calculate the elements of L as the least squares solution of the over-determined system

$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \\ l_5 \\ l_6 \end{bmatrix} = \begin{bmatrix} g(\hat{m}_{11}, \hat{m}_{11}) \\ g(\hat{m}_{12}, \hat{m}_{12}) \\ g(\hat{m}_{11}, \hat{m}_{12}) \\ \dots \\ g(\hat{m}_{F1}, \hat{m}_{F1}) \\ g(\hat{m}_{F2}, \hat{m}_{F2}) \\ g(\hat{m}_{F1}, \hat{m}_{F2}) \end{bmatrix}^{\dagger} \begin{bmatrix} 1 \\ 1 \\ 0 \\ \dots \\ 1 \\ 1 \\ 0 \end{bmatrix}, \tag{7}$$

or briefly $l = G^{\dagger}c$, where $G^{\dagger}$ is the pseudo-inverse of G and $g(a,b)$ is

$$[a_1b_1, a_1b_2 + a_2b_1, a_1b_3 + a_3b_1, a_2b_2, a_2b_3 + a_3b_2, a_3b_3]$$

The matrix Q is obtained by the eigen-decomposition of L.

2.2. Metric constraints under weak perspective

Under weak perspective a closed-form solution can also be obtained by least squares optimization [14]. In this case, we have only two constraints for each frame:

$$\begin{aligned} r_{f1}^T r_{f1} &= r_{f2}^T r_{f2} \\ r_{f1}^T r_{f2} &= 0 \end{aligned}$$

Equation (7) modifies as follows:

$$G_{weak} = \begin{bmatrix} g(\hat{m}_{11}, \hat{m}_{11}) - g(\hat{m}_{12}, \hat{m}_{12}) \\ g(\hat{m}_{11}, \hat{m}_{12}) \\ \dots \\ g(\hat{m}_{F1}, \hat{m}_{F1}) - g(\hat{m}_{F2}, \hat{m}_{F2}) \\ g(\hat{m}_{F1}, \hat{m}_{F2}) \end{bmatrix} \tag{8}$$

and $c_{weak} = [0, 0, \dots, 0, 0]^T$. $G_{weak}l = c_{weak}$ has an infinite number of solutions. The system has one degree of freedom. In particular, $l = 0$ is always a correct solution. An additional constraint is needed. It is well known [2] that the optimal least squares solution subject to $l^T l = 1$ is the eigenvector of matrix $G_{weak}^T G_{weak}$ corresponding to the least eigenvalue.

3. The proposed method

In this section, an updated version of the Tomasi-Kanade factorization is presented. As it is mentioned in the introduction, the weak point of the original method is the SVD step: the singular value decomposition reduces the space of the measurement matrix into three and after rank reduction, the factorization cannot leave the three-dimensional subspace.

The proposed algorithm is an iterative one. After initial values are determined, in each iteration step, the motion and the structure matrices are fined. The corresponding part of the motion matrix can be expressed as the function of three angles (α_f, β_f and γ_f) and a scaling parameter q_f in every frames as it is described in appendix A.

3.1. Initialization of the motion data

The intention of this section is to determine the initial angles α_f^0, β_f^0 and γ_f^0 and scale parameter q_f corresponding to the initial motion data represented by motion matrix $M^0 = \hat{M}Q$ obtained by the original factorization.

The motion matrix M^0 can be divided into submatrices corresponding to the frames. Let M_f^O be the motion submatrix of the f^{th} frame. It has a size of 2×3. The rows of the matrix represent the base vectors of the camera plane on the corresponding frame. The matrix can be completed with the third base vector: let the direction of the third vector be parallel to the cross product of the first two base vectors, let its length be unit (orthography) or the average of the other two vectors (weak-perspective). The completed matrix is denoted by $\tilde{M}_f^0$.

The measurement matrix is also constructed from submatrices: W_f denotes the submatrix of the measurement matrix W in the f^{th} frame as it is defined above. The matrix W_f is completed in the following way: $\tilde{W}_f = \tilde{M}_f^0 S$.

Each element of M_f^0 is a function of the rotation angles α_f^0, β_f^0 and γ_f^0 and the scale parameter q_f. In general case, M_f^0 is not an orthogonal matrix, since the factorization method described above does not guarantee the orthogonality of M_f^0. The matrix is only quasi-orthogonal.

The task is to find the closest orthonormal matrix to $\tilde{M}_0^f$. With the completed matrices $\tilde{M}_0^f$ and $\tilde{W}^f$, the problem can be transformed to an already solved data fitting problem: given two $3D$ data set (the columns of $\tilde{W}_f$ and S represent 3D points), an optimal orthogonal transformation have to be determined.

The 3D data fitting problem has already been solved. A good review about the possible solutions is given in [7]. In this paper, we use the results of [1] for determining the optimal rotation and that of [5] and [6] to determine optimal scale. Optimal scale parameter q_f of the f^{th} frame can be written as

$$q_f = \frac{\sum_{p=1}^{P} ||\tilde{w}_{fp}||^2}{\sum_{p=1}^{P} ||s_p||^2}, \tag{9}$$

where $\tilde{w}_{fp}$ is the p^{th} column of $\tilde{W}_f$. According to [13], optimal motion can be calculated as $\hat{M}_f^0 = VU^T$ if the matrix H is calculated as:

$$H = \sum_{p=1}^{P} s_p \tilde{w}_{fp}^T, \tag{10}$$

and the singular value decomposition of H is $H = U\Lambda V^T$.

3.2. The iteration

The iteration algorithm described here is to minimize the following so-called reprojection error in each step:

$$\varepsilon = ||W - M^{(k)} S^{(k)}||_2 \tag{11}$$

The algorithm consists of two main step: the S-step and the M-step. It runs until a given iteration number is reached or until the difference between the new and the previous reprojection errors is less than a given limit. The initial matrices are set to $M_f^{(0)} = \hat{M}_f^0$ and $S^{(0)} = S$.

3.2.1. S-step

Given $S^{(k-1)}$ and $M^{(k-1)}$, the goal is to determine $S^{(k)}$ optimally. Because the structure matrix has no constraint, its elements have arbitrary real values, the optimal value of $S^{(k)}$ is given by the well-known least-squares (LS) optimization:

$$S^{(k)} = {M^{(k-1)}}^{\dagger} W_f, \tag{12}$$

where ${M^{(k-1)}}^{\dagger}$ is the Moore-Penrose pseudo-inverse of $M^{(k-1)}$.

3.2.2. M-step

Estimating all $M_f^{(k)}$ from $M_f^{(k-1)}$ and $S^{(k)}$ is a more difficult task, because the pseudo-inverse is not applicable in this step: the motion submatrices $M_f^{(k)}$ must fulfill the orthonormality requirements: the first base vector represented by the first row of $M_f^{(k)}$ must be orthogonal to the second base vector represented by the second row, and the length of them must be unit (orthography) or the same (weak-perspective).

We can use nonlinear optimization methods to determine the angle parameters α_f, β_f and γ_f and the scale parameter q_f. In our environment, Levenberg-Marquardt technique is applied. For this optimization technique, the determination of the Jacobian matrix is needed. It is described in appendix B. The determination of the angles and scale parameters from motion submatrices is also described in appendix A.

Theorem 1 The algorithm described above converges.

Proof Both steps of the algorithm reduces the reprojection error and the error cannot be negative. Therefore, the algorithm converges. □

4. Tests on synthetic data

Experiments with synthetic data have been carried out to study the properties of the proposed iterative method. In this experiments, we compare the efficiency of the original and the improved factorization methods.

For all tests, an object is generated as a point clouds by a Gaussian random number generator with zero mean and standard deviation σ_{3D}. Then the object is rotated randomly.

No. of points	orig. error	new error	improvement (%)
4	18.457	16.170	12.39
6	11.934	11.059	7.33
8	39.685	39.296	0.98
10	23.675	21.628	8.65
12	9.7988	9.7540	0.46
14	15.123	15.532	-2.7
16	22.788	22.526	1.15
18	42.672	42.661	0.03
20	15.754	15.301	2.88

Table 1: *Reconstruction errors versus the number of points with 5% noise level.*

No. of points	orig. error	new error	improvement (%)
4	22.336	20.220	9.47
6	26.947	25.211	6.44
8	25.679	25.384	1.15
10	35.423	32.936	7.02
12	28.405	25.195	11.3
14	40.228	40.411	-0.45
16	37.558	37.180	1.01
18	26.991	26.072	3.4
20	22.902	21.768	4.95

Table 2: *Reconstruction errors versus the number of points with 10% noise level.*

The 3D points of the objects are projected onto the image planes by a weak-perspective projection. Finally, 2D noise is added to every 2D point. The 2D noise is generated by a zero-mean Gaussian random number generator with standard deviation σ_{2D}. The object is reconstructed by both the original and the improved factorization. Then the differences between the reconstructed objects and the original object are calculated. This difference comes from the fitting error value defined in [5,6].

This test is repeated with different noise levels: 5%,10%,20%, where the noise level is defined as the σ_{2D}/σ_{3D} quotient in percentage. The results are shown in Tables 1, 2 and 3.

No. of points	orig. error	new error	improvement (%)
4	17.756	16.979	4.39
6	37.484	26.528	29.23
8	35.816	31.930	10.85
10	39.480	40.445	-2.44
12	36.249	33.965	6.3
14	33.354	33.089	0.79
16	46.199	44.262	4.19
18	49.359	47.965	2.82
20	49.782	49.367	0.83

Table 3: *Reconstruction errors versus the number of points with 20% noise level.*

The following conclusion can be drawn: if an object consists of a few points, the improved factorization method serves significantly better results.

5. Improving outlier rejection with the proposed method

A very important component of our previously published outlier rejection method [4] is the determination of the motion matrix from four points by the original Tomasi-Kanade factorization. We exchanged that component for the proposed factorization method and compared the original and the modified outlier rejection methods.

In the first test, an object is rotated and projected to the image planes and noise are added to each 2D point. Then incorrect points with random data are inserted into the measurement matrix. Correct points are called inliers, while points of random data are called outliers. Then the outlier rejection method is run, and after it, the number of false positive errors (an inlier classified as an outlier) and false negative errors (an inlier classified as an outlier) are counted. We test the outlier rejection method with different noise levels. The measurement matrix contains 500 inliers and 250 outliers. The results of the first test are shown is Table 4.

The second test is similar to the first one, the only one difference is that the number of inliers is 250 and the number of outliers is 500. The results are presented in Table 5.

Based on the result of the tests, the following conclusion is drawn: The quality of the results are significantly improved compared to the original method.

The proposed algorithm has also been tested on video sequences. Sample frames from the widely-known synthetic sequence 'Hotel' are shown in figure 1. Outliers are plotted

Noise level (%)	Orig	New
4	2/1	0/0
5	1/1	0/0
6	0/1	0/0
7	0/2	0/0
8	0/165	0/1
9	0/96	0/84
10	0/171	0/6

Table 4: *False negative and false positive errors.*

Noise level (%)	Orig	New
4	1/0	0/0
5	1/0	0/0
6	0/0	0/0
7	0/4	0/0
9	0/49	0/5
8	0/107	0/1
10	0/200	0/13

Table 5: *False negative and false positive errors.*

with 'x', while inliers with dots. In figure 2, test on a home-made real sequence is presented.

6. Conclusion and future work

We have proposed and tested a novel iterative factorization technique for structure from motion under weak-perspective. It has been shown that the new algorithm is more efficient than the original one. We have successfully applied the reviewed factorization method for the problem of outlier rejection. In the future, we plan to extend the theory to the perspective case, and we try to develop a novel factorization method based on the key idea of this paper which can handle the problem of SfM with missing data.

References

1. K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987. 3
2. Åke Björck. *Numerical Methods for Least Squares Problems*. Siam, 1996. 2
3. M. Brand and R. Bhotika. Flexible Flow for 3D Non-rigid Tracking and Shape Recovery. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 312–322, December 2001. 1
4. Levente Hajder, Dmitry Chetverikov, and István Vajk. Robust Structure from Motion under Weak Perspective. In *2nd Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, Sept 2004. 1, 4
5. B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4:629–642, 1987. 3, 4
6. B.K.P. Horn, H.M. Hilden, and S. Negahdaripourt. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, 5(7):1127–1135, 1988. 3, 4
7. A. Lorusso, D. W. Eggert, and R. B. Fisher. A comparison of four algorithms for estimating 3-d rigid transformations. In *BMVC '95: Proceedings of the 1995 British conference on Machine vision (Vol. 1)*, pages 237–246, Surrey, UK, UK, 1995. BMVA Press. 3
8. T. Morita and T. Kanade. A Sequential Factorization Method for Recovering Shape and Motion from Image Streams. In *ARPA Image Understanding Workshops*, volume II, pages 1177–1188, November 1994. 2
9. C. J. Poelman and T. Kanade. A Paraperspective Factorization Method for Shape and Motion Recovery. *IEEE Trans. on PAMI*, 19(3):312–322, March 1997. 1
10. P. Sturm and B. Triggs. A Factorization Based Algorithm for Multi-Image Projective Structure and Motion. In *ECCV*, volume 2, pages 709–720, April 1996. 1
11. C. Tomasi and T. Kanade. Shape and Motion from Image Streams under orthography: A factorization approach. *Intl. Journal Computer Vision*, 9:137–154, November 1992. 1
12. L. Torresani, D.B. Yang, E.J. Alexander, and C. Bregler. Tracking and Modelling Nonrigid Objects with Rank Constraints. In *IEEE Conf. on Computer Vision and Patter Recognition*, 2001. 1
13. Michael W. Walker, Lejun Shao, and Richard A. Volz. Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Underst.*, 54(3):358–367, 1991. 3
14. D. Weinshall and C. Tomasi. Linear and Incremental Acquisition of Invariant Shape Models From Image Sequences. *IEEE Trans. on PAMI*, 17(5):512–517, 1995. 1, 2
15. J. Xiao, J.-X. Chai, and T. Kanade. A closed-form solution to non-rigid shape and motion recovery. In *ECCV (4)*, pages 573–587, 2004. 1

Appendix A: Elements of a 2×3 submatrix of a 3×3 orthogonal matrix.

If the 2×3 orthogonal matrix denoted by A is given, its elements can be written as the functions of the four parameters: α, β, γ and q (three rotation and a scale parameters). If α is the rotation around the x axis, β around y axis and γ around z axis, the elements of matrix A can be written as follows:

$$A = \begin{bmatrix} a_{11} & a_{22} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

where

$$\begin{aligned} a_{11} &= q\cos\alpha\cos\beta \\ a_{12} &= q(\sin\alpha\cos\gamma - \cos\alpha\sin\beta\sin\gamma) \\ a_{13} &= q(\sin\alpha\sin\gamma + \cos\alpha\sin\beta\cos\gamma) \\ a_{21} &= -q\sin\alpha\cos\beta \\ a_{22} &= q(\cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma) \\ a_{23} &= q(\cos\alpha\sin\gamma - \sin\alpha\sin\beta\cos\gamma) \end{aligned}$$

Appendix B: Jacobian matrix of the error function.

Given the error function $\varepsilon = ||W_f - M_f S||$ where S is a $3 \times P$, W_f a $2 \times P$ and M_f a 2×3 orthogonal matrix and all elements of M_f can be written as the functions of angles and scale, the Jacobian matrix of the error function with respect to the three angles and the scale is as follows:

$$J = \begin{bmatrix} \frac{\partial E_1}{\partial \alpha_f} & \frac{\partial E_1}{\partial \beta_f} & \frac{\partial E_1}{\partial \gamma_f} & \frac{\partial E_1}{\partial q_f} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E_{2F}}{\partial \alpha_f} & \frac{\partial E_{2F}}{\partial \beta_f} & \frac{\partial E_{2F}}{\partial \gamma_f} & \frac{\partial E_{2F}}{\partial q_f} \end{bmatrix}$$

where E_i is the i^{th} row of the error matrix $E = W - MS$. The elements of the Jacobian matrix can be expressed as follows: For the odd rows, the derivatives are:

$$\begin{aligned} \frac{\partial E_{2*f-1}}{\partial \alpha_f} &= \Sigma_{p=1}^{P} q_f[(-\sin\alpha_f\cos\beta_f)s_{p1} \\ &+ (\cos\alpha_f\cos\gamma_f + \sin\alpha_f\sin\beta_f\sin\gamma_f)s_{p2} \\ &+ (-\sin\alpha_f\sin\beta_f\cos\gamma_f + \cos\alpha_f\sin\gamma_f)s_{p3}] \\ \frac{\partial E_{2*f-1}}{\partial \beta_f} &= \Sigma_{p=1}^{P} q_f[(-\cos\alpha_f\sin\beta_f)s_{p1} \\ &- (\cos\alpha_f\cos\beta_f\sin\gamma_f)s_{p2} \\ &+ (\cos\alpha_f\cos\beta_f\cos\gamma_f)s_{p3}] \\ \frac{\partial E_{2*f-1}}{\partial \gamma_f} &= \Sigma_{p=1}^{P} q_f[0 \cdot s_{p1} \\ &- (\sin\alpha_f\sin\gamma_f + \cos\alpha_f\sin\beta_f\cos\gamma_f)s_{p2} \\ &+ (-\cos\alpha_f\sin\beta_f\sin\gamma_f + \sin\alpha_f\cos\gamma_f)s_{p3}] \\ \frac{\partial E_{2*f-1}}{\partial q_f} &= \Sigma_{p=1}^{P} q_f[(\cos\alpha_f\cos\beta_f)s_{p1} \\ &+ (\sin\alpha_f\cos\gamma_f - \cos\alpha_f\sin\beta_f\sin\gamma_f)s_{p2} \\ &+ (\cos\alpha_f\sin\beta_f\cos\gamma_f + \sin\alpha_f\sin\gamma_f)s_{p3}], \end{aligned}$$

while the derivatives of the even rows are expressed as

$$\begin{aligned} \frac{\partial E_{2*f}}{\partial \alpha_f} &= \Sigma_{p=1}^{P} q_f[(-\cos\alpha_f\cos\beta_f)s_{p1} \\ &+ (-\sin\alpha_f\cos\gamma_f + \cos\alpha_f\sin\beta_f\sin\gamma_f)s_{p2} \\ &- (\sin\alpha_f\sin\gamma_f + \cos\alpha_f\sin\beta_f\cos\gamma_f)s_{p3}] \\ \frac{\partial E_{2*f}}{\partial \beta_f} &= \Sigma_{p=1}^{P} q_f[(\sin\alpha_f\sin\beta_f)s_{p1} \\ &+ (\sin\alpha_f\cos\beta_f\sin\gamma_f)s_{p2} \\ &- (\sin\alpha_f\cos\beta_f\cos\gamma_f)s_{p3}] \\ \frac{\partial E_{2*f}}{\partial \gamma_f} &= \Sigma_{p=1}^{P} q_f[0 \cdot s_{p1} \\ &+ (-\cos\alpha_f\sin\gamma_f + \sin\alpha_f\sin\beta_f\cos\gamma_f)s_{p2} \\ &+ (\cos\alpha_f\cos\gamma_f + \sin\alpha_f\sin\beta_f\sin\gamma_f)s_{p3}] \\ \frac{\partial E_{2*f}}{\partial q_f} &= \Sigma_{p=1}^{P} q_f[(-\sin\alpha_f\cos\beta_f)s_{p1} \\ &+ (\cos\alpha_f\cos\gamma_f + \sin\alpha_f\sin\beta_f\sin\gamma_f)s_{p2} \\ &+ (\cos\alpha_f\sin\gamma_f - \sin\alpha_f\sin\beta_f\cos\gamma_f)s_{p3}]. \end{aligned}$$

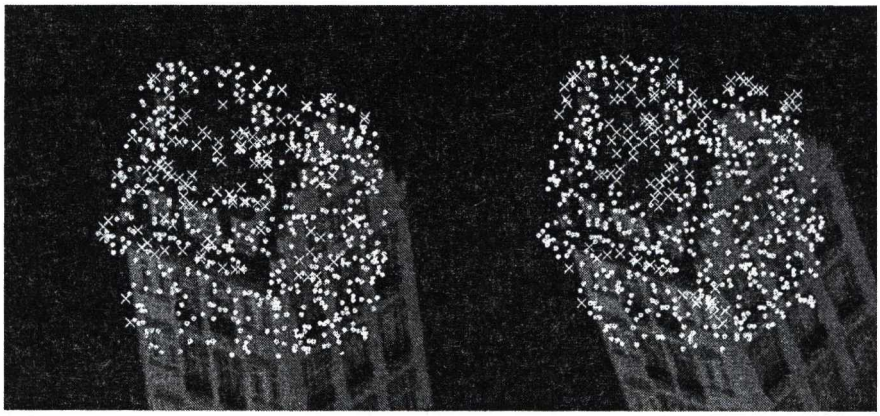


Figure 1: *Results for 'Hotel' sequence. 'x' is a detected outlier.*

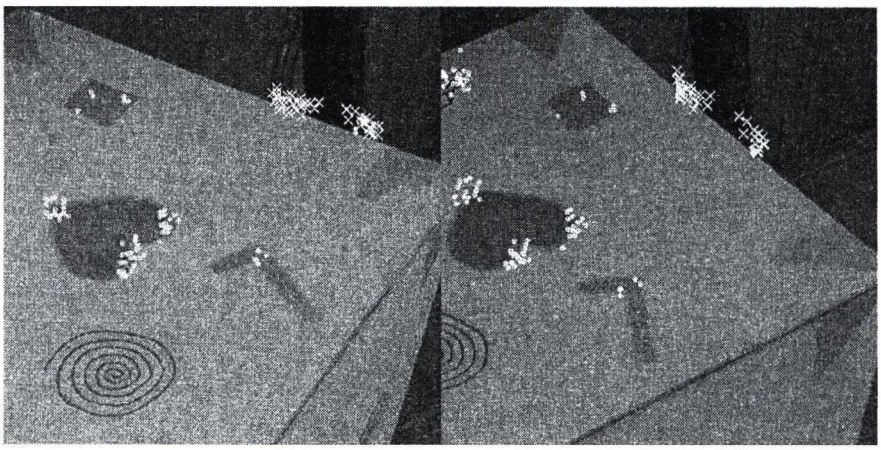


Figure 2: *Results for a real sequence. 'x' is a detected outlier.*

Building Photorealistic Models Using Data Fusion

Z. Jankó, E. Lomonosov and D. Chetverikov

Computer and Automation Research Institute, Budapest, Kende u.13-17, H-1111 Hungary
and Eötvös Loránd University, Budapest

Abstract

We are currently working on several projects related to the automatic fusion and high-level interpretation of 2D and 3D sensor data for building models of real-world objects and scenes. One of our major goals is to create rich and geometrically correct, scalable photorealistic 3D models based on multimodal data obtained by different sensors, such as camera and laser scanner. In this paper, we present a sophisticated software system that processes and fuses geometric and image data using genetic algorithms and efficient methods of computer vision.

1. Introduction

Thousands of cultural heritage objects around the world are in the danger of being lost. During the last years a number of ambitious projects have been started to preserve these objects by digitalising them. Examples of such projects are the Michelangelo Project [9], the Pieta Project [2] and the Great Buddha Project [3].

There exist different techniques to reconstruct the object surface and to build photorealistic 3D models. Although the geometry can be measured by various methods of computer vision, for precise measurements laser scanners are usually used. However, most of laser scanners do not provide texture and colour information, or if they do, the data is not accurate enough.

Our photorealistic modelling system receives as input two datasets of diverse origin: a number of partial measurements (3D point sets) of the object surface made by a hand-held laser scanner, and a collection of high quality images of the object acquired independently by a digital camera. The partial surface measurements overlap and cover the entire surface of the object; however, their relative orientations are unknown since they are obtained in different, unregistered coordinate systems. A specially designed genetic algorithm (GA) automatically pre-aligns the surfaces and estimates their overlap. Then a precise and robust iterative algorithm (Trimmed Iterative Closest Point, TrICP [1]) developed in our lab is applied to the roughly aligned surfaces to obtain a precise registration. Finally, a complete geometric model is created by triangulating the integrated point set.

The geometric model is precise, but it lacks texture and colour information. The latter is provided by the other dataset, the collection of digital images. The task of precise fusion of the geometric and the visual data is not trivial, since the pictures are taken freely from different viewpoints and with varying zoom. The data fusion problem is formulated as photo-consistency optimisation, which amounts to minimising a cost function with numerous variables which are the internal and the external parameters of the camera. Another dedicated genetic algorithm is used to minimise this cost function.

When the image-to-surface registration problem is solved, we still face the problem of seamless blending of multiple textures, that is, images of a surface patch appearing in different views. This problem is solved by a surface flattening algorithm that gives a 2D parameterization of the model. Using a measure of visibility as weight, we blend the textures providing a seamless and detail-preserving solution.

All major components of the described system are original, developed in our laboratory. Below, we briefly present the main algorithms and give examples of photorealistic model building using GA-based registration and fusion of spatial and pictorial data.

2. Pre-registration of surfaces using a genetic algorithm

This section deals with genetic pre-alignment of two arbitrarily oriented datasets, which are partial surface measurements of the object whose model we wish to build. The task is to quickly obtain a rough pre-alignment suitable for subsequent application of the robust Trimmed Iterative Closest

Point algorithm [1] developed in our lab earlier. Our experience with TrICP shows that, depending on the data processed, the algorithm can cope with initial angular misalignments up to 20°; 5° is certainly sufficient. This means that the genetic pre-registration should provide an angular accuracy of 5°, or better.

Our genetic pre-registration procedure minimises the same objective function, $\Psi(\xi, \mathbf{R}, \mathbf{t})$, as TrICP, but this time as a function of 7 parameters, namely, the overlap ξ, the three components of the translation vector $\mathbf{t}$, and the three Euler angles of the rotation matrix $\mathbf{R}$. The difference between the genetic solution and the overlap optimisation procedure [1] is essential. The former means evaluating $\Psi(\xi, \mathbf{R}, \mathbf{t})$ for different values of ξ, $\mathbf{R}$, and $\mathbf{t}$, while the latter means running TrICP for different values of ξ.

To minimise the objective function $\Psi(\xi, \mathbf{R}, \mathbf{t})$, we applied a genetic algorithm tuned to the problem. To evaluate the objective function, each integer parameter was mapped by normalisation onto a real-valued range. Simple one-point crossover was employed. Different population sizes were tested and an optimal value was selected for the final experiments. Two mutation operators were introduced. Shift mutation shifts one parameter randomly by a value not exceeding 10% of the parameter range, while replacement mutation replaces a parameter with a random value. The corresponding probabilities were also set after preliminary experimentation. Tournament selection was applied, as it is easy to implement and helps avoid premature convergence. An elitist genetic algorithm was employed, where one copy of the best individual was transferred without change from each generation to the next one.

The method is presented in detail in our paper [8]. The main steps of the genetic algorithm are as follows:

1. Generate initial population.
2. Calculate objective function values.
3. Apply genetic operators (crossover, mutation) to selected individuals. Next generation will contain the offspring and, additionally, the best fit individual from current generation.
4. Calculate objective function values for the new population.
5. If best fitness has not changed for N_g generations, stop and designate the best fit individual as solution, otherwise go to step 3.
6. If the solution obtained at step 5 is not of acceptable precision, restart from step 1.

We have tested the genetic pre-alignment and the combined method (GA followed by TrICP) on different data. Two examples, the Angel and the Bird data, are shown in figures 1 and 2, respectively. To test the method under arbitrary initial orientations, set $\mathcal{P}$ was randomly rotated prior to alignment in each of the 100 tests. Results of all tests were visually checked. No erroneous registration was observed. Typical examples of alignment are shown in figures 1 and 2. In each figure, the first two pictures show the two datasets to be registered. The datasets result from two separate measurements of the same object obtained from different angles.

The third picture of each figure (GA) displays the result of our genetic pre-registration algorithm. Here, the two datasets are shown in different colours. One can see that the datasets are roughly registered, but the registration quality is not high: the surfaces are displaced, and they occlude each other in large continuous areas instead of 'interweaving'. Finally, the rightmost picture is the result of the fine registration obtained by TrICP using the result of the genetic pre-registration. Here, the surfaces match much better, and they are interwoven, which is an indication of the good quality of the final registration.

3. Fusion of surface and image data

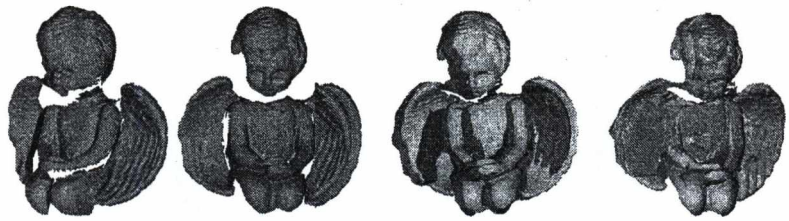
In this section, we address the problem of combining geometric and textural information of the object. As already mentioned, the two sources are independent in our system: the 3D geometric model is obtained by 3D scanner, then covered by high quality optical images. First, we discuss our photo-consistency based registration method with genetic algorithm based optimisation. Then we deal with the task of blending multiple texture mappings and present a novel method which combines the techniques of surface flattening and texture merging. Finally, test results on synthetic and real data are shown.

3.1. Registering images to a surface model

Based on our previous paper [5], let us discuss the registration of images to a 3D model. The input data consists of two colour images, I_1 and I_2, and a 3D surface model. They represent the same object. (See figure 3 for an example.) The images are acquired under fixed lighting conditions and with the same camera sensitivity. All other camera parameters may differ and are unknown. The raw data is acquired by a hand-held 3D scanner, then processed by the efficient and robust triangulator [6] developed in our lab. The 3D model obtained consists of a triangulated 3D point set (mesh) $\mathcal{P}$ with normal vectors assigned.

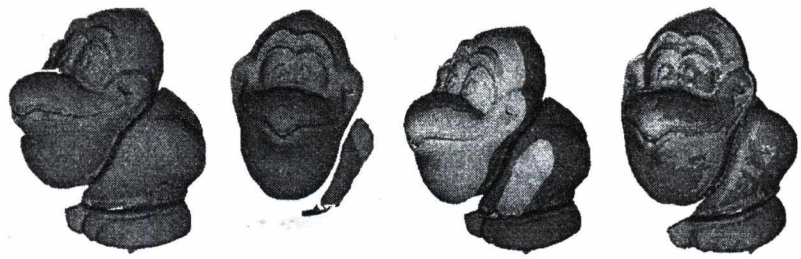
The finite projective camera model is used to project the object surface to the image plane: $\mathbf{u} \simeq P\mathbf{X}$, where $\mathbf{u}$ is an image point, P the 3×4 projection matrix and $\mathbf{X}$ a surface point. ($\simeq$ means that the projection is defined up to an unknown scale.)

The task of registration is to determine the precise projection matrices, P_1 and P_2, for both images. Since the projection matrix is up to a scale factor, it has only 11 degrees of freedom in spite of having 12 elements. The collection of the 11 unknown parameters is denoted by p, which represents the projection matrix P as an 11-dimensional parameter vector.



Angel data GA GA+TrICP

Figure 1: *The Angel dataset, GA alignment and final alignment.*



Bird data GA GA+TrICP

Figure 2: *The Bird dataset, GA alignment and final alignment.*

Values of the two parameter vectors p_1 and p_2 are sought such that the images are consistent in the sense that the corresponding points – different projections of the same 3D point – have the same colour value. (It is assumed that the surface is Lambertian.) This type of consistency is called **photo-consistency** [4]. A robustified photo-consistency based cost function, $C_\phi(p_1, p_2)$, was introduced in our paper [5]. The minimum of $C_\phi(p_1, p_2)$ gives a good estimation for the projection matrices.

Although the cost function is simple, it has unpredictable shape in the multidimensional parameter space. The standard local and global nonlinear minimisation techniques we tested failed to provide reliable results. Finally, we decided to apply a genetic algorithm, as a time-honoured global search strategy.

We pre-register the images and the 3D model manually. This yields a good initial state for the search, which narrows the search domain and accelerates the method. Manual pre-registration is reasonable since this operation is simple and fast compared to the 3D scanning, which is also done manually. The photo-consistency based registration makes the result more accurate.

The genetic algorithm starts by creating the initial population. The individuals of the population are chosen from the neighbourhood of the parameter vector obtained by the manual pre-registration. The values of the genes are from the intervals defined by the pre-registered values plus a margin of $\pm\varepsilon$. In our experiments ε was set to values between 1% and 3%, depending on the meaning and the importance of the corresponding parameter. The individual that encodes the pre-registered parameter vector is also inserted in the initial population to avoid losing it.

We have tested the method with a number of different genetic settings. With proper settings, the projection error of registration can be decreased from 18–20 pixels (the average error of the manual pre-registration) to 5–6 pixels. After preliminary testing with semi-synthetic data, the following genetic setting has been selected: Steady state algorithm with Tournament selector, Swap mutator and Arithmetic crossover, with 250 individuals in the population, with mutation probability of 0.1 and crossover probability of 0.7. The typical running time with a 3D model containing 1000 points was 5–6 minutes on a 2.40 GHz PC with 1 GB memory.

We applied the method to different real data. One of them, the Bear Dataset, is shown in figure 3. The precision of the registration can be best judged at the mouth, the eyes, the hand and the feet of the Bear. Figure 4 visualises the difference between the manual pre-registration and the photo-consistency based registration. The areas of the mouth, the eyes and the ears show the improvement of the quality.

3.2. Merging multiple textures

After registering the images to the 3D model, they can be mapped to the surface. Usually one image can show only one part of the model, but a number of images of the same object taken from different viewpoints can cover the whole.

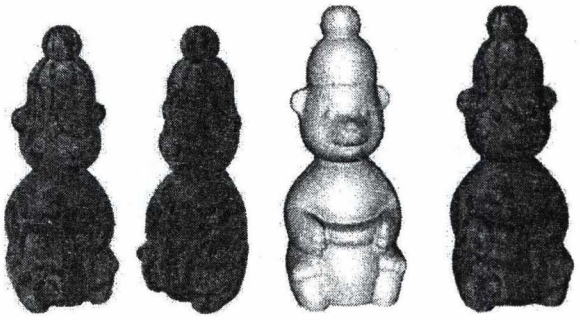


Figure 3: *The Bear Dataset and result of registration of images to surface.*

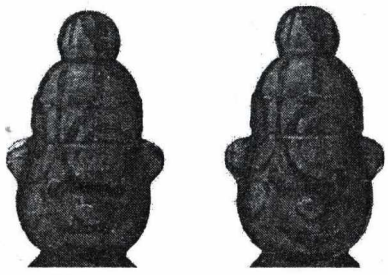


Figure 4: *Difference between manual pre-registration and genetic registration.*

Below, we discuss the problem of combining partial overlapping textures and present a novel method for it.

To paste texture to the surface of an object, we need two pieces of information: a *texture map* and *texture coordinates*. The former is the image we paste, while the latter specify where it is mapped to. Texture coordinates can be determined by a texture mapping function, for instance, by applying projection matrix P to 3D point $\mathbf{X}$.

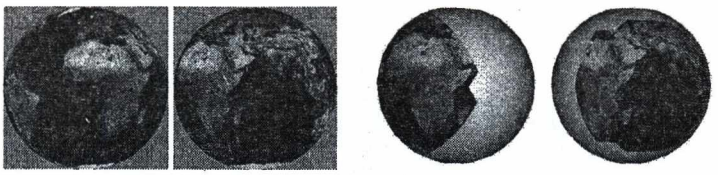
Figure 5a shows two images of the globe, which can be considered as texture maps. Merging the two texture maps to one is not obvious. Creating an image by appending the second image to the first one and modifying the second projection matrix with a translation yields gap between the border of the textures.

There exists an other way to create a texture map based on the images. Flattening the surface of the object yields also a two-dimensional parameterization. The advantage of this parameterization is that it preserves the topology of the three-dimensional mesh. A texture that covers entirely the flattened 2D surface covers also the original 3D surface. Converting optical images to flattened surfaces yields partially textured meshes, but since flattening preserves the structure of the 3D mesh, these texture maps can be merged, in contrast to the optical images.

We use the algorithm [7] developed in our lab to flatten and parameterize triangular meshes. After this one needs to convert the optical images to flattened texture maps. Since the transformation of flattening cannot be represented by a matrix, we have to use the mesh representation for conversion. Given a triangle of the mesh, consider the known corresponding triangles in the optical image and on the flattened surface, respectively. The affine transformation between them can be easily determined. This transformation gives the correspondence between the points of the triangles. Note that the affine transformation is unique for each triangle pair. The process of converting optical images to flattened texture maps is illustrated in figure 6.

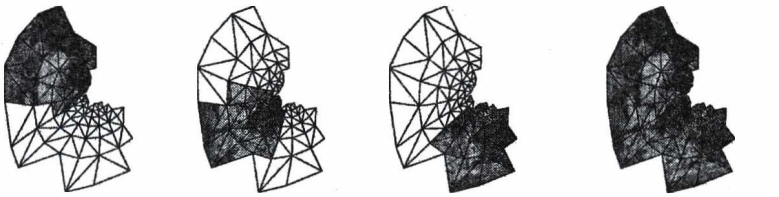
Merging partial texture maps may cause a problem in the overlapping areas. To eliminate the seams appearing at the borders of the texture maps, we blend the views as follows. For each triangle all the views are collected which the given triangle is entirely visible from. A measure of visibility of a 3D point is the scalar product of the normal vector and the unit vector pointing towards the camera. This measure is used to set a weight for each view: If the point is better visible from the view, the weight is greater. To set the colour of a point, all of these views with their weights are combined.

The method has been tested both on synthetic and real data. The Earth Dataset consists of 8 images of the globe



a. Input Images b. Partially Textured Model

Figure 5: *Textures cover only parts of the model.*



Partial textures Merged texture

Figure 6: *Partial and merged texture maps.*

and a synthetic 3D model. Figure 7 shows two of the input images, the merged texture map and a snapshot of the textured 3D model. Applying the method to real data is illustrated by figure 8. The projection matrices of the images of the Bear Dataset were obtained by our photo-consistency based registration method described earlier.

4. Conclusion

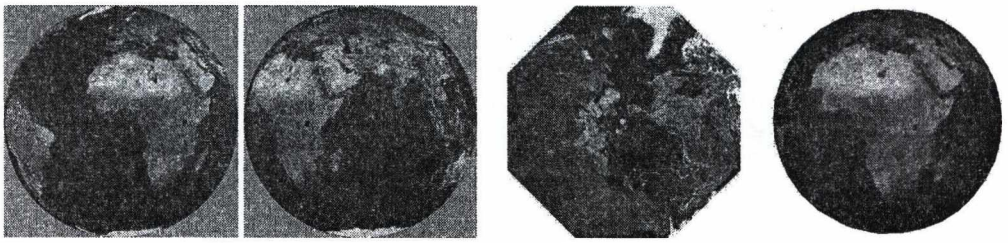
We have presented a software system for building photorealistic 3D models. It operates with accurate 3D model measured by laser scanner and high quality images of the object acquired separately by a digital camera. The complete 3D model is obtained from partial surface measurements using a genetic based pre-registration algorithm followed by a precise iterative registration procedure. The images are registered to the 3D model by minimising a photo-consistency based cost function using a genetic algorithm. Since textures extracted from images can only cover parts of the 3D model, they should be merged to a complete texture map. A novel method is used to combine partial texture mappings using surface flattening. Test results with synthetic and real data demonstrate the efficiency of the proposed methods. Our next step will be adding the surface roughness by measuring the bumps maps using a photometric stereo approach.

Acknowledgements

This work is supported by EU Network of Excellence MUSCLE (FP6-507752).

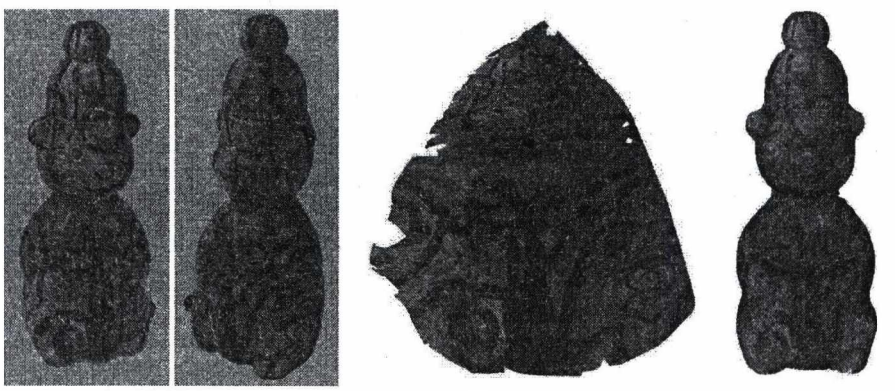
References

1. 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. 1, 2
2. F. Bernardini et al. Building a digital model of Michelangelo's Florentine Pietà. *IEEE Comp. Graphics & Applications*, 22(1):59–67, 2002. 1
3. K. Ikeuchi et al. The great Buddha project: Modeling cultural heritage for VR systems through observation. In *Proc. IEEE ISMAR03*, 2003. 1
4. M.J. Clarkson et al. Using photo-consistency to register 2D optical images of the human face to a 3D surface model. *IEEE Tr. on PAMI*, 23:1266–1280, 2001. 3
5. Z. Jankó and D. Chetverikov. Photo-consistency based registration of an uncalibrated image pair to a 3D surface model using genetic algorithm. In *Proc. 2nd Int. Symp. on 3D Data Processing, Visualization & Transmission*, pages 616–622, 2004. 2, 3
6. G. Kós. An algorithm to triangulate surfaces in 3D using unorganised point clouds. *Computing Suppl.*, 14:219–232, 2001. 2
7. G. Kós and T. Várady. Parameterizing complex triangular meshes. In *Proc. 5th International Conf. on Curves and Surfaces*, pages 265–274, 2003. 4
8. E. Lomonosov, D. Chetverikov, and A. Ekárt. Fully automatic, robust and precise alignment of measured 3D surfaces for arbitrary orientations. In *28th Workshop of the Austrian Association for Pattern Recognition*, pages 39–46, 2004. 2
9. M. Levoy et al. The digital Michelangelo project. *ACM Computer Graphics Proceedings*, pages 131–144, 2000. 1



Images Texture Map Textured Surface

Figure 7: *Result of texture map merging for the Earth data.*



Images Texture Map Textured Surface

Figure 8: *Result of texture map merging for the Bear data.*

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Road Traffic Monitoring by Video Processing

Szabolcs Czuczor

Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary

Abstract

The number of road vehicles and therefore the intensity of the road traffic is increasing day by day. Drivers need more accurate and real-time information about the traffic and the possibility of jam. The internet and the mobile technology make almost any information accessible inside the car. If there was an interactive city map with information about the number of moving vehicles on the road and their speed, drivers always would be able to calculate the most ideal route to reach their target and avoid the traffic jam. All around our capital city there are video cameras that provide real-time visual information about sections of roads, important crossings and road junctions. These video signals are monitored by the police and the public transport companies, but at the moment, because of technical reasons and privacy issues, there is no opportunity to give this information to the hands of the public. This paper presents an adaptive video processing application, which is currently under development, and whose task is to recognize the individual road vehicles, count and classify them and determine their speed independently on the weather and lighting conditions. Visualizing these numbers on a city map, we can easily avoid the above mentioned problems and can help drivers on the road interactively.

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Scene Analysis]: Motion, Tracking

1. Introduction

Today image sequence based road traffic monitoring is an actual research field of image processing. To have control on the roads and be able to provide enough information about the intensity of the road traffic all over the city, we need many viewpoints in different places, on different road junctions. Each one of these video cameras watches different areas of the city where also the position, orientation and perhaps other parameters of the cameras are different, so do the lighting and weather conditions. These criteria demand a very flexible and adaptive monitoring system that needs only a few tuning parameters as setup or pre-adjustment. The outdoor being of these cameras are also important when we want to design the dataflow between the camera and the video processing system. Could we deploy an outdoor computer near the camera, or it is too expensive and do we need to take it into an indoor station and transport the high bandwidth video stream through the city, which is also an expensive solution?

Our task was to implement an image processing application that monitors the downtown road traffic unaffectedly by lighting and visible weather conditions, can be run real-time on a relatively modern PC, and measures the average speed and the intensity of the traffic. In other words, its output is the number, the category and the average speed of the moving vehicles. The aerial method described in [1] is very effective in case of highway shots taken from the air. The speed and orientation of vehicles are nearly constant, and because of the big area that can be watched the 1.2 seconds processing time between image samples is also satisfying. In case of downtown traffic the observable area is much smaller, where the vehicles follow each other rapidly, thus the frame rate must be higher. In [2,4,5] we can meet powerful vehicle and object recognition and tracking systems, but one of their biggest problem (which is ours too) the occlusion of the objects. The problem of occlusion is also mentioned in [3], but we cannot exploit the behavioral and motion differences of objects as we could in case of an occluding human and a car. In [6] the problem of occlusion is solved by modeling the major color regions (blobs) of the body. In our case this method is replaced by feature (object or texture) tracking which is also used in [7] because of the unpredictable variety of color and shape of vehicles.

The structure of this paper is the following: in Section 2 we present the preprocessing and already operational steps of our algorithm. In Section 3 the functionalities and features

will shown, which are still under development. While Section 4 talks about our tasks for the next few months, which are to make the algorithm more flexible, robust and faster, Section 5 presents our actual results. At the end of our paper, in Section 6 we have a little discussion about the opportunities and the applications of this algorithm.

2. Algorithm steps

The most important input of the algorithm is a real-time, PAL system[†] digitized video stream. To avoid the artifacts caused by interlaced being of the signal and the 4:2:0 color coding, we use just CIF[‡] resolution. We can do that, because there is no need for so high resolution and besides this, working on lower resolution, we win a lot of process time. Currently the test sequences are loaded from CIF size MJPEG coded AVI files, which were pre-recorded from different cameras, in different viewpoints of Budapest. The other inputs of the algorithm are additional parameters, whose value depend on the actually processed scene and are set by the observer.

In the following subsections we describe each step of the algorithm, which is needed to provide traffic-related information from the source video.

2.1. Background generation

There are many different foreground-background separation techniques used in object recognition and object tracking. Of course, simple frame differencing and background subtraction techniques give usually low performance. In [8] a novel and very efficient approach of foreground-background-shadow separation is proposed, but it is very time consuming. 1–2 seconds or more time cannot be allowed for a real-time application, except when we record image sequences in specified periods and evaluate them off-line. This could cause 10–20 minutes delay in the flow of the traffic information, which is unacceptable.

Our frame differencing method might give sometimes hardly processable data, but it is very fast. One or two "dropped" frames caused by noise are still acceptable, which will be proven in Section 3.1. The video signals are provided by stable cameras. It means that subtracting two neighboring frames results a difference image, which shows only the elementary motion of moving objects and the sudden light changes. The width of these difference patches is nearly proportional to the time between the two frames. Because of the non-uniform surface of vehicles (the different parts of them: floor, engine hood, windscreen, dazzle lamps, wheels, tires, roof rack, door etc.) these patches have a lot of gaps. If we increase the time between the two neighboring frames used for differencing, these patches will be broaden. Overlapping these difference images, generating lists of them, choosing proper list length, then overlapping and masking these lists with each other, we can get nearly solid shapes of the moving objects that can be called *motion mask* (see Figure 1).

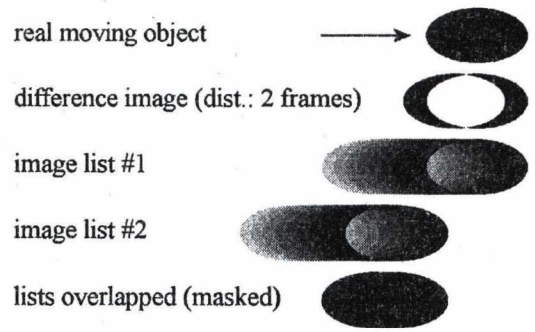


Figure 1: *Generating motion mask: (1) differencing frames from 2 frame-time distance, (2) overlapping difference images and generating image list, (3) another image list is made with a list-length delay, (4) masking the lists with each other the result is the motion mask*

If we work on gray-scale images and use well chosen differencing threshold and gamma correction, the resulting image can be a binary (black-and-white) image. Now it is sure that the motion mask covers moving objects, and the remaining pixels covers just static background (See Figure 2).

Gathering the static background pixels from the original frames, we can build an empty background image that does not contain any moving objects. Subtracting this static background from the actual frame, we can get only the moving objects of the scene as a difference (see Figure 3 top right).

One big problem with this background generation approach is that we are outdoor, and as the time goes by the lighting conditions are changing (the sunlight is changing, the clouds are moving etc.). We need to refresh the background with a well chosen latency (or inertia) to follow the slow fluctuation of lights, but avoid the effects of sudden changes (e.g. lens flare, sunlight twinkle on the windscreen of the cars etc.). That is why we have to create a FIFO[§]-type image buffer with a predefined length (in our case it's length is 12 seconds i.e. 300 frames) to make the background up-to-date.

Working with this static, but yet always refreshing background, we still cannot be happy. The cameras are self-calibrating devices that always try to balance the histogram

[†] PAL system: interlaced, 720 × 576 or 704 × 576 pixel resolution frames, 25 FPS (50 field per second) frame-rate, YUV 4:2:0 color coded video signal

[‡] CIF: Common Intermediate Format, quarter PAL (progressive, 352 × 288 pixel resolution frames, 25 FPS frame-rate)

[§] FIFO: First In First Out

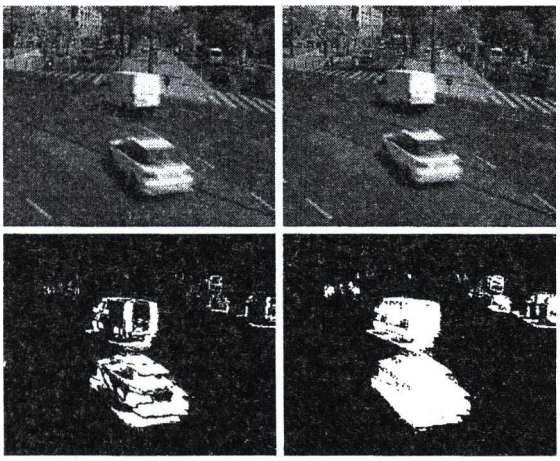


Figure 2: *Top left and right: two frames from the original color image sequence with 2 frame time (80 ms) between them; bottom left: their gamma corrected difference; bottom right: motion mask*

of the image to avoid the overflow of the pixel intensities. So when the camera looks at the asphalt in a sunny day and suddenly a big white van comes into the screen, the whole image goes into dark to compensate the huge amount of high intensity pixels. In this case the real-time generated static background should be darkened too.

2.2. Object detection

The subtraction of the real-time generated, lightness compensated static background and the actual frame results a difference image, which contains only the moving objects of the scene. After well chosen gamma correction and threshold, a noise filtering pass can be applied to keep only the useful information. Median filter (for instance with 3×3 kernel) is able to remove pixel size salt and pepper noise and can melt together areas with small gaps and holes between or inside them. This can remove the noise caused by birds or leafs of wind blown trees, and can correct the split silhouette of vehicles caused by their small background colored parts.

After noise filtering, a one pixel wide white-area-spread effect (also known as *dilate*) can be applied to broaden the visible parts of moving objects. As a result of this effect the size of patches is not increased especially, but considering visibility aspects, it can be useful.

This time we have several independent silhouettes of moving objects that can be big or small vehicles and of course pedestrians. In a lucky case the camera does not see any pavements or other places where pedestrians can be found, hence each silhouette belongs to a vehicle. In a lesser lucky case we can measure the area of the silhouettes, and knowing the distance of the given object from the camera, we can decide if it is a pedestrian or a vehicle. See Section 3.2 for a method that help pedestrian-vehicle separation. Now we can give IDs to the silhouettes (for example by painting them with different colors, see Figure 3) and then they can be tracked frame by frame.

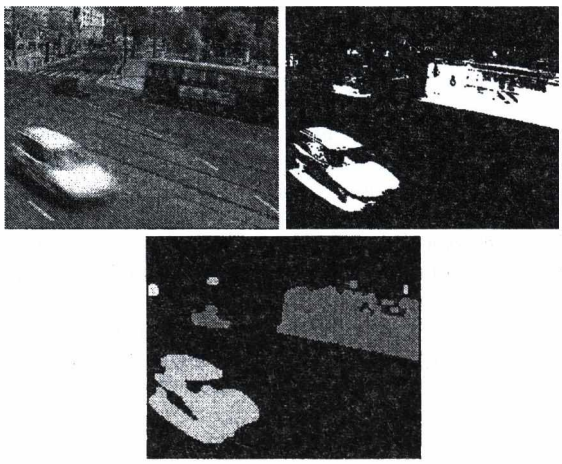


Figure 3: *Original frame, the difference image with the detected moving objects and their painted silhouettes*

Unfortunately, some of the objects are often too close or occlude each other. In this case their silhouettes are melted together and if we paint them by flood-fill method, those two (or more) vehicles will be recognized as one. To avoid this, we have to apply some pattern tracking method on the original image considering on the silhouettes on the difference image. Also the area of the painted silhouettes can hold useful information to detect attaching individuals or detaching previously occluding objects. More about this in Section 3. Examining the image sequence, it is very important that the object IDs (or silhouette colors) remain frame by frame consequently. We mustn't repaint each object in each frame but we have to take its past into attention. This task can be very difficult if we meet the problem of occlusion, because in that situation two or more objects have the same ID (or color) in the past.

3. Under development

In this Section those subsystems and tasks are presented that are under development at the moment. The deadline of the project is the end of year 2005, so after writing this article, many improvements are expected.

3.1. Managing objects' lifetime

Suppose that we have detected all of the possible moving objects in the scene well. It is time to count them and measure their speed. Well managed ID allocation makes us able

to define the lifetime of the different objects. When an object comes into the screen and get a new ID, we could say that it is born. Some frames later it goes out of the screen, it disappears, we could say that it dies. Between these two moments we track it, and because we know some 3D information of the scene (refer to Section 3.2), we can calculate its speed. The time is already known, it can be calculated easily, because in PAL system each frame follows the other in 40 ms.

What about the "dropped" frames mentioned in Section 2.1? The lifetime manager should take care of the continuity of appearance and movement of objects. It should be insensible to sudden appearance and/or disappearance caused by dropped frames and noise and, of course, to occlusion caused by other objects. The presence of an object can be modeled by measuring the area of its silhouette (counting the pixels in it). Figure 4 shows the lifetime of an object as the area of its silhouette in time. It should change continuously, and if a dirac-like change happens, it can be left out of consideration by a low-pass filter applied on it.

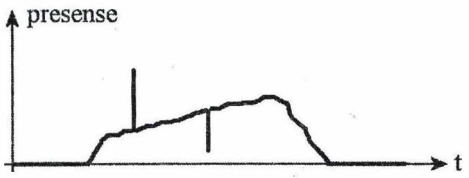


Figure 4: *The lifetime (or presence) of a moving object with two sudden change that should be by-passed*

After following the lifetime of the moving objects, we will be able to count the number of them in a given time unit, which will represent the first output of the algorithm. The second output is the average speed that can be calculated by averaging the speed of all moving objects according to their lifetime and tracking information. More about object tracking can be found in Section 3.3.

3.2. 3D description of the scene

Usually the locations watched by the cameras are flat surfaces. Aligning a simple grid to it (and, of course, taking the lens distortion [10] into account) helps us to determine the relative size and distance of the passing objects from the viewpoint. If we tell the system the size of the grid, then we can approximate the absolute size of the objects (or the area of their surface in the screen). See Figure 5.

Knowing the size of the objects, we are able to decide if an object is big enough to accept it as a vehicle or we should ignore it, because it is just a pedestrian or a smaller animal (dog, cat, bird etc.). Unfortunately this approach has several drawbacks and difficulties: for example a group of pedestrians can have relatively big surface, therefore it can be recognized as a vehicle. Sometimes a bike or a bicycle rider

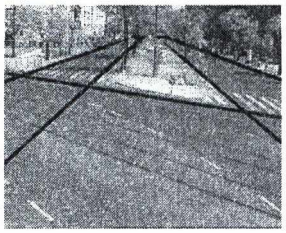


Figure 5: *Aligning a grid to the image to get 3D information from the scene*

passes in front of the camera, which is too small to accept as a vehicle. The worst two cases are when these smaller objects have almost exactly the same size as the threshold value of the size evaluator subsystem, or when these objects occlude one or more vehicles and results false decision about the kind of the object. In the first case the result is a lot of sudden appearance and disappearance of an object, which can also be perturbed by noise and causes jumps and breaks in the lifetime curve of an object (remember Figure 4). In the second case the simple flood-fill method used in object detection and object tracing (mentioned in Section 2.2) also melts more objects into one huge object. In this situation our only hope is the pattern tracking, which is described in the following point (Section 3.3).

3.3. Pattern tracking to avoid occlusion problems

Pattern tracking is a common method for motion detection by finding similarities between frames of a video source or an image sequence and bind them by motion vectors. This method compares the pixel values of a small region from the "source" frame with the corresponding pixels in its neighborhood from the "destination" frame. Usually the result is evaluated by calculating the minimal mismatch error as the summed square of difference of each corresponding pixel. This task is very time consuming depending on the size of the searched pattern and the size of its neighborhood.

It is a good question that how large pattern should we choose, what shape should it have, and how much neighborhood do we need to scan to find similarity. In the most simple case the moving vehicle has an easy to delimit silhouette that can be used as a binary mask to define those pixels on the "source" frame that we compare with the neighborhood on the "destination" frame when we look for similarities. This mask and its masked texture are stored in the memory as a small image defined by the axis-aligned bounding box of the silhouette (see Figure 6). The mask and the stored texture should be updated frame by frame to have always actual image of the vehicle.

In case of occlusion we cannot follow the pattern of the full vehicle because it is behind another vehicle (see Figure 7). Of course, we can predict its position even if we

Figure 6: *Detected object, its silhouette surrounded with its bounding box and the masked pattern*

cannot see it using its motion vectors, but what if its speed changes? The problem could be solved by the method proposed in [7] if the occlusion is just partial and we are able to follow the vehicle behind. It can also happen when two or more occluding vehicles pass by in front of the camera with the same relative speed. In this case we can refer to this group as one vehicle, however, it is not.

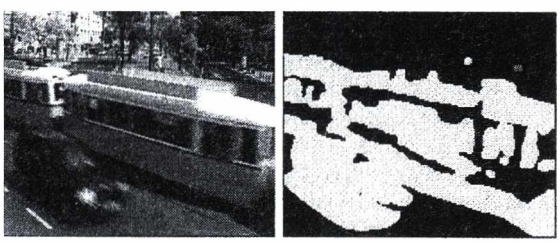


Figure 7: *The problem of occlusion: fused silhouettes of a car and a tram*

Another big problem is when the viewing axis of the camera is near along the observed street and the direction of the moving vehicles points to the viewpoint (see Figure 8). Fist we see one big patch in the distance that comes closer and splits up into several detached objects. This time it is very difficult to decide when to define more than one moving object and how to manage their lifetime after they split up. The reverse way, when the vehicles goes by from the viewpoint and their silhouettes melt together in the distance, is not so problematic, because they come into the screen as independent objects, so their lifetime can be followed easily.

Fortunately, there are opportunities to enhance the performance of the pattern- or motion tracking. As we mentioned above, the most time consuming part of this task is to find similarity of the tracked pattern. If we take the lifetime (refer to Section 3.1) of the object into attention and know the position and movement of it, we can predict its position on the following frame by extrapolation. Using this, we can smallen the size of the neighborhood to find similarity, because the probability that it will be somewhere at the extrapolated position is very high, seeing that the speed of a vehicle usually does not change suddenly.

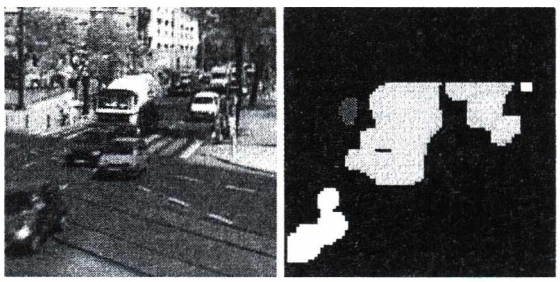


Figure 8: *The problem of common source: the tank-truck and the neighboring cars on the left have the same color (green)*

4. Future tasks

In this Section we describe our plans about extra features for the algorithm, which are to make it more flexible, more adaptive and much faster. Also an idea of a third output was born in the beginning of the project that is about to classify the passing vehicles (i.e. to order them into categories of cars, trucks, buses etc.).

4.1. Vehicle classification

In [2] a vehicle recognition and classification system is presented. It works on skeletal recognition of objects and it needs a lot of constrains (such as the orientation of the car) that have to be realized for the correct classification. The suggested approach also has some limitations including the problem of occlusion. Our project targets working on real-time PAL video, which means the whole tracking and measuring task (including the classification task) should be done in 40 ms. Naturally, some of these tasks could be completed only after the lifetime of the moving object reaches its end, because all of the information, measured while the object was visible, is needed for evaluation.

The real-time and flexible being of our algorithm demands not to spend too much time on recognition of vehicles. Besides, we cannot guarantee that the vehicles on the road are the most ideal orientation, because of the various setups, locations and orientations of cameras. So, at the moment our goal is just to classify the objects after their size, however, this is not the best way to decide whether a bus or a truck has passed in front of a camera. After deeper examination we try to teach the algorithm finding feature patterns on the texture of the objects to make more certain decisions on categories.

4.2. Problematic visibility cases

At the moment we have only test videos recorded in daylight. In many respects we are lucky, because of the relatively good lighting and visibility conditions on a good quality video. Unfortunately it is not absolutely true; there are many problems with daylight records:

- *shadows of the vehicles on the asphalt* – shadow enlarge the silhouette of a moving object that can cause fusion of multiple objects during object detection (see Section 2.2). The problem caused by this phenomenon could be eliminated if the *hue* information was also examined. At the moment we work on grayscale images. If we used also true color images in HLS[¶] color space it would hugely decrease the performance of the algorithm.
- *lens flare* – lens flare is caused by the multiple reflection and refraction of light between the lenses of the camera. This phenomenon can be seen as a relatively static artifact on the screen that changes with the change of sunlight. Hence, usually it causes very slow change in the foreground that can be interpreted as a slow change in the background. So, it can be easily eliminated by the foreground-background separation method mentioned in Section 2.1.
- *sunlight twinkle on the windscreen* – this phenomenon can be also originated in optical reasons, but it caused by the microscopic corrugation or rib on the surface of lens(es). The result is a bright line or curve on the screen (see Figure 9), which appears suddenly when the sunlight is reflected on the windscreen of a car or any other glossy surface (e.g. glazed coachwork). To eliminate this effect, we have to know the orientation of these twinkles. Then transforming the image into frequency domain we can filter them, but it is risky, if some line of the scene, which is important for evaluation, has the same orientation.

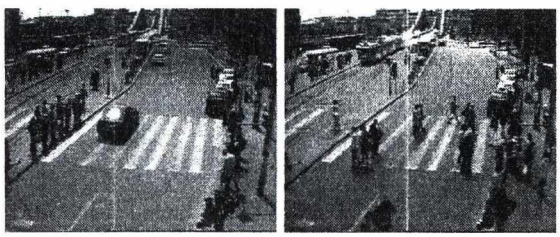


Figure 9: *The result of sunlight twinkle on the windscreen*

Now let us see the problems that we meet in other weather conditions:

- *nighttime scenes* – in the night we can meet the following problems:
  - *not visible coachwork* – if the CCD[‖] of the camera is not sensitive enough, or its histogram balancing system does not work well, the whole coachwork will not be visible in the screen, but only the two headlights of the car.
  - *shadows and light beams on the road* – the problem of shadow (now caused by the street-lighting) has been mentioned above. But in nighttime we can see also beam on the asphalt cast by the headlights of the car. This problem can be also solved by processing the hue channel of the color image (see above).
- *rainy, snowy scene* – the relatively ordered noise on the image caused by falling rain could be filtered in frequency domain (refer to the *sunlight twinkle* problem mentioned above). But in case of snow, which falls slower and in random directions, this solution does not work well.
- *blocky frames* – it is not decided at the moment by the project leaders, where the video processing unit will be deployed. Whether near to an analog camera, where the direct input is an analog video signal, or far from a digital camera where the input is an encoded, MPEG-like** digital video stream. When we save too much data transmission bandwidth, the encoded video could be blocky. This blockiness caused by the transformation-based being of the video encoding algorithm, where the blocks are the elementary units of the encoded data. On the other hand, in case of enough bandwidth they are unperceivable (see Figure 10).
  The blockiness is very harmful during motion tracking, because the edges of blocks influence the success of finding pattern similarities. There are algorithms that are to eliminate the blockiness of low bandwidth images, but usually their secondary effect is a little blur and they are also very time-consuming.

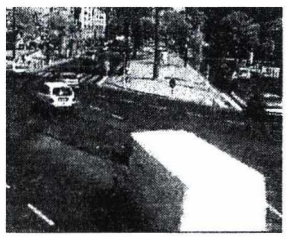


Figure 10: *Blocky image caused by the too low data transmission bandwidth*

4.3. Speeding up

The current implementation of our algorithm is written in C++ environment without any use of low-level instructions.

¶ HLS: Hue, Lightness, Saturation. This is an alternative color coordinate system derived from the RGB (Red, Green, Blue) system

‖ CCD: Charge-Coupled Device. CCDs convert light into proportional analog electrical current. 2D (or area array) CCD chips are responsible for capturing images in a camera or camcorder.

** MPEG: Motion Picture Experts Group. Today's digital videos are encoded into MPEG-1 (VideoCD), MPEG-2 (DVD, DVB), MPEG-4 (Camcorders, PC AVI codecs such as DivX, XviD) etc. format, which is a hybrid (transformation-based, differential) standard [11] video coding algorithm

Using low-level SIMD[††] instructions we can increase the speed of our algorithm and optimize image-related functions. Our plan is that we implement as much part of our source code into SSE[‡‡] instruction set assembly code as possible. Besides, we want to examine the possibility of porting some tasks to the GPU of today's modern graphics cards.

5. Actual results

At the time, when this article is being written, the algorithm is just able to clear the background, localize the moving objects, sign them with individual IDs and paint their silhouette with different colors. There is still no lifetime management, no counting, no speed measurement and no classification. The implementation and the tests are performed on a 1492 MHz Athlon XP 1700+ processor PC with 512 MB system memory running Windows XP operating system. As it is mentioned in Section 2 our test media is an MJPEG coded AVI file with CIF resolution. During the tests we process 1000 frames of the video. Observing the accuracy of the algorithm we generate control images that show the different phases of the algorithm frame by frame. These images are saved as 8-bit TGA (Targa) format image files with the resolution of 352×288. When we generate and save these images, the average frame time is 49.961 ms, which equals to 20.015 FPS frame rate. When the control image generation and saving is ignored (which is absolutely needless in the final version of the algorithm), the average frame time is 40.418 ms, which equals to 24.741 FPS frame rate.

6. Conclusions, discussion

The results show that after the development and test phase – when we work on direct video source and not from an AVI file, always reading it from the HDD, and when we implement the algorithm using SSE instruction set – we will have enough time between frames to do the remaining tasks such as lifetime management, object counting, speed measurement and vehicle classification.

The advantage of the algorithm is that it can be deployed everywhere in the capital city where a usable viewpoint can be found for a camera and a PC to observe the road traffic. The outputs of the application are just a few numbers in each minute, which means that this data can be easily collected, transmitted and broadcasted. On the other hand, one of its disadvantages is that some parameters need to be preset such as the 3D helper grid mentioned in Section 3.2.

However, this application could be the first city-wide road traffic management application, which provides public information about the city traffic that is accessible everywhere by a single WAP enabled mobile phone or the internet.

[††] SIMD: Single Instruction Multiple Data

[‡‡] SSE: Streaming SIMD Extension. It is an instruction set extension developed by Intel and used in every Pentium III (and later) PC CPUs

7. Acknowledgements

This project is managed by the Department of Highway and Railway Engineering at the Budapest University of Technology and Economics, Budapest, Hungary.

References

1. P. Mirchandani, M. Hickman, A. Angel and D. Chandnani, "Application of Aerial Video for Traffic Flow Monitoring and Management", *FIEOS 2002 Conference Proceedings, 2002.* 1
2. G. L. Foresti, V. Murino and C. Regazzoni, "Vehicle Recognition and Tracking from Road Image Sequences", *IEEE Transactions on Vehicular Technology, vol. 48, No. 1, January 1999* 1, 5
3. A. J. Lipton, "Local Application of Optic Flow to Analyse Rigid versus Non-Rigid Motion", *ICCV Workshop on Frame-Rate Vision, 1999* 1
4. E. Rivlin, M. Rudzsky, R. Goldenberg, U. Bogomolov, S. Lepchev, "A real-time system for classification of moving objects", *16th International Conference on Pattern Recognition, 2002. Proceedings* 1
5. Z. Zeng, S. Ma, "An Efficient Vision System for Multiple Car Tracking", *16th International Conference on Pattern Recognition, 2002. Proceedings* 1
6. A. M. Elgammal and L. S. Davis, "Probabilistic framework for segmenting people under occlusion", *8th IEEE International Conference on Computer Vision, Vancouver, 2001* 1
7. H. Fujiyoshi, T. Kanade, "Layered Detection for Multiple Overlapping Objects", *16th International Conference on Pattern Recognition, 2002. Proceedings* 1, 5
8. Cs. Benedek, T. Szirányi, "A Markov Random Field Model for Foreground-Background Separation", *Joint Hungarian-Austrian Conference on Image Processing and Pattern Recognition (5^{th} KÉPAF, 29^{th} OAGM/AAPR), Veszprém, Hungary, 2005* 2
9. Video Broadcast Standards – NTSC – PAL – SECAM, `http://www.alkenmrs.com/video/standards.html`, February 26. 2005
10. G. Vass, T. Perlaki, "Applying and removing lens distortion in post production", *Second Hungarian Conference on Computer Graphics and Geometry, 2003, Proceedings, pp. 90–97, Budapest, Hungary* 4
11. ISO/IEC 11172 (MPEG), International Standard, 1993 6

Enhanced Video Capture Support in OpenCV under Linux

Csaba, Kertesz

University of Szeged, Faculty of Science

Abstract

The wide-ranging multimedia support is very important in a desktop operating system if somebody would like to use the PC for entertainment or research projects (e.g. image processing, robot vision applications etc.). It is unavoidable to acquire sound or image sequences with different multimedia devices. The Intel's OpenCV (Open Source Computer Vision Library) library can be used for image processing and computer vision solutions. This library is available under different operating systems such as Linux and grown to a standard in the image processing applications last years. The program is constantly under development thus the author of this paper also significantly improved a few video capture features of this library. These results are summarized in the followings.

Subjects: *OpenCV, Video4Linux, Video4Linux2, video capture, webcam*

1. Introduction

Linux produced a noticeable advance last years, also in the area of driver programming. A lot of programmers recognized the possibility to make drivers for various multimedia devices and allowed of better work with Linux day by day. However the support for multimedia hardware is still far behind that for other operating systems, sometimes due to the lack of support and documentation by hardware manufacturers, sometimes due to licensing problems. Another major reason for this was the lack of a standardized programming interface.

A device driver is not usable for everybody if the author of driver writes it without collaborating with others using a common API. This purpose is served by V4L[1] and V4L2 specification[2]. Video for Linux is the first generation of standards for using video devices under Linux. The V4L API was firstly introduced in Linux 2.1 to unify and replace various TV and radio device related interfaces, developed independently by driver writers in prior years.

After a few years Video for Linux Two was a very good replacement for the V4L API that comes with the official newest Linux kernels. Starting with Linux 2.5 the much improved V4L2 API replaces the V4L API, although existing drivers continued to support V4L, either directly or through the V4L2 compatibility layer[3].

OpenCV maintained by Intel is an image processing library which is available under Windows and Linux too. The OpenCV implements a wide variety of tools for image interpretation. In spite of primitives such as filtering, image statistics, pyramids, OpenCV is mostly a high-level library implementing algorithms for calibration techniques, feature detection and tracking etc.[5]

OpenCV has a special sublibrary which does not contain image processing methods but it equips with wide-ranging facilities for handling image data like loading/saving images, video capture/storage functions. One of these features is video capture which has different support under Windows (DirectShow) and Unix-like systems (Videodev module, FireWire).

More improvements were written in video capture capabilities under Linux because there was only poor support in OpenCV regarding all prospects of V4L/V4L2 API. The following chapters describe the main features of V4L/V4L2, the old status of the video

capture support in OpenCV and the new improvements of the author in this part.

2. V4L/V4L2 Technical Background

There are two different parts in the Linux kernel regarding this multimedia programmable interface:

- API (Application Programming Interface),
- Videodev module.

The API of V4L is defined in videodev.h, Videodev module in videodev.c. The API of V4L2 in videodev2.h and module in videodev2.c respectively (files are parts of official kernels)[7]. The API is responsible for defining specification and Videodev module for handling several video drivers (registration and administration)[3].

The user reaches the interface through /dev logical directory under /dev/videoX nodes (X means an integer value). If a new device is plugged Linux will try to use an appropriate driver with Videodev module. The driver of device will register by Videodev module and V4L/V4L2 will create a special node (for example /dev/video0) which is a connection point between the user and the kernel driver of device. Abstraction layers are demonstrated in Figure 2 after References section. Nevertheless Videodev module only takes care that the right driver is reached with ioctl calls.

V4L/V4L2 supports following features with their API:

- Get/set capture main capabilities,
- Get/set image properties,
- Switch capture memory buffer modes,
- Synchronization of capture,
- Set audio mode etc.

The present situation is that many drivers of capture devices have got only Video for Linux support available. The V4L is the older specification and drivers are more stable in this mode thus V4L and V4L2 are living parallel and providing standard API layer for all video drivers.

3. Old V4L/V4L2 support in OpenCV

OpenCV did not support V4L2 till this summer. It handled only Video for Linux with following features:

- Automatic video source detection
(it was ready for V4L2 also),
- Automatic palette detection,
- Memory allocation with mmap mode,
- Synchronization of capture,
- Ability to change capture resolution
during capture process.

The biggest lack was no possibility to change image capture capabilities and adjust image properties (e.g. contrast, brightness) in OpenCV. This is very important in several applications to improve image quality with changing hardware capture control parameters which are useful many times instead of standard software algorithms because of e.g. decreasing CPU load or better image quality.

4. The developed novel image properties for OpenCV

Firstly the image properties for V4L and V4L2 devices were changed and the API of OpenCV (in concert with OpenCV developers) was modified with a few new constants in order to handle new properties:

- Brightness,
- Contrast,
- Saturation (it has a different name
"Colour" in V4L),
- Hue,
- Gain (this feature in V4L is not
supported).

These five image properties cover all features of the capture devices. Naturally, support of the several elements depends on implementing of the device drivers and the hardware design. If a driver or device does not support a property it generates an error message and functions of OpenCV return with value -1.

Nevertheless, there are differences between handling image properties in V4L and V4L2. Properties have a constant range 0-65535 in V4L but it changes in different driver implementations. Many times the limits of this range must be surveyed with manual modification of properties because of the lack of automatic detection of these values in V4L. On the other hand Video4Linux2 supports to get minimum and maximum limits of capture image properties that the new patches taking it into account.

The user scope is that the image features have float ranges 0-1 specified by the official API of OpenCV. The distinct value ranges in V4L/V4L2 solutions were handled and normalized in the new codes. But OpenCV detects range limits of properties only for V4L2 devices since the achievement would be very difficult in V4L.

5. New V4L2 support in OpenCV

Only another possibility was for the users if they did not have a V4L compatible device under Linux. Cam-

era with FireWire connection could be used instead of a V4L device but it was a partial solution because V4L2 provides also a very good multimedia support.

The first step of V4L2 development was the buying of a webcam (Genius VideoCam NB Security) which possesses a real stable V4L2 compatible device driver. After originating of hardware basis of development, initialization process of capture devices in OpenCV was rewritten and latter separation of V4L/V4L2 codes was designed. If user initials a new device in OpenCV the process is following:

1. Detection as V4L compatible device
2. Detection as V4L2 compatible device

Note that the API of V4L2 is more advanced than the API of V4L however many times device drivers are more stable in V4L mode therefore OpenCV tries to use every capture device as V4L at first.

A new palette (codec) detection for V4L2 was written. It supports following codecs/colorspaces:

- RGB (24 bit, uncompressed),
- YVU 4:2:0,
- YUV 4:1:1,
- SBGGR8 codec,
- SN9C10x codec.

First three standard codecs were implemented earlier in OpenCV for V4L devices and they are used by many webcams and capture cards. Last two codecs are specific by Genius VideoCams and they were introduced by the author in OpenCV however original SBGGR8 codec is part of a SN9C10x webcam application[8] and SN9C10X codec was written by Takafumi Mizuno[8].

The developed patches use mmap memory allocation function in image buffer. In that way the capture process will be faster as the user mapped function (latter provides only one frame in each capture period after clearing out image buffer). Another implemented feature is the synchronization of the capture process which is important to capture each frames after the others.

6. Test hardware/distributions

The developed patches for OpenCV were tested on several hardware platforms and distributions with various webcams (Genius NB Security, Creative Vista and some Logitech). That shows Figure 3 and Figure 4 after References section. Basically, test hardwares were i386 architectures although new 64 bit systems were presented also in tests (both AMD and Intel). It laid special emphasis on testing more versions of software environments:

- Different branches of Linux kernel (2.4.x and 2.6.x),
- Latest official release and CVS version of OpenCV,
- Debian (Stable and Testing) and Suse distributions.

The development in testing status was not met with big difficulties, there were only a few problems with compiling OpenCV and CamView (the later discussed sample application) caused by distinct ways of Linux distributions (e.g. library locations in filesystem). That was not needed to make a lot of changes in new patches because Linux systems have only different versions of V4L/V4L2 layer is provided by kernel and Videodev module which can be handled simply. Eventually all tests were successful without any problems.

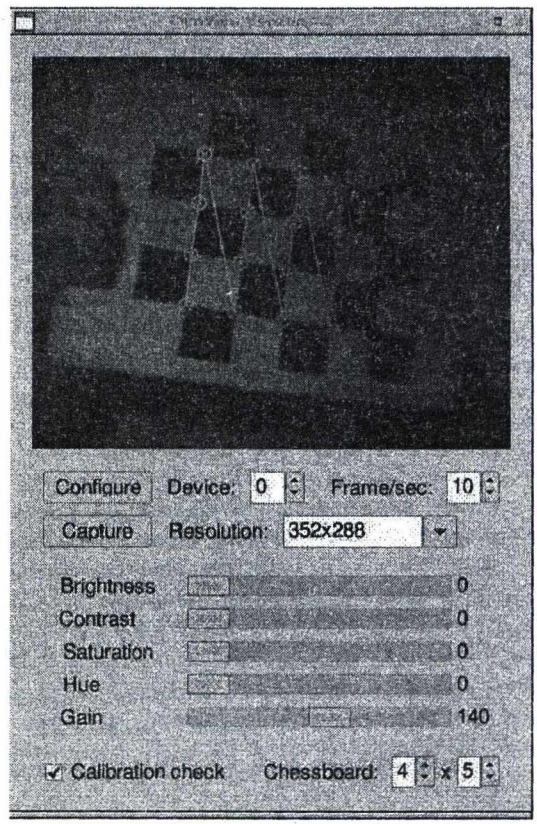


Figure 1: *Screenshot from CamView program.*

7. Sample application

A simple viewer application called CamView[6] was developed in GTK/GNOME graphical environment for webcams and other V4L and V4L2 compatible devices. It uses standard capture procedures of OpenCV and a few calibration functions in order to test easily effects of changing capture control parameters (Figure 1). The program can detect a chessboard on an image by help of OpenCV and show internal corner points detected by subpixel accuracy.

The window of CamView application was divided into four parts. The first part is the captured image of V4L/V4L2 device, in the second there is the main controls of capture features (device index /dev/videoX, fps and resolution), the third shows the properties of the captured image (contrast, brightness, saturation, hue, gain) and the task of the last part is the settings and to recognize and show the calibration chessboard.

Usage of program is very simple. After application was run the user can change the calibration control and the options of image properties and commit with Configure button. If an image property is not supported by capture device or driver the horizontal slider belongs to it will jump back to value zero. Program halts and prints error messages by strong situation (e.g. unsupported capture resolution).

The user can test the capture device in a few minutes with CamView and experience which capture capabilities are adjustable on it.

8. Summary

The paper have described the new API changes of OpenCV and the patch design with collaboration of two maintainers of Intel's library (Olivier Bornet and Vadim Pisarevsky). Four more important additions of OpenCV codes were created and tested on more types of computers and Linux distributions with several webcams. Test results were good and big differences can not be found between different computer platforms. This is a main advantage of the common standards and APIs in Open Source Community under Linux. New modifications are available on CVS page of OpenCV[4].

Current status of the V4L/V4L2 support patched by the author with maintainers of OpenCV was discussed and conclusion is the following: the new additions are stable and will take care of only future modifications of V4L/V4L2 kernel components or other standards.

This project confirmed V4L/V4L2 solutions have a very pleasant, good purposeful API for wide-range of the multimedia applications which can be implemented quickly and efficient. Eventually an open source project with good planned API and program architecture (e.g. OpenCV) is very useful and it can be improved by a programmer swimmingly.

References

1. Video4Linux resources: specifications, drivers for many devices etc.
URL: *http://www.exploits.org/v4l/*
2. Video4Linux2 main homepage: Specifications, drivers, programs etc.
URL: *http://linux.bytesex.org/v4l2/*
URL: *http://linuxtv.org/v4lwiki/index.php*
3. Metzler, Ralph: Video4Linux. 6th International Linux Kongress, September 8th to 10th, 1999, Augsburg, Germany.
4. OpenCV library homepages: General informations on Intel's pages and newest releases, CVS version of code on Sourceforge.net project page.
URL:
http://www.intel.com/technology/computing/opencv/
URL:
http://sourceforge.net/projects/opencvlibrary/
5. Open Source Computer Vision Library. Reference Manual. Intel Corporation, 1999-2001.
URL: *http://developer.intel.com*
6. CamView program. Part of Cognitive Vision project.
URL: *http://www.sourceforge.net/projects/aibo/*
7. Official kernel distributions of Linux systems.
URL: *http://www.kernel.org*
8. SN9C10x official driver and webcam application.
URL:
http://www.linux-projects.org/modules/news/

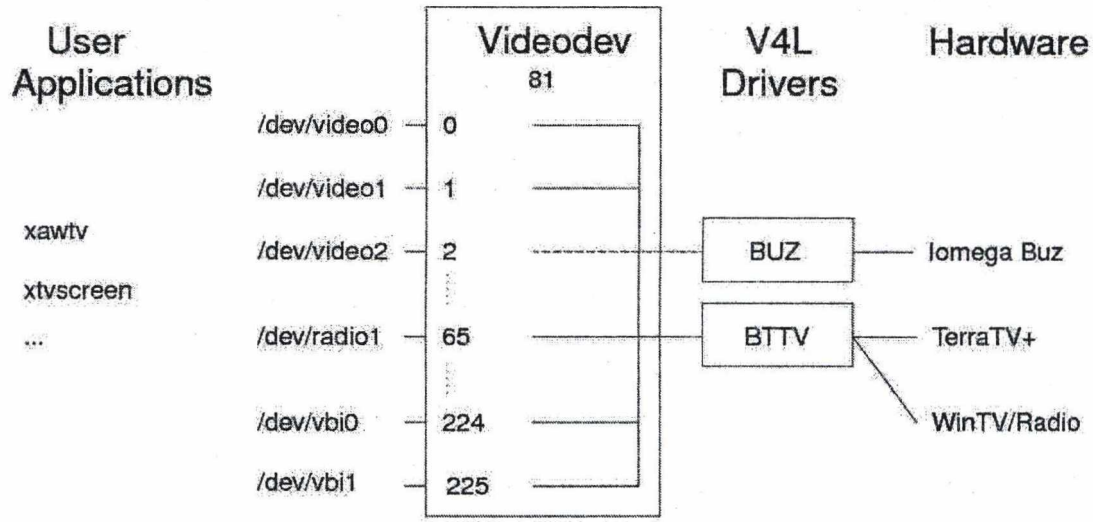


Figure 2: *Abstraction layers to handle video devices in Linux.*

Name	Processor	Kernel	Distribution	OpenCV	Tester
Toshiba Tecra 8200 Notebook	P3	2.6.12.2	Debian Stable	0.9.7	Csaba Kertesz
PC	AMD Athlon XP	2.6.12.2	Debian Stable	0.9.7	Csaba Kertesz
PC	Intel P4	2.6.10	Debian Stable	0.9.7	Csaba Kertesz
PC	Intel P3	2.4.21-99-default	Suse 9.0	0.9.7	Csaba Kertesz
PC	Intel P4	2.6.11.5	Debian Testing	CVS	Olivier Bornet
PC	64 bit Pentium	2.6.12	Debian Testing	CVS	Olivier Bornet
PC	64 bit AMD	2.6.12	Debian Testing	CVS	Olivier Bornet

Figure 3: *Features of tested platforms.*

Name	Driver name	Driver version	Driver compliance	Tester
Genius NB Security	sn9c10x	1.24	v4l2	Csaba Kertesz
Creative Vista	spca5xx	0.57	v4l	Csaba Kertesz
Logitech Quickcam Express	spca5xx	0.57	v4l	Csaba Kertesz
Logitech Messenger	qc-usb-messenger	0.8	v4l	Csaba Kertesz
Logitech Quickcam	pwc	10.0.7-2	v4l(/v4l2 unstable)	Olivier Bornet

Figure 4: *Tested webcams.*

Virtual Patient for Anatomical Ultrasound Guidance

Barnabás Takács
Digital Elite/WaveBand, Los Angeles
BTakacs@digitalElite.net

Gábor Szijártó, Balázs Benedek
VerAnim, Budapest
szijarto.gabor@freemail.hu , benedek@digitaldream.hu

Abstract

This paper presents a novel application of computer animated digital human technology applied to portable healthcare. It describes an advanced solution for patient-specific ultrasound guidance system which permits medics with minimal training to take highly accurate ultrasound scans in remote locations and transmit that information to doctors situated in another location. We implemented a demonstration system using a portable "back-pack" ultrasound device connected to our telemedicine program interface. It allows rapid diagnosis in the critical first hours after injury, without actually transporting the wounded person to a medical facility. Doctors are able to participate in diagnosis and treatment decisions through full access to the ultrasound scan from a remote hospital or field station via a digital network and use their own computer, PDA or even cell phone to view the images.

1. Introduction

This paper presents a novel application of digital human technology developed for the portable healthcare market. The system, called "*3D Anatomically Guided Ultrasound System (3D-AGUS)*" builds upon our *Virtual Human Interface* [1-3] technology to provide an advanced solution for patient-specific guidance. Through a simple to use interface it permits medics with minimal training to take highly accurate ultrasound scans in remote locations and to transmit that information to doctors via a digital network telemedicine interface. The demonstration system we developed employs a portable "back-pack" ultrasound device, which allows rapid diagnosis in the critical first hours after injury, without actually transporting the wounded person to a medical facility. Doctors are able to participate in diagnosis and treatment decisions through full access to the ultrasound scan from a remote hospital or field station via a digital network and use their own computer, PDA or even cell phone to view the images.

The *3D-AGUS* is an advanced real-time visual interface that provides expert assistance to medical personnel in performing diagnostic and therapeutic ultrasound on the site of an accident or in remote urban locations. The solution we developed is a rugged, portable and easily deployable "back pack" system that can be used to examine and treat the injured or sick in far-forward conditions. Using the built in guidance functions and telemedicine module, lower-skilled medics are capable of acquiring and transmitting ultrasound (US) scans to higher-skilled clinicians working at a remote telemedicine facility.

The prototype *3D-AGUS* combines the benefits of a generic virtual human anatomy model with visualizing 3D volumetric data sets (MRI & CT) of the patient that may be available from a central database. These volumetric data sets and a high fidelity generic 3D model of the human body together provide active computer assistance to the medic performing the examination by visually displaying the ways that the image plane of the ultrasound probe moves toward the specified anatomic location as she or he manipulates the ultrasound probe over the injured person's body. Our innovative approach employs measures the 3D motion of the probe and register its location to the precise features of the patient's body. The real-time visualization system was integrated with a *SonoSite C180 Plus* portable ultrasound (US) device [4]. Images obtained by the US probe are displayed in real time on an image plane precisely registered to the patient's body and internal organs. The 6DOF motion (x,y,z translation and yaw, pitch, and rotation) is tracked by an external device easily attached to the US probe itself.

2. Background

Ultrasound revolutionized the practice of diagnosis, patient care and internal surgery. In the field it provides a quick and reliable assessment of medical conditions that can be readily compared with pre-existing data sets. Ultrasound has also been in the forefront as a tool to expand the range of interventional procedures. As a result, during the past years many open operations have been replaced with less morbid, minimally invasive procedures. Today, surgical patients can often return home the same day and return to work in days, as compared to weeks with conventional surgery. The major obstacle facing medics as well as doctors learning to use ultrasound is understanding how the ultrasound images are oriented relative to the patient [5-8]. The basis of this problem is that the orientation techniques used to establish image context in the classical methods, such as cross section slicing, create a major difficulty for interpreting the data set. The techniques used by ultrasonographers to understand imagery rely heavily on knowing about the orientation of the patient and how/where the US probe is moving at any given moment time in relation to the body itself. The problems of interpreting the US data and performing diagnosis are particularly difficult when the person taking the scan is in a different location from the one who performs the diagnosis. This is frequently the case when medics on the field or in rapidly deployed ambulance cars need to care for injured patients. Furthermore, many of these medics do not have the skill and anatomical knowledge to ensure that the US scans obtained provide the most valuable information for remote diagnosis.

The solution to this problem is offered by a novel anatomical guidance and navigation system that displays the position and orientation of the ultrasound plane and images to the medic performing the scan in real time. The technique employed herein displays the required orientation information with the help of a generic 3D model of the human body as well as patient-specific volumetric data scans processed with the help of *Slicer* [9] as the main visual references. This technique is helpful because it provides both operators and doctors with important spatial cues. This has been shown to improve their ability to interpret the ultrasound. In addition, by using telemedicine and augmented reality technology, it also overcomes the problem of the physician having only limited information about the location of the probe with respect to the patient lying on the ground, and thus helps the mental representation of the ultrasound beam location inside the patient's body.

3. System Overview

The overall diagram of the prototype *3D-AGUS* system is shown in Figure 1. The system receives input from a commercially available ultrasound device that can be a hand held/portable or fixed platform installation. The motion of the ultrasound probe is tracked in 3D (position and orientation) using advanced computer low-cost tracking sensors. The measured 6 degrees of freedom (DOF - x,y,z position and yaw, pitch, and rotation) information is used to control the motion of the image plane slicing through the patient's body. The second component of the system is a generic highly detailed 3D human body model, which - depending on the level of detail required - includes internal organs, veins and limbic system, bones, and musculature. This generic 3D model is registered to the patient's body using a quick interactive selection of key feature points and statistically available anthropometric data sets. Furthermore, when available, patient specific volumetric data (MRI or CT) is readily included in the form of volumetric rendering yielding a final 3D patient model displayed by the 3D visualization engine. Finally, the detailed model and the tracked probe with the ultrasound image itself are displayed in an interactive 3D diagnostic imaging and therapeutic visualization environment which acts as an advanced human-machine interface, helping the medic see how the image plane moves toward the specified anatomic location as she or he manipulates the ultrasound probe over the patient's body.

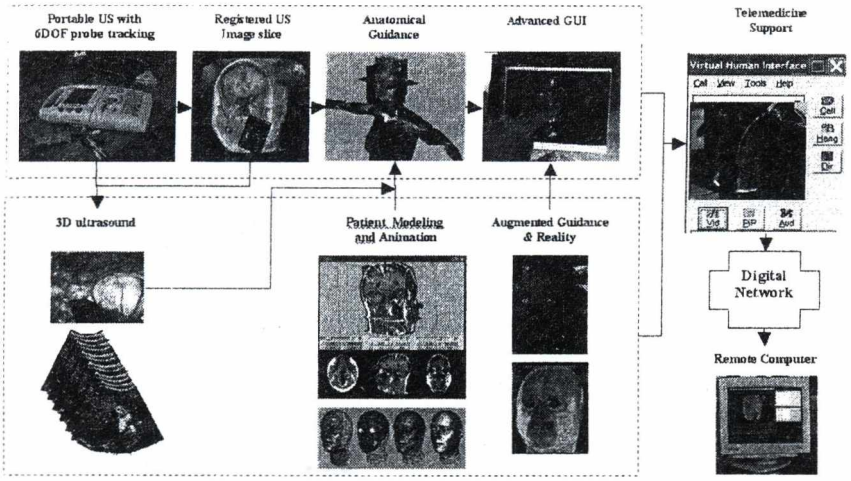


Figure 1. Functional diagram of the 3D Anatomically Guided Ultrasound System (3D-AGUS).

Figure 2 shows the portable *3D-AGUS* we developed to demonstrate the usability and applicability of our solution. In our experiments we used a *SonoSite C180 Plus* portable "back pack" field deployable device (bottom) with a small probe shown on the right. The US video image captured buy the US equipment is digitized and transmitted to a laptop computer or small footprint desktop computer at 30 frames per second (fps) via a USB 2.0 digital interface. The sequence of US images transmitted is directly mapped onto a visualization image plane in real time. To track the motion of the US probe multiple technical solutions were evaluated. The figure shows one of these external sensors comprised of a transmitter (placed on top of the computer) and a receiver, which is directly attached to the probe itself (right). The 6DOF motion of the US probe thus measured is subsequently mapped onto the respective geometric transformation axes controlling the position and orientation of the US image plane within the software. The following sections summarize the most important modules and aspects of the *3D-AGUS* solution.

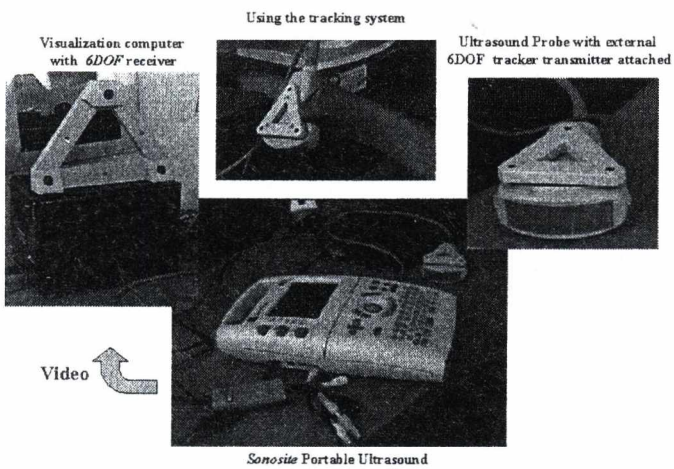


Figure 2. Portable ultrasound guidance demonstration system.

4. Basic Program Modules

4.1. Portable Ultrasound Interface

The prototype *3D-AGUS* system uses a portable US device manufactured by *SonoSite Inc* [4]. To interface the visual data stream output by the *SonoSite* 180 Plus device as a video signal, we developed a video interface using *dynamic texture* technology. Dynamic textures are high resolution images mapped onto the surface of any 3D object in a virtual environment at high speeds to create the effect of a live video feed. The technology was originally developed for projects in TV and film production and has been optimized to process the signal received from the portable US device. The *3D-AGUS* system accepts video signals from the ultrasound device via a standard RCA connector. The image is digitized and captured at 30 frames per second (fps) and subsequently mapped onto any object, in most cases a simple image plane, as a *dynamic texture* for display and further processing. The dynamic texture architecture is a unique feature of the anatomically guided ultrasound system. It allows the ultrasound image to be mapped onto any virtual object and at the same time to be processed in real time. We have implemented a variety of real-time image processing functions that can be used either to process and register the image to other objects, such as MRI data, or simply to enhance the US output for visualization purposes.

4.2. Spatial Calibration

The purpose of the calibration algorithm we developed is to tie the real-world coordinates in which the operator moves and rotates the ultrasound probe with the relative motion of the US image plane in the virtual guidance space. This image plane is then used to slice the virtual body to help the medic scan the patient in the locations required by the injury or guided remotely by the doctors. To map the real-world motion of the tracker attached to the ultrasound probe to the virtual patient we use an *anatomically correct 3D skeletal model.* Specifically, during calibration we ask the operator to interactively click on points on the patient's body when prompted. The prompt method we developed uses the anatomical bones of our virtual patient and indicates on the screen where the operator is requested to place the probe. As an example, the interface would prompt him or her to click on the left knee, the neck or the top of the head. Once all calibration points are collected, an automatic algorithm computes the appropriate affine transformation parameters that will be used for the process of obtaining the best possible US scan. This process in the simplest case uses two calibration points, one at the feet and one at the top of the head. The calibration algorithm prompts/asks the operator by showing a green marker on the body part and asking him to move the US probe to that location. A series of calibration points are quickly input this way to compute the global transformation parameters. The algorithm can use as many calibration points as necessary, thus allowing it to create a complete mapping of the real-world onto the digital patient model. Since the model refers to bone names, muscle groups etc. (using their Latin or English equivalent) it is very easy for the doctors situated at the remote telemedicine station to specify the area of interest for the operator.

4.3. Anatomically Correct 3D Virtual Human Model

The *3D-AGUS* prototype system contains both a generic human body model as well as person-specific 3D medical data. The former is represented as 3D model geometry and the latter with volumetric objects that directly display patient data obtained by MRI, CT scanners. We currently have implemented a detailed model that includes the skeletal system and the muscle systems. Models of internal organs as well as the limbic and vestibular systems are to be added later, as required. Figure 3 shows an example of the skeletal and muscle structures as well as the facial geometry of a female patient. Finally, the 3D model geometry can be combined by loading

multiple independent volumetric data sets obtained from MRI or CT scans. The volumetric data sets are linked to the bone system and move along as the virtual patient is animated. Figure 4 shows a pseudo-colored head MRI data set attached to the neck. The slices represent the *axial*, *coronal*, and *sagittal* image planes, respectively, and they can be moved to readily obtain any cross section in the data set. The complete virtual patient model is fully animatable thereby creating the foundation for accurately matching the posture, position, size, and even body type of the patient.

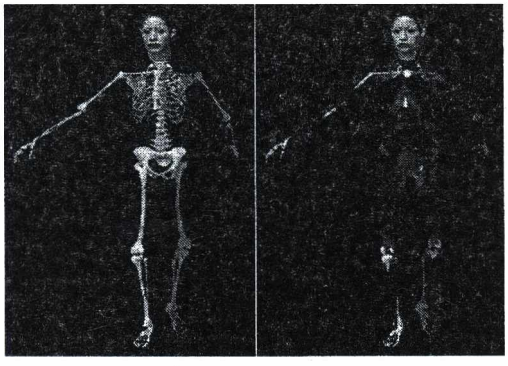


Figure 3. Example of anatomically correct detailed skeleton, muscles and facial geometry animated to match a specific body pose.

4.4. Volume Visualization and Slicing (MRI and CT)

Besides using generic human models and 3D geometry, one of the key advantages of the prototype ultrasound guidance system we developed is that it can display and process patient specific medical 3D data sets directly. Specifically, we implemented a volumetric visualization node that readily accept these kinds of data sets, such as MRI and CT slices, and displays them in an integrated fashion with other elements of the virtual anatomical guidance systems. The *volume node* uses 3D textures (u,v,w) as a basic data storage and takes advantage of the latest advances in hardware-supported rendering methodologies. However, our solution does not require the use of expensive computational platforms but rather a portable personal computer or laptop. The volume node supports multiple ways of displaying volumetric information. As an example, for flexible visualization properties, the image plane representing the ultrasound image can be used to glide through and penetrate this volume or to slice it directly. This is demonstrated in Figure 5. In the upper row the ultrasound image plane is shown inside the data set as it passes through the volume, while the lower row shows examples of using the image plane of the US scan to "slice" the volume and display the corresponding cross section.

Unlike other solutions, the volumetric visualization node is implemented as any other object within the 3D scene; thus it can be attached to the virtual humans's bone structure or linked to other geometries in the scene (see also Figure 4 above).

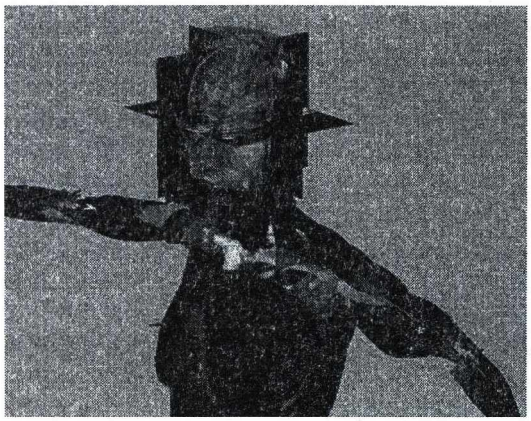


Figure 4. Combining volumetric information (pseudo colored) obtained from an MRI data set with 3D model geometry.

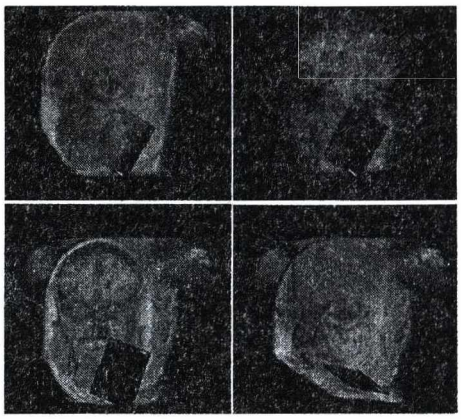


Figure 5. Visualizing volumetric MRI data set using different slicing methods.

5. Advanced Interfaces

5.1. Disc Controllers

The segmentation of large volumetric data sets of a person is a computationally intensive and time consuming task that needs to be done prior to a planned intervention. To provide anatomical guidance and US scanning capabilities on the field, the operator may not have the 3D models or the critical time required to generate these models readily available. To overcome this problem, we developed and demonstrated a novel technique, called *dynamic shading technology (DST)*. Dynamic shading provides *in-volume visualization* and US guidance for specific body parts *without(!) the explicit need for segmenting* the volume model. DST uses the same segmentation parameters based on gray scale values of different tissues selected manually or estimated automatically. However, DST works by computing these calculations in render time, directly on the graphics card hardware itself, thereby freeing up the CPU from performing the computations. As a result, DTS is a powerful method to quickly visualize inner organs and regions of the body to help guide the US acquisition process.

This technology was further enhanced by adding a novel interface, which allows medics to quickly access a wide range of internal body models while interactively changing the segmentation and imaging parameters. Specifically, to implement this interface we devised a unique and intuitive user interface called the *Disc Controller* (DC): The *Disc Controller* interface is a simple and intuitive device that will allow US operators to readily access different segmentation and algorithm parameters with ease. Figure 6 demonstrates the use of the DC for segmenting and visualizing the internal structure of a human head. The disc represents a continuous space of segmentation parameters. The red dots on the periphery and their associated labels refer to user defined names, such as "brain", "skull", "arteries", etc. The center of the disc is called the "neutral" position and it refers to the normal operation of the 3D anatomical display. The DC algorithm works by allowing the user to quickly access and navigate between these respective labels (and data sets referring to different internal organs) by using a mouse or simply touching the screen. The cross-shaped pointer displays the current settings/position in this image space. Since the specific labels on the periphery refer to unique combinations of segmentation parameters derived automatically or dialed in manually, these settings directly control the operation of the *real-time dynamic shader* pipeline by simply moving the pointer over any

location of the disc. *Vertex- and pixel shaders* are tiny program *fragments* running directly on the graphics card and determining the color and position of each pixel within the final rendered output. The algorithms implemented in vertex- and pixel shaders are controlled by *textures* and *constants* passed from the CPU when the operation is set up, typically once in every application. The dynamic pixel shader technology uses this basic framework, but instead of uploading the texture maps and associated constants only once, it does so 30 times per second, effectively providing very powerful means to constantly change and modify how the image appears and, consequently, how objects are displayed.

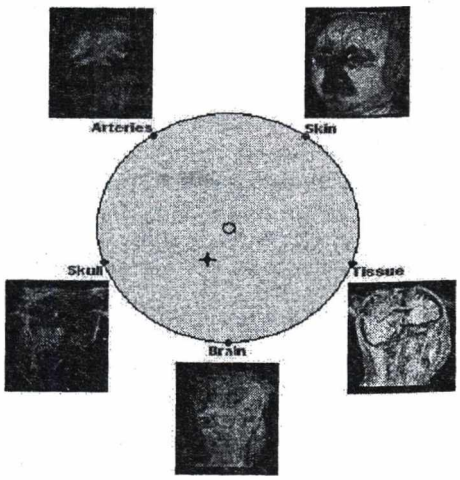


Figure 6. Disc Controller Interface to allow ultra sound operators to access different segmentation parameters in a simple and intuitive manner.

5.2. Touch Screen-based Interaction

Portable anatomical guidance systems like the one we developed are to be used by minimally trained operators in field conditions. Therefore, to ensure that they are capable of obtaining the best possible ultrasound images to be used by doctors in a remote location, it is critical to provide advanced user interfaces that simplify the scanning process and thus minimize the possibility of error.

Gesture-based input offers a simple touch and point interface that can be conveniently accessed any time by the operator. For the demonstration system we developed such a solution that allows the US operator to access areas of interest and navigate or explore the 3D virtual human model via a touch-screen device. Figure 7 demonstrates this interface. The *3D-AGUS* system was augmented with an external touch screen device that can be readily placed in front of the computer monitor (left). Once installed, the *3D-AGUS* system is capable of directly accessing the 2D location of the screen wherever the user might touch it (center). This information is subsequently mapped onto keyboard and mouse events, allowing the operator to select body parts of interest (click on), rotate the 3D model by dragging a finger, and/or access parameters. This interface is intuitive and very easy to use. It requires no special skill or training and can be readily configured to provide access to a broad range of information. As an example one may take an ultrasound scan using a US probe in the right hand while rotating the model on the screen via the touch-based interface with the left hand.

Figure 7. External touch screen interface to access 3D virtual human model during ultrasound scanning.

6. Conclusion

In this paper we described a novel *ultrasound guidance system* that employs high fidelity digital human models to provide medics with minimal training to take highly accurate ultrasound scans in remote locations. The system was designed then to transmit that information to doctors via a digital network with the help of a telemedicine interface. The demonstration system enables doctors to participate in diagnosis and treatment decisions through full access to the ultrasound scan from a remote hospital or field station via a digital network and use their own computer, PDA or even cell phone to view the images. The advanced capabilities of the *3D-AGUS* solution therefore allow for better patient care faster and higher quality medical response and as such it may become the foundation and the backbone architecture for a new kind of nationwide e-health solution.

Acknowledgement

This research has been partially funded by a contract from U.S. Army TATRC (ref: Ron Marchessault). The authors wish to thank to Dr. Steve Pieper and Dr. Kirby Vosburgh from Harvard University SPL for their valuable discussions and support.

7. References

[1] Digital Elite Inc. (2005), www.digitalElite.net.

[2] B. Takács, B. Kiss (2003), "Virtual Human Interface: a Photo-realistic Digital Human", *IEEE Computer Graphics and Applications*, **23**(5): pp. 38-45.

[3] B. Takács (2005), "Special Education and Rehabilitation: Teaching and Healing with Interactive Graphics", *IEEE Computer Graphics and Applications*, **25**(5): pp. 40-48.

[4] Sonosite Inc. (2005), www.sonosite.com.

[5] Birkett, D.H. (1998), "Advanced Display Devices, in Cybersurgery: Advanced Technologies for Surgical Practice," R.M. Satava, Editor., Wiley-Liss, New York.

[6] Sato, Y., M. Miyamoto, N. Nakamoto, Y. Nakajima, M. Shimada, M. Hashizume, S. Tamura (2001), "3D Ultrasound Image Acquisition Using Magneto-optic Hybrid Sensor for Laparoscopic Surgery," in Lecture Notes in Computer Science, **2208**, p. 1151-1153.

[7] Kane, R. (1999), "Intraoperative, Laparoscopic, and Endoluminal Ultrasound," Philadelphia: Churchill Livingstone, pp. 224.

[8] Boctor, E.M, A. Viswanathan, S. Pieper, M.A. Choti, R.H. Taylor, Ron Kikinis, and G. Fichtinger (2004), "CISUS: An Integrated 3D Ultrasound System for IGT Using a Modular Tracking API", SPIE Proceedings, **5367**, pp. 27.

[9] Slicer (2005), www.slicer.org

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Reconstruction of Optimally Sampled Volume Data

Balázs Csébfalvi†

Department of Control Engineering and Information Technology,
Budapest University of Technology and Economics

Abstract

Spatial signals are usually sampled on 3D rectilinear grids because of obvious practical advantages. Nevertheless, it is well known that a rectilinear grid is not optimal, even if the cells are congruent cubes resulting in a Cartesian Cubic (CC) grid. It has already been shown that an optimal Body-Centered Cubic (BCC) grid requires about 30% fewer samples to represent the same spatial information than an equivalent CC grid does. However, the reconstruction of an optimally sampled signal is problematic because of the non-separability of the BCC grid. In this paper different reconstruction techniques are reviewed which have been proposed to visualize BCC-sampled volumetric data, including our recently published Prefiltered Gaussian Reconstruction (PGR) technique.

Categories and Subject Descriptors (according to ACM CCS): I.4.5 [Image Processing]: Reconstruction–Transform Methods; I.4.10 [Image Processing]: Volumetric Image Representation; I.3.7 [Computer Graphics]: Three-dimensional Graphics and Realism.

1. Introduction

Volume data can be obtained from different sources. For example, it can contain the measured values of a physical property, like the X-ray attenuation coefficient (Hounsfield density) in a CT scan, or the result of a simulation, like Computational Fluid Dynamics (CFD). In order to smooth, warp, or morph surfaces, their geometrical representation can also be transformed into a volumetric model. Such a volume data contains samples of an implicit function (like a signed distance map) which defines the surface as a level set.

Generally volume data is supposed to be a discrete representation of an underlying continuous phenomenon. The accuracy of this discrete representation is strongly influenced by that where and how frequently the samples are taken from the original function (or signal). From a signal-processing point of view, the original continuous signal can be perfectly reconstructed from the samples if it is band-limited and the sampling frequency is above the Nyquist limit [13].

Samples are usually defined on a rectilinear grid, which has several practical advantages. For instance, samples are stored in 3D arrays, therefore they can be easily addressed. Furthermore, for many volume-processing or rendering algorithms, it is important to rapidly access the neighboring voxels of a certain voxel. Convolution-based filtering or interpolation, for example, can be efficiently implemented on a rectilinear grid.

Nevertheless, it is well known that a rectilinear grid is not optimal, even if the cells are congruent cubes resulting in a Cartesian Cubic (CC) grid [14]. If a band-limited signal is sampled on an optimal grid, the number of samples necessary for a perfect signal reconstruction is minimal. The optimal regular sampling of spatial signals is closely related to the classical sphere-packing problem [2, 15], which is also referred to as Kepler's problem in the literature. According to the Kepler conjecture, spheres can be optimally packed in 3D if the centers of the spheres are located on a Face-Centered Cubic (FCC) grid (see Figure 1). This almost 400-year-old conjecture was proven just in 1998 by Hales [5].

An optimal 3D sampling grid is derived from an optimal sphere-packing grid in the following way. Assume that the original signal has a spherical spectrum (there are no preferred directions). The spectrum of the sampled signal contains the replicas of the primary spectrum, centered at the points of the dual (or reciprocal) of the sampling grid [13]. The original signal can be reconstructed if there is no overlapping between the replicas. On the other hand, the spars-

† e-mail: cseb@iit.bme.hu http://www.iit.bme.hu/~cseb/

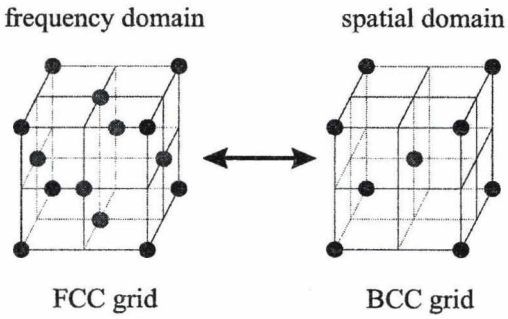


Figure 1: *Duality between the FCC and BCC grids.*

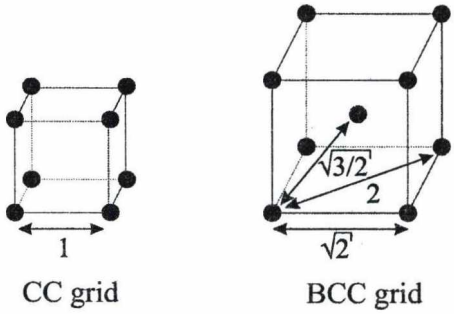


Figure 2: *Equivalent CC and BCC grids.*

est sampling in the spatial domain corresponds to the tightest arrangement of spheres in the frequency domain. The FCC grid is an optimal sphere-packing grid, therefore its dual, which is the Body-Centered Cubic (BCC) grid (see Figure 1), defines an optimal sampling pattern [18].

2. Reconstruction of CC-sampled signals

Many volume-processing and volume-rendering methods require the reconstruction of the underlying continuous function from the discrete samples stored in the volume data. The fidelity of the continuous reconstruction strongly depends on the quality of the applied resampling filter. Therefore several researchers analyzed different reconstruction and derivative filters, both in terms of accuracy and computational cost [9, 8, 10, 11]. Generally, the wider the support of the filter in the spatial domain, the better its quality. For reconstructing band-limited CC-sampled volumetric data, the best reconstruction filter is the 3D *sinc* kernel, since it represents an ideal low-pass filter. In practice, however, it is difficult to convolve a signal with the *sinc* kernel, because of its infinite support. Therefore practical filters either approximate it or truncate it by an appropriate windowing function [8, 16]. On the other hand, convolution of a CC-sampled signal with separable kernels, like the trilinear or tricubic filters, can be easily and efficiently implemented.

3. Reconstruction of BCC-sampled signals

Although the BCC grid requires about 30% fewer samples to represent the same amount of spatial information than an equivalent CC grid does (see Figure 2), it is not widely used in practice. This is mainly because of the lack of efficient adaptation of high-quality volume reconstruction methods developed for the rectilinear volume representation.

Especially interpolating filters are difficult to design for the BCC grid, since a 1D interpolating kernel cannot be extended to 3D by a separable (or tensor product) extension. The reason is that the BCC grid is not separable itself. A 3D interpolation kernel has to take a value of one at the origin and a value of zero at all the other discrete sample points. This criterion, however, does not make it easy to explicitly define the 3D interpolation kernel by a simple closed form, which could be efficiently evaluated.

Even the derivation of the ideal low-pass filter for the BCC grid is more complicated than for the CC grid. For the CC grid the ideal 3D kernel is simply the separable extension of the 1D *sinc* function. For the non-separable BCC grid, the ideal low-pass filter is derived in the following way [4]. The Voronoi cell of the dual FCC grid is a rhombic dodecahedron. The corresponding characteristic function takes a value of one inside the cell, and a value of zero outside the cell. The ideal low-pass kernel is defined as the inverse Fourier transform of this characteristic function.

From a practical point of view, it is also important how the BCC-sampled data should be stored to support convolution-based filtering. Anyway, due to the more complicated indexing scheme, the neighboring voxels cannot be as efficiently accessed as in the case of a CC, or a rectilinear grid.

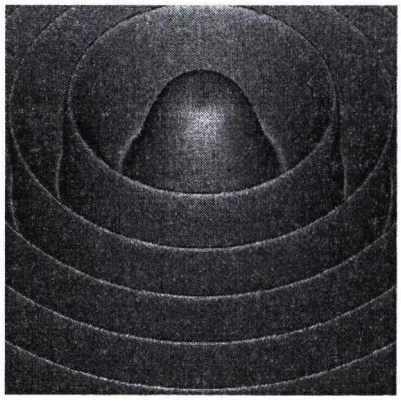


Figure 3: *The continuous Marschner-Lobb test signal.*

In the following sections different reconstruction methods are reviewed which have been proposed up to now in order to visualize BCC-sampled volume data. We compared these

techniques by rendering the Marschner-Lobb test signal [8] (see Figure 3) using a high-quality ray caster with the same rendering parameters. In order to demonstrate the gradient-estimation capability of the previous methods, for the shading computation, we estimated the gradients by calculating central differences on the reconstructed signal.

3.1. Spherically symmetric filters

Theußl et al. used spherical extension of 1D reconstruction filters to render BCC-sampled data by the splatting method [18]. This approach, however, resulted in blurry images. Although the spherical extension seems to be a natural choice to define a filter for the BCC grid, it does not fulfill the interpolation constraint, even if the extended 1D filter is interpolating.

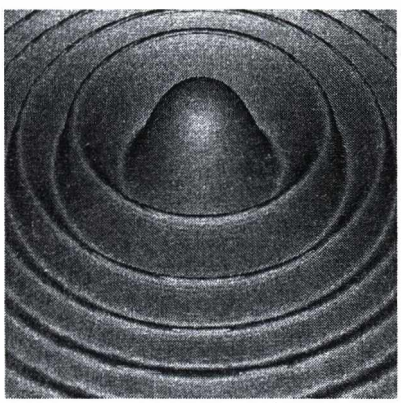


Figure 4: *Gaussian reconstruction from* $32 \times 32 \times 32 \times 2$ *BCC samples.*

Figure 4 shows the test signal reconstructed by a spherically symmetric 3D Gaussian kernel. As it is an approximating filter, the high-frequency details are blurred and the isosurface is significantly displaced.

3.2. Sheared trilinear interpolation

Theußl et al. implemented also ray casting for the BCC grid using various interpolation techniques [17]. Among these techniques the best results were produced by sheared trilinear interpolation. According to this approach, trilinear interpolation is implemented in the sheared space, exploiting that the BCC grid can be treated as a sheared rectilinear grid (see Figure 5). This method, however, is direction-dependent, therefore it causes view-dependent artifacts. Furthermore, central differences on the reconstructed signal produce poor-quality gradients (see Figure 6). Therefore, a more sophisticated gradient-estimation method has to be applied to precompute gradients at the grid points. Gradient samples at arbitrary sample positions can then be interpolated from the precomputed gradients by also using a sheared trilinear interpolation. Precomputed gradients, however, require additional storage.

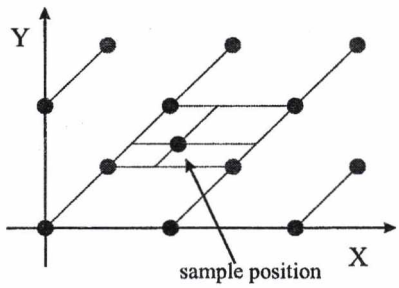


Figure 5: *Sheared trilinear interpolation.*

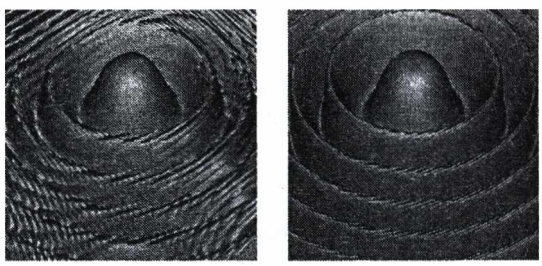


Figure 6: *Reconstruction by sheared trilinear interpolation.*

3.3. Linear box-spline reconstruction

Entezari et al. [4] derived a linear box-spline filter for the BCC grid. This approach is based on a clever extension of the 1D linear (or tent) filter to higher dimensions. The 1D linear kernel can be obtained by projecting a 2D box along its diagonal axis onto the 1D space (see Figure 7). Similarly, projecting a 3D box onto the 2D space, a linear kernel is obtained for the hexagonal grid, which is an optimal 2D sampling grid [20]. For the optimal BCC grid, a linear kernel is derived by projecting a 4D hypercube (tesseract) along its antipodal axis down to the 3D space. The obtained 3D filter is non-zero inside a rhombic dodecahedron, which can be interpreted as a 3D shadow of a tesseract.

Using the linear box-spline filter, a BCC-sampled volume data can be more efficiently rendered by a software-implemented ray caster than an equivalent CC-sampled data using the popular trilinear filter. On the other hand, CC-sampled data can be interactively rendered by the conventional graphics hardware, which supports trilinear interpolation. Hardware-accelerated rendering of BCC-sampled data based on the linear box-spline filter, however, has not been implemented yet.

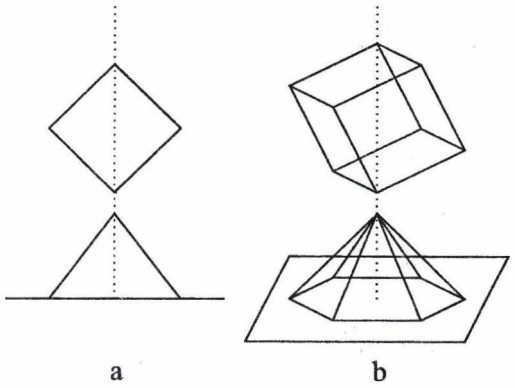


Figure 7: *(a): 1D linear box spline. (b): 2D hexagonal linear box spline.*

3.4. Cubic box-spline reconstruction

Entezari et al. [4] also derived a cubic box-spline filter for the BCC grid. In 1D the cubic box spline is obtained by convolving the linear box spline with itself. Similarly, a cubic box spline for the BCC grid is defined by convolving the linear box-spline kernel reviewed in Section 3.3 with itself. By further successive convolutions a family of odd-order box splines can be derived for the BCC grid.

Unlike the linear box spline, the cubic box spline is an approximating filter, since it does not fulfill the interpolation constraint. Therefore, the cubic box spline cannot exactly reconstruct the values of the original signal at the discrete sample points.

Furthermore, the cubic box-spline reconstruction, as it was derived in [4], requires the evaluation of 81 terms for each neighboring voxel covered by the filter kernel. Therefore, without any simplification, it is impractical because of its enormous computational cost.

3.5. Prefiltered Gaussian reconstruction

As it is difficult to explicitly define a filter for the BCC grid which fulfills the interpolation constraint, it seems to be straightforward to implicitly make an approximating filter interpolating by prefiltering the original discrete samples. This conception was introduced by Blu et al. as *Generalized Interpolation* [1]. Traditional interpolation can be formulated as a convolution of the discrete samples $f_i = f(x_i)$ with a continuous reconstruction kernel $\phi(x)$:

$$f(x) \approx \tilde{f}(x) = \sum_{i=0}^{N-1} f_i \cdot \phi(x - x_i). \tag{1}$$

In contrast, using generalized interpolation, the shifted basis functions are weighted by coefficients w_i, which are not necessarily equal to the original samples f_i:

$$f(x) \approx \tilde{f}(x) = \sum_{i=0}^{N-1} w_i \cdot \phi(x - x_i). \tag{2}$$

According to the interpolation constraint, weighting factors w_i have to be determined such that $\tilde{f}(x_i) = f_i$. This condition results in the following system of linear equations:

$$\mathbf{A} \cdot \mathbf{w} = \mathbf{f}, \tag{3}$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_{N-1}]$, $\mathbf{f} = [f_0, f_1, \ldots, f_{N-1}]$, and the elements of coefficient matrix $\mathbf{A}$ are defined as $a_{i,j} = \phi(x_i - x_j)$. Since Equation 2 actually defines a convolution, the unknown variables w_i can be easily determined by a deconvolution performed as a division in the frequency domain. Therfore, a computationally expensive linear algebra method, like the symmetric LU decomposition, does not have to applied to solve Equation 3.

Generalized interpolation can be easily adapted to volume data defined on a regular rectilinear grid by using a simple separable 3D extension. The implementation of a deconvolution for the non-separable BCC grid, however, is not that obvious. Nevertheless, applying our recently published method [3], the periodicity and symmetry of the BCC grid can still be exploited in order to efficiently perform a discrete deconvolution in the frequency domain.

Actually a BCC grid can be interpreted as two overlapping CC grids (Figure 8). One of them contains grid points $\mathbf{x}^{(1)}_{i,j,k}$ (red dots) and the other one consists of grid points $\mathbf{x}^{(2)}_{i,j,k}$ (blue dots), where $i \in \{0,1,\ldots,N_x-1\}$, $j \in \{0,1,\ldots,N_y-1\}$, and $k \in \{0,1,\ldots,N_z-1\}$. Due to the geometry of the BCC grid $\mathbf{x}^{(1)}_{i,j,k} = [i \cdot T, j \cdot T, k \cdot T]$, and $\mathbf{x}^{(2)}_{i,j,k} = [(i+0.5) \cdot T, (j+0.5) \cdot T, (k+0.5) \cdot T]$, where T is the sampling distance in the separate CC grids.

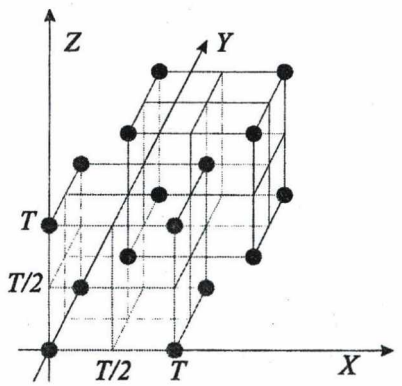


Figure 8: *The BCC grid as two overlapping CC grids.*

Radial Basis Function (RBF) reconstruction [19, 12, 6] of the original 3D function $f(\mathbf{x})$ sampled on a BCC grid is formulated as two separate convolutions on the overlapping CC

grids:

$$\tilde{f}(\mathbf{x}) = \sum_{i,j,k} w^{(1)}_{i,j,k} \cdot \phi\left(\frac{\mathbf{x} - \mathbf{x}^{(1)}_{i,j,k}}{T}\right) + \sum_{i,j,k} w^{(2)}_{i,j,k} \cdot \phi\left(\frac{\mathbf{x} - \mathbf{x}^{(2)}_{i,j,k}}{T}\right). \tag{4}$$

In our implementation we used an invertible 3D Gaussian kernel: $\phi(\mathbf{x}) = e^{-|\mathbf{x}|^2/(2\sigma^2)}$. In order to accurately reconstruct the samples $f^{(1)}_{i,j,k} = f(\mathbf{x}^{(1)}_{i,j,k})$, and $f^{(2)}_{i,j,k} = f(\mathbf{x}^{(2)}_{i,j,k})$ of function $f(\mathbf{x})$, the following conditions have to be fulfilled:

$$\tilde{f}(\mathbf{x}^{(1)}_{i,j,k}) = \sum_{l,m,n} w^{(1)}_{l,m,n} \cdot \phi\left(\frac{\mathbf{x}^{(1)}_{i,j,k} - \mathbf{x}^{(1)}_{l,m,n}}{T}\right) + \sum_{l,m,n} w^{(2)}_{l,m,n} \cdot \phi\left(\frac{\mathbf{x}^{(1)}_{i,j,k} - \mathbf{x}^{(2)}_{l,m,n}}{T}\right) = (w^{(1)} \otimes g)_{i,j,k} + (w^{(2)} \otimes g^{(2)})_{i,j,k} = f^{(1)}_{i,j,k}, \tag{5}$$

and

$$\tilde{f}(\mathbf{x}^{(2)}_{i,j,k}) = \sum_{l,m,n} w^{(1)}_{l,m,n} \cdot \phi\left(\frac{\mathbf{x}^{(2)}_{i,j,k} - \mathbf{x}^{(1)}_{l,m,n}}{T}\right) + \sum_{l,m,n} w^{(2)}_{l,m,n} \cdot \phi\left(\frac{\mathbf{x}^{(2)}_{i,j,k} - \mathbf{x}^{(2)}_{l,m,n}}{T}\right) = (w^{(1)} \otimes g^{(1)})_{i,j,k} + (w^{(2)} \otimes g)_{i,j,k} = f^{(2)}_{i,j,k}.$$

In Equation 5 discrete convolution kernel $g^{(1)}$ expresses the influence of weights $w^{(1)}_{l,m,n}$ (which are the weighting factors of the Gaussian kernels centered at positions $\mathbf{x}^{(1)}_{l,m,n}$) onto the reconstructed samples $\tilde{f}(\mathbf{x}^{(2)}_{i,j,k})$. Similarly, discrete convolution kernel $g^{(2)}$ expresses the influence of weights $w^{(2)}_{l,m,n}$ (which are the weighting factors of the Gaussian kernels centered at positions $\mathbf{x}^{(2)}_{l,m,n}$) onto the reconstructed samples $\tilde{f}(\mathbf{x}^{(1)}_{i,j,k})$. Furthermore, discrete convolution kernel g is obtained by sampling the continuous 3D Gaussian kernel $\phi(\mathbf{x})$. More precisely, discrete functions $g_{i,j,k}$, $g^{(1)}_{i,j,k}$, and $g^{(2)}_{i,j,k}$ are defined as follows:

$$g_{i \bmod N_x, j \bmod N_y, k \bmod N_z} = \phi([i,j,k]), \tag{6}$$

$$g^{(1)}_{i \bmod N_x, j \bmod N_y, k \bmod N_z} = \phi([i+0.5, j+0.5, k+0.5]),$$

$$g^{(2)}_{i \bmod N_x, j \bmod N_y, k \bmod N_z} = g^{(1)}_{(i-1) \bmod N_x, (j-1) \bmod N_y, (k-1) \bmod N_z},$$

where $i \in \{-N_x/2, \ldots, N_x/2-1\}$, $j \in \{-N_y/2, \ldots, N_y/2-1\}$, and $k \in \{-N_z/2, \ldots, N_z/2-1\}$.

The unknown weighting factors $w^{(1)}_{i,j,k}$ and $w^{(2)}_{i,j,k}$ are determined from the system of $N_x \cdot N_y \cdot N_z \cdot 2$ linear equations (5). Note that the number of unknown variables is the same as the number of conditions, therefore there is a unique solution. Since the unknown variables are formulated as two 3D discrete functions defined by convolution operations, the solution can be efficiently obtained by deconvolution operations performed in the frequency domain. Let us denote the discrete Fourier transforms of discrete functions $f^{(1)}_{i,j,k}$, $f^{(2)}_{i,j,k}$, $w^{(1)}_{i,j,k}$, $w^{(2)}_{i,j,k}$, $g_{i,j,k}$, $g^{(1)}_{i,j,k}$, and $g^{(2)}_{i,j,k}$ by $F^{(1)}_{\alpha,\beta,\gamma}$, $F^{(2)}_{\alpha,\beta,\gamma}$, $W^{(1)}_{\alpha,\beta,\gamma}$, $W^{(2)}_{\alpha,\beta,\gamma}$, $G_{\alpha,\beta,\gamma}$, $G^{(1)}_{\alpha,\beta,\gamma}$, and $G^{(2)}_{\alpha,\beta,\gamma}$ respectively. Since convolution in the spatial domain is equivalent to multiplication in the frequency domain, Equation 5 is formulated in the frequency domain as follows:

$$W^{(1)}_{\alpha,\beta,\gamma} G_{\alpha,\beta,\gamma} + W^{(2)}_{\alpha,\beta,\gamma} G^{(2)}_{\alpha,\beta,\gamma} = F^{(1)}_{\alpha,\beta,\gamma}, \tag{7}$$

$$W^{(1)}_{\alpha,\beta,\gamma} G^{(1)}_{\alpha,\beta,\gamma} + W^{(2)}_{\alpha,\beta,\gamma} G_{\alpha,\beta,\gamma} = F^{(2)}_{\alpha,\beta,\gamma}.$$

By solving Equation 7, $W^{(1)}_{\alpha,\beta,\gamma}$ and $W^{(2)}_{\alpha,\beta,\gamma}$ are obtained as:

$$W^{(1)}_{\alpha,\beta,\gamma} = (F^{(1)} G - F^{(2)} G^{(2)})_{\alpha,\beta,\gamma} / (GG - G^{(1)} G^{(2)})_{\alpha,\beta,\gamma},$$
$$W^{(2)}_{\alpha,\beta,\gamma} = (F^{(2)} G - F^{(1)} G^{(1)})_{\alpha,\beta,\gamma} / (GG - G^{(1)} G^{(2)})_{\alpha,\beta,\gamma}. \tag{8}$$

Finally, for RBF reconstruction of $f(\mathbf{x})$ (Equation 4), weighting factors $w^{(1)}_{i,j,k}$ and $w^{(2)}_{i,j,k}$ are calculated by inverse discrete Fourier transforms of $W^{(1)}_{\alpha,\beta,\gamma}$ and $W^{(2)}_{\alpha,\beta,\gamma}$ respectively.

Note that, using the above reconstruction scheme, a continuous Gaussian reconstruction follows a discrete Gaussian deconvolution, which is actually a prefiltering of the original discrete samples. Therefore we called this scheme *Prefiltered Gaussian Reconstruction* (PGR) [3]. The prefiltering has to be performed only once, and afterwards an arbitrary sample can be evaluated by a simple Gaussian convolution according to Equation 4. For this operation only the prefiltered BCC samples (weighting factors $w^{(1)}_{i,j,k}$ and $w^{(2)}_{i,j,k}$) need to be stored in the main memory. Although the continuous Gaussian kernel has an infinite extent, it can be well approximated by a truncated Gaussian kernel in order to efficiently implement spatial-domain convolution. Because of this truncation PGR is theoretically an approximation. However, the approximation error vanishes exponentially if the number of neighboring voxels taken into account is increased. Therefore, from a practical point of view, PGR is an interpolation method. The influence of the truncated kernel is usually restricted to the interval $[-3\sigma, 3\sigma]^3$, since outside this interval the Gaussian function practically equals to zero. In our implementation we wanted to limit the convolution window to $4 \times 4 \times 4 \times 2$ voxels, therefore we used $\sigma = 0.6$.

data set	MRI brain	CT engine
linear box spline	9.06 sec	13.17 sec
cubic box spline	34 min	51 min
discrete prefiltering	12.16 sec	3.34 sec
Gaussian reconstruction	2.38 min	3.48 min

Table 1: *Rendering times using different reconstruction methods.*

Figure 9 shows images rendered by using linear box-spline (a), cubic box-spline (b), and prefiltered Gaussian (c) reconstructions. The gradients for shading were estimated by central differencing on the reconstructed function. Alternatively the partial derivatives can also be obtained by convolving the discrete volume representation with the partial derivatives of the reconstruction kernel. Figure 10 shows the angular errors of the estimated gradients, where angular error of 30 degrees is mapped onto white. Note that linear box-spline reconstruction causes severe aliasing, while cubic box-spline reconstruction results in smoothing, and therefore slight displacement of the isosurface. In contrast, PGR ($\sigma = 0.6$) well preserves the depth of the wave valleys without significant aliasing.

PGR was tested also on practical data sets. As the BCC grid is a subgrid of the CC grid, we artificially produced BCC-sampled data by simply downsampling CC-sampled real-world data sets on a BCC subgrid. Nevertheless, in order to naturally generate BCC-sampled data directly by a CT or MRI scanner, only the tomographic reconstruction software of such devices need to be modified. Since the slices are usually computed by the classical Filtered Back-Projection (FBP) algorithm [7], the discrete tomographic reconstruction can be easily performed on a translated 2D square grid for every second slice in order to provide BCC samples. However, until commercial BCC-sampling scanners are available, such a natural data acquisition on an optimal BCC grid is not possible.

Figure 11 shows the reconstructions of our artificially produced BCC-sampled data sets. The resolution of the original MRI scan of a human brain was $256 \times 256 \times 166$, while the resolution of the original CT scan of an engine block was $256 \times 256 \times 110$. From these CC-sampled data sets we selected $128 \times 128 \times 83 \times 2$ and $128 \times 128 \times 55 \times 2$ BCC samples respectively. Note that, using linear box-spline reconstruction (a, b) aliasing appears as a regular pattern on the surfaces, while cubic box-spline reconstruction (c, d) blurs the high-frequency details. As it is clearly visible in Figure 11 (e, f), PGR does not smooth the surfaces, and preserves the fine details even better than linear box-spline reconstruction does.

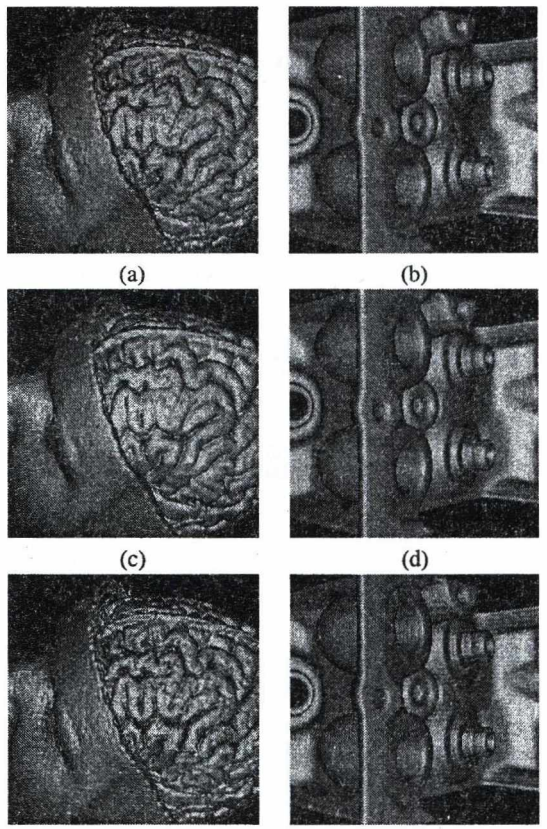
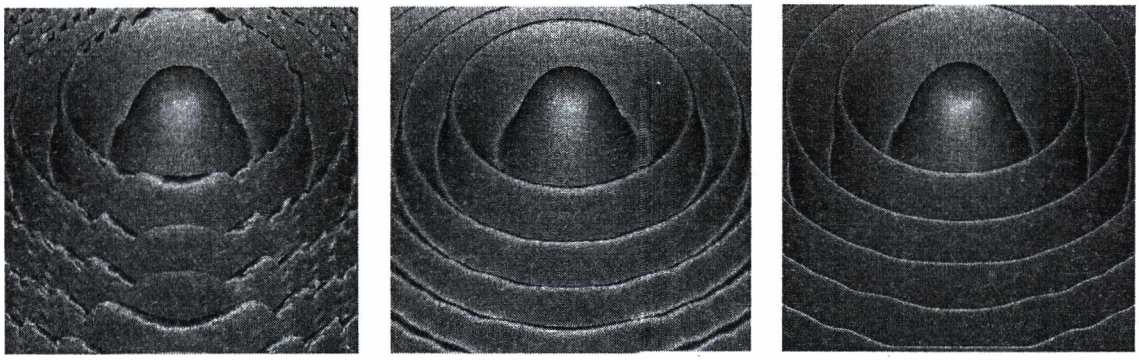


Figure 11: *Reconstruction of BCC-sampled practical data. (a, c, e): MRI data of a human brain containing* $128 \times 128 \times 83 \times 2$ *BCC samples. (b, d, f): CT data of an engine block containing* $128 \times 128 \times 55 \times 2$ *BCC samples. (a, b): Linear box-spline reconstruction. (c, d): Cubic box-spline reconstruction. (e, f): Prefiltered Gaussian reconstruction.*

Rendering times measured on a 3GHz Intel Pentium 4 PC with 1GB of RAM are shown in Table 1. Images of resolution 256×256 were generated by the same ray caster using different reconstruction techniques for resampling. Although PGR is slower than linear box-spline reconstruction, it is significantly faster than cubic box-spline reconstruction. Note that the time cost of the preprocessing for PGR, which is the discrete prefiltering, is relatively low compared to the rendering time. Nevertheless, the preprocessing has to be performed only once for each data set, and afterwards the preprocessed volume can be rendered from an arbitrary viewing direction.

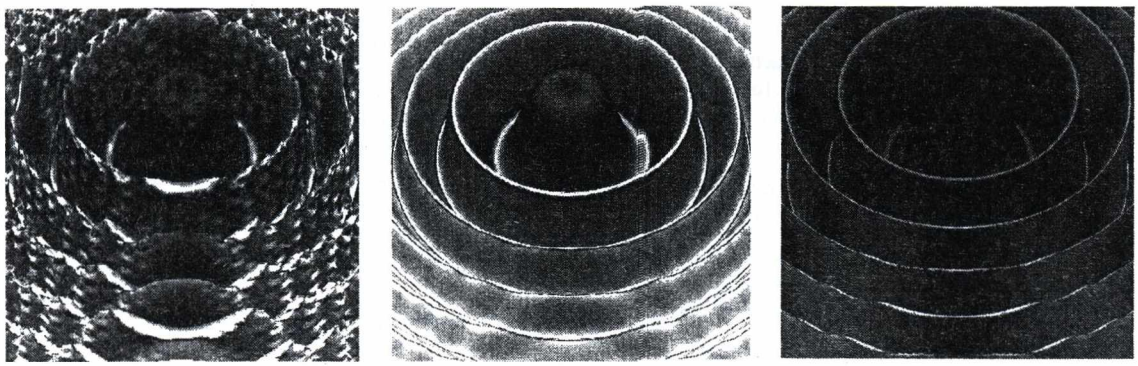
4. Conclusion

In this paper several reconstruction techniques proposed for BCC-sampled volume data have been reviewed and com-



(a) Linear box spline. (b) Cubic box spline. (c) Prefiltered Gaussian.

Figure 9: *Reconstruction of the Marschner-Lobb test signal from* $32 \times 32 \times 32 \times 2$ *BCC samples.*



(a) Linear box spline. (b) Cubic box spline. (c) Prefiltered Gaussian.

Figure 10: *Angular errors of the estimated normals and the exact normals. Angular error of* 30 *degrees is mapped to white.*

pared. As it has been demonstrated, the most important drawback of the BCC grid is its non-separability, which makes it difficult to extend the well-known 1D reconstruction filters, like the linear tent filter or popular cubic filters by a simple separable extension.

In order to define exact interpolating filters for the BCC grid, two strategies can be followed. One possibility is to derive an explicit kernel which is easy to evaluate and fulfills the interpolation constraint, like the linear box spline. Another alternative is to implicitly construct an interpolating filter from an approximating filter by adapting the conception of generalized interpolation to the BCC grid.

Both approaches have disadvantages. For example, higher-order interpolating kernels are rather difficult to explicitly define for the BCC grid, therefore such a filter has not been explored yet. On the other hand, generalized interpolation requires the frequency-domain prefiltering of the original samples.

According to our experiments, currently the best results can be achieved by using our prefiltered Gaussian reconstruction technique. Although PGR is an order of magnitude faster than the cubic box-spline reconstruction, it is still too expensive computationally for practical applications.

Although the superiority of the BCC grid over the CC grid has already been demonstrated in several papers, CC-sampled volumes can still be more efficiently rendered by using hardware-accelerated rendering techniques, like 3D texture mapping. Therefore, until high-quality reconstruction methods proposed for the BCC grid are adapted to the recent programmable graphics cards, the traditional CC-sampled volume representation will probably remain more popular.

Acknowledgements

This work has been supported by the Postdoctoral Fellowship Program of the Hungarian Ministry of Education.

References

1. T. Blu, P. Thévenaz, and M. Unser. Generalized interpolation: Higher quality at no additional cost. In *Pro-*

ceedings of IEEE International Conference on Image Processing, pages 667–671, 1999. 4

2. J. H. Conway, N. J. A. Sloane, and E. Bannai. *Sphere-packings, lattices, and groups*. Springer-Verlag New York, Inc., 1987. 1
3. B. Csébfalvi. Prefiltered Gaussian reconstruction for high-quality rendering of volumetric data sampled on a body-centered cubic grid. In *Proceedings of IEEE Visualization*, pages 311–318, 2005. 4, 5
4. A. Entezari, R. Dyer, and T. Möller. Linear and cubic box splines for the body centered cubic lattice. In *Proceedings of IEEE Visualization*, pages 11–18, 2004. 2, 3, 4
5. T. C. Hales. Cannonballs and honeycombs. *AMS*, 47(4):440–449, 1998. 1
6. Y. Jang, M. Weiler, M. Hopf, J. Huang, D. S. Ebert, K. P. Gaither, and T. Ertl. Interactively visualizing procedurally encoded scalar fields. In *Proceedings of Joint EUROGRAPHICS-IEEE TCVG Symposium on Visualization*, pages 35–44, 2004. 4
7. A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988. 6
8. S. Marschner and R. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of IEEE Visualization*, pages 100–107, 1994. 2, 3
9. D. Mitchell and A. Netravali. Reconstruction filters in computer graphics. In *Proceedings of SIGGRAPH*, pages 221–228, 1988. 2
10. T. Möller, R. Machiraju, K. Mueller, and R. Yagel. A comparison of normal estimation schemes. In *Proceedings of IEEE Visualization*, pages 19–26, 1997. 2
11. T. Möller, K. Mueller, Y. Kurzion, R. Machiraju, and R. Yagel. Design of accurate and smooth filters for function and derivative reconstruction. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 143–151, 1998. 2
12. B. S. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings of Shape Modeling International*, pages 89–98, 2001. 4
13. A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Inc., Englewood Cliffs, 2nd edition, 1989. 1
14. D. P. Petersen and D. Middleton. Sampling and reconstruction of wave-number-limited functions in n-dimensional euclidean spaces. *Information and Control*, 5(4):279–323, 1962. 1
15. N. J. A. Sloane. The sphere packing problem. In *Proceedings of International Congress of Mathematicians*, pages 387–396, 1998. 1
16. T. Theußl, H. Hauser, and M. E. Gröller. Mastering windows: Improving reconstruction. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 101–108, 2000. 2
17. T. Theußl, O. Mattausch, T. Möller, and M. E. Gröller. Reconstruction schemes for high quality raycasting of the body-centered cubic grid. *TR-186-2-02-11, Institute of Computer Graphics and Algorithms, Vienna University of Technology*, 2002. 3
18. T. Theußl, T. Möller, and M. E. Gröller. Optimal regular volume sampling. In *Proceedings of IEEE Visualization*, pages 91–98, 2001. 2, 3
19. G. Turk and J. F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH*, pages 335–342, 1999. 4
20. D. Van De Ville, T. Blu, M. Unser, W. Philips, I. Lemahieu, and R. Van de Walle. Hex-splines: A novel spline family for hexagonal lattices. *IEEE Transactions on Image Processing*, 13(6):758–772, 2004. 3

Computer Aided Diagnosis based on Second Derivatives of Volume Data

József Koloszár, László Szirmay-Kalos, Zsolt Tarján, and Dávid Jocha

Department of Control Engineering and Information Technology, Technical University of Budapest
Budapest, Magyar Tudósok krt. 2., H-1117, HUNGARY
Email: szirmay@iit.bme.hu

Abstract

This paper proposes a robust algorithm to detect colon tumors using the second derivatives of volume data. We obtain the eigenvectors of the Hessian matrix, which represent the second order derivatives of a vector variate scalar valued function. Based on the sign and scale of the eigenvalues blobby tumors can be selected on a given scale. In order to reduce noise and set the scale of the analysis, Gaussian filtering is applied together with the computation of the derivatives. Our final filter shows how blobby a neighborhood is on several scales. The identified regions can classify CT data to contain tumors and also to control the navigation of the virtual camera of the medical doctor.

Keywords: Volume visualization, Second derivatives, Computer aided diagnosis

1. Introduction

Medical devices, such as CT and MRI equipment, can provide density values of an object at regularly placed locations. The density values are stored in voxel arrays. Voxel arrays are visualized by direct or indirect algorithms, allowing the medical doctor to examine the measured data[2, 4, 1]. Based on the presented images, the physician identifies tumors and decides on the subsequent actions. This process can be speeded up and made easier if the data is preprocessed and the attention of the physician is drawn to those critical points, which are primary candidates for being tumors. This helping process is called computer aided diagnosis.

Based on the voxel data computer aided diagnosis identifies certain locations or neighborhoods where the critical artifacts show up. This process can also be imagined as filtering. The voxel data is taken as a discrete representation of the originally continuous density field. At each possible location this density field is combined with a recognition filter, which generates one or several values describing the local neighborhood. Then these values are used to make the final yes or no decision. On the other hand, the selected local neighborhood may also affect the path of the virtual camera through which the medical doctor sees the internal organs.

In order to design such an algorithm, the recognition filter should be carefully selected to clearly separate those cases, which might be identified as tumors. On the other hand, the decision based on the filtered values should also be as accurate as possible. A promising way is to mimic how an experienced medical doctor locates the shapes that can be tumors, and how he makes the final conclusion from the shape characteristics. When the process of the diagnosis is modeled, we should be specific and reflect all important factors while ignoring unimportant ones. The inclusion of important factors is necessary to make the method robust and reliable, while ignoring unimportant factors is essential to establish a fast and real-time method.

Automatic diagnosis may result in two types of false decisions. The decision is false positive if it identifies a shape as a tumor, which is not. On the other hand, the decision is false negative if the process misses a tumor. In medical diagnosis the reduction of false negative cases is essential.

In this paper we present an algorithm to detect potential tumors inside a colon. If this investigation were made by a medical doctor, he would search for blobby structures on the colon walls, and examining the size and the form of these

blobs he could decide whether or not the particular blob is a tumor. In order to mimic this procedure we design a recognition filter that is particularly sensitive to blobs of given size and provides numeric data representing the form of the blob. The recognition filter is based on the second order derivatives of the reconstructed density function.

2. Previous work

While shape analysis and shape detection have received significant attention in 2D image processing, they are relatively new in 3D volume analysis. Two dimensional shape detection methods usually measure certain features that can characterize the shapes of interest and make decisions based on the measured values. These features are required to be translation and rotation invariant since translation and rotation do not alter shapes. The practically useful features include, among others, perimeter, area, circularity, spatial moments, main inertia, derivatives, etc.

Having computed the feature values, they are grouped in a vector and this vector is compared with the properties of the shape or shapes to be identified. If the distance from the feature vector of the shape to be identified is small, then we can report that the shape is found. Setting the distance threshold, we can make a compromise between false negative and positive decisions. If there are several candidate shapes, nearest neighbor search can be applied to find the closest candidate. When such algorithm is designed, the critical problems is the definition of the feature vector, especially when the class of target shapes is not well defined. This is always the case in medical diagnosis since different tumors are not exactly similar from geometric point of view.

In order to cope with this problem, we can avoid information extraction and suppose the image in a smaller neighborhood to be the feature vector. In the case of 5×5 pixel neighborhoods of a gray-scale image, the dimension of the feature vector is 25. To select from such high-dimensional vectors, *neural nets* can be used, which are trained by real-world examples (e.g. by images of real tumors). Unfortunately, the image itself is not rotation invariant, thus the neural net should be trained with different rotations of the images. Neural nets are very elegant and require no a priori knowledge of the features to be identified, but their practical application also has disadvantages. To train a net to identify high dimensional vectors, very many training cases are needed, which are not usually available in medical practice. The other problem is that probability of false positive and negative decisions is not easy to control.

The best results can be obtained by the combination of feature extraction and neural net based identification. A relatively low dimensional feature vector is computed, which contains rotation and translation invariant measures. Then instead of distance comparison, the neural net is fed by this feature vector to make the final decision.

2.1. Local measures

Those features that are based on the Taylor series expansion of the volume data are called local measures since the Taylor series is a good local approximation of a function. For example, the second-degree approximation of a function $f(x)$ around x_0 is:

$$f(x) \approx f(x_0) + \left.\frac{df}{dx}\right|_{x_0} \cdot (x - x_0) + \frac{1}{2} \cdot \left.\frac{d^2 f}{dx^2}\right|_{x_0} \cdot (x - x_0)^2.$$

This approximation depends on function value $f(x_0)$, derivative df/dx and second derivative d^f/dx^2. The first derivative is the slope of the function. The second derivative is the speed describing how fast the slope of the function is changing, thus is a measure of the function curvature (figure 1). In fact, the absolute value of the second order derivative is inversely proportional to the radius of a tangent circle. The sign of the second derivative also shows whether the high curvature area is a *hill* or a *valley*.

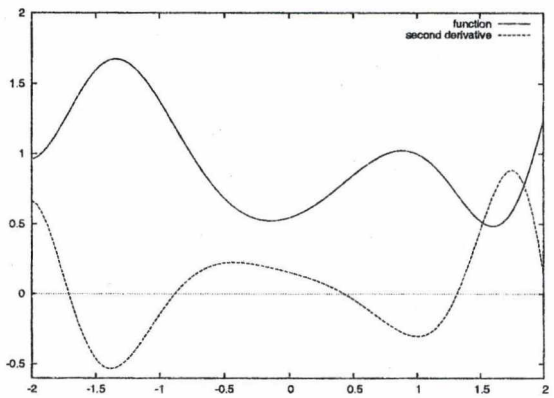


Figure 1: *A function and its second-order derivative. Note that the absolute value of the second derivative is inversely proportional to the curvature radius. The sign of the second derivative makes a distinction between hills and valleys. The second derivative is scaled in the figure to improve visual comprehension.*

Sato et al have extended this idea to the analysis of 3D density data [5]. An excellent review and an application to data compression can be found in [3]. Suppose that the density value of the examined object at point $\vec{r} = (x, y, z)$ can be described by function $g(\vec{r})$. The second-order approximation of the density function around $\vec{r}_0$ is the following:

$$\tilde{g}(\vec{r}) = g(\vec{r}_0) + (\vec{r} - \vec{r}_0)^T \cdot (\nabla g)(\vec{r}_0) +$$

$$\frac{1}{2} \cdot (\vec{r} - \vec{r}_0)^T \cdot (\nabla^2 g)(\vec{r}_0) \cdot (\vec{r} - \vec{r}_0),$$

where

$$\nabla g = (\partial g/\partial x, \partial g/\partial y, \partial g/\partial z)$$

is the gradient vector and

$$\nabla^2 g = \begin{bmatrix} \partial^2 g/\partial x^2, & \partial^2 g/\partial x \partial y, & \partial^2 g/\partial x \partial z \\ \partial^2 g/\partial y \partial x, & \partial^2 g/\partial y^2, & \partial^2 g/\partial y \partial z \\ \partial^2 g/\partial z \partial x, & \partial^2 g/\partial z \partial y, & \partial^2 g/\partial z^2 \end{bmatrix} \quad (1)$$

is the Hessian matrix. The Hessian matrix allows to express the second derivative in an arbitrary direction represented by unit vector $\vec{v}$

$$\frac{d^2 g}{d\vec{v}^2} = \vec{v}^T \cdot H \cdot \vec{v}.$$

Unfortunately, the gradient and the Hessian matrix are not invariant to rotations, and they strongly depend on the axes of the coordinate system. Such dependence can be eliminated for the gradient if its absolute value is considered since $|\nabla g|$ becomes independent of the orientation of the axes.

In order to make the second derivative measures also rotation independent, we have to select those directions, which correspond to the maximal and minimal second derivatives. As can be proven [3], these directions are the eigenvectors of the Hessian matrix, where the corresponding second derivatives are the eigenvalues. Since the Hessian matrix is symmetric, the eigenvalues are real.

The eigenvalues can be obtained by solving equation

$$det(\lambda I - H) = 0,$$

where I is the unit matrix. Since $det(\lambda I - H)$ is a third-degree polynomial, the roots can be found analytically in an efficient way (see Appendix).

Let us denote the three eigenvalues by $\lambda_1, \lambda_2, \lambda_3$, and assume that $\lambda_1 \geq \lambda_2 \geq \lambda_3$. These eigenvalues express the minimum and maximum of the second derivatives, or the maximum and the minimum of the curvature radii. If the three eigenvalues have similar sign, then the function is locally approximated by an ellipsoid. If the sign is negative, then we have a hill, otherwise a valley. The minimum and maximum curvatures of an ellipsoid are proportional to the lengths of the main axes. A small $|\lambda|$ value corresponds to a long axis, while a large $|\lambda|$ to a short axis. If all axes are short (i.e. the absolute values of all eigenvalues are large), then the ellipsoid is a small blob (figure 2). If all axes are long, then the ellipsoid cannot be recognized, and is said to be homogeneous on the local level. If one axis of an ellipsoid is much longer than the other two (i.e. the absolute value of one eigenvalue is much smaller than the other two), then the ellipsoid has a tubular shape. If two axes are much longer than the third one, then the ellipsoid looks like a sheet.

Table 1 summarizes the possible conclusions that can be drawn by inspecting the three eigenvalues.

Summarizing second order local measures include the density value $g(\vec{r}_0)$, the gradient magnitude $|(\nabla g)(\vec{r}_0)|$, and the eigenvalues of the Hessian matrix $\lambda_1, \lambda_2, \lambda_3$. All of them are translation and rotation invariant, thus are primary candidates for features.

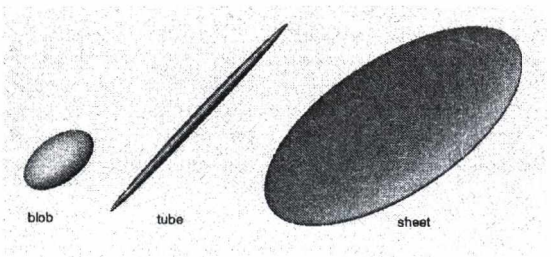


Figure 2: *Local quadratic approximation. Based on the relative absolute values of the eigenvalues, blob, tubular and sheet-like structures can be identified.*

	classification
$\lambda_3 \approx \lambda_2 \approx \lambda_1 \ll 0$	blob-like hill
$\lambda_3 \approx \lambda_2 \ll \lambda_1 \approx 0$	tubular-like hill
$\lambda_3 \ll \lambda_2 \approx \lambda_1 \approx 0$	sheet-like hill
$0 \approx \lambda_3 \approx \lambda_2 \approx \lambda_1$	homogeneous area
$0 \approx \lambda_3 \approx \lambda_2 \ll \lambda_1$	sheet-like valley
$0 \approx \lambda_3 \ll \lambda_2 \approx \lambda_1$	tubular-like valley
$0 \ll \lambda_3 \approx \lambda_2 \approx \lambda_1$	blob-like valley

Table 1: *Classification of the blob-, tubular-, and sheet-like structures*

2.2. Computation of the derivatives of sampled datasets

The measuring process takes discrete samples at regular points $\vec{p} \in P$, thus the function stored in the voxel array is:

$$g_s(\vec{r}) = g(\vec{r}) \cdot \sum_{\vec{p} \in P} \delta(\vec{r} - \vec{p})$$

where δ is the Dirac-delta function. From the sampled signal, the continuous signal can be reconstructed by convolving $g_s(\vec{r})$ with the impulse response of an ideal low-pass filter of limiting frequency equal to the upper band limit of the original signal (the sampling theorem requires the band limit to be lower than the half of the sampling frequency).

The impulse response of the ideal low-pass filter is $sin(\pi x)/\pi x$, which is rather difficult to convolve with, because of its oscillating shape and infinite support. Due to computational issues, the ideal low-pass filter is approximated, for example, by the Gaussian-filter, thus the reconstructed signal is:

$$g(\vec{r}) \approx (k \otimes g_s)(\vec{r})$$

where k is the impulse response (kernel) of the 3D Gaussian-

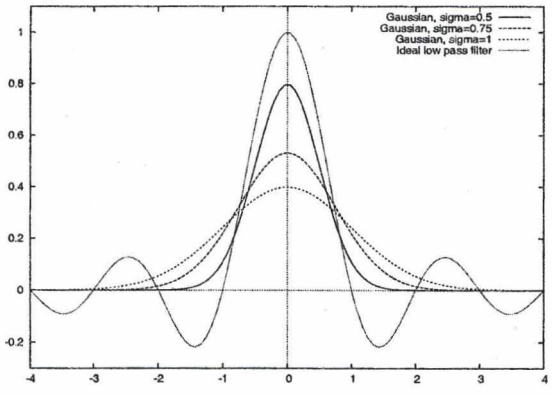


Figure 3: *The impulse responses of the Gaussian filters with different standard deviations and of the ideal low pass filter* $(sin(\pi x)/\pi x)$

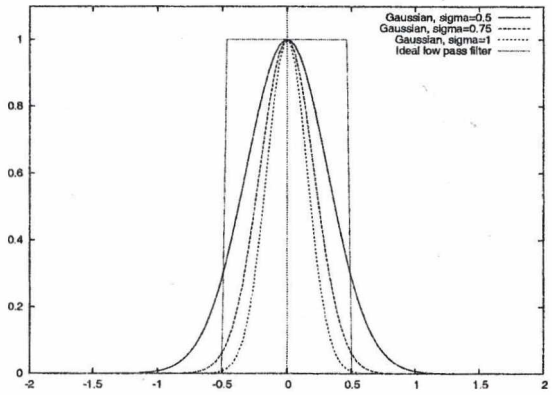


Figure 4: *Frequency domain characteristics of the Gaussian filters with different standard deviations and of the ideal low pass filter*

filter having standard deviation σ:

$$k(\vec{r},\sigma) = \frac{e^{-|\vec{r}|^2/2\sigma^2}}{(\sigma\sqrt{2\pi})^3} = \frac{e^{-x^2/2\sigma^2}}{\sigma\sqrt{2\pi}} \cdot \frac{e^{-y^2/2\sigma^2}}{\sigma\sqrt{2\pi}} \cdot \frac{e^{-z^2/2\sigma^2}}{\sigma\sqrt{2\pi}}.$$

Standard deviation σ should be set to make the filter efficiently cut off frequencies higher than the upper band limit of the measured signal. If the upper band limit is about half of the sampling frequency, which is usually 1 (the samples are at unit distance), then $\sigma \in [0.5, 1]$ is usually satisfactory (figure 4). If σ is smaller than 0.5, then high-frequency aliasing occurs. On the other hand, if σ is bigger than 1, the higher frequency components of the signal are eliminated. Large σ values represent filters that are combinations of a low-pass filter and the reconstruction filter.

Convolution commutes with differentiation, thus the derivatives of the reconstructed signal can be obtained by convolving with the derivative of the filter kernel. For example, the derivative of $g(\vec{r})$ according to x is (the derivatives according to y and z are similar):

$$\frac{\partial g(\vec{r})}{\partial x} \approx \frac{\partial (k \otimes g_s)(\vec{r})}{\partial x} = \frac{\partial k(\vec{r})}{\partial x} \otimes g_s.$$

Thus to compute all second partial derivatives, the partial derivations of the filter kernel $(\partial^2 k/\partial x^2, \partial^2 k/\partial x \partial y, \partial^2 k/\partial x \partial z, \ldots)$ should be pre-computed, and the sampled data is convolved with the required derivative kernels. The Gaussian-kernel as well as its derivatives are separable, which means that the three-variate function can be expressed as the product of three one-variate functions parameterized by x, y and z, respectively. Using x as a parameter, the one-variate function, and its first and second derivatives are (figure 5):

$$k(x,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-x^2/2\sigma^2},$$

$$\frac{dk(x,\sigma)}{dx} = \frac{-x}{\sigma^3\sqrt{2\pi}} \cdot e^{-x^2/2\sigma^2},$$

$$\frac{d^2k(x,\sigma)}{dx^2} = \frac{x^2-\sigma^2}{\sigma^5\sqrt{2\pi}} \cdot e^{-x^2/2\sigma^2},$$

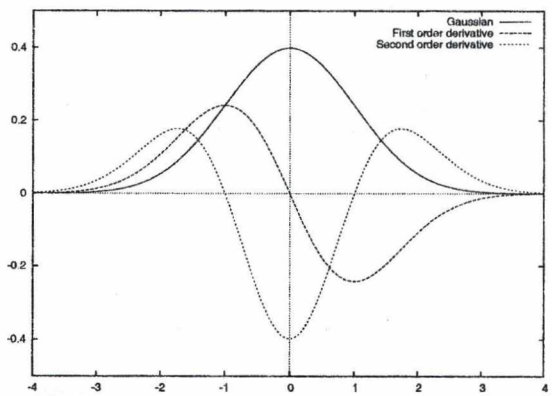


Figure 5: *Gaussian-kernel and its first and second order derivatives* ($\sigma = 1$)

3. Design of the tumor detection filter

Colon tumors are blob-like hill features in a given size range. The proposed detection algorithm looks for cases corresponding the first row of table 1, but it also has to take into

account the density of the tumor, the shapness of its boundary (gradient), and the size of the blob. In the following section the size control problem is discussed.

The Gaussian-kernel is not a perfect reconstruction kernel, thus the derivatives are just approximations. For example, if the sampled values are in $[0,1]$, then the maximal values of the first and second derivatives that can be reconstructed is approximately

$$\max\left|\delta(\vec{x}) \otimes \frac{dk(x,\sigma)}{dx}\right| = \frac{1}{\sigma^3\sqrt{2\pi}}.$$

$$\max\left|\delta(\vec{x}) \otimes \frac{d^2k(x,\sigma)}{dx^2}\right| = \frac{1}{\sigma^3\sqrt{2\pi}}.$$

Note that standard deviation σ can be used to control the possible range of the derivatives. If we are interested in features where the second derivatives are in a prescribed range, then σ should be set to make the maximum derivatives close to the top of the prescribed range. In this way we can maximize the sensitivity of our filter. This can mean that σ is greater than the value needed for the optimal reconstruction, thus all those features that are smaller than the interesting ones are eliminated.

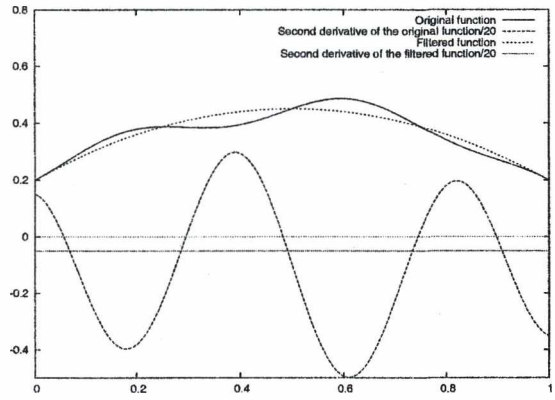


Figure 6: *Low pass filtering makes larger features detectable. Note that the second derivatives are scaled in the figure to improve visual comprehension.*

Figure 6 demonstrates why the low pass filtering is necessary to allow the detection of larger features. The original function contains a single hill, which is modulated by a high-frequency wave. The curvature values fluctuate according to the high-frequency wave, not permitting to detect the average curvature that corresponds to the hill. However, when low-pass filtering eliminates the high frequency wave, the curvature clearly identifies the original hill. From this observation we can conclude that a crucial part of feature recognition is the definition of the appropriate scale or scales if we examine different sizes simultaneously. Sato et al approached this problem assuming that the signal is also a Gaussian function. However, in tumor diagnosis, this assumption is not necessarily true, since tumors have a well recognizable boundary.

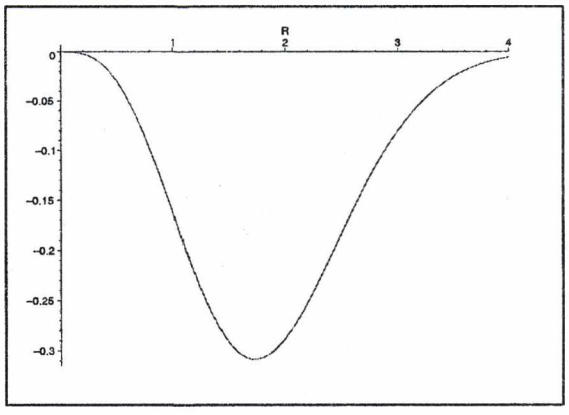


Figure 7: *The eigenvalue at a center of a sphere of radius $r = R\sigma$ computed by a Gaussian filter of standard deviation σ*

Thus to present a more accurate analysis, let us suppose that the volume data contains a sphere of radius r. The density value is 1 inside the sphere and zero outside. Let us also assume that the sampling frequency is high enough to make the discrete sums close to the continuous integrals. The first derivatives at the center of the sphere is approximated by the convolution of the density function and the filter kernel $\frac{dk(x,\sigma)}{dx} \cdot k(y,\sigma) \cdot k(z,\sigma)$, which can be expressed by the following integrals

$$\frac{dg}{dx} = \int\limits_{x^2+y^2+z^2 \le r^2} \frac{-x}{\sigma^5(2\pi)^{3/2}} \cdot e^{-(x^2+y^2+z^2)/2\sigma^2}\, dxdydz.$$

Applying the $X = x/\sigma, Y = y/\sigma, Z = z/\sigma, R = r/\sigma$ substitutions, we obtain:

$$\frac{dg}{dx} = \frac{1}{\sigma} \cdot \int\limits_{X^2+Y^2+Z^2 \le R^2} \frac{-X}{(2\pi)^{3/2}} \cdot e^{-(X^2+Y^2+Z^2)/2}\, dXdYdZ =$$

$$-\frac{1}{\sigma^2} \cdot \sqrt{\frac{2}{9\pi}} \cdot e^{-R^2/2} \cdot R^3$$

The first and second derivatives at the center of the sphere are approximated by the convolution of the density function and the filter kernel $\frac{dk(x,\sigma)}{dx} \cdot k(y,\sigma) \cdot k(z,\sigma)$ and $\frac{d^2k(x,\sigma)}{dx^2} \cdot k(y,\sigma) \cdot k(z,\sigma)$, respectively, which can be expressed by the following integrals

$$\lambda(r) = \int\limits_{x^2+y^2+z^2 \le r^2} \frac{x^2-\sigma^2}{\sigma^7(2\pi)^{3/2}} \cdot e^{-(x^2+y^2+z^2)/2\sigma^2}\, dxdydz.$$

Applying the $X = x/\sigma, Y = y/\sigma, Z = z/\sigma, R = r/\sigma$ substitutions, we obtain:

$$\lambda(R) = \frac{1}{\sigma^2} \cdot \int_{X^2+Y^2+Z^2 \leq R^2} \frac{X^2-1}{(2\pi)^{3/2}} \cdot e^{-(X^2+Y^2+Z^2)/2}\, dXdYdZ =$$

$$-\frac{1}{\sigma^2} \cdot \sqrt{\frac{2}{9\pi}} \cdot e^{-R^2/2} \cdot R^3$$

Note that product $\lambda\sigma^2$ is a function of $R = r/\sigma$, which is depicted in figure 7. This function has a minimum at $R = r/\sigma = \sqrt{3}$, which corresponds to the largest curvature the filter can appropriately reconstruct. If $r/\sigma > \sqrt{3}$, then the curvature of the sphere decreases, and the reconstructed value follow this change. However, when $r/\sigma < \sqrt{3}$, the curvature of the sphere increases, but the absolute value of the reconstructed eigenvalue decreases. Here, the Gaussian acts as a low pass filter, and instead of computing the second derivative, it gradually eliminates the sphere itself.

Suppose that the smallest feature that should be detected can be approximated by a sphere of radius r. Figure 7 tells us to set the standard deviation of the Gaussian filter to $\sigma \approx r/1.74$. Such setting allows us to detect features of radius $[r, 3r]$ while eliminating features of radius smaller than r.

There is one final issue of the Gaussian-kernel, which should be addressed here. The Gaussian-kernel has infinite support, which is impossible to handle during convolutions. To cope with this problem, the Gaussian-kernel is truncated, which means that it is assumed to be zero, where the original kernel is small. The size of the support domain depends on the standard deviation σ. For example, if $\sigma = 0.5$, then the kernel values at the sampling positions $0, 1, 2$ are $0.797, 0.108, 0.00027$, respectively. It means that we can suppose that the kernel is zero outside region $[-2, 2]$. Larger σ values, however, require filters with greater support. If truncation happens at value $t = x/\sigma$, then the integral of the cut off part is $1 - \Phi(t)$, where Φ is the Gaussian distribution function. As a rule of thumb we can truncate at $t = 4$, i.e. $x = 4\sigma$ since in this case the integral of the cut off part is less than 0.00003.

The truncation of the kernel might have the undesirable effect that the reconstructed second derivatives of constant functions will not be zero. This is due to the fact that the integral of the second derivative of the truncated kernel is not necessarily zero. Note that this can never happen when the first derivative is computed since the first derivative is anti-symmetric ($dk(-x,\sigma)/dx = -dk(x,\sigma)/dx$) To solve the problem of second derivatives, σ should be fine tuned in a way that the sampled second derivatives of the truncated kernel sums up to zero.

4. The detection algorithm

In the preprocessing phase the standard deviation σ is approximated from the expected size of the tumors relative to the sampling resolution of the device. If it turns out that σ is too small (smaller than 0.5), then the criteria of the sampling theorem cannot be met, thus such features cannot be recognized from the given data set. If σ is not too small, then the corresponding window size is determined, and σ is tuned to give zero second derivatives for homogeneous regions.

The diagnosis process starts with identifying the voxels corresponding to the colon boundary. This way the feature detection can be restricted to the neighborhood of the color walls, since colon tumors can show up only at these regions. In the selected neighborhoods, the shape detection filter is run at each voxel. Tumors correspond to bright blob-like structures, thus the maximum eigenvalue is taken and is compared to a limiting value. If the maximum eigenvalue is less than the threshold, than this voxel is marked. During the visualization step the neighborhood of the marked pixels are colored to ease visual detection.

In order to test the algorithm we made measurements on a phantom (figures 8, 9). The sampling resolution was 2mm. The tumor identification process searched the 20mm neighborhood of the colon wall. The expected tumors are of 1cm size, thus σ was set to 6, which corresponds to $25 \times 25 \times 25$ filter. Finally, the 10mm neighborhood of the marked voxels were colored to red.

The selected regions can also control the virtual camera of the user. Suppose that each selected region is assigned a weight that is inversely proportional to the square distance of the camera and the center of this region. Let us compute a vector from the camera toward each selected region and multiply the vectors by the weight. This operation will result in a vector that always points toward the closest selected region. Adding this vector to the derivative of the path, we can obtain the heading of the camera. Note that at neighborhoods where there are no selected regions, the camera would head at where it is going to, but would turn to the tumor if it is close.

5. Conclusions

This paper presented a robust filter to identify tumors in a CT data. The filters are based on second derivatives computed on several scales. The regions selected by this filter are highlighted and the camera heading is also changed accordingly.

Thus the proposed method can be used to quickly process a patient's examination data and decide whether it contains blobby features that require further investigation. When the doctor flies through the CT data, the same information forces him to carefully look at the selected blobby features.

6. Acknowledgements

This work has been supported by the National Scientific Research Fund (OTKA ref. No.: T042735), and IKTA ref. No.: 00159/2002.

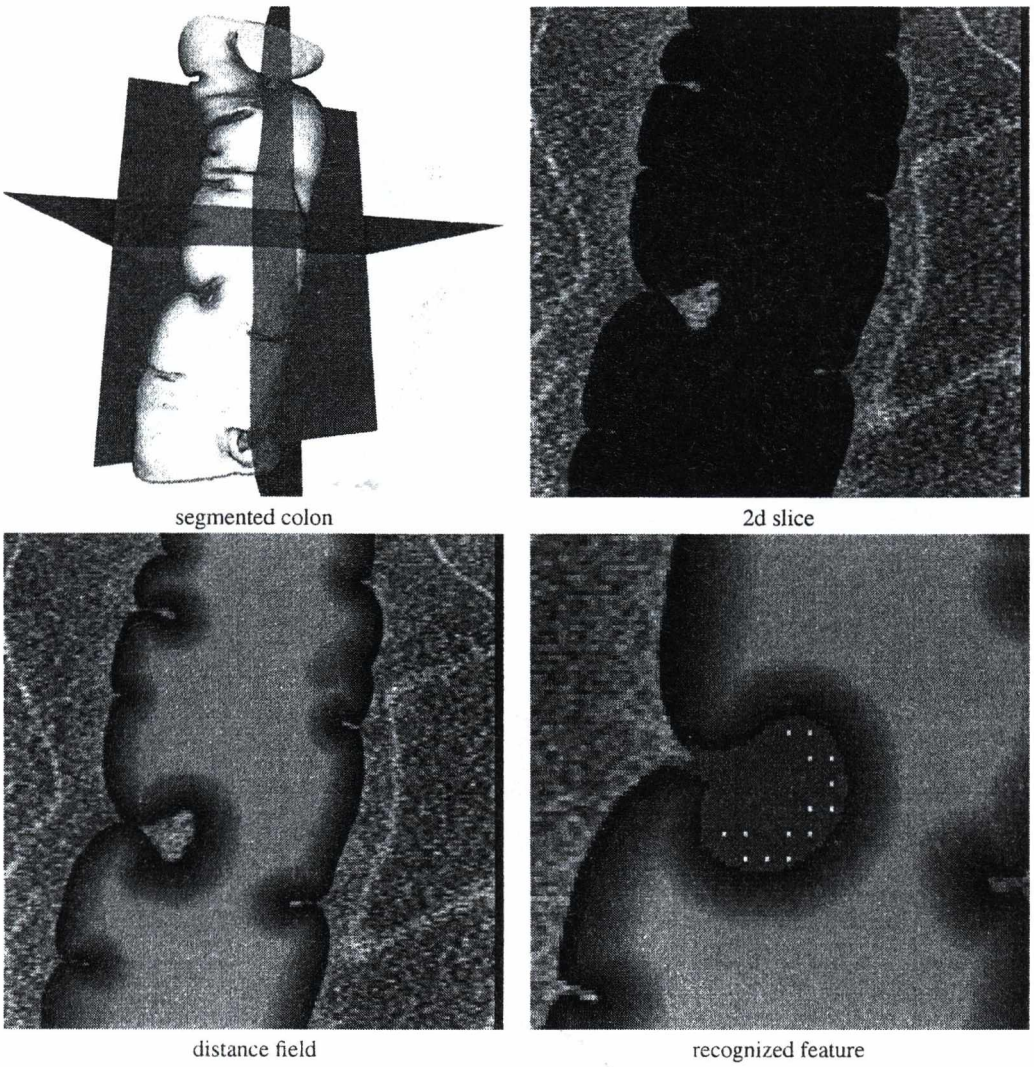


Figure 8: *Steps of the diagnosis process: Segmentation, colon wall identification, distance field generation, and shape recognition. The yellow points show those voxel centers where the filter reported blobs. The red area is the colored neighborhood.*

References

1. B. Csébfalvi. *Interactive Volume-Rendering Techniques for Medical Data Visualization.* PhD thesis, Technische Universität Wien, Institut für Computergraphik und Algorithmen, 2001. http://www.cg.tuwien.ac.at/research/theses/. 1

2. A.D. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, 22(4), 1988. 1

3. J. Hladuvka. *Derivatives and Eigensystems for Volume-Data Analysis and Visualization.* PhD thesis, Institute of Computer Graphics, Vienna University of Technology, Vienna, Austria, 2002. 2, 3

4. M. Levoy. Efficient ray tracing of volume data. *ATG*, 9(3):245–261, 1990. 1

5. Y. Sato, C. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S. Tamura, and R. Kikinis. Tissue classification based on 3D local intensity structures for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):160–180, 2000. 2

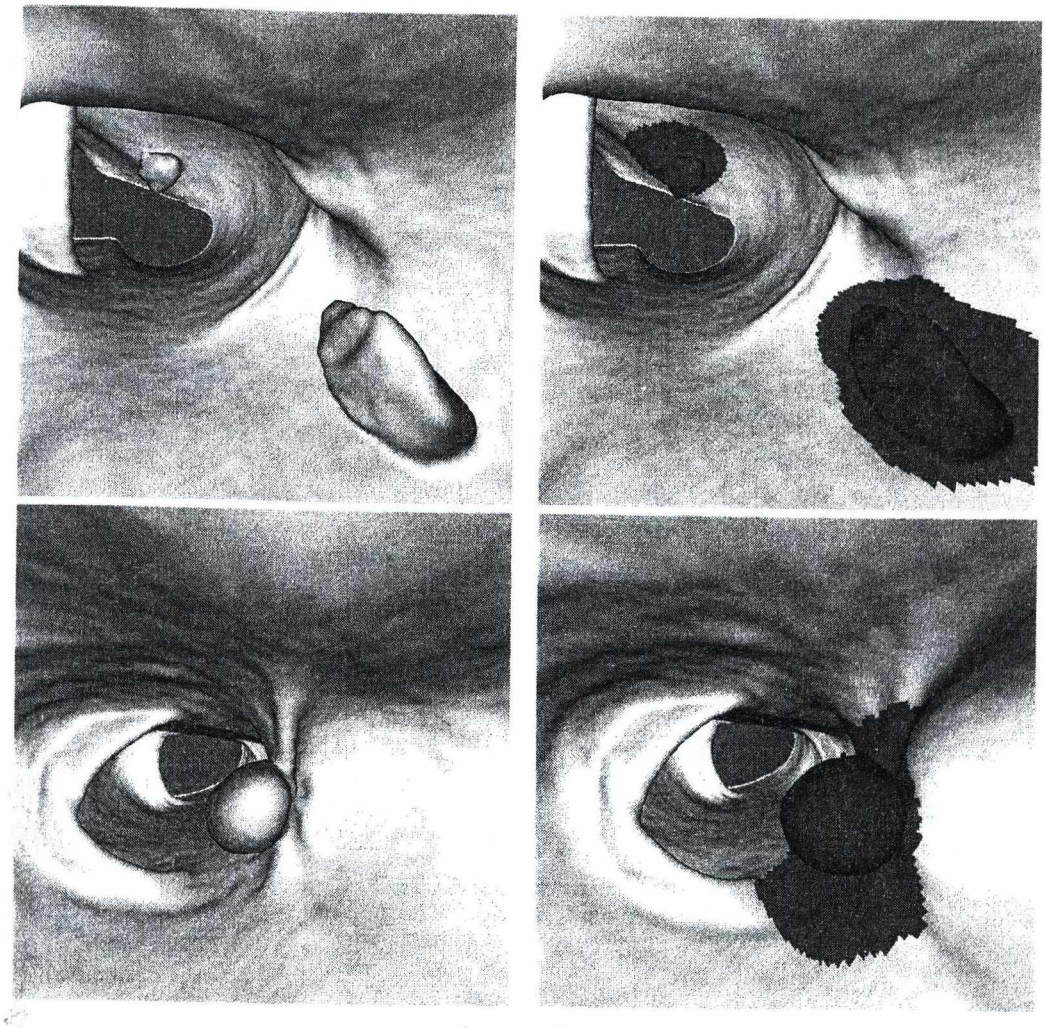


Figure 9: *3D views of the diagnosis process.*

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Conservative rasterization of texture atlases

László Szécsi

szecsi@iit.bme.hu
Department of Control Engineering and Information Technology,
Budapest University of Technology and Economics,
Budapest, Hungary

Abstract

Render-to-texture has become a very basic operation in graphics hardware programming, allowing for computations to be decomposed into consequent passes. Most prominently, all kinds of lighting algorithms, including stochastic iteration, photon maps, caustics, or light animation can be transposed into the two-dimensional surface domain by using texture atlases. Computations are carried out using a render-to-atlas operation, and data is later accessed by texturing. However, the rasterization algorithm supported by the hardware does not necessarily trigger rendering to every texel that can be addressed by surface texture coordinates. This causes artifacts near texturing seams. In order to eliminate them, we have to simulate conservative overestimated rasterization of triangle charts when rendering to the texture atlas.

In this paper, we propose a method that processes the original model geometry and constructs a minimal set of line segments that will rasterize to complete the texture atlas. There is no increase in the complexity of the geometry, and the overlying computation algorithm does not have to be modified.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Bitmap and framebuffer operations

1. Introduction

One of today's ongoing trends is that global illumination algorithms are appearing in all kinds of everyday applications to an increasing extent, making use of the capabilities of the graphics hardware. Global illumination, in essence, is the uncompromising solution of the rendering equation, where the inherently recursive nature of light transport is respected. Research has shown that taking the solution to interactive speeds must rely on data reuse, where the data structure can range from finite elements[7] through photon maps[4] to ray caches[8]. On the other hand, graphics hardware was engineered to effectively support texture mapping. Both the opportunity and the challenge of real-time global illumination are to store reusable data in textures. Along with all stages of the hardware pipeline, texturing is also continuously growing more customizable. Most importantly, images can be rendered to textures, and with programmable vertex and pixel shaders, this allows computations generating the data textures also to be implemented on the GPU.

1.1. Texture atlases

The most straightforward and often unavoidable means to pre-compute and store illumination data is the usage of texture atlases. Whenever we need to know the illumination of surface elements without fixing a specific viewing direction, we may carry out computations at sample points of the surface: these will be the texels of our texture atlas. In case of a fixed light position, we may also pre-compute a shadow map to speed up lighting, but it may not be as effective as the using the atlas. Furthermore, if we wish to pre-compute interreflections independent of both light position and viewing direction, the use of a texture atlas is imperative. Similarly, if we have to blend impostors on surfaces, like in the caustic generation method using approximate ray tracing[6], it also has to be performed in the texture atlas domain. To name a further example, photon map gathering[2] was shown to be effectively performed by storing photon hits in a texture atlas.

An important feature of textures is the possibility of filtering. While a texture atlas may not sample surface points very densely, bilinear interpolation of pre-computed values

comes practically for free. Still, quality will depend on the resolution, and poor sampling may appear as texturing artifacts. However, similar effects appear for any algorithm that reuses stored data.

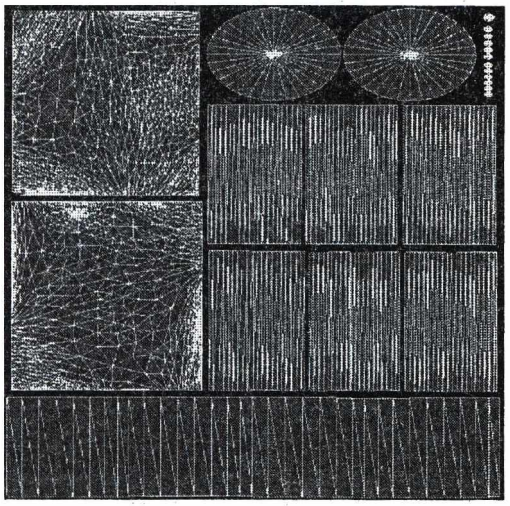


Figure 1: *An atlas texture with triangle edges. The mesh is decomposed to triangle charts.*

A texture atlas has to ensure a unique mapping of texture coordinates to surface points. While some of the texture space may be empty, every surface point has to have unique texture coordinates. It is also desirable to avoid discontinuities. Generally, the texture atlas mapping for a triangular mesh will contain a number of flattened charts of adjacent triangles, separated by a few texels wide empty margin (Figure 1). The contours of these charts are the seams of the mapping: that is where adjacent surface points are mapped to distant coordinates. Seams are usually unavoidable without extremely distorting triangles. There are several approaches to generate atlas mappings with as few seams as possible or to find those seams that are visually least disturbing [5]. While these approaches are very useful to reduce artifacts, it cannot always be assumed that the virtual world model an algorithm has to work on is practically seamlessly textured.

1.2. Render to texture

Using a texture atlas involves two basic steps. The first one is to render the triangles of the model onto the texture atlas, to invoke vertex and pixel shaders that compute illumination data and output it to atlas texels. This is the render-to-atlas operation, which is simply performed by issuing a draw call with the original vertex and index buffers of the mesh. In the vertex shader, however, the vertices are displaced to their respective texture atlas coordinates, transforming the triangles to texture space. Theoretically, the pixel shaders should be invoked for all valuable texels of the atlas, filling it with the desired data. Having computed the atlas, it can be accessed from the pixel shaders of later passes, typically addressed by texture coordinates interpolated between vertices. The problem this paper addresses is how to ensure that these texture reads will always address a texel for with the render-to-atlas has been performed and thus contains valid information.

1.3. Triangle rasterization

How to convert render primitives (points, line segments and polygons) to screen pixels effectively was exhaustively researched by pioneers of computer graphics[1]. Variant algorithms to render smooth anti-aliased edges, or to include pixels only touching the area of primitives were also invented. However, only the most effective and clear-cut variant could make it to current incredibly fine-tuned graphics hardware. The issue of anti-aliasing is more effectively solved by the multi-sampling post-filtering approach. On the other hand, this specialization forces a need for workarounds when we need a different rasterization logic.

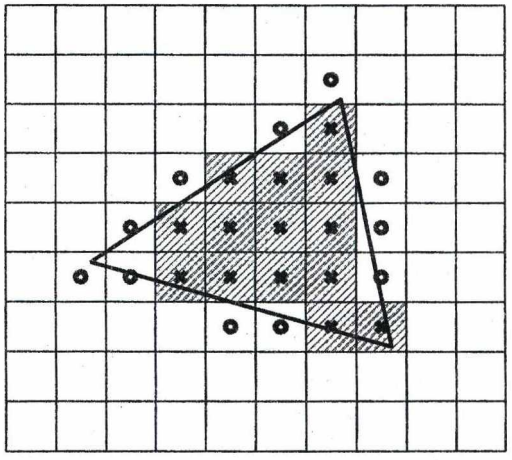


Figure 2: *Hardware supported triangle rasterization. Pixels marked with circles are not coloured, even though they overlap with the triangle.*

The general rule for rasterizing a triangle is to colour those pixels, whose centre is within the triangle (Figure 2. With adjacent triangles, this ensures there are no holes and no doubly coloured pixels in the image. However, if we apply this rasterization rule when rendering to the texture atlas, there may be some texels whose centre is not within the triangle, but they are overlapping with it. The texture coordinates in these areas will address a texel which was not considered by the rasterization, and therefore the pixel shader computing the value that should be there was not invoked. These texels will appear as non-initialized or black blocks or stripes near the seams of the texture atlas. These cases can be avoided by very careful texturing, if the seams are all horizontal or vertical in texture space, but they will always appear in the

general case of slant-edged triangle mesh atlases. Furthermore, if the texture is read using linear filtering, these non-initialized values will influence and even larger area.

2. Previous work

Our objective is to find the missing texels along the seams and invoke the pixel shader for them. Figure 3 shows the desired result, with white pixels indicating seam texels that should also be rasterized. One way to do it would be to change the rasterization rules to colour all the pixels overlapping with the triangle, known as overestimated rasterization. This is not supported by the hardware, but can be simulated using the approach of Hasselgren et al.[3], discussed in the Section 2.1. However, overestimated rasterization would extend all triangles including those not near seams, resulting in multiple pixel shaders invoked for the same texel, causing both unnecessary overhead and artifacts.

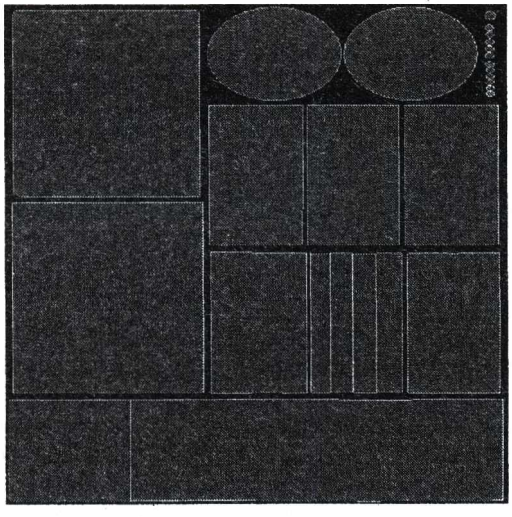


Figure 3: *A texture atlas. Gray texels are rasterized by conventional rasterization. White texels should also be computed.*

A simple but neither effective nor accurate method is to filter the texture atlas, and copy valid neighbouring texel values to non-initialized ones. This approach is really easy to implement, but it needs a full-screen rendering of the texture atlas, invoking a moderately expensive pixel shader using a number of texture reads for every texel. The overhead may be critical for a real-time algorithm. Furthermore, it is impossible to plausibly decide which neighbour's value is actually valid for a texel without knowledge of the geometry. Usually the heuristics of choosing the maximum is applied to keep visual artifacts low.

2.1. Overestimated triangle rasterization

Hasselgren et al.[3] have proposed a technique to conservatively rasterize separate triangles. While they explain that any GPU computation that decomposes its continuous problem domain into discrete triangles may benefit from the technique, they primarily consider applications like collision detection, where accurate rasterization is indeed a priority, but the shaders computing the rasterized colour are inexpensive and uniform. The difficult part of such problems is the processing of the results. Therefore, they are not concerned by adjacent triangles, where the area along the edge will be rasterized twice. For rendering complex computation results to atlases, however, this is a nuisance.

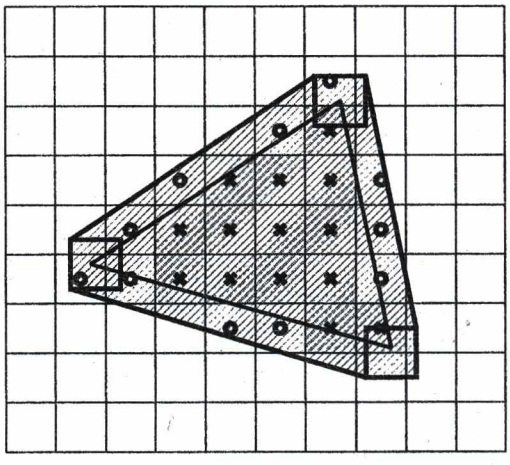


Figure 4: *Overestimated rasterization[3]. More complex geometry is rendered to include centres of overlapping pixels.*

Furthermore, quite heavy geometrical processing is required, as triangles are converted into polygons containing all texel centres that should be conservatively rasterized. Resulting polygons are defined by nine vertices, though some of them may be overlapping. The preprocessing cost is not critical compared to the time required to load the mesh model, and processing the geometry can not be avoided for an accurate solution anyway. However, it is not laudable that the triangle count will increase eightfold compared to the original mesh, possibly having some impact on performance. Realizing this flaw, the authors also devised an improved algorithm that creates a single overly extended triangle, but needs the modification of the pixel shaders to eliminate pixels that should not really be rasterized. However, the only way to bail out from a pixel shader without any effect is to render a transparent value with proper blending. Therefore, this imposes limitations and transfers responsibilities to the computation algorithm using the rasterization.

3. The proposed algorithm

Looking at the texture atlas, we can realize that we actually need to extend the contours of the charts of adjacent triangles. Instead of exchanging all the triangles with more complex extended polygons, we only need to rasterize a few line segments along the seams. In order to do this, we need to identify those edges of the model mesh which are duplicated and placed on seams, and find out how to offset the line segments to augment the triangles just like conservative rasterization would.

3.1. Line rasterization

A line segment is rasterized according to the principles set by the DDA or the Bresanham[1] algorithms. Cases are separated depending on whether the line segment is closer to the vertical or the horizontal direction. Then, in every line or every column respectively, that pixel is coloured the centre of which is closest to the line (Figure 5). We always measure distances on the scan line or the pixel column, thus we can state that pixels whose centre is within the half-a-pixel distance form the line will be coloured.

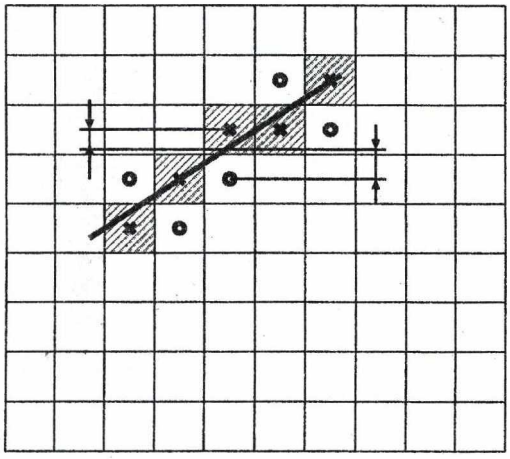


Figure 5: *Line segment rasterization[1]. The pixel whose centre is nearest to the line is coloured is every column of pixels.*

Fortunately, albeit not by coincidence, this produces the exact same pattern as the one that appears at the edge of a rasterized triangle. In order to extend the triangle by one pixel, we can simply draw a line next to it. This line has to cover those pixels, the centre of which is not within the triangle, but is within the one pixel distance. As line rasterization will colour the pixels within half-a-pixel distance to the line, we have to offset the edge exactly by half a pixel. Therefore, the vertices serving as the endpoints of the line segment can be derived from the original triangle vertices by offsetting their texture coordinates by an amount corresponding to half a pixel, in the positive or negative u or v direction, depending on the orientation of the edge (Figure 6).

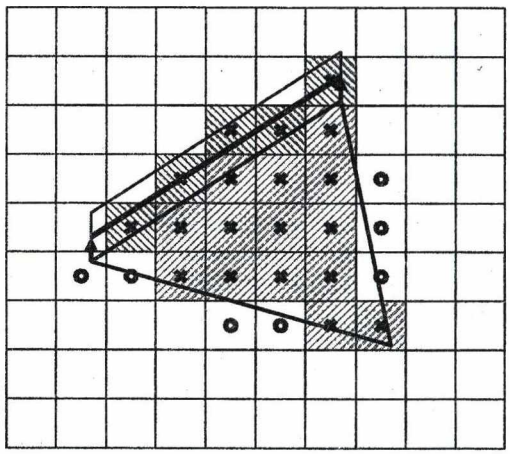


Figure 6: *Line segment drawn to rasterize overlapping pixels along edge.*

It is possible that two consecutive edges along the contour of a chart have different orientations, and the pixels of offseted edges do not cover the corner (Figure 7). To avoid missing these small areas, the edge should be extended by one pixel measured perpendicular to the offset direction at both ends.

Thus we acquire a supplemental set of geometry made up of line segments. Whenever rendering the original geometry, we can also render the supplemental geometry as a vertex buffer describing line primitives. As all vertex data are derived from the original triangle vertices, and the texture coordinates have been manipulated, there is no need to change the shaders. Neither the vertex shader nor the pixel shader will experience any difference from what happens when rendering the original triangles.

3.2. Finding the seam edges

In order to assemble the geometry of offseted seam edges, we have to identify such edges in the mesh texture mapping. In mesh description file formats or data structures, seam vertices are already replaced by two identical vertices with different texture coordinates, referenced by different triangles. Adjacency of triangles is usually not stored. However, we may build an adjacency graph based on the index buffer, disregarding the fact that duplicated edges actually would connect adjacent triangles. What we get is exactly the adjacency graph of the mesh charts in the texture domain. For this graph it is easy to identify those triangles that miss a neighbour, and thus, the edges that are on the seams of the texture atlas. We also need the texture coordinates of the

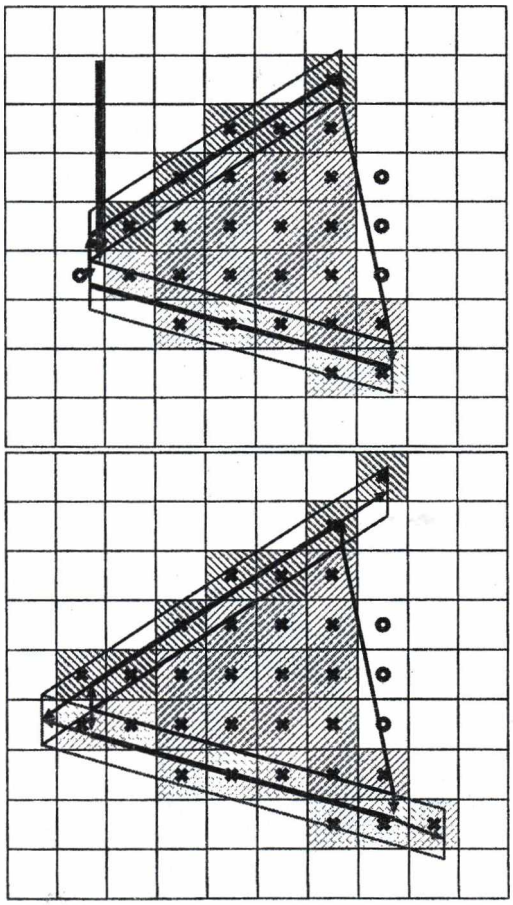


Figure 7: *Non-covered area may appear at corners where two differently aligned edges meet (above). Extending the edge line segment to cover corner pixels (below).*

third vertex of the triangle, to be able to find the proper offset direction for the extra line segment.

4. Results

The implementation of the proposed method is straightforward. It may be notable that the offset applied to edges depends on the resolution of the atlas. It is also required to have two texels wide free space between triangle charts to avoid rending them over each other. We have tested the method with meshes modelled with conventional tools using Maya, without taking any special care of placing texture coordinates. The texture atlas resolution was only 256 × 256 to emphasize artifacts (Figure 8).

We concluded that the artifacts (Figure 9) were completely covered by texels rendered using the supplemental line segments (Figure 10). However, the method failed for degenerate triangles, where the texture coordinates contained insufficient of false information about the orientation of an edge. Therefore, it is required that the texturing of the mesh is uniformly mapped, and there are no zero or negative texture area triangles.

Extending the line segments to cover edges accomplishes its goal, but it renders a few texels superfluously. As we may consider texels rendered from triangles more valuable, it is beneficial to render the supplemental geometry before the real one. This will also suppress edges overlapping with other charts of triangles, if the two pixels wide gap requirement is violated by the model.

We have also integrated the new algorithm into an implementation of the approximate ray tracing method, featuring soft shadows, multiple refractions and caustics all at high frame rates. Caustic and shadow impostors are rendered to the texture atlas. Formerly, the costly filtering method was used to eliminate the artifacts near texturing seams, sending frame rates down by up to 9% (48 FPS to 44 FPS on average). Implementing the proposed conservative rasterization algorithm, the artifacts were removed without any measurable performance impact. Figure 11 shows screenshots of the application with and without the new algorithm.

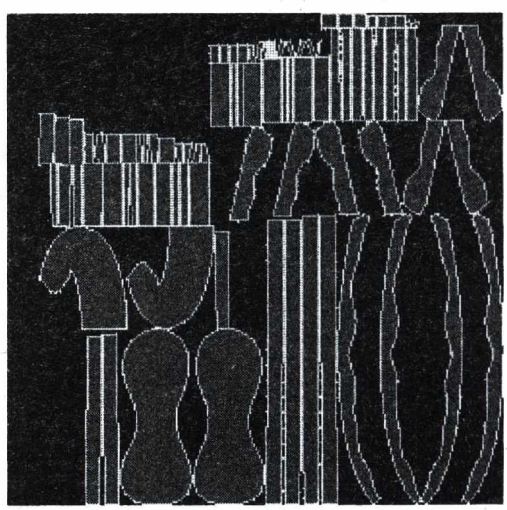


Figure 8: *The texture atlas of the rocking horse mesh. White texels were generated by rasterizing the supplemental line segments.*

5. Future work

When rendering to the atlas with blending enabled it is critical not render to any texel more than once. This is respected by the edge offsetting algorithm, but if we extend the line segments to cover corners, we loose this property. Further processing of the geometry would be required to identify

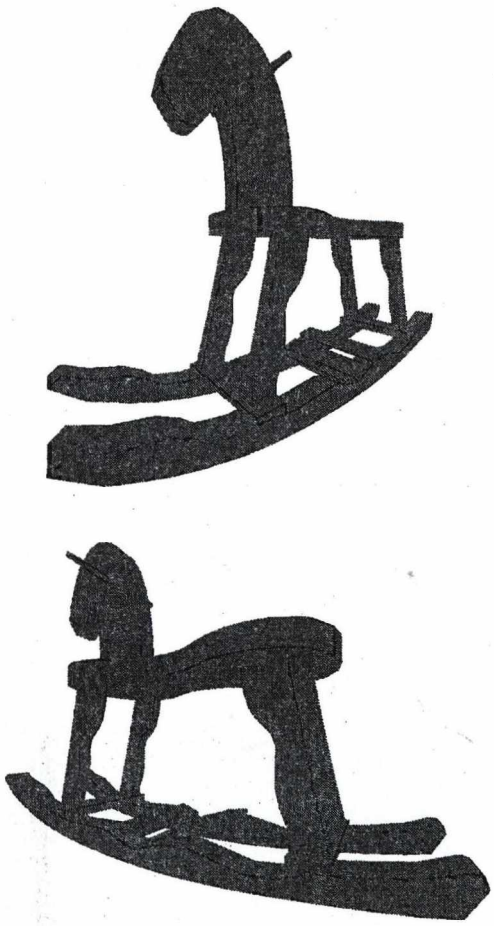


Figure 9: *The rocking horse rendered with conventional rasterization. Black areas have no valid values in the atlas.*

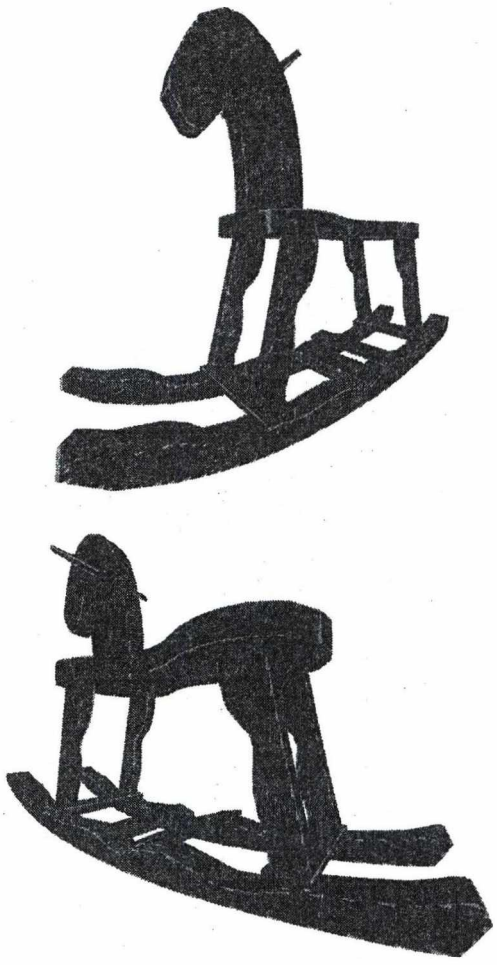


Figure 10: *The rocking horse rendered with conservative rasterization. Grey areas address texels computed by rasterizing supplemental line segments.*

corner texels exactly, and rasterize them using a supplemental vertex buffer of point primitives.

6. Acknowledgements

This work has been supported by OTKA (TO42735), GameTools FP6 (IST-2-004363) project, and the Spanish-Hungarian Fund (E-26/04).

We also wish to thank the authors of the approximate ray tracing method[6] for their invaluable help at showing the applicability of our algorithm in an advanced lighting computation framework.

References

1. Jack Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. 2, 4
2. Szabolcs Czuczor, László Szirmay-Kalos, László Szécsi, and László Neumann. Photon map gathering on the gpu. In *Eurographics short presentations*, Dublin, Ireland, 2005. Eurographics. 1
3. Jon Hasselgren, Tomas Akenine-Möller, and Lennart Ohlsson. Conservative rasterization. In Matt Pharr, editor, *GPUGems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 677–690. Addison-Wesley, 2005. 3

4. H. W. Jensen. Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30, 1996. 1

5. Alla Sheffer and John C. Hart. Seamster: inconspicuous low-distortion texture seam layout. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 291–298, Washington, DC, USA, 2002. IEEE Computer Society. 2

6. László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate ray-tracing on the GPU with distance impostors. In *Computer Graphics Forum, Proceedings of Eurographics 2005*, volume 24, Dublin, Ireland, 2005. Eurographics, Blackwell. 1, 6, 7

7. László Szirmay-Kalos and Balázs Benedek. Stochastic iteration for nondiffuse global illumination. pages 237–248, 2003. 1

8. Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In D. Lischinski and G.W. Larson, editors, *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)*, volume 10, pages 235–246, New York, NY, Jun 1999. Springer-Verlag/Wien. 1

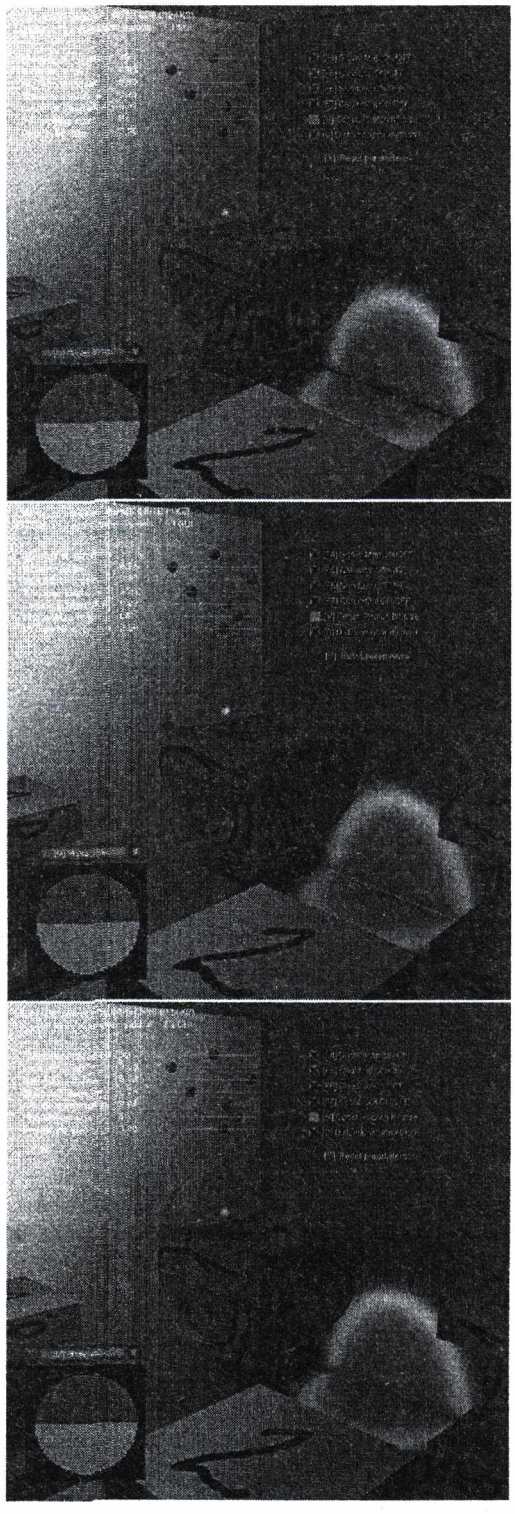


Figure 11: *Screenshots from the approximate ray tracing demo application[6] with atlas rasterization artifacts (above), with the texels rendered by the supplemental edge geometry in lilac(middle), and with the proposed conservative rasterization algorithm (below).*

Real-Time Indirect Illumination Gathering with Localized Cube Maps

István Lazányi, László Szirmay-Kalos

Department of Control Engineering and Information Technology, Budapest University of Technology, Hungary
Email: szirmay@iit.bme.hu

Abstract

This paper presents a fast approximation method to obtain the indirect diffuse or glossy reflection on a dynamic object, caused by a diffuse or a moderately glossy environment. Instead of tracing rays to find the incoming illumination, we look up the indirect illumination from a cube map rendered from the reference point of the object. However, to cope with the difference between the incoming illumination of the reference point and of the shaded point, we apply a correction that uses geometric information also stored in cube map texels. This geometric information is the distance between the reference point and the surface visible from a cube map texel. The method computes indirect illumination although approximately, but providing very pleasing visual quality. The method suits very well to the GPU architecture, and can render these effects interactively. The primary application area of the proposed method is the introduction of diffuse and specular interreflections in games.

1. Introduction

Final gathering, i.e. the computation of the reflection of the indirect illumination toward the eye, is one of the most time consuming steps of realistic rendering. According to the rendering equation, the reflected radiance of point $\vec{x}$ in viewing direction $\vec{\omega}$ can be expressed by the following integral

$$L^r(\vec{x} \to \vec{\omega}) = \int_{\Omega'} L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}')) \cdot f_r(\vec{\omega}' \to \vec{x} \to \vec{\omega}) \cdot \cos\theta_{\vec{x}} \, d\omega',$$

where Ω' is the set of possible illumination directions, $L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}'))$ is the incoming radiance arriving at point $\vec{x}$ from point $\vec{y}$ visible in illumination direction $\vec{\omega}'$, f_r is the BRDF, and $\theta_{\vec{x}}$ is the angle between the surface normal at point $\vec{x}$ and the illumination direction.

The evaluation of this integral usually requires many sampling rays from each point visible in some pixel. Ray casting finds *illuminating points* $\vec{y}$ for sampling point $\vec{x}$ at different directions (Figure 1), and the radiance of these illumination points is inserted into a numerical quadrature approximating the rendering equation. In practical cases, number P of sample points visible from the camera is over hundred thousands or millions, while number D of sample directions is about a hundred or a thousand to eliminate annoying sampling artifacts. In games and in real-time systems, rendering cannot take more than a few tens of milliseconds. This time does not allow tracing $P \cdot D$, i.e. a large number of rays.

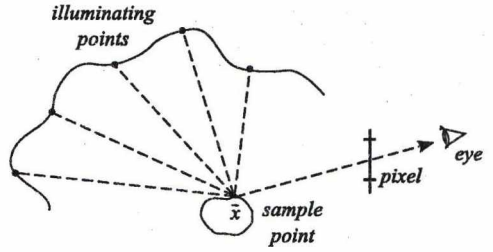


Figure 1: *Determining indirect illumination with sampling rays.*

To solve this complexity problem, we can exploit the fact that in games the dynamic objects are usually significantly smaller than their environment. Thus the global indirect illumination of the environment can be computed in a preprocessing phase, since it is not really affected by the smaller dynamic objects. On the other hand, when the indirect illumination of the dynamic objects is evaluated, their small size makes it possible to reuse illumination information obtained when shading other sample points. The first idea of this paper is to trace rays with the graphics hardware just

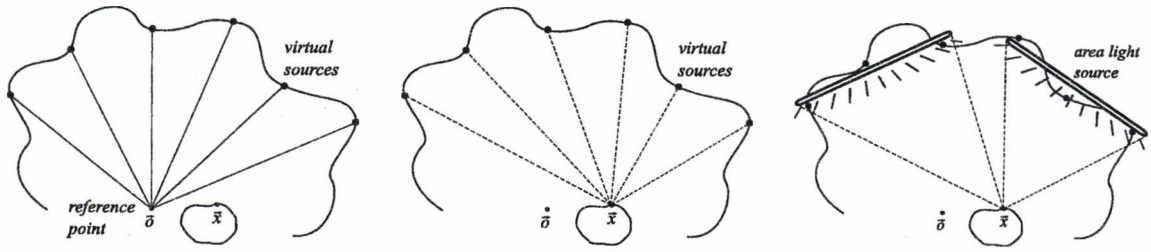


Figure 2: *The basic idea of the proposed method: first virtual lights sampled from reference point $\vec{o}$ are identified, then these point lights are grouped into large area lights. The illumination of a relatively small number of area lights at shaded points $\vec{x}$ is computed without visibility tests.*

from a single reference point being in the neighborhood of the dynamic object. Then the illumination points selected by these rays and their radiance are used not only for the reference point, but for all visible points of the dynamic object. From a different point of view, tracing rays locates *virtual light sources* which illuminate the origin of these rays. We propose to find a collection of virtual light sources from a reference point, and then use these virtual light sources to illuminate all shaded points (Figure 2).

This approach has two advantages. On the one hand, instead of tracing $P \cdot D$ rays, we solve the rendering problem by tracing only D rays. On the other hand, these rays form a bundle meeting in the reference point and are regularly spaced. Such ray bundles can be very efficiently traced by the graphics hardware. On the other hand, this simplification also has disadvantages. Assuming the same set of illuminating points visible from each sample point, self-shadowing effects are ignored. However, while shadows are crucial for direct lighting, shadows from indirect lighting are not so visually important. Thus the user or the gamer finds this simplification acceptable. Additionally, the used virtual light sources must be visible from the reference point. In concave environments, however, it may happen that an environment point is not visible from the reference point but may illuminate the shaded point. However, indirect illumination is usually quite smooth, thus ignoring smaller parts of the potentially illuminating surfaces does not lead to noticeable errors.

Unfortunately, this simplification alone cannot allow real time frame rates. The evaluation of the reflected radiance at a sample point still requires the evaluation of the BRDF and the orientation angle, and the multiplication with the radiance of the illuminating point by D times. Although the number of rays traced to obtain indirect illumination is reduced from $P \cdot D$ to D, but the illumination formula is still evaluated $P \cdot D$ times. These computations would still need too much time.

In order to further increase the rendering speed, we propose to carry out as much computation globally for all sample points, as possible. Clearly, this is again an approximation, since the weighting of the radiance of each illumination point at each sample point is different. From mathematical point of view, we need to evaluate an integral of the product of the illumination and the local reflection for every sample point. To allow global computation, the integral of these products is approximated by the product of the integrals of the illumination and the local reflection, thus the illumination can be handled globally for all sample points.

Intuitively, global computation means that the sets of virtual light sources are replaced by larger homogeneous area light sources. Since the total area of these lights is assumed to be visible, the reflected radiance can be analytically evaluated once for a whole set of virtual light sources.

The organization of this paper is as follows. First we review previous work that used global computations and data structures to make local reflections interactive. Then we discuss when factoring of integrals is acceptable to simplify illumination computations in section 3. Section 4 presents the formulae of illumination reuse. Section 5 discusses implementation details, and finally results and conclusions are presented.

2. Previous Work

Environment mapping[2] has been originally proposed to render ideal mirrors in local illumination frameworks, then extended to approximate general secondary rays without expensive ray-tracing[5, 9, 11]. Environment mapping has also become a standard technique of *image based lighting*[7, 3].

Classical environment mapping can also be applied for glossy and diffuse reflections as well. The usual trick is the convolution of the angular variation of the BRDF with the environment map during preprocessing[8]. This step enables us to determine the illumination of an arbitrarily oriented surface patch with a single environment map lookup during rendering.

A fundamental problem of this approach is that the generated environment map correctly represents the direction

dependent illumination only at a single point, the reference point of the object. For other points, the environment map is only an approximation, where the error depends on the ratio of the distances between the point of interest and the reference point, and between the point of interest and the surfaces composing the environment. Accurate results can be expected if the distance of the point of interest from the reference point is negligible compared to the distance from the surrounding geometry. However, when the object size and the scale of its movements are comparable with the distance from the surrounding surface — which is the typical situation in games — errors occur, which create the impression that the object is independent of its illuminating environment (Figure 3).

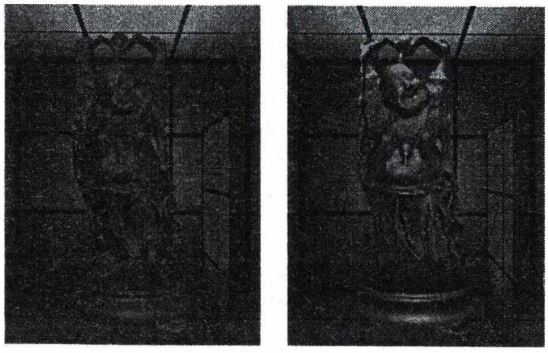


Figure 3: *Specular buddha rendered with the classical environment mapping method (left) and a reference image (right). The reference point for the environment mapping is located in the middle of the room. Although the buddha's head is very close to the green ceiling, thus it is expected to be greenish, the classical environment mapping is unable to cope with this expectation.*

One possible solution is to use multiple environment maps. For example, Greger et al.[4] calculate and store the direction dependent illumination in the vertices of a bi-level grid subdividing the object scene. During run-time, irradiance values of an arbitrary point are calculated by trilinearly interpolating the values obtained from the neighboring grid vertices. They reported good results even with surprisingly coarse subdivisions. This means that the smooth irradiance function is especially suitable for interpolation. While Greger et al. used a precomputed radiosity solution to initialize the data structures, Mantiuk et al.[6] calculated these values during run-time using an iterative algorithm that simulates the multiple bounces of light. For each bounce of light, cube maps are used to record the incoming radiance at the grid vertices. Then, irradiance values are calculated and stored at these grid vertices using the spherical harmonics representation[8]. Then, the scene is rendered again using tri-linearly interpolated irradiance values. Using this progressive approach, they could achieve global illumination effects in dynamic scenes with dynamic light sources at nearly-interactive frame rates.

Instead of working with multiple environment maps, another possible approach is to use a single environment map "smartly", so that it provides different illumination information for every point, based on the relative location from the reference point. At a given time, we have just a single "global" environment map, but make "localized" illumination lookups. GPU based localized image based lighting has been suggested by Bjorke[1], where a proxy geometry (e.g. a sphere or a cube) of the environment is intersected by the reflection ray to obtain the visible point. The *Approximate Ray-Tracing* approach[10], on the other hand, stores the distance values between the reference point and the environment and applies an iterative process to identify the real hit point of those rays that are not originated in the reference point.

3. Radiance computation in the new algorithm

Let us assume that we use a single environment map that records illumination information for reference point $\vec{o}$. Our goal is to reuse this illumination information for other nearby points as well. To do so, we use approximations that will allow us to factor out those components from the rendering equation, which strongly depend on actual point $\vec{x}$.

In order to estimate the integral of the rendering equation, directional domain Ω' is decomposed to solid angles $\Delta\omega'_i, i = 1, \ldots, N$, where the radiance is roughly uniform in each domain. After decomposing the directional domain, the reflected radiance is expressed as the following sum:

$$L^r(\vec{x} \rightarrow \vec{\omega}) =$$

$$\sum_{i=1}^{N} \int_{\Delta\omega'_i} L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}')) \cdot f_r(\vec{\omega}' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}} \, d\omega'.$$

Let us consider a single term of this sum representing the radiance reflected from $\Delta\omega'_i$. If $\Delta\omega'_i$ is small, then L^{in} has small variation. Furthermore, considering the illumination and reflectance as random variables, the variation of the illumination depends on the surface of the *illuminating* points, while reflectance $f_r(\vec{\omega}' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}}$ depends on the *receiver* point, thus they can be supposed to be independent. The expected value of the products of independent random variables equals to the product of the expected values, thus we can use the following approximation:

$$\int_{\Delta\omega'_i} L^{in}(\vec{x} \leftarrow \vec{\omega}') \cdot f_r(\vec{\omega}' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}} \, d\omega' \approx$$

$$\approx \tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i(\Delta\omega'_i)) \cdot \int_{\Delta\omega'_i} f_r(\vec{\omega}' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}} \, d\omega' \quad (1)$$

where $\tilde{L}_{in}$ is the *average incoming radiance* coming from surface Δy_i seen at solid angle $\Delta\omega_i'$:

$$\tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i(\Delta\omega_i')) = \frac{1}{\Delta\omega_i'} \cdot \int_{\Delta\omega_i'} L^{in}(\vec{x} \leftarrow \vec{y}(\vec{\omega}'))\, d\omega'.$$

The average incoming radiance can also be expressed with the properties of the surface visible from $\vec{x}$ at the given solid angle. Let us denote the center of this area by $\vec{y}_i$. The visible area and the solid angle have the following relationship:

$$\Delta\omega_i' = \frac{\Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{x}}}{|\vec{x} - \vec{y}_i|^2}$$

where $\theta_{\vec{y}_i,\vec{x}}$ is the angle between the normal vector at $\vec{y}_i$ and the direction from $\vec{y}_i$ to $\vec{x}$.

Expressing average incoming radiance $\tilde{L}_{in}$ with surface area Δy_i, we obtain:

$$\tilde{L}_{in}(\vec{x} \leftarrow \Delta\omega_i') \approx \tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i) = \frac{1}{\Delta y_i} \cdot \int_{\Delta y_i} L(\vec{y} \rightarrow \vec{\omega}_{\vec{y}\rightarrow\vec{x}})\, dy.$$

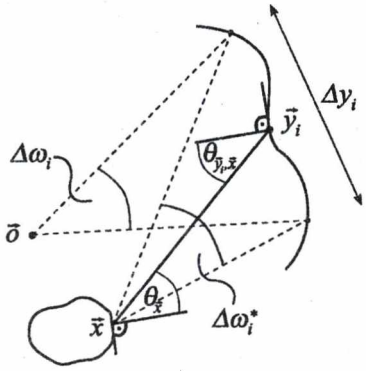


Figure 4: *The notations of the evaluation of factors*

The second factor of equation 1 is the *reflectivity* integral, which is also expressed as product of the average integrand and the size of the integration domain:

$$\int_{\Delta\omega_i'} f_r(\vec{\omega}' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega' = a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \Delta\omega_i'$$

where

$$a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega}) = \frac{1}{\Delta\omega_i'} \cdot \int_{\Delta\omega_i'} f_r(\vec{\omega}' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}}\, d\omega'$$

is the *average reflectivity* from solid angle $\Delta\omega_i'$.

The average reflectivity is evaluated using only one directional sample $\omega_{\vec{y}_i\rightarrow\vec{x}}$ pointing from $\vec{y}_i$ toward $\vec{x}$, that is, the integrand is assumed to be constant in domain $\Delta\omega_i'$:

$$a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega}) \approx f_r(\omega_{\vec{y}_i\rightarrow\vec{x}} \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \cos\theta_{\vec{x}}.$$

Clearly, such approximation is acceptable for diffuse and moderately glossy materials, but fails for highly specular surfaces.

Putting the results of the average incoming radiance and the reflectivity formula together, the reflected radiance can be approximately expressed as

$$L^r(\vec{x} \rightarrow \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i) \cdot a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \Delta\omega_i'. \tag{2}$$

4. Reusing illumination information

Let us find reference point $\vec{o}$ and identify elementary surfaces Δy_i which may illuminate the reference point, and evaluate incoming radiance $\tilde{L}_{in}(\vec{o} \leftarrow \Delta y_i)$. To do this, we render the scene from reference point $\vec{o}$ onto the six sides of a cube. In each pixel of these images we store the radiance of the visible point and the distance from the reference point. The pixels of the cube map thus store the radiance and also encode the position of small indirect lights.

To simplify the indirect illumination calculation, these small lights are clustered into larger area light sources, which corresponds to downsampling the cube maps. A pixel of the lower resolution cube map is computed as the average of the included higher resolution pixels. Note that both radiance and distance values are averaged, thus finally we have larger lights having the average radiance of the small lights and placed at their average position. The total area corresponding to a pixel of a lower resolution cube map will be elementary surface Δy_i.

According to equation 2 the reflected radiance at the reference point is:

$$L^r(\vec{o} \rightarrow \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}_{in}(\vec{o} \leftarrow \Delta y_i) \cdot a(\Delta\omega_i' \rightarrow \vec{o} \rightarrow \vec{\omega}) \cdot \Delta\omega_i'.$$

Let us now consider another point $\vec{x}$ close to the reference point $\vec{o}$ and evaluate a similar integral for point $\vec{x}$ while making exactly the same assumption on the surface radiance, i.e. it is constant in areas Δy_i:

$$L^r(\vec{x} \rightarrow \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i) \cdot a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \Delta\omega_i^*.$$

Solid angles $\Delta\omega_i'$ and $\Delta\omega_i^*$ can be expressed as

$$\Delta\omega_i' = \frac{\Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{o}}}{|\vec{o} - \vec{y}_i|^2}, \qquad \Delta\omega_i^* = \frac{\Delta y_i \cdot \cos\theta_{\vec{y}_i,\vec{x}}}{|\vec{x} - \vec{y}_i|^2}.$$

Assume that the environment surface is not very close compared to the distances of the reference and shaded points, thus the angles between the normal vector at $\vec{y}_i$ and reflection vectors from $\vec{o}$ and from $\vec{x}$ are similar ($\cos\theta_{\vec{y}_i,\vec{x}} \approx \cos\theta_{\vec{y}_i,\vec{o}}$). In this case, we can establish the following relationship between $\Delta\omega_i^*$ and $\Delta\omega_i'$:

$$\Delta\omega_i^* \approx \Delta\omega_i' \cdot \frac{|\vec{o} - \vec{y}_i|^2}{|\vec{x} - \vec{y}_i|^2}.$$

On the other hand, supposing diffuse or moderately specular environment, receiving point $\vec{x}$ can be translated a little while $\tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i)$ remains approximately the same, making $\tilde{L}_{in}$ independent of the receiving point $\vec{x}$ and allowing its global computation, that is

$$\tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i) \approx \tilde{L}_{in}(\vec{o} \leftarrow \Delta y_i).$$

Using these considerations, the reflected radiance at point $\vec{x}$ can be expressed as

$$L^r(\vec{x} \rightarrow \vec{\omega}) \approx \sum_{i=1}^{N} \tilde{L}_{in}(\vec{x} \leftarrow \Delta y_i) \cdot a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega}) \cdot \Delta\omega_i' \cdot \frac{|\vec{o} - \vec{y}_i|^2}{|\vec{x} - \vec{y}_i|^2}.$$

Note that we do not have to know anything about the orientation of the environment surface Δy_i. Those factors that are independent of point $\vec{x}$ can be pre-computed in

$$L_i^* = \tilde{L}_{in}(\vec{o} \leftarrow \Delta y_i) \cdot \Delta\omega_i' \cdot |\vec{o} - \vec{y}_i|^2,$$

and stored in the texels of the environment map. Then the summation of

$$L^r(\vec{x} \rightarrow \vec{\omega}) \approx \sum_{i=1}^{N} \frac{L_i^* \cdot a(\Delta\omega_i' \rightarrow \vec{x} \rightarrow \vec{\omega})}{|\vec{x} - \vec{y}_i|^2} \qquad (3)$$

is executed on-the-fly for each visible point $\vec{x}$.

5. Implementation

The proposed algorithm first computes an environment cube map from the reference point and stores the radiance and distance values of the points visible in its pixels. We usually generate $6 \times 256 \times 256$ pixel resolution cube maps. Then the cube map is downsampled to have 4×4 (or 2×2) pixel resolution faces. The resulting low-resolution environment map shall be called `LREnvMap` and its resolution will be denoted by `M`. To keep the presented shader code as simple as possible, we will omit the precomputation of L_i^* and calculate the value of the reflected radiance entirely on-the-fly:

```
float4 ReflRadPS (
          float3 N   : TEXCOORD0,
          float3 V   : TEXCOORD1,
          float3 pos : TEXCOORD2 ) : COLOR0
{
  float4 Lr = 0;
  V = normalize( V );  N = normalize( N );

  for (int x = 0; x < M; x++) // for each texel
   for (int y = 0; y < M; y++) {

     float2 t = float2((x+0.5f)/M, (y+0.5f)/M);
     float2 p = 2*t-1;  // -1..1

     Lr += Contrib(float3(p.x,p.y, 1), pos, N);
     Lr += Contrib(float3(p.x,p.y,-1), pos, N);
     Lr += Contrib(float3(p.x, 1,p.y), pos, N);
     // + similarly for the 3 remaining sides
  }
  return intens;
}
```

The `Contrib` function calculates the contribution of a single texel in `LREnvMap` to the illumination of the shaded point. Arguments `pos` and `N` are the position and the surface normal of the shaded point, respectively. Assuming diffuse BRDF the implementation is as follows:

```
float4 Contrib(float3 L,
               float3 pos,
               float3 N) {
    float l = length(L);  L = normalize(L);
    float dw_i = 4.0 / (M*M) / (l*l*l);

    float4 Lin = texCUBE(LRCubeMap, L);
    float  a   = k_d * max(dot(N,L),0);
    float4 Lr  = Lin * a * dw_i;

    float r = texCUBE(LRCubeMap, L).a;
    float r_= length(pos - L * r);
    Lr *= (r*r) / (r_*r_);    // localization
    return Lr;
}
```

The solid angle subtended by the given texel can be described as:

$$\Delta\omega_i = \frac{\Delta A \cdot \cos\theta}{|\vec{L}|^2},$$

where ΔA is the area of the texel, $\Delta A \cdot \cos\theta$ is the area of the texel visible from the reference point, and $\vec{L}$ is a vector pointing to the texel center. Assuming that the largest component of vector $\vec{L}$ is equal to 1 (as seen in the `ReflRadPS` function), the solid angle becomes

$$\Delta\omega_i = \frac{(2/M)^2}{|L|^3},$$

since the cosine value describing the orientation of the texel equals to $1/|\vec{L}|$.

6. Results

Let us consider a simple environment consisting of a cubic room with a divider face in it (Figure 5). Assuming that the reference point for the environment mapping is located in the middle of the room, the original and the downsampled environment maps are shown in Figures 6 and 7.

The object to be indirectly illuminated is the bunny, the dragon, and the happy buddha, respectively. Each of these models consists of approximately 50-60 thousand triangles. Frame rates were measured in 700×700 windowed mode on an NV6800GT graphics card and P4/3GHz CPU.

The first set of pictures (Figure 8) shows a diffuse bunny inside the cubic room. The first column is rendered using the traditional environment mapping technique for diffuse materials where a precalculated convolution enables us to determine the irradiance at the reference point with a single lookup. Clearly, these precalculated values cannot deal with the movement of the object: the bunny looks similar at any position of the room. The other columns show the results

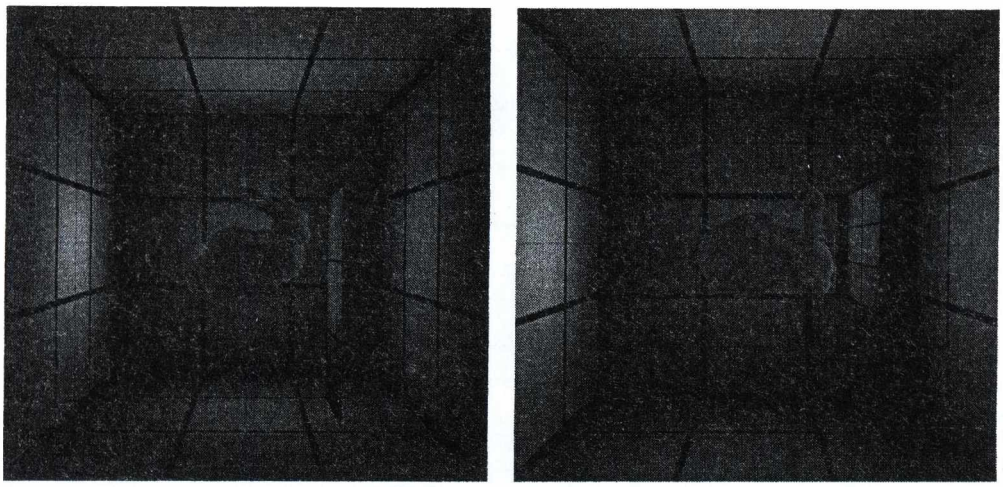


Figure 5: *A bunny in a simple environment (side and top view).*

of our method using different sized cube maps. Note that even with an extremely low resolution (2×2) we get images similar to the large-resolution reference.

The second set of pictures (Figure 9) shows a specular dragon inside a room. The first column presents the traditional environment mapping technique while the other three columns present our localized algorithm. Similarly to the diffuse case, even cube map resolution of 2×2 produced more pleasing results than the classical technique.

7. Conclusions

This paper presented a localization method for computing diffuse and glossy reflections of the incoming radiance stored in environment maps. The localization uses the distance values stored in the environment map texels. The presented method runs in real-time and provides visually pleasing results.

Acknowledgements

This work has been supported by OTKA (T042735) and GameTools FP6 (IST-2-004363) project.

References

1. K. Bjorke. Image-based lighting. In R. Fernando, editor, *GPU Gems*, pages 307–322. NVidia, 2004.

2. J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

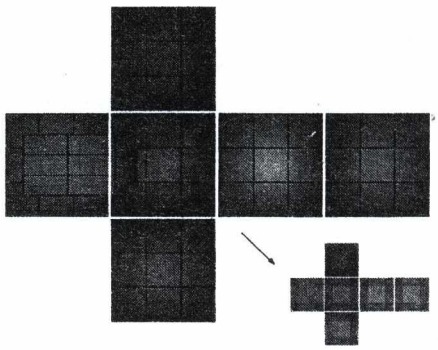


Figure 6: *Large-resolution cube map taken from the reference point and its downsampled version.*

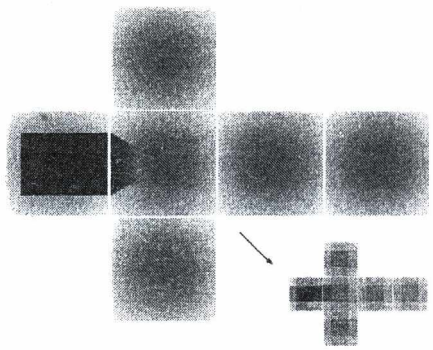


Figure 7: *Alpha channel of the large-resolution cube map stores the distances from the illuminating environment. The downsampled version contains averaged distances.*

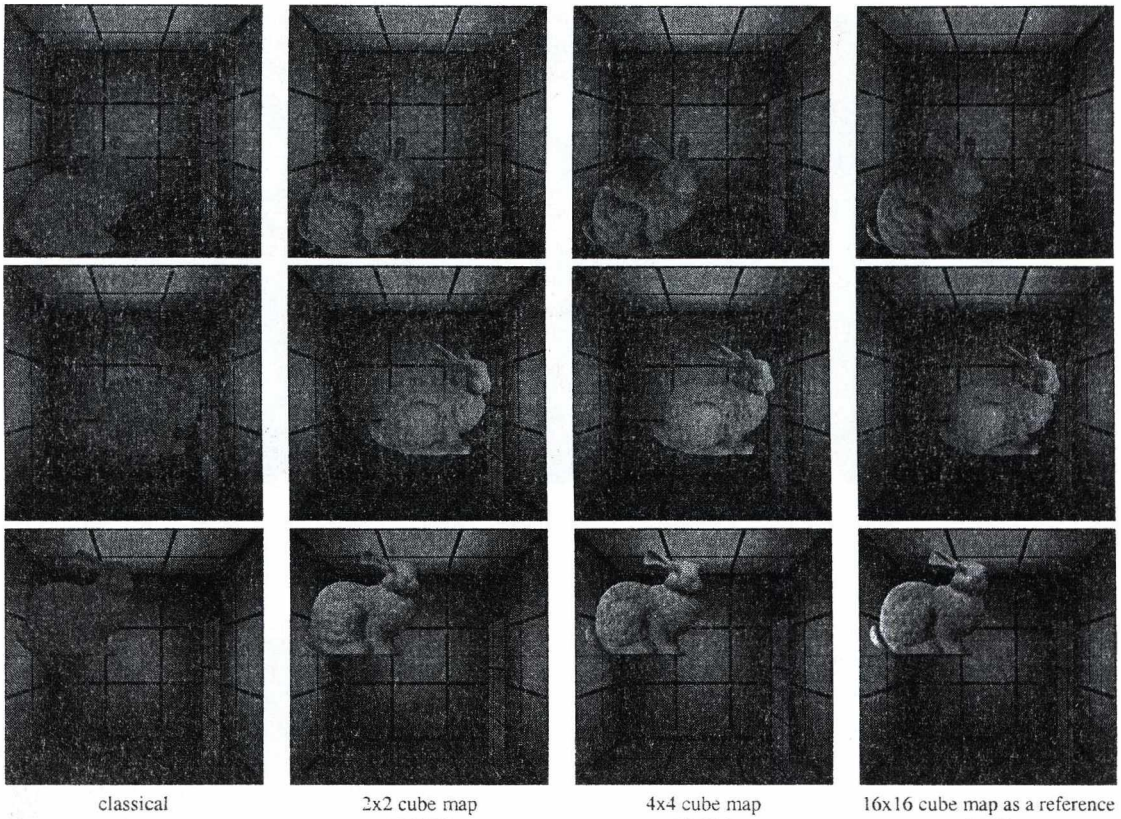
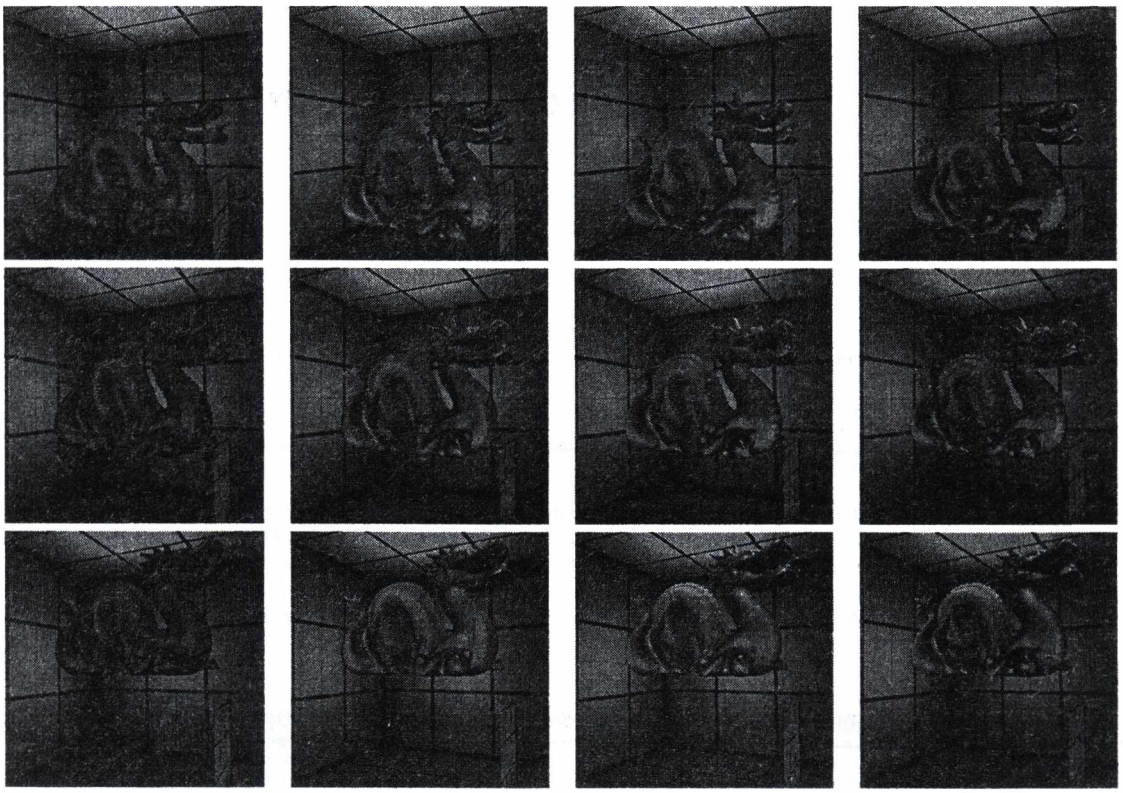


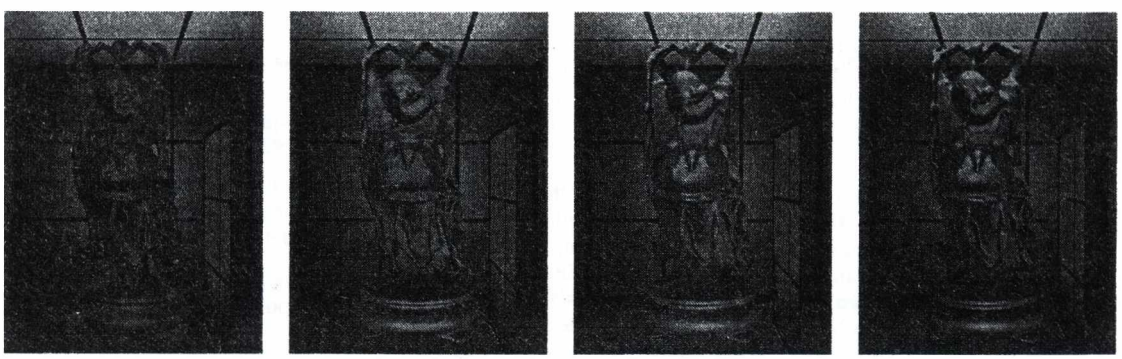
Figure 8: *Diffuse bunny rendered with the classical environment mapping (left column) and with our proposed algorithm.*

3. P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98*, pages 189–198, 1998.
4. Greger G., P. Shirley, P. Hubbard, and D. Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.
5. N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, 1984.
6. R. Mantiuk, S. Pattanaik, and K. Myszkowski. Cube-map data structure for interactive global illumination computation in dynamic diffuse environments. In *International Conference on Computer Vision and Graphics*, pages 530–538, 2002.
7. G. S. Miller and C. R. Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environment. In *SIGGRAPH '84*, 1984.
8. R. Ramamoorthi and P. Hanrahan. An efficient representation for irrandiance environment maps. *SIGGRAPH 2001*, pages 497–500, 2001.
9. E. Reinhard, L. U. Tijssen, and W. Jansen. Environment mapping for efficient sampling of the diffuse interreflection. In *Photorealistic Rendering Techniques*, pages 410–422. Springer, 1994.
10. L. Szirmay-Kalos, B. Aszódi, I. Lazányi, and P. Mátyás. Approximate ray-tracing on the GPU with distance impostors. *Computer Graphics Forum*, 24(3), 2005.
11. A. Wilkie. *Photon Tracing for Complex Environments*. PhD thesis, Institute of Computer Graphics, Vienna University of Technology, 2001.



classical | 2x2 cube map 21 FPS | 4x4 cube map 7 FPS | 16x16 cube map as a reference 1 FPS

Figure 9: *Specular dragon (shininess = 5).*



classical | 2x2 cube map 22 FPS | 4x4 cube map 6 FPS | 16x16 cube map as a reference 1 FPS

Figure 10: *Specular buddha (shininess = 5).*

Scalable Rasterizer Unit

P. Szántó and B. Fehér

Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary

Abstract

Modeling the light-surface interaction in real time 3D applications becomes more and more complex, as users require more lifelike images. Segmented screen rendering offers a viable solution to minimize the unnecessary work done in traditional rendering architectures. However, increasing the efficiency of the rendering pipeline also increases the required hardware resources for the 3D rendering unit. This paper presents a modular, scalable rasterizer architecture, which makes it appropriate in a wide range of applications.

Categories and Subject Descriptors: B.2.4 High-Speed Arithmetic, Algorithms

1. Introduction

In real-time graphics rendering two approaches are prevalent. Immediate Mode Rendering (IMR) renders the scene triangle by triangle; rasterization and shading of a triangle immediately starts after it has been transformed into screen space. Contrary to this, Deferred Rendering (DR) waits for all triangles to be transformed before beginning the per-pixel operations. The latter method has two main advantages.

First, it guarantees maximum efficiency when computing the output color values (the shading part of the rendering process, which clearly becomes the most time consuming), as only the truly visible values are shaded – unlike IMRs, where pixels not visible on the final image may be also processed. This is possible by first doing the visibility test, and deferring the rasterization process, so it only starts when the whole frame is analyzed and the visible objects are determined for every screen pixel.

Second, it allows using on-chip memory for the Depth-, Stencil- and Frame Buffer, thus reducing external bandwidth requirements, lowering cost and power consumption. It must be noted, that theoretically IMRs can also use on-chip buffers, but these buffers have to be the same size as the final, rendered image – which currently cannot be manufactured. The ability to segment the screen into small rectangles and then render these rectangles as "independent, small screens" allows small, implementable buffers to be used.

The DR rendering process is just a little different from the IMR one:

```
for every triangle in the given frame{
    transform the triangle into screen space
    find overlapped segments
}
for every segment on the screen{
    for every pixel in the segment{
        do visibility test
    }
    for every pixel in the segment{
        compute output color values
    }
}
```

The first part of the article reviews different segmenting strategies and presents hardware architecture for segmenting. The second part presents a modular Depth/Stencil Unit.

2. Segmentation

Basically, there are three possibilities when deciding about the segmentation strategy [1].

The simplest solution is to process all triangles in all segments, therefore completely skipping the segmentation part (SGI had architectures which work this way). This method has clear disadvantages, as the effective depth/stencil fill rate is especially decreased due to the unnecessary work done during the visibility test. On the other side, the hardware architecture is simplified, and there is no need to store a triangle list for the segments.

Bounding box method uses the bounding box of the triangles to define the overlapped segments. Even this simple method can increase efficiency considerably – especially with small triangles –, however there are cases when a lot of unnecessary segments are marked as overlapped. Figure 1 shows such a case. There are known architectures doing software segmenting this way, for example Intel Extreme Graphics [2] or Microsoft Talisman [3]. Hardware solutions are rarer, the PixelFlow [4] is surely employing bounding box method and benchmark results indicate that the only commercial deferred renderer (PowerVR Kyro [5]) also prefers this way.

The most efficient method is exact segmenting, when only segments having at least one pixel overlapped with the triangle are marked. The disadvantage is the required hardware resources to implement the functionality.

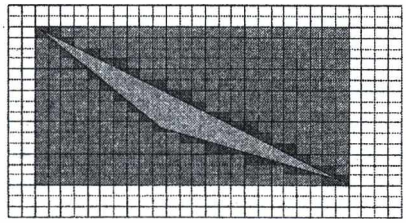


Figure 1. *Overlapped segments using bounding box and exact segmenting*

The following section shows the details of a hardware bounding box segmenting unit, and compares it with an exact segmenting solution, detailed in [6].

2.1 Bounding Box Segmenting Unit

Just as the Exact Segmenting Unit (ESU), the bounding box version (BBSU) consist of three main parts: the first part (Input Pipeline) computes the necessary input values for the main processing unit (Segment Generator), while the third part (Address Generator) handles communication with the external memory. To – possibly – increase efficiency, the unit supports programmable segment size, which can be set as an application specific parameter, or can be even adjusted adaptively, based on the statistics of a previous frame(s). Although selecting an overall appropriate segment size was already discussed in [7], the effects of variable segment size require further research to be correctly analyzed as there is no known academic or commercial architecture supporting this feature.

2.1.1 Input Pipeline

The first unit receives screen space vertex data (x, y coordinates) from the transformation part and, as a first step, generates primitives – triangles – from them. For primitive

without additional requirements, while for triangle fans the shar
times (thus, generating a triangle strip from the fan). The architecture of the Input Pipeline is shown on Figure 2.

For load balancing with the transformation part, vertex data is immediately written into a small, 64 word deep FIFO. The FIFO is followed by two 3-
input every clock cycle. After reading the appropriate number of vertices from the FIFO (eg. one for triangle strip, three for triangle list), the sorters output the minimal and maximal x and y coordinates of the current triangle (which define the bounding box with high precision). These values

vertical resolution of the segment, generating the corner segments of the bounding box. To limit resource usage, only

clock cycles to generate the four new values.

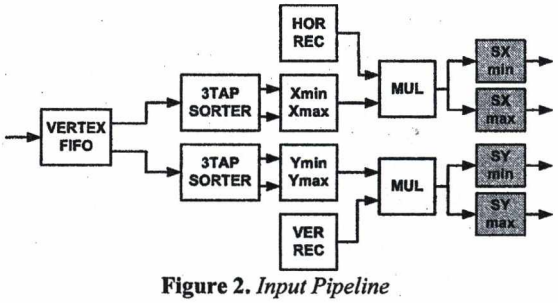


Figure 2. *Input Pipeline*

2.1.2 Segment Generator

The Segment Generator itself is very simple: it consists of two adders: one for incrementing the segment x coordinate, and one for incrementing the segment y coordinate. In the current implementation, the Segment Generator generates new (SX, SY) segment coordinate pairs every clock cycle, but with multiple adders it can be easily parallelized further

than the average maximum{bounding box height, width}).

2.1.3 Address Generator

The Address Generator builds a chained list for every segment. The list itself consists of 32-word blocks, from which 31 words are pointers to triangles, while the 32nd word is a pointer to the next 32 word block. The hardware

which was presented in details in [6].

2.1.4 Bounding Box vs. Exact Segmenting

Table 1. shows the main advantage of the BBSU, namely resource requirement.

All in all, the BBSU requires about quarter as many FPGA resources as the ESU. Efficiency is more complex to answer, as it largely depends on the frame to be rendered, not to mention that it is not enough to analyze the Segmenting Unit alone, but together with the Hidden Surface Removal Unit.

		FF	LUT	MUL	RAM
Input Pipeline	**ESU**	1100	1200	4	-
	BBSU	310	540	2	-
Segment Generator	**ESU**	900	2800	-	-
	BBSU	40	50	-	-
Address Gen.		270	380	1	4
ALL	**ESU**	2270	4380	5	4
	BBSU	620	970	3	4
BBSU/ESU, %		27.3	22.1	60	100

If average triangle size is comparable to the segment size (eg. one triangle overlaps only 3-4 segments) the BBSU can be just as effective as the ESU. As triangle size increases, ESU becomes at least twice as effective. If the scene contains a lot of triangles with high aspect ratio (just as the one on Figure 1), the efficiency advantage of the ESU version increases further. To fully answer this question, real-world applications should be analyzed, as widely accepted fill rate tests (such as 3DMark [8]) use full screen quads – in this case the ESU is twice as effective as the BBSU.

3. Hidden Surface Removal Unit

The Hidden Surface Removal Unit (HSRU) consists of two main parts: a Vertex Processing Unit (VPU), which receives vertex data and computes all the necessary values for the next part, which does overlapping determination, depth buffering and stencil buffering.

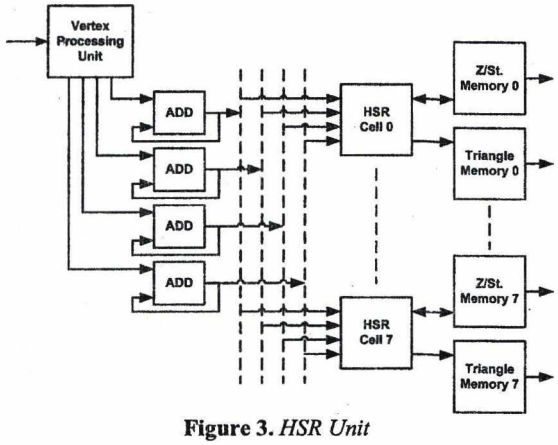


Figure 3. *HSR Unit*

The latter block is made up from several, similar processing elements (HSR Cells), as Figure 3 shows. The function of the adders between the two blocks will be discussed later.

Depending on the specified segment size, and the number of HSR Cells, each of them works on one or more segment lines. To correctly identify covered pixels and interpolate the depth values, the cells require initial values and delta values, which are generated by the VPU.

3.1 Vertex Processing Unit

Overlapping determination is based on variables generated from the explicit equation of the triangles sides:

$$S_j(x,y) = (x_i - x) * \Delta y - (y_i - y) * \Delta x \quad (1)$$

where x_i and y_i are points on the side, Δx and Δy are the

side. During interpolation, $S_j(x,y)$ is incremented or

through the pixels assigned to them. Covering is determined using the sign of the three S variables (S_0, S_1, S_2 represents the three variables for the three sides):

$$(sign(S_0(x,y)) \; XOR \; sign(S_1(x,y))) \; AND \; sign((S_1(x,y)) \; XOR \; sign(S_2(x,y)) \quad (2)$$

All in all, for the three sides the following calculations are required:

$$\begin{aligned} S_0(x_{strt}, y_{strt}) &= (x_{strt} - x_0) * (y_0 - y_1) - \\ &- (y_{strt} - y_0) * (x_0 - x_1) \\ S_1(x_{strt}, y_{strt}) &= (x_{strt} - x_0) * (y_0 - y_2) - \\ &- (y_{strt} - y_0) * (x_0 - x_2) \\ S_2(x_{strt}, y_{strt}) &= (x_{strt} - x_1) * (y_1 - y_2) - \\ &- (y_{strt} - y_1) * (x_1 - x_2) \end{aligned} \quad (3)$$

In (Eq. 3) x_{strt} and y_{strt} are the coordinates of the starting pixel for the HSR Cells (without anti aliasing, the top-left pixel of the processed segment).

Initial depth values and incremental values are generated using the following equations:

$$\begin{aligned} z(x_{strt}, y_{strt}) &= E_z * x_{strt} + F_z * y_{strt} + G_z = \\ &= (-\frac{A_z}{C_z}) * x_{strt} + (-\frac{B_z}{C_z}) * y_{strt} + (-\frac{D_z}{C_z}) = \\ &= (-\frac{A_z}{C_z}) * (x_{strt} - x_1) + (-\frac{A_z}{C_z}) * (y_{strt} - y_1) + z_1 \end{aligned} \quad (4)$$

The coefficients in (Eq. 4) are computed using the depth values defined at the three vertices and the plane equation of the triangle.

$$\begin{aligned} A_z &= (z_1 - z_2) * (y_1 - y_0) - (y_1 - y_2) * (z_1 - z_0) \\ B_z &= (x_1 - x_2) * (z_1 - z_0) - (z_1 - z_2) * (x_1 - x_0) \\ C_z &= (x_1 - x_2) * (y_1 - y_0) - (y_1 - y_2) * (x_1 - x_0) \\ D_z &= -(A_z * x_1 + B_z * y_1 + C_z * z_1) \end{aligned} \quad (5)$$

In the hardware realization clipped screen space x, y coordinates are 16 bit fixed point values, while vertex depth (z)

computations, the VPU uses these formats, but at the last step $z(x_{strt},y_{strt})$, E_z and F_z are converted to 24 bit fixed point format, preserving only the fractional part of the generated

depth values (the transformation of triangles from 3D world space to screen space maps depth values to [0, 1] range).

3.1.1 Hardware Architecture

Because of the architecture of the HSR Cells, the above computations can be done in 16 clock cycles without limiting performance. Therefore, the trivial dataflow implementation is not the best option, as it requires too many resources, while it is needlessly fast.

A programmable solution not only requires fewer resources, but it is also more flexible, which – together with the HSR Cell architecture – allows flexible anti aliasing implementation (more on this later). Figure 4 shows the architecture of the arithmetic unit.

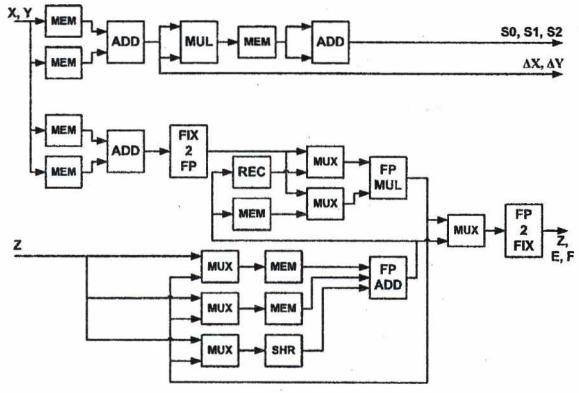


Figure 4. *Vertex Processing Unit*

The inputs of the VPU are the vertices' x, y and z coordinates and the screen coordinates of the top-left pixel in the processed segment.

The whole processing unit consists of two, almost separated portions. The upper part on Figure 4 is able to compute the required values for covering determination, while the lower part is responsible for the depth related computations.

The first units in the overlapping determination part are small, dual-ported memories to store the three x and y coordinate pairs of the currently processed triangle. The two memories fed a 16 bit adder/subtractor which has write enable signals at the input registers – its function is to compute the delta coordinate values in Eq. 3. The output is routed to a multiplier (which also has write enable signals at the input registers) to compute the partial products in Eq. 3 – to increase flexibility, these values are stored in a small memory. The final 32 bit adder can calculate the final edge variables (S_0, S_1, S_2), which – together with the delta values – are the input values for the covering determination part of the HSR Cells.

The depth related part supports mixed formats; in order to reduce latency, the subtraction of the screen coordinates are done using fixed point arithmetic, but all other calculations are performed using the 24 bit floating point format of the vertex depth values. Accordingly, the fix point adder which generates the coordinate differences is followed by a fixed point to floating point converter before connecting to the floating point multiplier. The coordinate differences are

differences), generating the partial products of A_z, B_z and C_z. The output of the multiplier is routed to the input memories of the adder to be able to compute the A_z, B_z, C_z results. The A_z, B_z results are then multiplied with the reciprocal of the C_z value (hence the multiplier can use the result of the adder and the reciprocal unit), generating the final E_z and F_z values. To generate the initial depth value, E_z and F_z are multiplied with the starting x and y coordinates, and then these results are added together with the z_1
value. To avoid the operand cancellation when adding together these values (that is when two large, almost equal values with different sign hide the impact of a small third operand, but then cancels each other), the adder has three inputs, and it always generates correct results; however, the third input is only used when the initial depth value is computed, otherwise it is set to zero (the shift register at the input allows the appropriate delay to be applied to z_1 compensate for the latency of the computations).

the read address of the memories; the select signals of the multiplexers; the add/subtract control signals and the write enable signals of the different units. Scheduling analysis shown that the depth computation has nearly 48 clock latency (obviously, this depends on the program), but it is able to start processing a new triangle every 16th clock cycle. Of course, when programmed differently, latency and performance may differ, the above reported results are valid for an actual program which can handle anti aliasing.

As the different arithmetic units have different latencies which are not hidden from the programmer in any way, programming the unit is not the easiest task. However, the creation of simple compiler (data flow compiler) is obviously possible.

3.2 HSR Cell

The unit doing the actual hidden surface removal is made up from a number of similar cells. Every cell has its own buffer (storing depth, stencil and triangle pointer), and some segment lines assigned to it; using N cells, every nth line is processed by the nth cell (for example, with 8 cells and 32*16 resolution segment, the 1st and 9th line is processed by the 1st cell, the 2nd and 10th lines are processed by the 1st cell, and so on). After processing one of the associated lines, the

interpolate in the y direction. Irrespectively of the number of

However, this makes scheduling predicta
the VPU can have 16 clock cycles to generate new input data for the cells.

When the VPU has finished generating the new values for the starting pixel, the 1st cell loads these data. At the same time the input data is modified for the next cell (stepping one line in the y direction), which loads them one clock cycle later compared to the 1st cell. This method only requires one input at any time, so only one adder per variable is needed to interpolate them in the y direction (these are the adders on Figure 3).

A cell itself consists of three distinct units. Covering determination identifies if a pixel is inside the processed triangle – and allows the write enable signal of the buffers to become active.

3.2.1 *Covering Unit*

After loading the initial values of the three S_j variables, a HSR Cell decrements this value with Δy (see Eq. 2-3) every clock cycle. The decision about overlapping is done using Eq. 2.

3.2.2 *Depth Unit*

The Depth Unit reads the depth buffer, compares the read value with the interpolated one (using the set comparison function) and in case the comparison returns true, allows the write back to the depth buffer. The pipelined architecture is shown on Figure 5, gray blocks represent registers, and white blocks are logic functions.

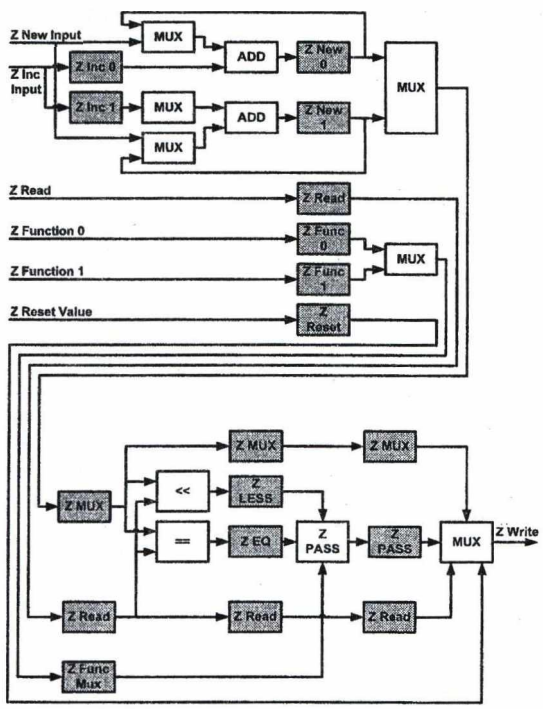


Figure 5. *Depth Unit*

This unit supports all possible comparison functions (Z Func – always, never, less, less-or-equal, equal, greater-or-equal, greater). The Depth Unit has two processing modes: opaque and transparent.

In opaque mode, two pixels are processed per clock cycle, the two Z Inc registers stores the same delta value and the depth functions are also similar.

Transparent, multi pass mode can be activated after all opaque triangles are processed. In this mode, the unit can process one pixel per clock cycle. In every pass, the transparent triangle which is farthest from the camera, but closer than the already processed triangles is determined for every pixel. This is done with two comparisons and using two depth buffer location: one location stores the depth value of the already processed (shaded) triangle, while the other is the working buffer. The former is compared with the interpolated depth value using less comparison function, while the latter is compared using larger comparison function. After finishing a pass, the two buffer locations changes function.

3.2.3 *Stencil Unit*

To keep up with the Depth Unit, the Stencil Unit also supports two stencil tests per system clock using 8 bit stencil values. As these two units share the same memory, the pipeline latency is also the same. Just as the Covering- and Depth Unit, it also gen

value, write mask value, comparison function and stencil operation. The comparison function has the same options which were listed for the Depth Unit, while the stencil operation can be set for "stencil test fails", "stencil test passes and depth test fails" and "stencil test passes and depth test passes" cases. Available operations are: keep previous value, set to zero, replace with reference value, increment (with or without saturation), decrement (with or without saturation) and invert.

4. Arbitrary Anti Aliasing

The programmable Vertex Processing Unit together with the programmable segment size allows implementing anti aliasing (AA) with arbitrary number of samples and arbitrary sample positions.

Without AA, the x_{strt} and y_{strt} coordinates in Eq. 3 and Eq. 4 are set to the screen space coordinates of the top-left segment pixel, and the segment size is set to the real (pixel) resolution of the segment. The VPU processes a triangle once, computing the required delta values and the initial values for the top-left segment pixel. The HSR Cells use these values to determine covering and generate depth values at pixel centers.

Setting for example two-times AA requires only minor programming changes. Let define sampling positions as shown on Figure 6. The desired effect can be achieved by setting the vertical resolution of the segment to twice of the real size, and then modifying only the x_{strt} and y_{strt} values. First, the VPU computes the same delta values as without

AA, but generates the start values using (*top left pixel x + pix_size/4*) as x_{strt} and (*top left pixel y – pix_size/4*) as y_{strt}. As long as the HSR Cells process the first sample positions, the VPU modifies x_{strt} to (*top left pixel x – pix_size/4*) and y_{strt} to (*top left pixel + pix_size/4*).

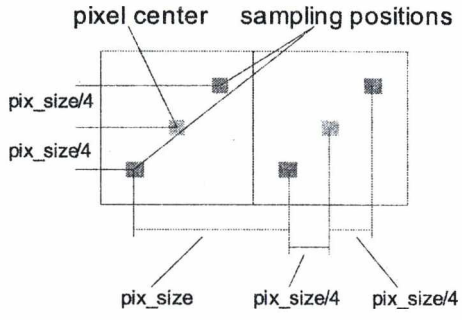


Figure 6. *AA sampling pattern*

Using this method, any number of AA samples can be generated within a pixel, as long as the precision of the coordinate computation allows it, and there is enough buffer memory. For example, a 32*16 resolution segment allows 64-times AA to be used. Obviously, depth/stencil buffering fill rate decreases linearly as the number of samples increases. It must be noted that the presented architecture only deals with oversampled covering determination and depth testing. To generate AA-ed images, the other units (especially the shader unit) have to support this arbitrary mode. Actually, the type of AA (multisampling – only object edges are filtered; supersampling – the whole image is filtered) applied is also dependent on those units – the output of the HSR allows both methods.

5. Conclusion

The article presented some hardware units which can be used to create an efficient rasterizer unit. In an FPGA implementation using XC2V6000-4 FPGA, all units can achieve 100 MHz system clock speed (with parts of the HSR Cells operating at double frequency), translating into 100 million segments/sec for the segmenting units, and up to 1.6 GPixels/sec depth/stencil fill rate for an 8 cell HSR Unit.

The two segmenting methods have to be analyzed using real-world applications to explore the overall performance increase exact segmenting offers. Similarly, to identify the potential benefits of the program
performance of real applications should be measured under different conditions. These measurements then may be used to create an algorithm to adaptively alter segment size between frames, which is simple enough to be implemented in hardware.

References

[1] Michael Cox, Narendra Bhandari, *Architectural Implications of Hardware-Accelerated Bucket Rendering On the PC*, Siggraph/Eurographics Workshop On Graphics Hardware, 1997

[2] *Intel Zone Rendering Technology 3 Whitepaper*, http://support.intel.com/design/chipsets/applnots/302625.htm

[3] J. Torborg and J.T. Kajiya, *Talisman: Commodity Realtime 3D Graphics for the PC*, Proc. ACM Conf. on Computer Graphics Conference(SIGGRAPH '96), 1996

[4] Eyles, John, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, and Nick England, *PixelFlow: The Realization*, Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, 1997

[5] *PowerVR Tile Based Rendering Whitepaper*, http://www.pvrdev.com/pub/PC/doc/idx/whitepapers.htm

[6] Péter Szántó, Béla Fehér, *Exact Bucket Sorting for Segmented Screen Rendering*, GSPX 2005 Pervasive Signal Processing, 2005

[7] I. Antochi, B.H.H. Juurlink, S. Vassiliadis, P. Liuha, *Scene Management Models and Overlap Tests for Tile-Based Rendering*, EUROMICRO Symposium on Digital System Design, 2004

[8] *3DMark05 Whitepaper*, http://www.futuremark.com/companyinfo/?companypdfs

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Real-time Physically Accurate Soft Shadows

Barnabás Aszódi, László Szirmay-Kalos

Department of Control Engineering and Information Technology,
Budapest University of Technology, Hungary
Email: szirmay@iit.bme.hu

Abstract

This paper presents a GPU based real-time algorithm to render physically accurate soft shadows for spherical light sources. The algorithm shoots shadow photons which are splat onto the shadow receiver surfaces at their hit points. The method needs a Shader 3.0 compatible driver and graphics card because of the complexity the required shader instructions like texture reading in the vertex shader. Thanks to the high computational power of the GPU, the algorithm can render realistic soft shadowing at very high frame rates, and can be included in games.

1. Introduction

Shadows are important not only to make the image realistic, but also to allow humans to perceive depth and distances. Shadows occur when an object called *shadow caster* occludes the light source from another object, called *shadow receiver*, thus prevents the light source from illuminating the shadow receiver. In games and real time applications shadow casters and receivers are often distinguished, which excludes self shadowing effects. However, in real life all objects may act as both shadow receiver and shadow caster.

Point and directional light source generate *hard shadows* having discontinuous boundaries. However, realistic light sources have non zero area, resulting in *soft shadows* having continuous transition between the fully illuminated region and the occluded region, called *umbra*. The transition is called the *penumbra* region. With hard shadows only the depth order can be perceived, but not the distance relations (Figure 1). Shadows should have real penumbra regions with physically accurate size and density to allow the observer to reconstruct the 3D scene.

The width and the density of the penumbra regions depend on the size of the area light source, on the distance between the light source and shadow caster object, and on the distance between the shadow caster and shadow receiver object.

The algorithm proposed in this paper generates shadows by casting shadow photons and visualizing them rendering small quadrilaterals into the texture atlases of the shadow receiver objects. The size and the density of a quadrilateral are controlled by the distances of the light sources and the shadow caster and the shadow receiver, respectively.

The algorithm first casts shadow photons and finds where they hit the shadow receiver. The hit positions are computed in texture space. During photon tracing, the distances of the light source, shadow caster and receiver are also calculated. Based on these distances, the sizes of the quads representing the shadow photon splats are determined. The quads are rendered into light maps, and finally the scene is shown with the product of the material textures and the light maps already containing the darkening of the shadow photons.

2. Previous Work

A lot of different methods have been proposed to generate soft shadows. Simplest approaches consider the area light source as a collection of point light sources and blend their hard shadows together to obtain smooth transitions. However, this approach can significantly degrade the rendering speed or the quality if the light sources are really large.

Samuli Laine et al.[8] developed an algorithm for physically correct soft shadows with shadow volumes and ray tracing. Their algorithm is targeted to production-quality ray tracers. Although, this method generates high quality shadows – the same result as stochastic ray tracing – but is far from real time.

Zhengming Ying et al.[10] developed a real-time and real-

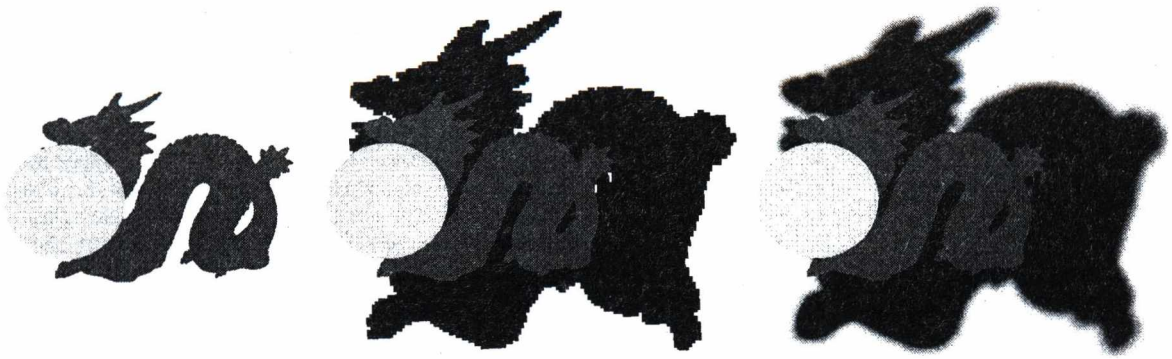


Figure 1: *The importance of soft shadows. Left: without shadows only the depth order can be perceived. Middle: hard shadows are unable to visualize the distance between the shadow caster and the receiver. Right: soft shadows help the observer to recognize distance relations.*

istic method for linear light sources. They sample the light sources and use several shadow maps. In their algorithm the quality of shadows depends on the number of samples.

Jukka and Jan[2] created real-time soft shadow with penumbra quads. Their algorithm generates physically not correct soft-shadow. They store additional information in a *penumbra map* not only z-coordinate information[4, 6, 9].

In addition to physically based approaches, there are also fake methods. The simplest technique is to use depth mapped shadow algorithm to obtain a hard shadow map and to filter it using a low pass filter. However, such approach does not take into account the distances between the light sources shadow casters and receivers, which would also be crucial to determine the density and size of the penumbra region.

Eric Chan and Frédo Durand[4] improved the basic concept and made more accurate. They use shadow map, identify silhouette edges, create "smoothies" (areas between the original and the enlarged silhouette edges), and render them with appropriate transparency values. These smoothies create penumbra regions. Arvo et al. [1] proposed an image space algorithm which also starts with a conventional shadow map, finds its edges, extends it with a flood fill algorithm, and computes the penumbra density in each pixel by a fragment program. Another interesting technique applies the concept of "blurring hard shadows" in the context of shadow volume algorithms, where the stencil buffer is filtered instead of the depth map [7].

Assarsson and Möller[3] used shadow volume based algorithm for generating physically based soft shadows. The algorithm creates additional penumbra volumes. It demands a lot of computational power.

The interested reader should also refer to the state of the art report on soft shadows [5] written by Hasenfratz et al.

3. The new soft shadow algorithm

The basic idea of the proposed algorithm is to shoot shadow photons that "darken" the lighting of the light source. Shadow photons are splat onto the shadow receiver surface. The process should meet the following requirements. At hard shadow regions of the shadow receiver, from where no point of the light source is visible, the total effect of shadow photons is expected to completely cancel the contribution of the light source. However, at penumbra regions, from where the light source is partially visible, the "darkening" must be proportional to the visible portion of the light source. The darkening is thus continuously decreasing from the hard shadow edge to the regions where the full light source is visible. The shape and size of the penumbra region depend on many factors, including the shape and size of the light source, of the shadow caster, and also on the geometry of the shadow receiver. In order to cope with these complex relationships, the size of the splat shadow photons is controlled to make the penumbra regions meet the geometric requirements.

For the sake of simplicity, the light source is assumed to have spherical geometry, thus its projected area and shape are similar in all directions. Should we have planar light sources, the algorithm can still be used, but the projected size of the light source depends on the direction between the light source and the shadow caster.

In case of a spherical light source, the penumbra region responsible for soft shadows depends on the following factors: the size of the light source that is described by diameter d, distance s between the light source and the shadow caster, distance r between the light source and the shadow receiver, and orientation angle θ between the surface normal at the shadow receiver and the illumination direction. Examining the geometry of figure 2, the linear size of the penumbra can

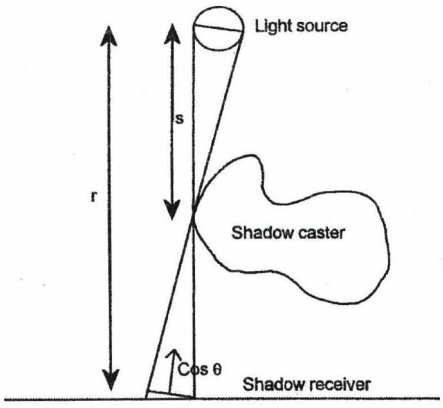


Figure 2: *Penumbra generated by a spherical light source.*

be obtained as:

$$u = d \cdot \frac{r - s}{s \cdot \cos\theta}. \qquad (1)$$

We propose to set the size of the shadow photon splat according to this formula. Since shadow photons which are generated by the silhouette of the shadow caster are responsible for the penumbra, the obtained penumbra will have approximately the expected size.

Equation 1 gives the size of the shadow photon splat in world space. However, if splats are rendered into texture space to obtain light maps, the expansion between the texture space and world space should also be taken into account. To represent this expansion, in the pre-processing phase we render a texture for each object where texels store the ratio of world space and texture space triangle areas. When the size of the splat in world space is computed, the expansion texture is used to convert this result to texture space.

The shadow generation algorithm has three phases. The first phase of the algorithm renders the shadow casters from the point of view of the light source. Now we are interested in whether a shadow caster is visible in a direction (i.e. pixel), and in distance s is from the light source center. This information can be put into a monochromatic texture, called the *caster map*, which stores distance s or -1 if no shadow caster is visible in a pixel. The shadow caster map is generated and stored by the GPU.

The second phase renders the shadow receivers from the point of view of the light source center using the same camera setting and viewport resolution as in the first phase. This step identifies those points that are potentially occluded from the light source. To identify these points, the object id and the texture coordinates of the points are written into r, g, b channels of the rendering target, respectively. Additionally, to prepare for the penumbra size computation (equation 1), distance r between the shadow receiver surface and the center of the light source, as well as the cosine of the orientation angle ($\cos\theta$) are also computed. Reading the caster map at this pixel to obtain shadow caster distance s, we have all values needed to evaluate equation 1. The result of this formula, that is size u of the photon splat in world coordinates, is multiplied with the expansion between the texture and world space, and the product is written into the alpha channel.

If no shadow caster is visible, i.e. the alpha channel of the caster map is negative, or distance s between the light source and the caster is not smaller than distance r between the light source and the shadow receiver, then no occlusion occurs and therefore no shadow photon is generated. To indicate this case, the alpha channel is set to a negative value. Thus the result of the second rendering phase is the identification and the size of shadow photon splats in texture space. This image is called the *receiver map*.

Having the receiver map, we know where and how large shadow photon splats should be placed. Placing the shadow photons, either the original surface textures can be modified or a separate light map can be generated.

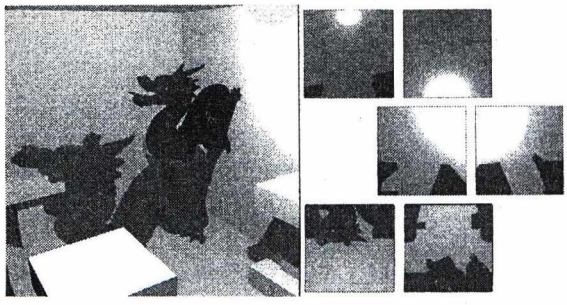


Figure 3: *A scene with hard shadow and direct illumination (left). The product of the texture and the light maps of the shadow receiver room (right). In order to avoid shadow leaking the faces are separated by void regions in texture space.*

To obtain the shadow, shadow photons are splat, that is, they are rendered as small quadrilaterals having semi-transparent Gaussian filter textures. We render as many quadrilaterals as many pixels the receiver map has. A single pixel of the receiver map thus represents a single shadow photon.

The vertex shader reads the receiver map and sets the coordinates of these quadrilaterals according to the center of the shadow photon and its size. Note that texture reading in the vertex shader is possible only with Shader 3.0 compatible graphics cards. The transformation is set to render to texture space, thus photons are splat onto the surface texture.

The simplified vertex shader code in HLSL is the following:

```
float4 ph = tex2Dlod(receivermap, IN.phcoord);
OUT.filtcoord =IN.pos.xy; // filter coords
float size = ph.a;        // splat size
OUT.hpos.x = ph.x * 2 - 1 + IN.pos.x * size;
OUT.hpos.y = 1 - ph.y * 2 + IN.pos.y * size;
OUT.hpos.w = 1;
if (ph.a < 0) OUT.hpos.z = 2; // ignore
else          OUT.hpos.z = 0; // valid
```

The original quadrilateral vertices passed in (`IN.pos`) are set to the vertices of a unit rectangle by the CPU. Simultaneously, the CPU passes the coordinates of the receiver map pixels in variable `IN.phcoord` one by one. The original quadrilateral is scaled and translated according to the shadow photon splat size and position, respectively, and the transformed vertices are put to the output position register `IN.hpos`. Note that setting the z component outside the clipping region (`OUT.hpos.z = 2`), we can let the clipping hardware ignore those pixels of the receiver map which does not contain shadow casters. Original vertices (`IN.pos`) are also used to address the Gaussian filter texture in the pixel shader, therefore passed in a texture register (`OUT.filtcoord`).

The pixel shader then computes the light map of the point using the partial visibility factor of the area light source:

```
float  shadow = tex2d(filter, filtcoord);
return illum * shadow;
```

The rendering target is the light map, which is multiplied with the BRDF texture during the rendering from the camera.

Rendering a small quadrilateral in texture space may result in shadow leaking, that is darkening show up in surface regions that are close to the shadow photon in texture space, but not necessarily in world space. To eliminate these artifacts, the texture atlas should be carefully defined. As shown by figure 3, we should introduce void regions between valid texture regions if shadow leaking between these regions should be prevented. The pixel shader code can identify the void regions and reject the rendering of shadow photon splats here.

4. Results and further improvements

The speed of the algorithm mainly depends on the resolution of the caster and receiver maps, and on the complexity of the geometry. The higher the resolution the better the shadow quality. The right image of figure 4 has been generated with 256×256 resolution. It runs about 100 FPS on an NV6800GT with 2.6GHz AMD CPU for the dragon object enclosed by a box. If 64×64 resolution is enough, even 200 FPS can be achieved.

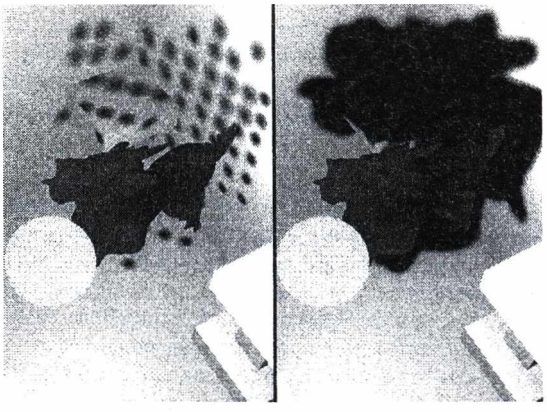


Figure 4: *Controlling of the size and the density of the splat quads as the function of the caster and receiver distances from the light source. Left: the resolution of the caster and receiver maps is 8×8 pixel. Right: the very same scene rendered with 256×256 resolution maps. The tail of the dragon is close to the left wall, thus shadow photon splats are small but strong here. The head of the dragon is close to the light source and far from the wall, thus its shadow photon splat is blurry, that is weak and large.*

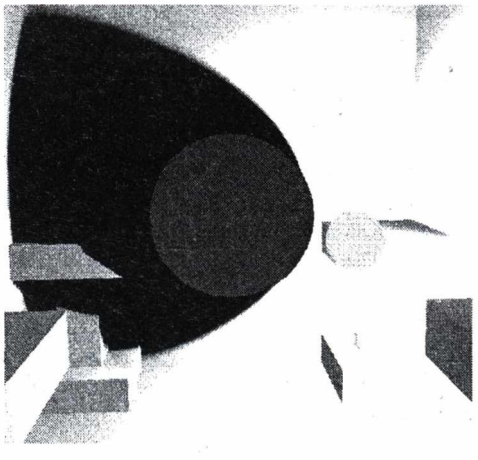


Figure 5: *This image has been rendered using 256×256 resolution caster and receiver maps at 80 FPS on NV6800GT/2.8GHz.*

4.1. Combination with hard shadow algorithms

Note that according to equation 1 the size of the splats is proportional to the size of the light source. Small light sources lead to small splats, which means that we need many shadow photons to constantly fill up the fully occluded shadow regions. Many shadow photons, in turn, correspond to high resolution caster and receiver maps, which degrade performance. Let us recognize that the proposed algorithm is good

for large area light sources for which hard shadow algorithms are bad, and poor for small and point sources for which even hard shadows are accurate. This recognition leads to the combination of the proposed method with a hard shadow algorithm, for example, with hardware supported z-buffer shadows.

The combined algorithm uses the classical z-buffer shadow algorithm to render the fully occluded umbra, and the new algorithm to generate the penumbra. If we use the new algorithm also to compute the umbra simultaneously to the hard shadow algorithm, the result is still correct, but we have to carry out unnecessary computations. The performance can be increased by letting the new algorithm place shadow photons only around the silhouette edges. To recognize these places, when the receiver map is generated, the caster map is read not only in a single pixel but the neighboring pixels as well. If the shadow caster is seen in all neighboring pixels and is closer than the receiver, then the point occluded by the caster in the center pixel is assumed to be in the umbra, thus no shadow photon is generated here. Instead, this point is handled by the hard shadow algorithm. However, when the neighborhood of the pixel contains pixels where no caster is seen, the shadow photon is generated as discussed earlier.

5. Conclusions

This paper presented a GPU algorithm for realistic soft shadow generation. The algorithm can render inter object shadows. The algorithm is particularly effective when combined with a hard shadow algorithm, such as the GPU supported depth mapped shadow method. The algorithm runs at high frame rates and can be applicable in games. The drawback of the method is the requirement of Shader 3.0 support.

Acknowledgements

This work has been supported by OTKA (T042735), GameTools FP6 (IST-2-004363) project, by the Spanish- Hungarian Fund (E-26/04).

References

1. Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. Approximate Soft Shadows win an Image-Space Flood-Fill Algorithm. *Computer Graphics Forum*, 23(3):271–279, 2004. 2
2. Jukka Arvo and Jan Westerholm. Hardware accelerated soft shadows using penumbra quads. *Journal of WSCG*, 2004. 2
3. U. Assarsson and T. Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 2003. 2
4. Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *Eurographics Symposium on Rendering*, 2003. 2
5. Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François X. Sillion. A survey of realtime soft shadow algorithms. In *Eurographics Conference. State of the Art Reports*, 2003. 2
6. F. Kirsch and Doellner J. Real-time soft shadows using a single light source sample. *Journal of WSCG*, 2003. 2
7. T. Kovács and Gy. Antal. Soft-edged stencil shadow in CAD applications. In *Harmadik Magyar Számítógépes Grafika és Geometria Konferencia*, 2005. to appear. 2
8. Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen, and Tomas Akenine-Möller. Soft shadow volumes for ray tracing. In *ACM SIGGRAPH*, 2005. 1
9. C. Wyman and C. Hansen. Penumbra maps. In *Eurographics symposium on rendering*, 2003. 2
10. Zhengming Ying, Min Tang, and Jinxiang Dong. Soft shadow maps for area light by area approximation. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, 2002. 1

Soft-Edged Stencil Shadow in CAD applications

Péter Tamás Kovács,[1] György Antal[2]

[1] Archi-Data Ltd.
[2] Faculty of Informatics, Eötvös Loránd Science University

Abstract

This paper describes a shadow generation technique that generates soft edges of shadow boundaries. The algorithm works in object space and based on the shadow volume method. We do not claim that the algorithm generates physically plausible shadow boundaries, umbras and penumbras. However, the generated images are visually pleasing and compared with the shadow volume algorithm that generate only hard shadow boundaries the computational overhead is negligible and the method works in real-time. Additionally, we extend our algorithm and overcome a typical artifact of this type of approaches, namely the halo, which appear at light intensity discontinuities.

1. Introduction

Creating shadows in interactive and real-time applications has been subject to serious investigation for years. The ever increasing computational power of the hardware makes it possible that nowadays we are witnessing a shift of focus from the performance centric approach to algorithms that deem it proper that the quality is more important. Special effects like realistic water, fire, smoke, alias-free edges, accurate shadow boundaries and soft shadows slowly become commonplace. However there is a performance cost when the rendering system supports these effects, no wonder that CAD applications rarely implement them. For the user of a CAM software this may be acceptable, but in architectural design there is a real need for photorealism. Consider the situation in which the architect is able to quickly sketch a house with garden and a fence or a room interior with furniture according to the preference of the customer and he is instantly able to show the image to the customer that resemble the real product as accurate as possible. This was one of the motivations which inspired the application of soft shadow into our CAD application. However, we have wanted a solution which causes little performance drop and suits well to our well tested shadow volume algorithm.

Shadow casting [SzK03] [SzM04] is crucial for human perception. To corroborate this belief Mamassian at al. [Mam98] undertook different psychophysical experiments and concluded that shadow helps to

- understand the relative position of an object from the shadow receiver and it helps to estimate the object size (see figure 1)
- understand the directly not visible geometry of an occluder (see figure 2)
- understand the geometry of the receiver (see figure 3).

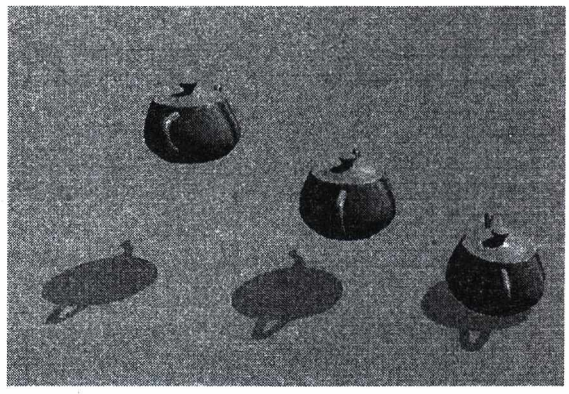


Figure 1: *Object relative position from the ground is revealed.*

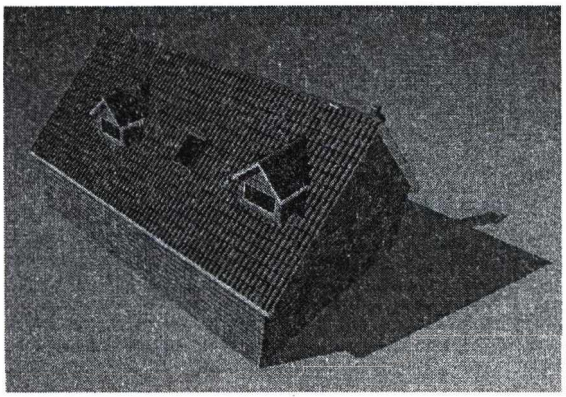


Figure 2: *Hidden geometry of the object is revealed.*

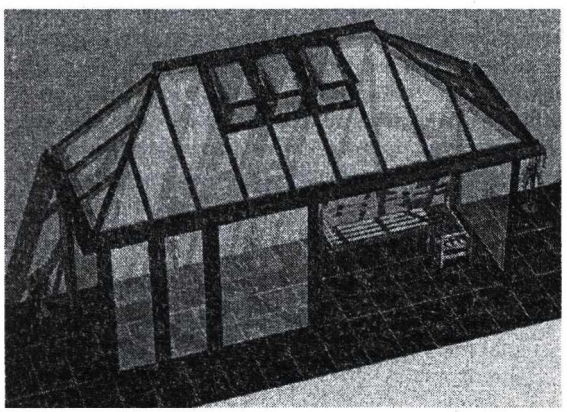


Figure 3: *The geometry of the ground is revealed.*

2. Hard shadows and soft shadow

The importance of shadow is beyond question, however the fixed function graphics pipeline of the video card does not consider the shadowing factor. Considering a usual point light source – which is a preferred lighting term used by the graphics pipeline – a point on the ground is either lit or is in shadow, which means the light source is occluded by an object. This is a binary relationship, which results hard shadow (see figure 4)

There are a number of approaches for shadow generation of this type.

Per-vertex shadow

This kind of method requires high tessellation level of shadow receivers. Before polygon rasterization, it precalculates how much light a vertex receives, determines the vertex color and stores it with the vertex position. The actual rendering of the graphics pipeline is fast, since the lighting is disabled and the GPU is free from any illumination calculation. This approach is most useful in radiosity settings, in which after a costly precalculation stage the walkthrough of the scene should be fast. However, the method cannot be used in dynamic environments in real time.

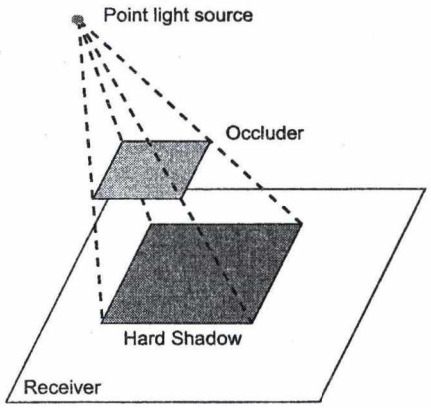


Figure 4: *Hard shadow of a point light source.*

Lightmaps

The idea is the same as before, but now the light contribution is stored in a small resolution texture and not in the vertices. Rendering of the high resolution diffuse textures are modulated by these lightmaps. The application of lightmaps was common in '96 when the first Quake engine appeared, but the significance of this technique slowly dropped. This approach does not require heavily tessellated geometry, but it requires a lot of texture memory and as before, it cannot be used in dynamic scenes.

Projected shadow

The projected shadow algorithm works by generating the silhouette of the shadow caster object into a texture. Then project this texture from the light's point of view to the receiver. This receiver is usually the ground plane. The method requires the categorization of objects to occluders or receivers, and therefore casting self-shadow are not easily performed.

Shadow maps

Probably, the most popular shadow generation algorithm is shadow mapping [Wil78], which is certified by the fact that it can handle shadows of dynamic scenes in real time. It works by first generating a depth image from the light's point of view. This is called depth map or shadow map. Rendering a pixel of the image from the camera's point of view invokes a query about the shadow status of the pixel. The pixel P (from the camera space) is transformed to the light's space P' and the Z value P'_z in the light's coordinate system is determined. This P'_z is then compared to the corresponding shadow map pixel that contains the Z value that is actually visible to the light source. If the

Z value of the shadow map is less than the transformed Z value P_z' the point P is in shadow.
It is a straightforward and simple algorithm, which does not generate any additional scene geometry. This is the main advantage. On the other hand, since shadow map is a discretized representation of the occlusions seen from the light source, it is not free from aliasing artifacts. The limited bit precision of the Z buffer can raise Z-fighting problems. And finally, omni-directional light sources pose another problem, since it is not possible to generate the image from the light's point of view with 360 degrees view angle. The most straightforward remedy is using a cube map and rendering 6 shadow maps, but it costs too much. If some kind of distortion at the edges is acceptable, dual-paraboloid shadow maps can be used, which only requires 2 shadow maps to render.

Shadow volumes

This method [Cro77] provides accurate hard shadows and it is also quite popular due to one of its most famous advocate John Carmack, who uses stencil shadow volumes extensively in his impressive Doom III engine. Shadow volumes (or stencil shadows) are not easy to implement and the naive implementation suffers from heavy pixel fillrate problem [Llo04]. However it gains popularity in professional games and applications fast. Since our algorithm is based on shadow volumes, we dedicate a separate subsection for it.

Hard shadow generation algorithms are popular in games of today, but abstract light sources like point, spot and directional lights do not exist in reality, since it is not possible that a finite measure of light energy could concentrate in a space of zero measure. Since all real lights have some extent (sun, filament lamp, luminaries), their shadow cannot be simulated by point lights.

Soft shadows [Has03] are generated from spatially extended light sources. In this situation (see figure 5) the fully shadowed area is called *umbra*. Every point of the umbra is hidden from the light source and therefore cannot see any part of it. There is a smooth transition from umbra to the fully lit region, this in-between area is called *penumbra*. Surface points of this area can view part of the light source. In this scenario the *shadow* is formulated by the union of the umbra and the penumbra.

Sometimes the hard edged shadow can be mistakenly misinterpreted as a new geometric object, which can never occur with soft shadows.

3. Shadow volume algorithm

We briefly detail the shadow volume [Eng03] algorithm in this section, since our algorithm is based on this method.

The main idea of the shadow volume algorithm is determining the areas of the world that are in shadow using shadow volumes.

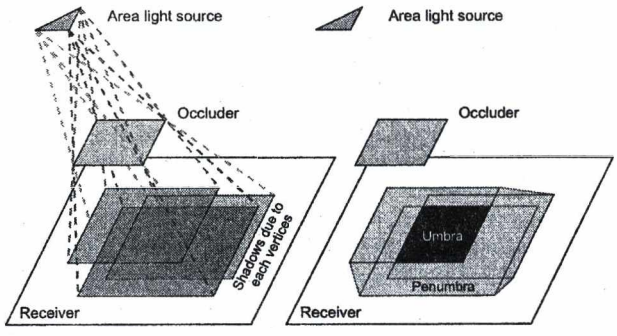


Figure 5: *Area light source generates umbra and penumbra.*

The shadowed areas are those that are behind the occluder from the light point of view. If we look at the occluder, we can determine the edges that form its contour from that view. If we extrude the object away from this point, we get the shadow volume, and everything inside the shadow volume is in shadow.

Now, from the eye's point of view, we have to determine the actual pixels that are in shadow. We start a ray from the eye, and start counting how many times do we enter or leave a volume. If we have entered more volumes than we have left when we hit an object, then we are in shadow (see figure 6).

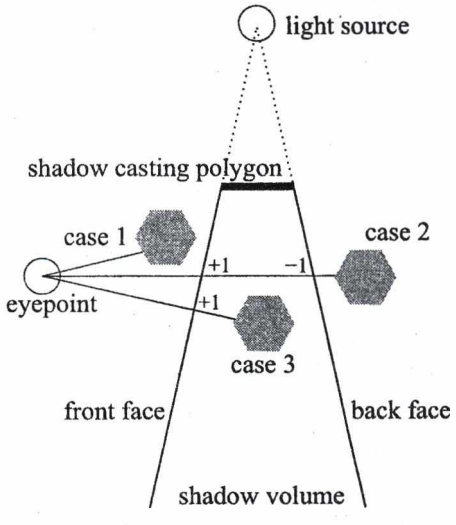


Figure 6: *The shadow volume counting. In case 1 and case 2 the pixel seen by the camera is not in the shadow volume. In case 3 the pixel is in shadow since the stencil value is more than zero.*

Fortunately, we actually don't have to start rays and com-

pute collisions with the volumes, because all this can be simulated using the stencil buffer. First, we render the scene with ambient lighting only, with depth buffer enabled. Then, we switch depth buffer writing off (but leave depth testing enabled), clear the stencil buffer to zero and configure the stencil in the following way: whenever a pixel passes the depth test, we increment the stencil value. Using this configuration, we render the front faces of the shadow volume. This equals with counting how many times do we enter a volume. Then, we configure it so that every pixel passing the depth test decrements the stencil value, and render the back faces of the shadow volumes. That is, we count the number of shadow volumes we leave, and subtract this value from the previous one. At this point, pixels having a stencil value more than zero are in shadow.

Now we configure the stencil not to increment or decrement in any case, but enable stencil test, that is, a pixel gets updated only if it passes the criterion which will be equality with zero in our case. Then, we switch off ambient lighting, and switch on our directional light, and render the scene again, using additive blending. As a result, we have the final shadowed image on the screen.

The method described here is called depth-pass, because of the stencil configuration used. Unfortunately this method does not work when the eye is inside a shadow volume, as it assumes that we are not in shadow when counting volume hits from the eye. Shadows will be inverted when entering a volume with the eye.

The artifacts which appear when the camera is in shadow can be avoided by using the depth-fail technique, which was independently presented by John Carmack [Car99] and Bill Bilodeau [Bil99]. †

4. Soft shadows methods

Generating soft shadows (see figure 5) is quite a complicated task.

First of all, it is not enough to determine the shadow silhouette from only one point (usually the center), since this silhouette can be different for each point of the extended light source. Identifying all parts of the occluders which can be visible from any point of the light source is algorithmically much more daunting than identifying it only from the center. Since very few methods can handle this problem accurately, no wonder that very few methods can claim to be physically plausible. However, we do not mind fake soft shadows if they serve images at interactive frame rates. Note also that in fast moving animations the shadow errors are usually less observable. A common failing example of fake soft shadow methods occurs in the situation when a large light source is placed directly above the occluder. In this case the physically exact image does not contain totally dark umbra regions. It seems to happen that simulating this effect correctly is not beyond the reach any real-time soft shadow algorithm.

On the other hand, shadows from several light sources pose another problem. In contrast to point lights, where the shadow area is the simple union of all shadows, the area lights can produce complete dark area (umbra) on surface region that none of the light sources blocked completely (they belonged to the penumbras).

The most physically accurate solutions are based on ray-tracing, however these are not usable in our CAD context. Apparently there is no single method that can render physically correct soft shadows in real-time for any dynamic scene. All the methods have one or other drawbacks. However there are some approaches which look promising. We name a few of them here. At first we consider the shadow map based then the shadow volume based approaches:

Combining several shadow maps
One of the first method [Her97] regularly places sample points on the light source and generates binary occlusion maps from these points. The binary maps are then combined into an attenuation map. This is repeated for each shadow receiver geometry, therefore this method requires the classification of the scene into occluders and receivers beforehand. However with 8 samples it still shows visible artifacts. Usually the combination of 64 or more samples gives nice result. The extension of this method is the Layered Attenuation Maps [Agr00] which generates only one attenuation map for all shadow receivers.

Heidrich's method
This method [Hei00] works by not only computing the shadow map, but a special map called visibility channel, which contains the percentage of visible light source area. This method is best suited for linear lights, on which shadow maps and visibility channel are generated for the two endpoints of the light source. The method was later extended to support polygonal light sources by Ying [Yin02].

Single sample soft shadows
This technique was developed originally for ray-tracing [Par98] for generating soft shadows using only a single shadow ray sample for a light source. This method was later extended to support graphics hardware by Brabec.

Percentage-closer soft shadow
It is a relatively new method [Fer05] published in Siggraph 2005 based on an old idea of percentage-closer filtering (PCF) whis is usually used to overcome the aliasing defect of ordinary shadow maps. This method solves the problem of large light sources and it produces umbras that do shrink as penumbras grow.

Combining several shadow volumes
The same brute force technique can be done with shadow

† This explain why the game developer community calls the depht-fail technique as *Carmack's Reverse*.

volumes alike with shadow maps. The drawback again is that this technique requires at least 64 or 128 samples to generate good-looking soft shadows.

Plateaus and Smoothies

Two object space soft shadow algorithms, which aim to extend the shadow volumes of the occluders by different geometries (cones, planes). Unfortunately, these methods generate only the outer penumbra, therefore occluders always has an umbra, even when using a very large light source.

Penumbra wedges

Möller [Mol04] gave forth another extension of shadow volumes which has a fancy promise of being a real object based quantatization-free soft shadow volume algorithm. Instead of one quad, the method constructs a wedge for each silhouette edge of the occluder. The wedge contains a front face, a back face and two side faces. These shadow volume objects need special treatment when rendered to the shadow buffer. The first version of the method handled only spherical light sources, but it was later extended for polygonal lights, where even the light source intensity can come from an animated texture. The current version of the method uses vertex and pixel shaders and achieves real-time performance with moderately complex scenes.

5. Soft-edged stencil shadow

Our algorithm is based on the shadow volume approach and we incorporated techniques from the work of [Bre03] and [Mit04], but the greatest influence on our work is caused by Shastry [Sha05].

The main idea is to use the contents of the stencil buffer as a shadow mask, but blur it before using. As we cannot directly blur the contents of the stencil buffer, we have to modify the original method in many aspects. The method consists of the following steps: Render the scene to a texture using ambient lighting, with depth writing and testing on. Switch off depth writing, configure stencil as mentioned at the simple shadow volume algorithm, and render the shadow volumes. Reset the stencil configuration. Then switch to another texture, but keep the depth/stencil buffer of the previous pass. Render a full-screen quad with stencil test enabled. Now we have the contents of the stencil buffer in the texture what we call a shadow mask. We do a blur on this mask in order to get the soft edges. Then we render the scene with diffuse and specular lighting only. Finally, using a pixel shader we combine the ambient image, the diffuse-specular image and the blurred shadow mask in the following way: FinalColor = Ambient + ShadowMask * DiffSpec

In our implementation we used a horizontal and a vertical blurring pass on the shadow mask.

We used the following pixel shader for the horizontal blur filter. The vertical blur shader is very similar.

```
extern float shadowMaskSizeX;

#define width 4

struct VS_OUTPUT
{
   float4 Position : POSITION0;
   float2 Texcoord : TEXCOORD0;
};

sampler shadowMaskSampler: register(s0);

float4 main(VS_OUTPUT IN) : COLOR0
{
   float4 sum = 0.0;

   for (int x = -width; x <= +width; x++) {
        sum += tex2D(shadowMaskSampler, IN.Texcoord
            + float2(x / shadowMaskSizeX, 0));
   }

   return(sum / (width * 2 + 1));
}
```

Figure 8: *Shadow mask before blur, after horizontal blur and after horizontal and vertical blur.*

Unfortunately this method - just like the one of Shastry for shadow map - suffers from the halo effect. If an object that is lit if placed before an object in shadow will have a bright contour, the so called halo. The same applies for the reversed case.

The more advanced method presented next successfully avoids the halo effect.

6. Haloless soft-edged stencil shadow

What is the main cause of the halo effect? At the edge of a lit object, the bright area is blurred onto the darker object in the back, and in the same way, dark is blurred back onto the lit surface. So somehow, we should avoid blurring. But if we don't blur, we get back to the original hard shadow.

But there is a way to detect whether we should blur or not. The halo is caused by blurring shadow mask values that correspond to different objects, which are close together on the screen, but possibly far away in world space. These different objects can be detected using their depth values. In theory they can be detected by rendering objectIDs to another rendertarget. However, we have the depth values for free, since without any computation the z-buffer is generated in the first pass. So we use the depth buffer in the blurring pass, and don't blur when the difference in depth values exceeds a given threshold. To be more exact, we test the depth value of every pixel in the filter kernel against the depth value of the center of the filter, and use the center color instead, if the depth difference is too large. This way, we avoid the halo effect, and still get the blurring where appropriate.

The horizontal blur pixel shader of the haloless method is

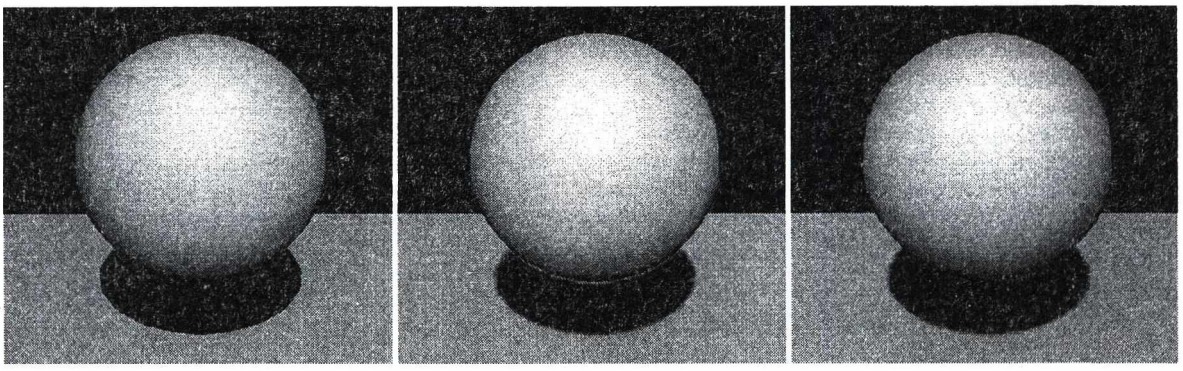


Figure 7: *Simple test scene with hard shadow, soft-edged shadow and halo-free soft-edge shadow.*

```
extern float shadowMaskSizeX;
extern float depthThreshold;

#define width 4

struct VS_OUTPUT
{
   float4 Position : POSITION0;
   float2 Texcoord : TEXCOORD0;
};

sampler shadowMaskSampler: register(s0);
sampler depthMap: register(s1);

float4 main(VS_OUTPUT IN) : COLOR0
{
   float4 sum = 0.0;
   float4 centerColor = tex2D(shadowMaskSampler, IN.Texcoord);
   float centerDepth = tex2D(depthMap, IN.Texcoord).r;

   for (int x = -width; x <= +width; x++) {
        float sampledDepth = tex2D(depthMap, IN.Texcoord
           + float2(x / shadowMaskSizeX, 0)).r;
        sum +=
           abs(sampledDepth - centerDepth) > depthThreshold)
           ? centerColor :
           tex2D(shadowMaskSampler,
           IN.Texcoord + float2(x / shadowMaskSizeX, 0);
   }

   return(sum / (width * 2 + 1));
}
```

Figure 9: *Shadow mask before blur, after horizontal blur and after horizontal and vertical blur. Note that shadow mask is not blurred at the upper edge, because here the occluder meets the shadow receiver.*

7. Results

The presented algorithm has been implemented in C++ in DirectX environment. We made our measurements on a Pentium IV 3 GHz computer with ATI Radeon X600 graphics card having 128MB VGA memory onboard. The rendering was performed windowed and the window resolution was 640*x*480.

We tried our algorithm with different scenes from which we present only two in this paper. Both scene consists only 1 light source, but extending the algorithm for multi-light environments is trivial. We have chosen a simple scene of a single sphere for occluder to demonstrate the basic idea of the algorithm. This scene contains 772 triangles. A more complex scene shows an interior with a table in the center. It contains 23K triangles. Each scene comes from DirectX .X files.

	simple scene	interior scene
without shadow	1175 fps	642 fps
hard shadow	825 fps	274 fps
soft-edged shadow	79 fps	68 fps
halo-free soft-edged shadow	51 fps	45 fps

Table 1: *Rendering speed for the test scenes.*

We present time measurements in table 1 for the cases. We chose walkthrough animation, that means that the objects do not move. Therefore, the measured times are free from complexity of the shadow volume generation.

8. Conclusions

This paper has presented a new rendering technique for rasterization hardware that generates a special type of soft shadows we called soft-edged shadow. The algorithm is based on shadow volume, which is very popular in game-development. After hard shadow mask calculation we applied two passes of blurring. One of the drawbacks of blur filters is the halo effect, which until now restricted the

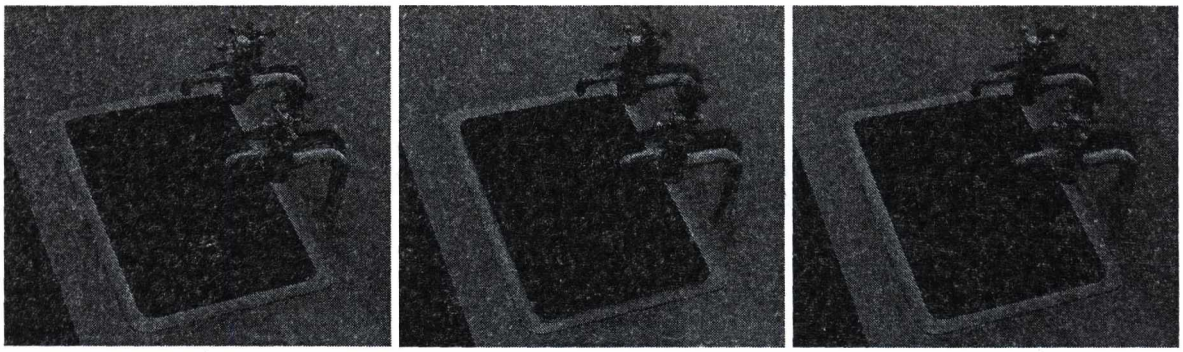


Figure 10: *Interior scene with hard shadow, soft-edged shadow and halo-free soft-edges shadow.*

real usage of soft shadows in games. We proposed a solution to overcome this artifact. Therefore, the main contribution of this paper are twofold: we applied the soft-edge technique to shadow volumes and created halo-free soft-shadows. Our results shows that the new extension does not require much more computational power compared to the soft-edged method and the elimination of the disturbing halo effect worths that extra time.

Acknowledgements

A big thank you goes to *Ferenc Csonka*, who were very helpful in reviewing the publication and gave invaluable comments.

This work has been supported by ArchiData Ltd. The scenes have been modelled in CarolineCAD using a modified GDL Geometric Description Language that was originally developed by Graphisoft.

References

[Mam98] Pascal Mamassian, David C. Knill, Daniel Kersten The Perception of Cast Shadows. *Trends in Cognitive Sciences vol. 2* (8), 288-295, 1998. 1

[Eng03] Hun Yen Kwoon The Theory of Stencil Shadow Volumes. *ShaderX2 Introductions & Tutorials with DirectX 9*, Wolfgang F. Engel (editor), *Wordware Publishing, Inc.*, 2003. 3

[Bre03] Flavien Brebion Soft Shadows. *ShaderX2 : Shader Programming Tips and Tricks with DirectX 9.0*, Wolfgang F. Engel (editor), *Wordware Publishing, Inc.*, 2003. 5

[Sha05] Anirudh.S Shastry Soft-Edged Shadows. *http://www.gamedev.net/reference/articles/article2193.asp*, 2005. 5

[Mit04] Jason L. Mitchell Poisson Shadow Blur. *ShaderX3 : Advanced Rendering with DirectX and OpenGL*, Wolfgang F. Engel (editor), *Delmar Thomson Learning*, 2004. 5

[Wil78] Lance Williams Casting curved shadows on curved surfaces. *ACM Siggraph conference*, pp. 270–274, 1978. 2

[Cro77] Franklin C. Crow Shadow algorithms for computer graphics. *ACM Siggraph conference*, pp. 242–248, 1977. 3

[Car99] John Carmack Carmack's Reverse (CarmackOnShadowVolumes.txt) *http://developer.nvidia.com/attach/6832*, 1999. 4

[Bil99] Bill Bilodeau, Mike Songy Real Time Shadows. *Creativity'99, Creative Labs, Inc. Sponsored game developer conferences, Los Angeles, California, and Surrey, England*, 1999. 4

[Has03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion A survey of Real-Time Soft Shadows Algorithms. *Eurographics conference*, State-of-the-Art Report, 2003. 3

[Her97] Michael Herf Efficient generation of soft shadow textures. *Technical Report CMU-CS-97-138*, Carnegie Mellon University, 1997. 4

[Agr00] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, Laurent Moll Efficient image-based methods for rendering soft shadows. *ACM Siggraph conference*, pp. 375–384, 2000. 4

[Hei00] Wolfgang Heidrich, Stefan Brabec, Hans-Peter Seidel Soft shadow maps for linear lights high-quality. *Rendering Techniques, 11th Eurographics Workshop on Rendering*, pp. 269–280, 2000. 4

[Yin02] Zhengming Ying, Min Tang, Jinxiang Dong Soft

shadow maps for area light by area approximation. *10th Pacific Conference on Computer Graphics and Applications*, pp. 442Ű443, 2002. 4

[Par98] Steven Parker, Peter Shirley, Brian Smits Single sample soft shadows. *Technical Report UUCS-98-019*, Computer Science Department, University of Utah, 1998. 4

[Mol04] Tomas Akenine-Möller, Ulf Assarsson Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges. *Thirteenth Eurographics Workshop on Rendering*, 2003. 5

[SzK03] Szirmay-Kalos László, Antal György, Csonka Ferenc Háromdimenziós grafika, animáció és játékfejlesztés. *ComputerBooks*, 2003. 1

[SzM04] Márton Szabó Hardware generated shadows. *(CESCG) Central European Seminar on Computer Graphics*, Budmerice, 2004. 1

[Llo04] Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, Dinesh Manocha CC Shadow Volumes. *Eurographics Symposium on Rendering*, 2004. 3

[Fer05] Randima Fernando Percentage-Closer Soft Shadows. *ACM Siggraph conference*, Nvidia Exhibitor Tech Sketches, 2005. 4

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Local Fairing of Freeform Curves and Surfaces

Péter Salvi[1,2] and Tamás Várady[2]

[1] Loránd Eötvös Science University, Budapest
[2] Geomagic Hungary, Budapest

Abstract

After reviewing different approaches, a new algorithm is presented for fairing B-spline curves and surfaces. It is based on a special target curvature, *computed from the not-yet-faired curve or surface. The method is parameter invariant and local. It moves a single control point at a time, so to find a global optimum iterative methods with appropriate heuristics need to be applied. The results are illustrated by a few examples.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling

1. Introduction

Digital Shape Reconstruction (formerly called Reverse Engineering) is a fast growing area in Computer Aided Geometric Design, which deals with the creation of geometric models using measured data. In many practical applications of DSR, surface fairness is a crucial matter, in particular in the automobile industry. Although fairness does not have an exact mathematical definition, researchers agree that it is inherent to pleasing aesthetics, and that the curvature of fair surfaces must be distributed evenly. A wide range of graphical *interrogation methods* (e.g. curvature maps, isophote and reflection lines) has been developed to detect small surface artifacts, but even with these, today fairing is a laborious manual process that needs a lot of skill. This is why there is a natural need for (semi) automatic fairing algorithms.

There are various approaches that can make a surface more fair. (i) Variational methods integrate fairing into the surface approximation process. (ii) Postprocessing methods usually define some *fairness measure*, and try to minimize it by changing the existing surface geometry, while maintaining some constraints, such as the maximum deviation from the original surface. Here we will only deal with the latter approach that seems to be more useful in the DSR context. It is important to note, that though the faired surfaces should be smooth, the highly curved *features* of the original shapes must be preserved.

Since research on surface fairing methods inevitably involves research on two-dimensional curve (spline) fairing, here we will summarize the previous work on fairing both curves and surfaces, introducing and comparing the most important fairness measures and algorithms.

In Section 2 different concepts of fairness will be presented. Section 3 gives an overview on previous work in the literature. Section 4 presents our proposed algorithm, followed by test results in Section 5.

2. Fairness

Fairness may have different meanings in different applications. Depending on the requirements, even the same surface can be qualified as fair or unfair, however, there are general guidelines that can be applied to most of the cases.

One widely used criterion of fairness is the smoothness and smooth distribution of reflection lines. If a fair object was placed into a room lit by parallel lights, the reflections of the light source should bend smoothly and evenly over the surface. This effect can be simulated using computer graphics — the most modern modelling systems offer these kinds of rendering options.

Isophote lines are very similar to reflection lines, since their smoothness depends on the change of the first derivative of the surface. An isophote line is a set of surface points where the angle between the normal vector and the viewing direction is the same (within a given tolerance). Since this

map is much simpler than the previous, but reveals just about the same flaws, it is more often seen in practice. Another variant is the highlight band,[2] which can also be computed very fast.

While these maps simulate something that is visible to the eye under special conditions, there are others that only have a mathematical meaning, but still proved to be crucial in examining fairness. These are the curvature maps and the curvature combs, which depend on the second derivatives. Curvature maps colour-code the curvature values and can have various types (Gaussian, mean, minimum, maximum, etc.), curvature combs display the values as orthogonal straight line segments along a curve.

These methods are also called visual interrogation tools, because they help the user to find minor discontinuities or wiggles on the surface. Since these are found by looking at the *changes* of the map, they show flaws of one degree higher than the derivatives the tool depends on.

In order to create a fairing algorithm, it is common to define a *fairness measure*, i.e. a functional that represents the fairness of the surface. In other words, we can say that a surface S is fair, if

$$\mathcal{F}(S) < \tau$$

applies, where τ is a user-defined tolerance. One "classical" definition of fairness by Farin and Sapidis is as follows:

> A curve is fair if its curvature plot is continuous and consists of only a few monotone pieces.[4]

In the next section, we will review what sort of other alternatives exist.

3. Previous Work

In this section we introduce the most well-known or prominent measures and algorithms in the literature. We will also present curve-fairing methods, since most results for surfaces can be easily generalized from them.

As Roulier and Rando point out, we cannot hope to have a universal fairness measure or algorithm, but we should strive to create new ones, in order to give designers the freedom of choosing the most suitable algorithms for their tasks.[14]

3.1. Fairness measures

A natural measure for curve fairness is the strain energy, that is based on a drawing technique used in ship design, from the 18th century until today. To create a smooth curve, metal weights were placed at the interpolation points and a flexible spline was spanned between them. The resulting curve $\mathbf{c}$ minimizes the strain energy, yielding the measure

$$E = \int (\kappa(s))^2 \mathrm{d}s, \tag{1}$$

where $\kappa(s)$ is the curvature of the curve as a function of the arc length. This minimizes the mean curvature, while giving a penalty to the extreme values by squaring.[14]

Computing the curvature can be difficult, so it is often replaced by a simpler, parameter-dependent formula:

$$\hat{E} = \int (\mathbf{c}''(t))^2 \mathrm{d}t. \tag{2}$$

The third degree interpolating spline minimizes this value, but has the drawback that in cases where the parameterization substantially differs from the arc-length parameterization, fairness is not guaranteed, and unexpected results may occur.

Since neither (1) nor (2) penalizes the sign change of the curvature, curves faired by these measures may preserve unwanted inflections. Different variations of the interpolating spline were devised to avoid this, e.g. the spline in tension or the ν-spline.[4]

Moreton and Séquin introduced another measure, called Minimum Variation Curve, optimizing the variation of the curvature:[12]

$$E_{MVC} = \int (\kappa'(s))^2 \mathrm{d}s. \tag{3}$$

This has the advantage that it does not create unnecessary inflection points due to its convexity-preserving property.

The curve-fairing measures introduced so far all have their surface-fairing equivalents. Similar to the strain energy, in the surface case we can minimize the thin plate energy

$$\Pi_P = \iint_S a(\kappa_1^2 + \kappa_2^2) + 2(1-b)\,\kappa_1\kappa_2 \,\mathrm{d}S, \tag{4}$$

where κ_1 and κ_2 are the principal curvatures of the surface S and a, b are material-specific constants that usually take the values $a = 1$ and $b = 0$ or $b = 1$.[7]

This also has a simpler variant

$$\Pi = \iint_A S_{uu}^2 + 2S_{uv}^2 + S_{vv}^2 \,\mathrm{d}u\,\mathrm{d}v, \tag{5}$$

which is parameter dependent, so it can only be used safely for isometric parameterization.

Moreton and Séquin suggested an alternative measure based on the variation of the curvature:

$$\Pi_{MVS} = \iint_S \left(\frac{\partial \kappa_1}{\partial e_1}\right)^2 + \left(\frac{\partial \kappa_2}{\partial e_2}\right)^2 \mathrm{d}S, \tag{6}$$

that vanishes on spheres, cones and tori. Although this measure gives excellent results, it requires very complex computations.

3.2. Fairing algorithms

One of the simplest, widely used curve-fairing method is *knot removal and reinsertion* (KRR), originally conceived by Kjellander and later made local by Sapidis and Farin.[3]

If we define an order k B-spline as

$$\mathbf{x}(t) = \sum_{i=0}^{n} \mathbf{d}_i N_{i,k}(t), \quad t \in [t_{k-1}, t_{n+1}], \tag{7}$$

where $\mathbf{d}_i$ are the control points, $N_{i,k}$ are the B-spline bases and $T = (t_j)_{j=0}^{n+k}$ are the knots, then it is at most C^{k-2}-continuous at the knot points.

We can add knots to a B-spline in a way that its shape does not change. This can be unambiguously done by multiplying the control points with a matrix. On the other hand, if we take out a knot, we can only preserve the shape if the curve was originally C^{k-1}-continuous at that knot point.

So the problem is locating the control points of

$$\tilde{\mathbf{x}} = \sum_{i=0}^{n-1} \tilde{\mathbf{d}}_i N_{i,k,\tilde{T}} \qquad (\tilde{T} \subset T) \tag{8}$$

in such a way that $\mathbf{d} = A\tilde{\mathbf{d}}$ applies, where A is the knot insertion matrix.[8] This breaks down to an overdefined equation that can have several approximate solutions. Farin gives the most local solution for third degree B-splines, ensuring C^3 continuity at the knot by repositioning only one control point.[4]

This gives the idea of the KRR algorithm, i.e. to find, remove and then reinsert the knot where the third derivative has the largest discontinuity. The process may be iterated until a suitable end condition is met. Finding such a condition is not a trivial task. A vast range of heuristics can be applied, including best-first-search[8] and simulated annealing.[13]

Eck and Hadenfeld fix all but one control points and locally minimize the fairness measure

$$E_l = \int_{t_{k-1}}^{t_{n+1}} \left(\tilde{\mathbf{x}}^{(l)}(t)\right)^2 \quad l = 2,3 \tag{9}$$

while keeping the distance from the original curve under a δ tolerance:[3]

$$\max\{|\mathbf{x}(t) - \tilde{\mathbf{x}}(t)| \mid t \in [t_{k-1}, t_{n+1}]\} \leq \delta. \tag{10}$$

Because of the convex hull property, this is easily done by preserving the $|\mathbf{d}_r - \tilde{\mathbf{d}}_r| \leq \delta$ inequality:

$$\tilde{\mathbf{d}}_r^* = \mathbf{d}_r + \delta \cdot \frac{\tilde{\mathbf{d}}_r - \mathbf{d}_r}{|\tilde{\mathbf{d}}_r - \mathbf{d}_r|}. \tag{11}$$

Both of these methods have equivalents in surface fairing. The main disadvantage of the KRR algorithm is that removing a knot changes a whole line of control points in the other parametric direction, e.g. if we have a surface

$$X(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{d}_{ij} N_{i,k,U}(u) N_{j,l,V}(v), \quad (u,v) \in [u_{k-1}, u_{n+1}] \times [v_{l-1}, v_{m+1}], \tag{12}$$

where $U = (u_i)_{i=0}^{n+k}$ and $V = (v_j)_{j=0}^{m+l}$ represent the knots, removing a knot v_s means removing a knot from all of the B-splines $\mathbf{x}_i = \sum_{j=0}^{m} \mathbf{d}_{ij} N_{j,l,V}(t)$, where $i = k, \ldots, n$. Furthermore, the generalized KRR only ensures C^3 continuity in one parametric direction.

Hahmann proves that it is sufficient to remove and reinsert a knot in only three rows or columns of B-splines, thus the algorithm can be made local for surfaces.[8] However, C^3 continuity in only one direction is not satisfactory in real-life applications.

Hadenfeld proposed a fairing method using the measure (5), as above only one control point is moved at a time and the largest deviation is constrained from the original.[7]

In a recent publication[1] fairing was performed through the optimization of knot vectors; in our research, however, we preserve the original knots.

4. The New Algorithm

In this section we first sketch a fairing algorithm for curves, then we generalize it for surfaces. We can expect that the curvature comb of a fair curve is smooth, without any jumps or sudden changes. Therefore we can smooth the curve defined by the curvature comb's endpoints, which is practically the same as the evolute, due the $\kappa = 1/\rho$ equality. We will call the smoothed curve the *target evolute*.

Now we want to find a curve that is close to the original, but whose evolute is the target evolute. This also defines a fairness measure: the closer the evolute is to the target evolute, the fairer is the curve. Let $\mathbf{n}$ denote the normal and $\mathbf{e}$ the target evolute, then our fairness measure is

$$E = \int \|(\mathbf{c}(t) + \rho\mathbf{n}(t)) - \mathbf{e}(t)\|^2 dt, \tag{13}$$

assuming that the two curves have a common parameterization. The algorithm for finding the minimum of this functional will be presented later. Controlling the deviation from the original curve can be managed in the same manner as written in the previous section.

Since the evolute (and the curvature comb) may be self-intersecting, we use directly the curvatures instead, so our measure becomes

$$\hat{E} = \sum_i |\kappa(t_i) - g(t_i)|^2, \tag{14}$$

where g is the smoothed (target) curvature.

In the surface case the single curvature need to be replaced by the two principal curvatures. Let g_1 and g_2 be the target curvatures based on κ_1 and κ_2, then

$$\hat{\Pi} = \sum_i \sum_j (|\kappa_1(u_i, v_j) - g_1(u_i, v_j)|^2 + |\kappa_2(u_i, v_j) - g_2(u_i, v_j)|^2) \tag{15}$$

is a meaningful fairness measure.

4.1. Determining the target curvature

Any simple and fast smoothing method can be effectively used for defining the target curvature, for example averaging the consecutive sample points of the evolute. Smoothness of the target curvature is much more important than to be close to the evolute of the original curve, therefore we should use a loose sampling rate. Global averaging can remove parts of the curvature that represent features, so the user should be allowed to restrict the smoothing or edit the target curvature manually.

Another possibility is to fit a NURBS curve over the sampled points. For surfaces this leads to the solution of a system of linear equations that minimizes the functional

$$F(\mathbf{g}_q) = \sum_i \sum_j \|\mathbf{g}_q(u_i, v_j) - \kappa_q^0(u_i, v_j)\|^2 \quad (q = 1,2), \quad (16)$$

where κ_q^0 is a principal curvature of the original surface (here the curvatures are interpreted as surfaces over the parameter domain). To get smooth results, we should also minimize the curvature of the fitted surface:

$$\hat{F}(\mathbf{g}_q) = \sum_i \sum_j \|\mathbf{g}_q(u_i, v_j) - \kappa_q^0(u_i, v_j)\|^2 + \int_u \int_v \hat{\kappa_q}(u, v)\, \mathrm{d}u\, \mathrm{d}v, \quad (17)$$

where $\hat{\kappa_g}$ is the curvature of $\mathbf{g}_q$.

Having the target curvatures for our curves and surfaces, the next step is to modify the current entities in such a way, that their curvature gradually gets closer to the target.

4.2. Finding the optimal solution

For simplicity's sake here we present an iterative method that moves only one control point in every iteration. This has the advantage of locality in exchange for speed, but this drawback is countered by our choice of minimization algorithm — the downhill simplex method, which is simple and very fast.[13] An iteration consists of the following steps:

1. Select the next control point to move from a priority queue.
2. Minimize the fairness measure by moving the selected control point.
3. Calculate a new control point position if it falls too far from the original (see (11)).

Selection of the next control point has great influence on the quality of fairness. Selecting the control point where the largest deviation of the target curvature occurs is a natural choice. However, this can lead to a deadlock, if the same control point is chosen over and over again. A list of the recently moved control points may be kept in order to avoid this. Also, boundary control points should not be selected for most applications.

Other minimization procedures to find the global optimum can also be used, such as simulated annealing.

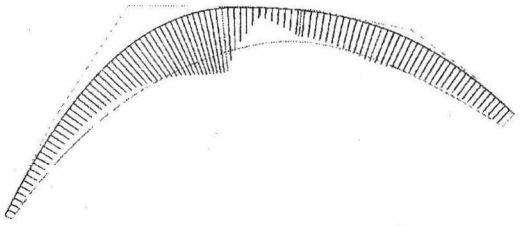


Figure 1: *The initial curve with its target curvature in green.*

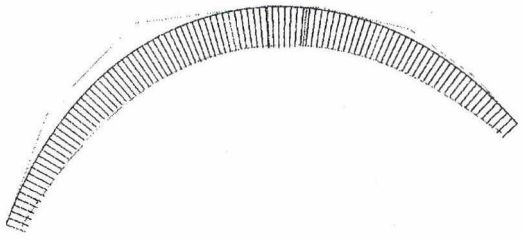


Figure 2: *The faired curve.*

5. Results

Figure 1 shows a curve before fairing. The original curvature comb and the original control polygon are also shown. The green line is a smooth version of the given curvature comb, this is the target curvature we want to approximate. Figure 2 shows the curve after fairing, the final curvature distribution is clearly much better, though it is not necessarily identical to the target function due to tolerance constraints and the final number of steps. Figures 3–4 highlight the effect of fairing through extrusion surfaces.

Figure 5 shows the actual and desired target curvature distributions for a car body surface element. Table 1 summarizes the numerical results. As we can see, there is minimal

Figure 3: *Extrusion surface of the initial curve.*

Figure 4: *Extrusion surface of the faired curve.*

Iteration	Fairness measure	Max. distance (mm)
0	0.901531	0.00000
200	0.244919	2.91302
400	0.223582	4.40624
600	0.215052	5.02049
800	0.201543	5.14342
1000	0.198280	5.85641

Table 1: *Fairing phases.*

improvement after 400 iterations, which justifies to stop iterating. Figure 6 shows the isophote lines of the original surface before and after fairing. Figure 7 is a deviation map to show where the largest positional modifications took place in order to get the final faired surface.

6. Conclusions

Fairing curves and surfaces is a complex problem. Unfortunately, the goal of generating perfectly fair shapes cannot be unambiguously formulated with mathematical terms,

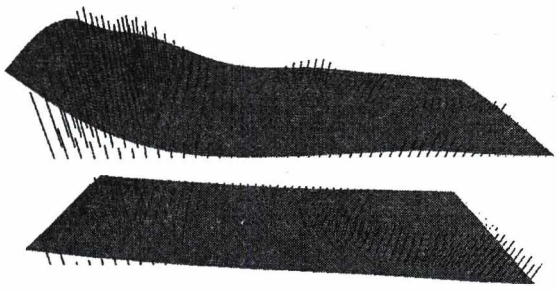


Figure 5: κ_1 *and* κ_2 *principal curvatures (teeth) and the target curvature functions.*

Figure 6: *Isophote map of the initial and final surface.*

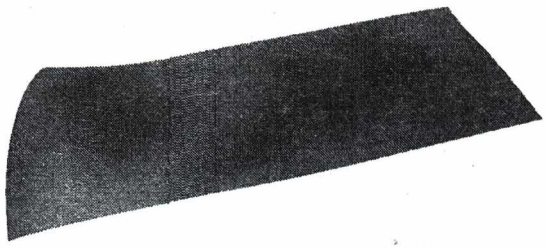


Figure 7: *Distance map showing deviations from the original surface.*

and there are many alternatives. Authors propose a method where a smoothed target curvature function is approximated step by step. The algorithm modifies a single control point at a time. Applying an iterative strategy, the global shape is optimized until the magnitude of the improvement becomes negligible. Our future research is going to replace the current iterative methods by direct methods, that can efficiently lead to fair curves and surfaces.

Acknowledgements

This research has been conducted within Geomagic Hungary, Ltd., Budapest. The first author is a PhD student at the Information Faculty of the Loránd Eötvös Science University.

References

1. B. Aszódi, Sz. Czuczor, L. Szirmay-Kalos, *NURBS Fairing by Knot Vector Optimization.* Journal of WSCG, Volume 10, pp. 19–26, 2004.
2. K-P. Beier, Y. Chen, *The Highlight Band, a Simplified*

Reflection Model for Interactive Smoothness Evaluation. In: N. S. Sapidis (Ed.), Designing Fair Curves and Surfaces, pp. 213–230, SIAM, ISBN 0-898-71332-3, 1994.

3. M. Eck, J. Hadenfeld, *Local Energy Fairing of B-Spline Curves*. Computing Supplement 10, pp. 129–147, 1995.
4. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide*. Academic Press, 5th edition, 2002.
5. G. Farin, J. Hoschek, M.-S. Kim (Eds.), *Handbook of Computer Aided Geometric Design*. North-Holland, 2002.
6. J. Hadenfeld, *Fairing of B-Spline Curves and Surfaces*. In: J. Hoschek, P. Kaklis (Eds.), Advanced Course on FAIRSHAPE, pp. 59–75, Teubner Stuttgart, ISBN 3-519-02634-1, 1996.
7. J. Hadenfeld, *Local Energy Fairing of B-Spline Surfaces*. In: M. Daehlen, T. Lyche, L. L. Schumaker (Eds.), Mathematical Methods for Curves and Surfaces, pp. 203–212, Vanderbilt University Press, ISBN 0-826-51268-2, 1995.
8. S. Hahmann, S. Konz, *Knot-Removal Surface Fairing using Search Strategies*. Computer Aided Design 30, pp. 131–138, 1998.
9. S. Hahmann, *Shape improvement of surfaces*. Computing Supplement 13, pp. 135–152, 1998.
10. M. Hoffmann, *Geometric and Solid Modeling. An Introduction*. Morgan Kaufmann Publishers, 1989.
11. H. P. Moreton, C. H. Sequin, *Functional Optimization for Fair Surface Design*. ACM SIGGRAPH Computer Graphics, Volume 26, Issue 2, pp. 167–176, 1992.
12. 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, pp. 123–159, SIAM, ISBN 0-898-71332-3, 1994.
13. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*. Cambridge University Press, 2nd Edition, ISBN 0-521-43108-5, 1992.
14. J. Roulier, T. Rando, *Measures of Fairness for Curves and Surfaces*. In: N. S. Sapidis (Ed.), Designing Fair Curves and Surfaces, pp. 75–122, SIAM, ISBN 0-898-71332-3, 1994.

Computing smoothness parameters

Gábor Renner

Computer and Automation Research Institute
Hungarian Academy of Sciences

Abstract

The key problem in reconstruction of complex free-form surfaces is to choose appropriate values for the parameters (degree, knots) of the continuous surface which approximates the set of measured data points, and also for the parameters of the reconstruction process (tolerances, smoothing weight). The paper describes a reconstruction strategy, which gradually increases the number of free surface parameters, while keeping the smoothness under control. We focus in the paper on the automatic adjustment of the contribution ratio of the distance term and the smoothness terms in the functional to be minimized. A new method for controlling the ratio - which is based on the gradient of the functional - has been developed and tested for reconstructing complex shapes.

Keywords: reconstruction, smoothing

1 INTRODUCTION

Reconstruction means to convert a large number of discrete data points generated by an appropriate measurement into continuous surfaces. The quality of the conversion depends on two basic factors: from functional point of view, the deviation of the surface from the data points should be as small as possible; from the point of view of aesthetic appearance, the smoothness and fairness of the surfaces are most important. Unfortunately, these two factors are in fundamental contradiction to each other. If a surface is `too smooth', it cannot meet tight tolerances at highly curved areas. On the other hand, if the surface is `too accurate', it will remain close to the data points, but unwanted oscillations may occur.

The most appropriate tool to solve the reconstruction problem is to minimize a functional, which depends on the free parameters of the surface and consists of two terms corresponding to the above two factors. The first term is the squared deviation of the surface from the data points, and expresses accuracy. The second term corresponds to the waviness of the surface. Although there are well-established methods to solve the functional minimization [1], [2], [3] one fundamental problem remains. The final shape of the surface strongly depends on the ratio between the two terms in the functional. The acceptable shape can be defined as the smoothest shape within a given tolerance. This can be achieved by carefully adjusting the ratio of the two terms in the functional, which is difficult to realize automatically.

We have developed a reconstruction strategy, which gradually increases the number of free surface parameters, while keeping the smoothness under control. Beyond this, the process improves the surface parameterization and handles the problem of weakly defined portions of the surface. The key question in this process is the automatic adjustment of the contribution ratio of the two terms in the functional. A new method for controlling the ratio - which is based on the gradient of the functional - has been worked out. The paper describes the strategy of the surface reconstruction and discusses the algorithmic details and its realization.

2 SURFACE FITTING BASICS

A detailed review of mathematical techniques to fit smooth surfaces over point clouds is given in our previous paper [4], which discusses the topic from three main points of view; the strategy of the reconstruction, the functionals to be minimized and the parameterization of the surface. Here we repeat only the basic ideas of fitting continuous surfaces on

a (usually large) set of data points, coming from measurements.

The surfaces to be fitted are represented by a tensor product B-spline, as the conventional CAD representation of free form surfaces

$$S(u,v) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} s_{i,j} N_i(u) N_j(v),$$

with control points $s_{i,j}$ and B-spline basis functions N_i and N_j, which are fully determined by their degrees and knot vectors. The unknown control points of the approximating surface are determined by minimizing the functional

$$\mathrm{F} = \mathrm{F}_{\mathrm{lsq}} + \lambda \mathrm{F}_{\mathrm{sm}}$$

where

$$\mathrm{F}_{lsq}(S) := \sum_{r=1}^{m} \left(S(u,v) - p_r\right)^2$$

$$\mathrm{F}_{sm}(S) := \iint_{\Omega} \left(S_{uu}^2 + S_{uv}^2 + S_{vv}^2\right) du\, dv$$

Here the first term $\mathrm{F}_{\mathrm{lsq}}$ the least square deviation of the data points p_r from the surface. The second term F_{sm} is an approximation of the thin plate energy, which is big if the surface is highly curved. In this way the first corresponds to accuracy, while the second to the smoothness of the surface.

If we substitute surface equation into the functional and assume, that for each data points a pair of surface parameter values *(u,v)* are associated, we can write $\mathrm{F}_{\mathrm{lsq}}$ and F_{sm} in matrix notation. We get

$$\mathrm{F}_{\mathrm{lsq}}(s) = s^T \mathbf{M}_{\mathrm{lsq}} s - 2 s^T \mathbf{N}^T p + p^T p$$

and

$$\mathrm{F}_{\mathrm{sm}}(s) = s^T \mathbf{M}_{\mathrm{sm}} s,$$

where the elements of matrix $\mathbf{N}$ are the basis function values at the parameter value of the data points, $\mathbf{M}_{\mathrm{lsq}} = \mathbf{N}^T\mathbf{N}$ and $\mathbf{M}_{\mathrm{sm}}$ is the quadratic matrix corresponding to the smoothness term.

Assuming degrees, knot values and smoothing weights are given and constant values, minimization of the above functional by setting its gradient to zero:

$$\frac{\partial \mathrm{F}}{\partial s} = 2(\underbrace{\mathbf{M}_{\mathrm{lsq}} + \lambda \mathbf{M}_{\mathrm{sm}}}_{\mathbf{M}}) s + 2\mathbf{N}^T p = 0,$$

results in the linear system

$$\mathbf{M}s = \mathbf{N}^T p,$$

which can efficiently be solved for the unknown control points $s_{i,j}$.

Unfortunately, for practical cases (industrial or biological surfaces) the above simplifying assumptions do not hold, which makes the reconstruction process a highly non-linear one, and difficult to automatize. In order to obtain a solution, first we fix a proper subset of the above quantities, and compute the best solution. Then the complementer subset is fixed and the rest is determined. A strategy is needed, what to fix and what and how to compute. In the next chapter we give a strategy for searching for an optimal solution in the high dimensional non-linear parameter space of the reconstruction problem.

3 STRATEGY OF RECONSTRUCTION

The process of reconstruction consists of multiple levels of iterations. It starts with the computation of an initial parametrization of the data points. Usually this is done by means of a reference surface [5], to which the data points are projected. An advanced method to create such a surface, which results in a valid parametrization, is given in [4].

Within the outer *cycles*, the degrees of freedom of the surface are increased, by refining the knot vectors. During the internal *iterations* the knot vectors are maintained and the weight of the smoothness functional is adjusted.

The key point in the above process is to keep the balance between accuracy and smoothness. In the next section we describe how to perform *one cycle*. Within this cycle we compute one best fit least-squares surface and several smoothed least-squares surfaces during the iterations for optimization of the smoothness weight. If the resulting *candidate* surface is within the given tolerance, it is accepted. Otherwise the process continues by reparametrizing the data points, adding new knots and starting a new cycle.

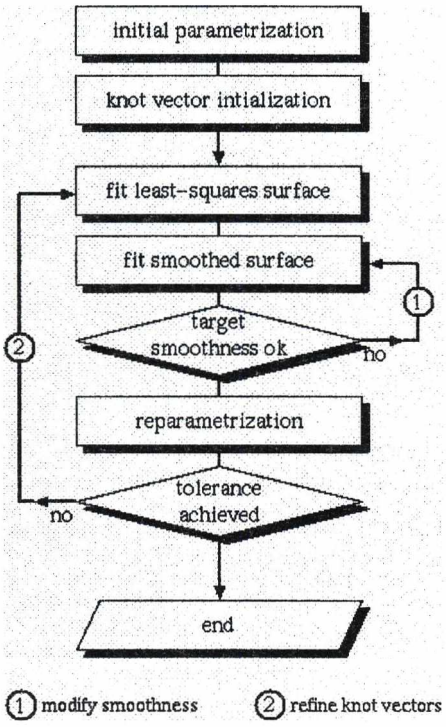


Figure 1: Flowchart of reconstruction

4 COMPUTING SMOOTHING WEIGHTS

Determination of the smoothness weight λ_i within the *i*-th cycle is done in a two steps:

1. At the beginning of the *i*-th cycle, a pure least-squares surface is fitted. This surface is used to compute the least-squares residual $r_{\mathrm{lsq},i}$ of the best approximation with the given knot vectors and point parameters.

2. Now we want to determine a smoothness weight $\overline{\lambda}_i$ in such a way, that the resulting surface is significantly improved, but not over dominated by the smoothness functional. This means, that we allow a growth of the residual $r_{\mathrm{lsq},i}$ to a value $r_{\mathrm{sm},i}$ in order to obtain a smoother result. We achieved good results by setting $r_{\mathrm{sm},i}$ to $1.15\, r_{\mathrm{lsq},i}$. Since the knot vectors and the point parameters are not modified during this cycle, the matrices $\mathbf{M}_{\mathrm{lsq}}$ and $\mathbf{M}_{\mathrm{sm}}$ remain unchanged and the search for $\overline{\lambda}_i$ can be done efficiently by incremental modifications of the linear system.

Figure 2 shows our experience concerning the dependency of the least-squares residual with respect to the smoothness weight at different cycles with a double logarithmic scale. Each graph can be divided into two almost horizontal regions and one middle region. In the first region smoothness has no effect, in the last region the shape is determined only by the smoothness and in between there is variation. The goal of the adaptive weight setting strategy is to keep the smoothness weight at the lower part of the middle region (working region), where smoothness has its positive effect and does not destroy accuracy.

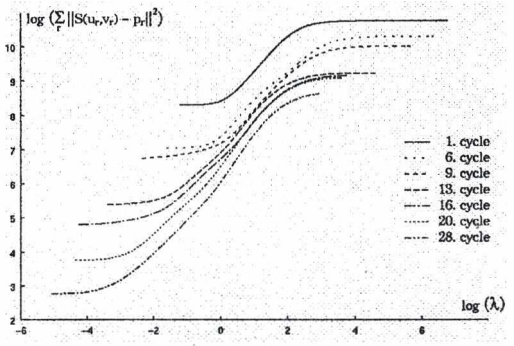


Figure 2: Residual vs. smoothing weight

Realizing the above process a method is needed to solve the non-linear problem of finding a smoothing weight which results in a surface with given least-squares residual (Step 2). We developed a method which works by computing the derivative of the gradient of the composite functional with respect to λ:

$$\frac{\partial \mathrm{F}}{\partial \boldsymbol{s}\partial\lambda} = \mathbf{M}_{\mathrm{sm}}\boldsymbol{s} + \mathbf{M}\frac{\partial \boldsymbol{s}}{\partial\lambda} = 0$$

or

$$\mathbf{M}\frac{\partial \boldsymbol{s}}{\partial\lambda} = -\mathbf{M}_{\mathrm{sm}}\boldsymbol{s},$$

which is a linear equation for the derivative $\partial \boldsymbol{s} / \partial\lambda$. This can be very efficiently solved, since the system matrix $\mathbf{M}$ and its factorization have already been computed in Step 1. By using this, Newton iteration

can be applied to determine the smoothing weight λ.

In order to successfully apply the Newton iteration we have to ensure that the starting point lies within the middle, working region of Figure 2. To detect this we evaluate the norm of the gradient by solving the above equation for $\partial s / \partial \lambda$ and check if it is greater then a threshold value. If this is not the case, we apply a simple search method, which uses two previous and the current values of the residual, instead of its gradient.

5 RESULTS

We have tested the fitting strategy and the automatic smoothing weight adjustment by reconstructing objects with various shapes. They are taken from industry, biology and medical applications. The measurements were performed by a 3D laser scanner (Modelmaker) produced by the company 3DScanners. The size of the point set varied between several thousand and several hundred thousand data points.

As an example, we discuss results obtained by reconstructing a part of a human knee bone, the end of the femur. The set of measured data point consists of 18158 points with a relatively high level of noise, and can be seen on Figure 3.

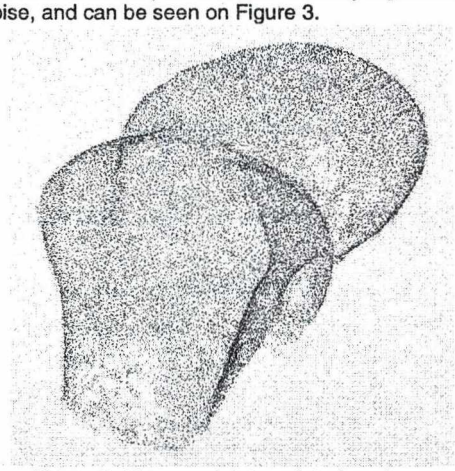


Figure 3: Measured data points for a bone

Bicubic B-spline surfaces were fitted to the data points of a part of the surface, which is in connection with the counterpart (tibia) during motion. Figure 4, 5, 6 show the influence of different smoothing weights on the shape of the generated surfaces. Color coded curvature maps are used to visualize the difference, and to evaluate surface quality.

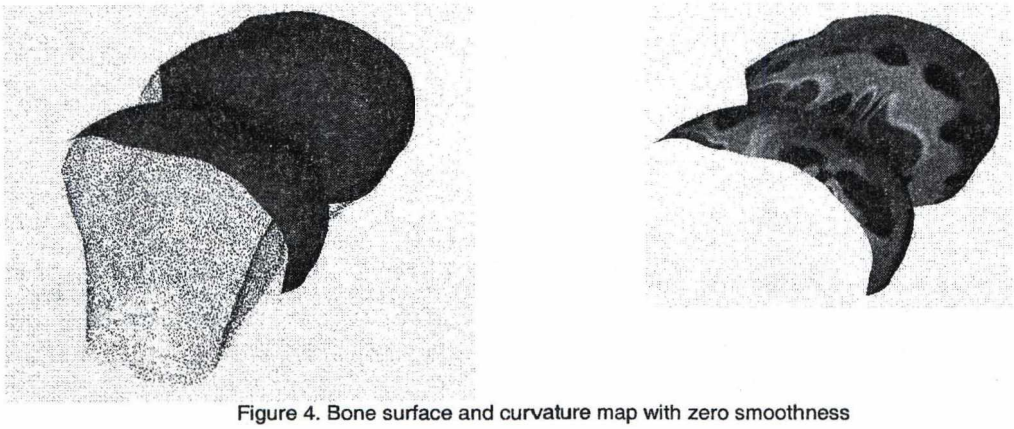


Figure 4. Bone surface and curvature map with zero smoothness

The surface shown in Figure 4. lies everywhere within the specified tolerance, which was set to 0.5 mm. The actual deviation is less than 0.38 mm. Because the smoothness weight was set to zero in this case, this is the least square error achieved by the selected surface parameters (degree, knots). However, the unsmooth behavior of the surface is not acceptable, especially along the borders and in the neighboring areas the shape exhibits a wide range of strongly varying curvatures.

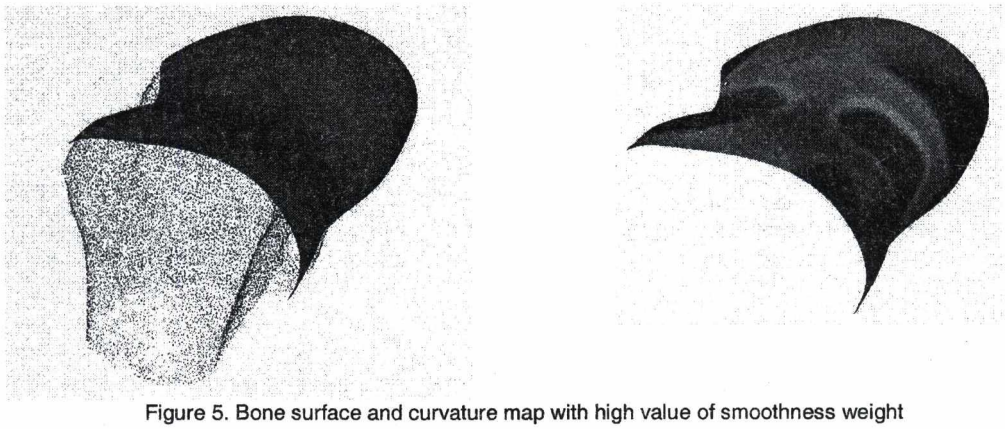


Figure 5. Bone surface and curvature map with high value of smoothness weight

High smoothness weight was applied in case of the surface of Figure 5. which resulted in a very smooth surface, as it can be recognized in the curvature distribution. However, the surface far out of the specified tolerance, the average deviation is more than 1.1 mm. This effect is especially strong in the highly curved region (see right side) where the difference between the shape of the generated surface and that of the point set can be seen by eyes.

Figure 6. shows the result with an optimal smoothing weight, computed by the above automatic smoothing weight adjustment. The generated surface fulfills the accuracy requirement, and has a shape with high overall smoothness. Adjusting the smoothing weight allows keeping the balance between the smoothing effect and the ability of reproducing complex shapes. Only in this way it is possible to fully exploit the potential of the B-spline representation with given degrees of freedom.

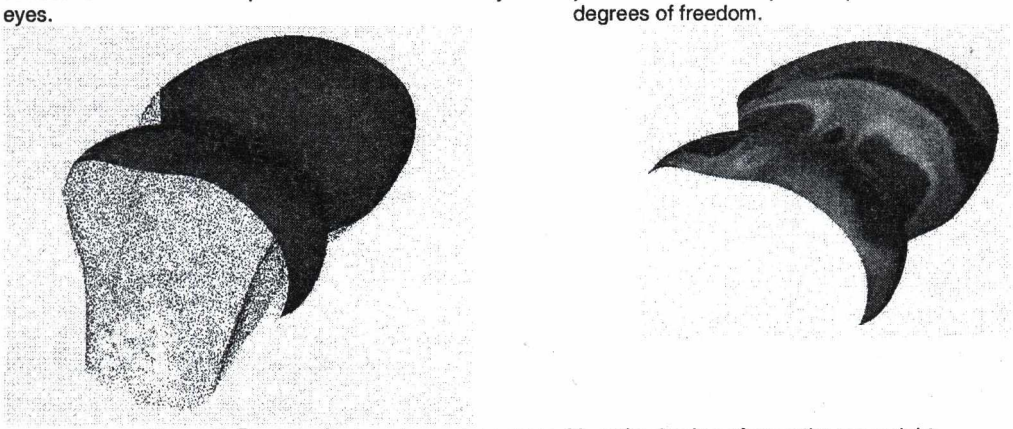


Figure 6. Bone surface and curvature map with optimal value of smoothness weight

6 CONCLUSION

Fitting of smooth surfaces with given accuracy over a cloud of data points is a difficult task, because the influence of the various parameters on the resulting surface has a complex nonlinear nature. In the paper we described a strategy for the reconstruction process, which decomposes the nonlinear optimization into a sequence of linear steps. A key problem of the surface fitting is the setting of the smoothing weight. An efficient method for the automatic adjustment of the smoothing weight is suggested. As a result, a high quality, smooth surface can be obtained, which approximates the data points with prescribed tolerances.

7 ACKNOWLEDGEMENT

This project was supported by the Hungarian Research Grant NKFP/1B/0009/2002. Special thank is due to V. Weiss and L. Szobonya for programming and for preparation of the manuscript.

8 REFERENCES

[1] Dietz, U 1998, Fair surface reconstruction from point clouds, Mathematical Methods for Curves and Surfaces, Vanderbilt Univ. Press, Nashville. 79-86

[2] Sarkar, B. Menq, C.. 1991, Smooth surface approximation and reverse engineering, Computer Aided Design, 23, 9, 623-628

[3] Dierckx, P., 1995, Curve and surface fitting with splines, Clarendon Press, Oxford

[4] Weiss, V, Andor, L., Renner, G., Várady, T., 2001, Advanced surface fitting techniques, Computer Aided Geometric Design, 19, (2002), pp. 19-42

[5] Ma, W., Kruth, J., 1995, Parametrization of randomly measured points for least squares fitting of B-spline curves and surfaces, Computer Aided Design, 27, 9, 663-675

Removing errors from triangle meshes by slicing

M. Szilvási-Nagy,[1†]

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

Abstract

Many surface-oriented representations, e.g. layered manufacturing use triangle meshes. The data structures of such meshes may contain errors in numerical data and topological relations. For removing wrong triangles and filling simple holes a slicing algorithm will be presented.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometric Algorithms

1. Introduction

Triangle meshes are used in many surface-oriented applications, e.g. layered manufacturing. In CAD systems different triangulation algorithms have been implemented to create discrete representations of surfaces. Unfortunately, the data structures of the created triangle meshes may contain errors in numerical data and topological relations [4]. Discrete geometry algorithms and discrete differential-geometry operators for estimating simple geometric attributes such as curvatures [6], for generating characteristic surface curves [3] and for smoothing meshes [1] require errorfree representations. For rapid prototyping (layered manufacturing) and tool path generation in milling accurate plane sections consisting of one or more closed polygonal lines is a basic requirement. A procedure for generating valid tool paths from section lines of defective triangle meshes can be found in [2].

With an appropriate polyhedral data structure built on a triangular mesh topological errors such as holes and gaps can be detected efficiently [5]. For removing invalid triangles and filling simple holes a slicing algorithm will be presented. It works semi-interactively, and computes in each step only with the triangles that are effected by the slicing plane.

2. Error detecting in the mesh

A possible numerical representation of a triangle mesh generated by CAD systems is described in an STL (stereo litography) format which became practically industrial standard. Each STL file contains a set of triangles the elements of which are stored one by one independently of each other during the triangulation process. The data contained by an STL file are the following: for each triangle three coordinates of the normal vector and three coordinates of the vertices ordered in counter clockwise direction. There are no topological informations about neighbouring relations in this data structure. The identification of the vertices is possible within a tolerance value of their distances. This can be done while reading the file by compairing the vertices of the actual triangle with those of read earlier. At the end of the reading process the set of triangles is described by the following data:

the number of the triangles (faces) N_F,

the array of mesh points $V_i : [x,y,z]_i$, $i = 1,\ldots,N_V$,

for each triangle denoted by F_i, $i = 1,\ldots,N_F$ the vertex cycle, i.e. three pointers to the vertices $j1, j2, j3 \in \{1,\ldots,N_V\}$ and three coordinates of its normal vector $[n_x,n_y,n_z]_i$.

This is the so-called TRI format of the mesh. It still does not contain any topological informations. We complement this data structure with triangle edges e_i, $i = 1,\ldots,N_e$, ($N_e = 3 * N_F$), and generate a "half edge" polyhedral data structure on the mesh in the following way [5]. For each edge e_i we search an edge joining the same vertices. If such an edge exists, we link them to each other. If not, the edge is signed as boundary edge. A half edge is described by five pointers: the pointers to its starting and end point $j1, j2 \in \{1,\ldots,N_v\}$, the pointer to the next edge in the edge cycle of the containing triangle $next \in \{1,\ldots,N_e\}$, the pointer to the containing triangle face $f \in \{1,\ldots,N_F\}$ and to the linked edge $link \in \{1,\ldots,N_e\}$ if exists. Boundary edges are linked to -1.

† supported by the Hungarian National Foundation OTKA No. T047276

This data structure is redundant, e.g. the end point of an edge is at the same time the starting point of the next edge in the edge cycle of a triangle, but it is very efficient for computing intersections.

In a correct triangle mesh representing the surface of an object each triangle has three neighbouring triangles, one along each of its edges, and each edge is contained exactly by two triangles. The existence of boundary edges shows gaps and holes in the mesh. Gaps and holes may arise from different reasons mainly along connection lines of joining surfaces or around sharp vertices due to numerical inaccuracy or errors in the triangulation (Fig. 1).

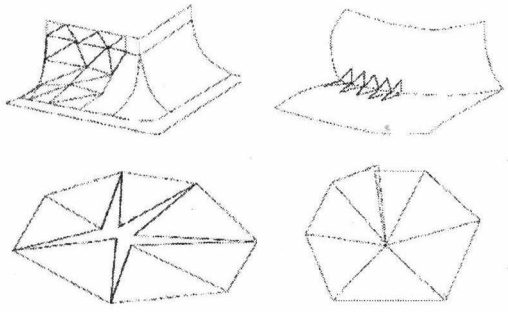


Figure 1: *Errors in a mesh.*

In layered manufacturing the material is removed or stiffened along the contour curves of consequtive plane sections. If these contour lines are not closed, the manufacturing can not proceed. Obviously, gaps and holes in the mesh result broken contour lines in plane sections.

Our aim is to detect such errors in the triangle mesh and to correct the triangulation.

In Fig. 2 a corrupted triangulation af a sphere is shown with different errors. A wrong triangle with a vertex outside of the sphere, two triangular holes and one invalid triangle (practically a double edge) are in the mesh. In a coloured representation the boundary edges can be seen immediately, consequently the user can check the mesh for holes.

3. Slicing the mesh

In this chapter we present a checking and healing algorithm by moving a slicing plane through the mesh and stopping it at defective areas. We compute the line of intersection at such user-defined positions of the slicing plane. The segments of a plane intersection are cut by the slicing plane from the triangles which have vertices on both sides of the plane. Let M_{ij} and M_{ik} denote the end points of the line of intersection on the edges e_j and e_k of the triangle F_i (Fig. 3). By the help of the pointers to the linked edges of e_j and e_k the neighbouring sides of the polygonal line are determined.

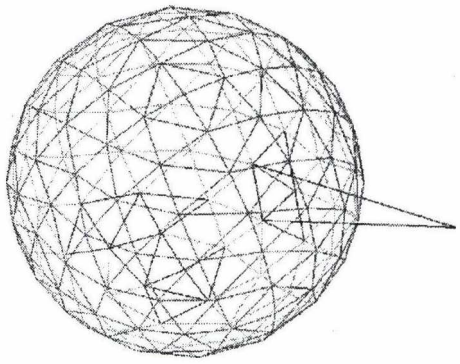


Figure 2: *Defective mesh of a sphere.*

If no boundary edges have been cut, then the line of intersection is a closed polygon or consists of more closed polygons. Otherwise chains of connected line segments or isolated line segments arise. The face containing an isolated line segment is obviously a wrong triangle in the mesh, therefore, it will be deleted. Two closest points of intersection on boundary edges show which edges are bordering a hole in the mesh. This is an essential information for filling holes, and allows a more effective healing algorithm than other methods searching through the whole mesh. Simple holes can be filled by defining new triangles between such boundary edges.

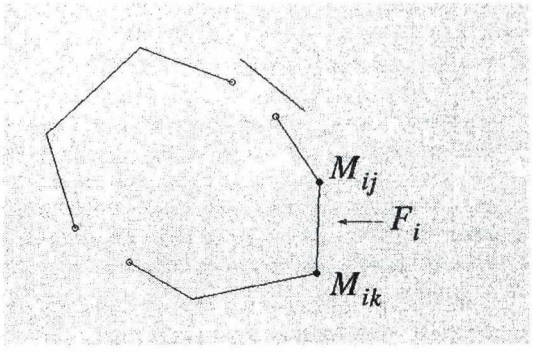


Figure 3: *Plane section of a mesh.*

In Fig. 4 the line of intersection of the mesh in Fig. 2 is shown containing one isolated segment and one gap. By removing the wrong triangle and adding one triangle to the mesh which fits the hole, the errors in this area disappear.

This slicing method works in a semi-interactive way. The user defines the cutting plane and gives commands to the algorithm about removing and adding triangles. The data of the line of intersection contain all 3D informations about the part of the mesh that is effected by the slicing plane, but searching for proximities (i.e. corresponding boundary

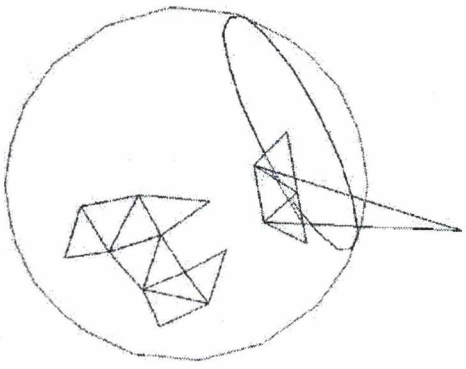


Figure 4: *Slicing the mesh of the sphere.*

edges) is made in the slicing plane. In this way 2D procedures are applied in a 3D algorithm.

In Fig. 5 two rectangular holes are shown in the mesh of the surface of a cylinder and a vertical slicing plane. By choosing appropriate slicing planes these holes are filled, hereby the mesh is healed in few steps. The slicing can be made parallelly to each coordinate plane, and the moving steps are specified by the user. In this way the user can influence in the filling procedure how a new triangle is defined by choosing which two edges of a hole are cut by the slicing plane.

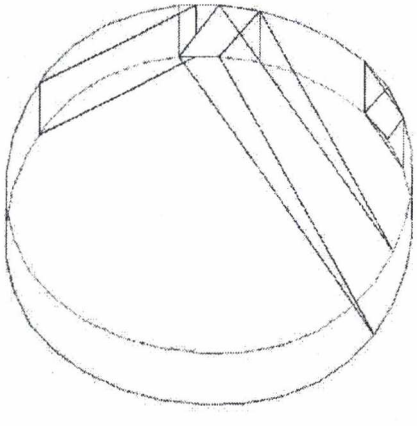


Figure 5: *Boundary triangles in the mesh of a cylinder and a slicing plane.*

In Fig. 6 a half of a cake baking form is shown with an intersecting plane, where only the boundary edges and their containing faces are drawn. There are errors of different types in this mesh: gaps, holes, wrong triangles and too many, too small triangles along stiching lines of surface parts. Consequently, its plane sections are too confused for making unambigious decisions for the healing algorithm. Perhaps a decimation algorithm should be applied first in order to get well arranged intersections.

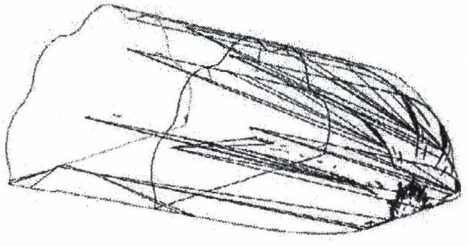


Figure 6: *Boundary triangles in a mesh and a plane section.*

4. Conclusions

We have shown a semi-interactive algorithm for visualizing, checking and improving a triangle mesh by generating a suitable data structure. The algorithm is implemented in the programing language Java 1.2 and works a PC for meshes with about 70000 faces.

References

1. M. Desbrun, M. Meyer, P. Schröder and A.H. Barr. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. *Computer Graphics Proceedings (Annual Conference Series, 1999)*: 317-324. 1
2. Sang C. Park. Sculpured surface mashining using triangular mesh slicing. *Computer-Aided Design*, **36**:279-288, 2004. 1
3. K. Polthier and M. Schmies. Straightest Geodesics on Polyhedral Surfaces. In H.C. Hege and K. Polthier, editors *Mathematical Visualization*. Springer Verlag, 1998. 1
4. M. Szilvási-Nagy and Gy. Mátyási. Analysis of STL Files. *Mathematical and Computer Modelling*, **38**:945-960, 2003. 1
5. M. Szilvási-Nagy, I. Szabó and Gy. Mátyási. A Polyhedral Data Structure for Shape Characterization of Meshed Surfaces. *II. Magyar Számítógépes Grafika és Geometria Konferencia (Budapest, 30.06-1.07. 2004)*:71-77. 1
6. G. Taubin. Estimating the Tensor of Curvature of a Surface from Polyhedral Approximation. In Proc. *5th Int. Conf. on Computer Vision (ICCV'95) (June 1995)*:902-907. 1

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Digital shape reconstruction using a variety of local geometric filters

Zsolt Terék[1,2] and Tamás Várady[2]

[1] Department of Computer Science, Technical University of Budapest
[2] Geomagic Hungary, Budapest

Abstract

After briefly reviewing the steps of digital shape reconstruction (DSR), we focus on describing various filters. Filters extract different geometric properties over a triangulated mesh based on local estimations. Filters typically create connected regions in order to (i) determine the global structure of measured objects, (ii) classify primary regions by surface type and (iii) highlight connecting features, such as, blending surfaces and sharp edges. Difficulties of setting the parameters of filtering will also be discussed. Filters assist in achieving our final goal to automate the DSR process and provide high quality surfaces for reconstructed CAD models.

1. Introduction

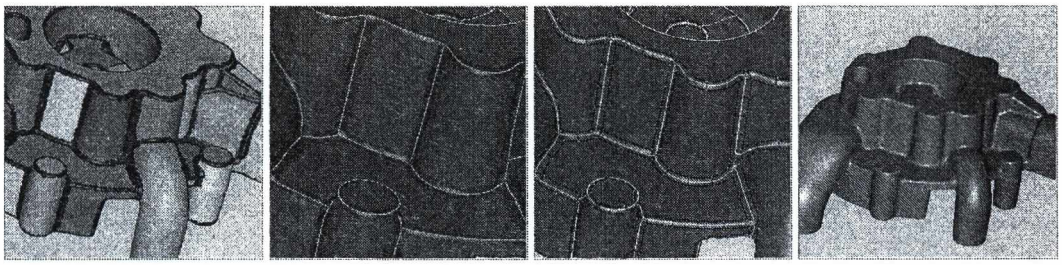
Digital shape reconstruction systems (formerly reverse engineering) create CAD models from measured data sets of various objects.[9] Our primary interest here is the reconstruction of mechanical engineering objects, which obey certain general principles of computer aided geometric design. First large functional or aesthetic surfaces are created; these can be of simple analytic types, such as planes, or cylinders; or swept surfaces, such as extrusions or surfaces of revolution; or free-from surfaces. These surfaces are often called *primary surfaces*. Once these are defined they need to be integrated by various operations, such as Booleans, or intersections. Between the primary surfaces relatively small connecting surfaces are created, which smoothly join the primaries and satisfy special geometric or functional requirements, such as blending surfaces, step surfaces, slots, etc. These surface elements are often called *connecting features* or *secondary surfaces* due to their dependency on the primaries.

In the digital shape reconstruction process our goal is to discover the original topological structure of the object and fit surfaces accordingly. This is practically equivalent to extracting the unknown regions from the measured data and add type information. This process is called *segmentation* which fundamentally determines the quality of the surfaces being fitted and stitched together in the consecutive phases of DSR.[9]

For our discussions let us assume that after data acquisition the scanned point clouds have been filtered, aligned and merged, and a decimated triangular mesh has been created which reflects the geometry of the object with sufficient point density and accuracy. In order to build up the global structure and set the type of the individual regions we extract various geometric properties based on local neighborhoods over the triangular mesh.

These quantities—often called as *indicators*—may be scalars or vectors; they may characterize simple properties from elementary differential geometry, such as a local normal vector or the principal curvatures; or may describe more complex properties, such as the best fit local translational direction or rotational axis. They may also characterize complex properties, such as *similarity* of a given point related to its neighborhood.[1] *Filtering* is the process of performing region separation or highlighting by the indicators. In our context the filters fall into three groups:

(i) extract the topological structure of the object by highlighting highly curved portions of the mesh, which are likely to represent connecting feature regions.
(ii) classify primary regions by assigning a likely surface type information, for example, planar, cylindrical, extruded or free-form, etc.
(iii) classify connecting features with particular emphasis to constant and variable radius/range blending surfaces, step surfaces and sharp edges.



(a) (b) (c) (d)

Figure 1: *Steps of digital shape reconstruction*

This paper is structured in the following way. In Section 2 we overview the basic phases of digital shape reconstruction and show a few examples. In Section 3 we discuss the most important filters applied by the above grouping. In Section 4 we briefly discuss the difficulties of parameter setting for filtering.

2. Steps of digital shape reconstruction

The starting point of our algorithm is a decimated triangular mesh. To separate the primary and secondary regions first we apply filters which separate the highly curved point regions from the relatively flat parts. As it is shown in Figure 1(a) the result is a collection of disjoint primary regions. In between there are highly curved, so-called *separator sets*, which are likely to represent secondary surface elements. By extracting the mid-curves of the separator sets we get a graph called *feature skeleton*, which globally segments the triangular mesh, see Figure 1(b).

The edges of the feature skeleton will be widened if we have a "real" connecting feature. Figure 1(c) shows an object where all edges of the feature skeleton are replaced by two quasi-parallel curves, which will serve as boundaries for blend surfaces to be computed later. Note, that though we cannot go into details here, further connecting elements, such as vertex blends need to be inserted as well, to join the connecting features at a given vertex of the skeleton. This will yield the final surface model, as can be depicted in Figure 1(d).

In other cases, the feature skeleton may represent sharp, i.e. zero-width features, when there is no need to widen, and the accurate location of the sharp edge needs to be computed. Such an example is shown in Figure 2(a) and Figure 2(b); the orange edges will be replaced by a blending surface, the purple edges remain sharp.

In this paper we do not discuss how one can fit surfaces onto the points of the segmented regions. There are two basic approaches to approximate the segmented structures.[8] *Automatic surfacing* produces watertight quadrilaterals in a computationally very efficient way. *Functional decomposition* creates trimmed surfaces with more computation and more user assistance, but provides better surface quality. The 'pros and cons' of the two approaches can also be found in the above paper.

3. Filters

This section describes the estimation methods for various indicators based on a set of sample points. Let p be the point for which the filter value is estimated, n denotes the size of the neighborhood and p_i are the points in the neighborhood ($i \in [1,n]$).

3.1. Filters to characterize high curvature and flatness

3.1.1. Planarity

Planarity is to measure the flatness of a point set.[1] This value is defined to be the root mean square error of the points to the LSQ plane. The average error is the least eigenvalue of

$$M = \frac{\sum_i p_i p_i^T - \frac{1}{n}(\sum_i p_i)(\sum_i p_i)^T}{n}.$$

Assuming $\varepsilon_0 \leq \varepsilon_1 \leq \varepsilon_2$ being the eigenvalues of M, the planarity is defined as

$$planarity = \sqrt{\varepsilon_0}.$$

Note, that the LSQ plane itself does not need be determined.

Figure 3(a) shows an example object rendered according to the planarity value. The color range from red to blue corresponds to the planarity interval $[0, m_p]$, where m_p denotes the maximum planarity estimated on the object.

3.1.2. Curvatures

There are several curvature-based scalar indicators derived from the principal curvatures (κ_1 and κ_2), such as the *mean curvature* $(\kappa_1+\kappa_2)/2$ and the *Gaussian curvature* $\kappa_1\kappa_2$. The principal curvatures, *minimum* (κ_1) and *maximum* (κ_2), are also often utilized.

(a) (b) (c)

Figure 2: *An object having both sharp and blended edges*

There are different ways to determine the principal curvatures. κ_1 and κ_2 are the eigenvalues of the so-called Weingarten matrix. To estimate the Weingarten matrix[7], the coordinate system should be transformed so that point p is the origin and the normal vector is the z axis. Then the best *explicit quadric surface* is fitted to the points in least square sense. The Weingarten matrix consists of the partial derivatives of the quadric at the origin.

Fitting an implicit quadric to the points[7] also gives us a way of curvature estimation, but this is less stable than the estimation described above. However, fitting an implicit quadric is often used for stable normal vector estimations.

Another method of curvature estimation using a polygonal mesh[6] is to integrate the curvature tensors assigned to the edges of the polygons. The advantage of this method is that arbitrary region of the polygonal surface can be used for the estimation, including parts of triangles.

3.2. Filters to classify primary regions

Here we follow the direct segmentation algorithm[1] is a hierarchical method of simple tests to classify primary regions. First, the region is determined to be *flat, ruled* or *doubly curved.* After that, several sub-categories are defined.

The top-level decision is made based on a so-called *dimensionality test.* This basically analyzes the distribution of the normal vectors (estimated at the neighborhood of the location examined) on the Gaussian sphere. The result of the test assigns 0, 1 or 2 as a dimensionality value, showing whether the normal vector represents a point cluster, a curve or a larger area on the sphere.

Another measure for the dimensionality[2] examines the *covariance* values at the point of estimation. After projecting the normals onto the tangent plane, the *covariance matrix* of the planar coordinates are calculated. The dimensionality is derived from the eigenvalues of this matrix.

3.3. Filters to classify connecting features

For practical objects in mechanical engineering, the following types of connecting features occur most frequently:

Rolling-ball blends are defined by a moving sphere that is swept while touching two adjacent primary surfaces. The ball can have either *constant* or *variable radius.* Rolling balls create G1 continuity along the connections between the primary and the secondary features.

Constant or variable range blends are used when G2/C2 continuity is needed between the surfaces, here the width of the blend needs to be extracted.

Sharp edges result from no connecting feature. The edge is defined by the intersection of the primary surfaces. There is only positional continuity along a sharp edge.

General swept surfaces are defined by a 3 dimensional *spine curve* and a 2 dimensional *profile curve* that is swept along the spine.

The *blend filter* described below (Sec. 3.3.1) is useful for finding blends, if the result of simple thresholding (see Sec. 4.1) is sufficient. The *sharp edge filter* (Sec. 3.3.2) evaluated in the middle of a feature reasonably separates sharp and non-sharp features.

In order to distinguish between general swept surfaces and other blends, the minimum curvature estimation is combined with similarity indicators[1] in the middle of the feature along the *transversal* direction. If there is no radical change in the curvature value, the feature can be considered a rolling-ball blend.[4] The transversal direction is the direction associated with the minimum principal curvature.

As for the distinction between constant and variable radius/range blends, the similarity of the minimum curvature estimations along the *longitudinal* direction should be determined.

3.3.1. Blend filter

The purpose of the blend filter is to locate blends within the mesh. The maximum curvature is the inverse of the radius

of the blends and it is very small for the flat areas, but it is also significant at vertex blends. We define the blend value as the product of the maximum curvature and a factor that vanishes for vertex blends and is around 1 for blends. Since the minimum curvature is significantly smaller on the blends than the maximum curvature, the factor is defined by

$$\begin{aligned} r(\kappa_1,\kappa_2) &= |\kappa_1|/|\kappa_2|, \\ f(r) &= e^{r/(r-1)} \end{aligned}$$

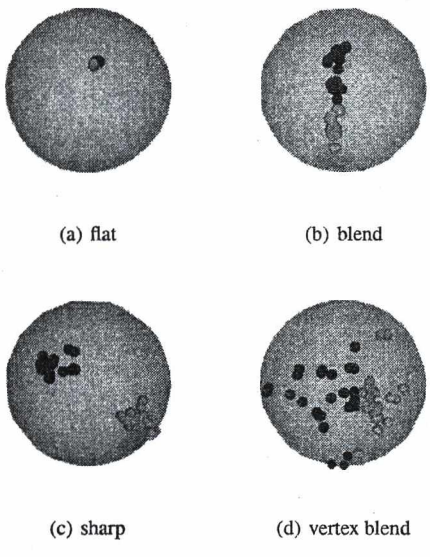
Here $r \in [0,1)$ and $\lim_{r\to 0+} f(r) = 1$ and $\lim_{r\to 1-} f(r) = 0$. In other words, $f(r(\kappa_1,\kappa_2))$ is small, if both curvatures have the same magnitude, and is close to 1, if $\kappa_2 \gg \kappa_1$.

In Figure 3(c), the rainbow colors assigned to the vertices correspond to the estimated blend filter values: pure red is the most concave, green is about zero and pure blue is the most convex blend.

3.3.2. Sharp Edge Filter

The aim of the sharp edge filter is to distinguish between sharp edges and small blends. The sharp edge filter is a variant of the covariance filter, but there are certain differences.

The basic idea is to examine the normal vectors of *triangles* in the vicinity of a vertex. In case of a sharp edge, the majority of the triangles are on one of the neighboring relatively flat surfaces and only a few reside on the edge. On blends and other non-flat areas, more triangles come from the nearby smooth edge. Therefore in the vicinity of sharp edges the normal vectors of triangles form two well-defined clusters on the Gaussian Sphere. Figure 4 shows some typical layouts of the face normals on the unit sphere.



(a) flat (b) blend

(c) sharp (d) vertex blend

Figure 4: *Normal vectors on the Gaussian sphere*

The sharp edge filter should return a value that reflects the extent how the normals on the Gaussian sphere are distributed into two separate clusters. Call this value ***bicentricity.***

First, the translational direction is determined: this is the normal of the plane that goes through the origin and minimizes the sum of squared distances from the points on the Gaussian sphere. The indicator value is set 0, if the translation is not well-defined, i.e. smaller than a threshold.

The translational direction defines a great circle, which is orthogonal to the translational direction. The normal vectors are projected to the plane of this circle and their angle to some predefined direction is taken.

Bicentricity is calculated from this set of 1 dimensional values. Take the projection of the normal vector at the current point, it separates the values into two partitions. (The different colors of the points in Figure 4 indicate these partitions.) Assuming the two parts are the two clusters, let v_i ($i \in [1,n_1]$) and w_j ($j \in [1,n_2]$) denote the values of the two groups. Then bicentricity is defined as

$$\begin{aligned} a_1 &= \tfrac{1}{n_1}\textstyle\sum v_i \\ a_2 &= \tfrac{1}{n_2}\textstyle\sum w_j \\ \alpha &= a_1 - a_2 \\ \delta_1 &= \tfrac{1}{n_1}\textstyle\sum (v_i - a_1) \\ \delta_2 &= \tfrac{1}{n_2}\textstyle\sum (w_j - a_2) \\ b &= \alpha/2 - \delta_1 - \delta_2 \\ b^* &= b\tfrac{n_1 n_2}{(n_1+n_2)^2} \end{aligned}$$

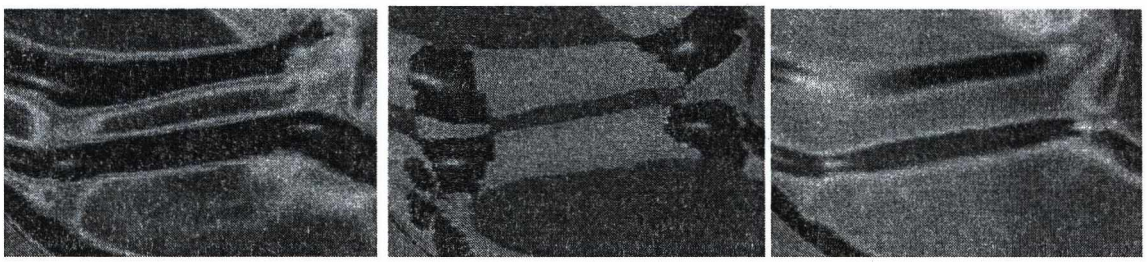
Rationale. On blends, the normals are equally distributed, thus their projection can be assumed to be equally distributed. In this case, α is about the half of the total angle and both δ_1 and δ_2 are about $\alpha/2$. Therefore b is near 0. On the other hand, assuming an ideal sharp edge $\delta_i = 0$, so b is $\alpha/2$. The value b^* incorporates a factor that emphasizes the center of sharp edges by taking the sizes of the clusters, which are nearly equal at the center and differ when going off the edges.

4. Difficulties of parameter setting

This section deals with the difficulties and problems arising when estimating indicator values. First filter-based segmentation methods are briefly reviewed. Then different environment computation methods are studied, and finally the stability of estimations is considered.

4.1. Segmentation

The basic method for separating regions and features is *thresholding.* Given a limit value, the separator set is defined by the vertices associated with an indicator value greater than that limit. Ideally the primary regions correspond to the connected components of the vertices not part of the separator set.



(a) Planarity (b) Dimensionality (c) Blend

Figure 3: *Different filters applied to the same object*

There are several problems with global thresholding. It is very difficult to determine the appropriate limit value for an unknown object with unknown dimensions and unknown shape variation. Moreover, the noise and the instability of the estimations result in false-regions and dummy-features. Generally the feature-region decomposition defined by thresholding is inconsistent.

More general approaches are needed to make a consistent segmentation of the triangulated data. *Region growing* methods[5] work with a set of seed points and an associated information concerning the type of the regions. Starting from the seed points, more and more triangles are added to the regions until they satisfy the type criteria. Once the regions cannot be extended anymore, the rest of the surface outside the regions forms the separator set.

A novel algorithm used in Geomagic Studio 8 is based on Morse theory. Given a function on a 2-manifold, a special structure, the Morse complex can be created by connecting corresponding critical points (i.e. maximum, minimum and saddle points) of the function. Minimum points will represent the "middle" of the regions, while loops of alternating segments connecting maximum points and saddles will represent region boundaries. From this initial structure a topologically consistent natural region segmentation can be derived, as was shown earlier in Figure 1(a). A good summary of the method was recently published[3].

A novel algorithm[3] used in Geomagic Studio 8 is based on Morse theory. Given a function on a 2-manifold, there is a mapping between the critical points of the function, which defines the primary regions and separator sets. The method guarantees that always a topologically consistent region structure is created.

The quality of filter estimations has a fundamental effect on the resulting segmentation and indirectly the quality of surfaces.

4.2. Computing the extent of the point environment

Two basic methods exist for collecting sample points in the neighborhood of a vertex. The first collects points based on their distance from the neighborhood center (n closest points). The other method, based on the mesh topology uses the connection information of the triangles and takes the points that are reachable within k steps.

Neither method can directly be applied since there is no general assumption on the distribution of the vertices on the polygonal mesh. There are several reasons for assuming uneven arrangement, either related to point capturing or post-processing.

If the data source is a line scanner, uneven point distribution is likely. The points are sampled on the scan lines. If the density of the scan lines differs from the density of points on the scan lines, the vertices of the mesh will be lined up. In this case, the distance-based collection might only collect points from a single scan line, which definitely destabilizes the estimations.

Post-processed (decimated) triangulations might contain triangles of different magnitude around the same vertex. In this case, the starting point of the estimation might be far from the center of the points collected by a purely topology-based method.

One way to overcome these problems is to equally resample the surface, for example take a specific point pattern and project it onto the triangulation. As a consequence of this, the n closest points will result in neighborhoods of approximately the same diameter. This method works fine with some of the filters (e.g. planarity), while it is not applicable for others, for example those that use normal vectors.

Another solution is to collect faces topologically, and post-process the neighborhoods to have the same diameter. In this case new sample points should be taken instead of the vertices that are too far from the center, while original inner points are kept.

4.3. Stability of the estimations

It is often desirable to have information on the precision of the estimation. Instable estimations are usually caused by noise in the data, outlier sample points, or radical change

in the surface shape. Some estimations provide an error term for fitting, some others define the filter value by the error term itself (e.g. planarity).

A general method to estimate stability[1] takes two neighborhoods with a single and a double size. After performing the estimation for both sets of points, the estimated values are compared. If the two values are close, the estimation is considered to be stable. Otherwise, either a better neighborhood is computed, or the estimations are thrown out and the point is marked as unstable.

5. Conclusion

In the course of digital shape reconstruction a set of filters needs to be used to create a global segmentation over a discrete triangular mesh. We discussed how these can separate adjacent regions from each other and help highlighting different types of subregions. Computational difficulties on how to set related filter parameters have also been discussed. The result is a good segmentation of the mesh, which is particularly critical for the final surface approximation, independently of whether surfaces using quadrilaterals or trimmed patches are fitted.

Acknowledgements

This research was conducted within Geomagic, Inc., North Carolina[10] and Geomagic Hungary, Budapest. Most of the pictures were generated by Studio 8, the latest digital shape reconstruction program of Geomagic. Partial support from the Ministry of Education is also acknowledged (OMFB-01979/2002, ADREN).

References

1. P. Benkő and T. Várady, *Segmentation methods for smooth point regions of conventional engineering objects,* Computer-Aided Design, 36, 2004, pp 511–523 1, 2, 3, 6
2. P. Csákany, A.M. Wallace, *Computation of local differential parameters on irregular meshes,* In: The Mathematics of Surfaces, IX, Eds: R. Cippola, R.R. Martin, Springer, 2002, pp 19–33 3
3. H. Edelsbrunner, *Surface Tiling with Differential Topology,* Eurographics Symposium on Geometry Processing 2005, 2005, pp 9–11 5
4. G. Kós, R.R. Martin and T. Várady, *Methods to recover constant radius rolling ball blends in reverse engineering,* Computer Aided Geometric Design, 17, 2000, pp 127–160 3
5. N.S. Sapidis and P.J. Besl, *Direct construction of polynomial surfaces from dense range images through region growing,* ACM Transactions on Graphics, 14, 1995, pp 171–200 5
6. D. Cohen-Steiner and J.-M. Morvan, *Restricted Delaunay triangulations and normal cycle,* In: Proc. 19th Annu. ACM Sympos. Comput. Geom., 2003, pp 124–133 3
7. Zs. Terék, *Filters to detect blends and sharp edges,* Technical Report RGI-TECH-2004-062, 2004 3
8. T. Várady and M.A. Facello, *New trends in digital shape reconstruction,* In: The Mathematics of Surfaces, XI, Eds: R.R. Martin, H. Bez, M. Sabin, Springer, 2005, pp 395–412 2
9. T. Várady and R.R. Martin, *Reverse Engineering, Chapter 26,* In: Handbook of Computer Aided Geometric Design, Eds: G. Farin, J. Hoschek, M. S. Kim; Springer, 2002, pp 651–681 1
10. `http://www.geomagic.com` 6

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Industrial Styling Based on Free-form Curve Networks

Gergely Várady, Tamás Várady and Tamás Zombori

Geomagic Hungary, Budapest

Abstract

Creating computer models for very complex, aesthetically pleasing objects is still a hot topic in computer aided geometric design. Traditional design techniques are often found dissatisfactory when feature lines drawn by stylists need to be converted into standard CAD models. The technique proposed in this paper puts emphasis on merging hand-drawn sketches into a consistent 3D structure, which is transformed into a free-form 3D curve network without topological limitations. This curve network uniquely defines an aesthetically pleasing smooth surface, comprised of N-sided patches. Instead of gridded control point modifications, here the surface model is adjusted through its defining curve network. Results are demonstrated by a prototype styling system called Sketches.

1. Introduction

In spite of the enormous progress of CAx technologies in the last decades, the design of complex free-form objects has remained a difficult task, in particular, when aesthetic requirements dominate, such as in car-body design. The design process is split into two phases. (i) Stylists typically use conventional graphic tools, such as pencils and rubbers to define their concepts. Unfortunately, the related (artistic) sketches are not precise, and not consistent in mathematical sense; moreover, the collection of curves is often insufficient to uniquely define surfaces. (ii) Design engineers use a limited set of operations to convert these sketches into a meaningful 3D surface model. This is supposed to reflect the original concept, but the created computer model usually differs from the original one, and many iterations are needed until the desired shape is achieved.

Current CAD systems have strong limitations in free-form surface design. Most surfaces are created by extruding or rotating planar, free-form profile curves. Lofting operations provide more generality, but remain still topologically rectangular. The most widespread free-form surfaces, such as, Bezier or NURBS surfaces are also arranged into a topologically rectangular grid. The problem is that real objects consists of not only four-sided faces. Though intersections and trimming operations can create N-sided surface portions, trim-lines are not suitable for modifying the initial surfaces, and for shape adjustments the designer has to go back and start again. Further difficulties arise when trimmed elements need to be stitched together.

A new paradigm has been evolving to overcome the above difficulties, where 3-dimensional shapes are created by means of a general topology network of free-form, space curves. The surface geometry is uniquely determined by the curve network. This is a hard task from both mathematical and user interface points of view. The curve network has to be filled up with such N-sided patches that flow naturally and get connected smoothly.

The current paper focuses on problems that fit into the above area. *Sketches* is a prototype styling system for processing hand-drawn sketches, which need to be converted into a smooth, high quality surface model. The design process is based on a new user tool, called *construction box*. This allows the creation of 3D curves that can satisfy projectional constraints simultaneously. Sketches uses a new mathematical model, which allows fast, interactive design of complex surface models. Section 2 gives a general overview on the design procedure and the modelling paradigm. Section 3 describes the creation of 3D curve networks, which approximate aligned 2D sketches being "glued" onto a construction box. The mathematical algorithms of N-sided patch generation are summarized in Section 4.

2. Design Process in Sketches

The workflow of shape design in Sketches is shown in Figure 1.

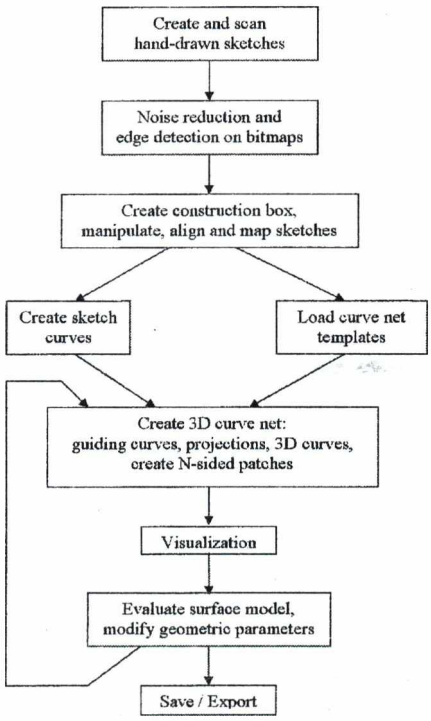


Figure 1: *Sketches – Design Flowchart*

We want to create a computer model of an object, which is indirectly defined by a set of images containing hand-drawn or computer generated *sketches* of silhouette and feature curves. (If we wish to reconstruct existing objects photographs can also be used.) The images typically show the object from different orthogonal views (front, back, right, left, top or bottom) and are sufficiently detailed for 3D design.

After noise reduction, edges are extracted using well-known image processing operations[4]. The images are aligned by scaling, rotation and positioning operations to obtain a coherent set of views being mapped onto the faces of the *construction box*.

The pixel information of the images needs to be converted into continuous 2D curves represented by markers, which will be interpolated by B-spline curves. The 3D curve network will be built by matching pairs of these 2D curves. Eventually the projections of the final 3D curve network will correspond to the feature lines of the original sketch input.

Another option to start design is to retrieve curve network *templates* of previously designed models. Designers are usually specialized in certain fields, such as car-body design, and the use of parametric templates can save a lot of work. In Sketches, users can set various parameters to control the actual shape of the template. Of course, a complete car-body cannot be defined in this way, but its main characteristics like the basic dimensions, the distance between the wheel axes, etc. are often sufficient to obtain a good initial curve network. The template curves may be projected back to the construction box and are tailored until the given images and projections get close to each other.

In order to modify or refine a network, users can add, remove or reposition individual markers, or perform complex curve operations. Once the curve network is satisfactory, Sketches fills the selected loops of curves by *N-sided patches*. In this way there is no need to create artificial edges due to rectangular topology, that often makes design tedious in other styling systems.

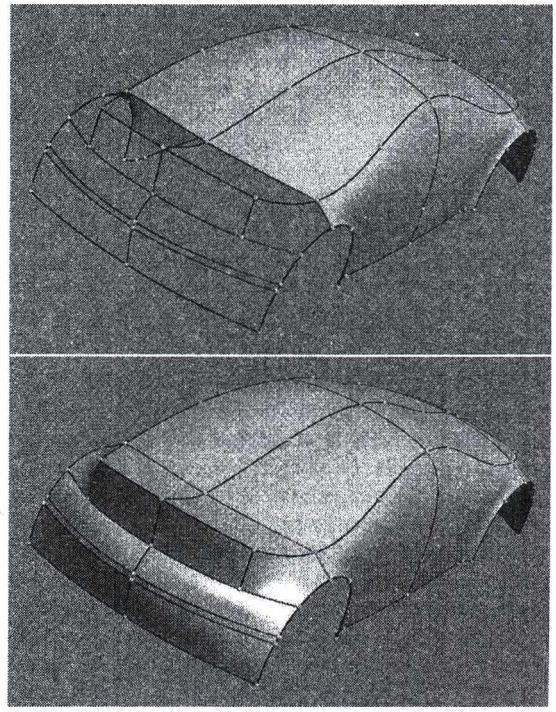


Figure 2: *Curve network and completed surface model of a car-body*

Surface generation in Sketches is exclusively based on the curve network and the derived cross derivative functions, which ensure watertight connections with $G1$ continuity between the adjacent patches. If necessary, sharp edges ($G0$) can be incorporated into the patch structure or higher order smoothness ($G2$) can be assured.

The internal representation of Sketches is based on a vari-

ant of Gregory patches. Due to the fact that the structure is defined by a few meaningful parameters only, curve and surface modifications show up in real-time. Several tools are available to measure and analyze the smoothness and surface quality of the resulting patches including slicing, isophotes, curvature and environment maps.

In order to export models, the last step is to convert the surfaces into a standard format. Sketches uses IGES, and the N-sided patches can be very tightly approximated by NURBS surfaces.

To illustrate the above process, the final steps of designing a concept car are shown in Figure 2. The last few patches have been generated by automatically interpolating the rest of the unfilled curve network.

In the next two sections we present further details on the algorithms applied.

3. Generating 3D curves from 2D input

The first step is to grab basic shape information from the 2D sketches and convert them into a consistent 3D curve network. The images, which may have been taken from six orthogonal views (front, back, left, right, top and bottom), are mapped onto the faces of the construction box. The actual curve network definition will take place later inside the box utilizing so-called *guiding curves*, which have been extracted from the sketches. At any stage of the process, the curve network can be projected to the box providing visual feedback about the correlation between the model being built and the sketches.

As discussed earlier, these hand-drawn sketches are usually neither precise, nor consistent. This is due to the fact that they typically come from different sketches of an imagined object, which never existed. In order to get a meaningful arrangement, the images have to be properly aligned on the faces of the construction box. This requires basic operations like image scaling, rotation and translation. After these, further steps are applied to reduce noise and detect edges[4]. For curve extraction, the user clicks on the image and Sketches will automatically trace the underlying curve. Based on the previously computed transformations, 3D guiding curves are created on the corresponding face of the construction box (Figure 3).

The 3D curves and the curve network can be modified not only by moving its 3D markers, but also by moving their *projective markers* sliding on the faces of the construction box. A modification on a face affects two coordinates, which propagate back to the 3D marker leaving the 3rd coordinate unchanged. The projected markers can automatically be snapped to two guiding curves; in this way a 3D curve can easily be constructed matching two orthogonal views. This curve will approximate two sketched – maybe slightly inconsistent – curves as close as possible. (Figure 4).

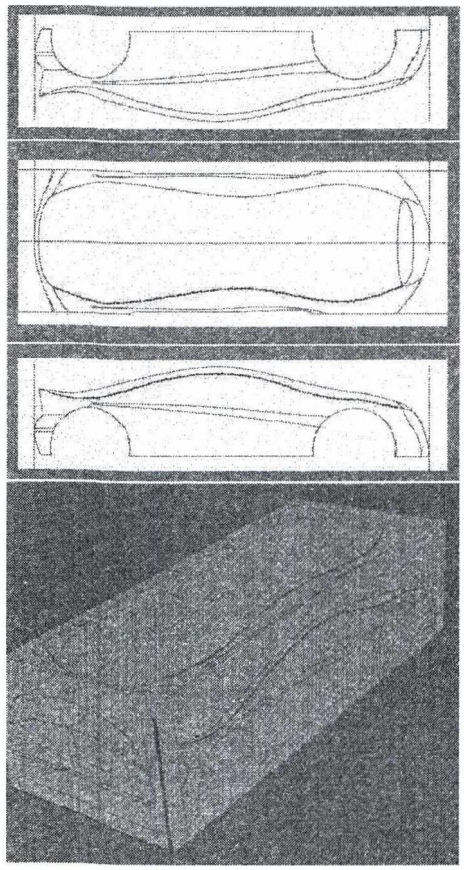


Figure 3: *Guiding curves on the construction box*

Another problem is occlusion, i.e. certain feature curves are visible only from a single view. To create a corresponding 3D curve, the user should first define the basic 3D curve network and the related surfaces, then the user can project the single view curve onto the existing structure, and incorporate it into the network for further refinements.

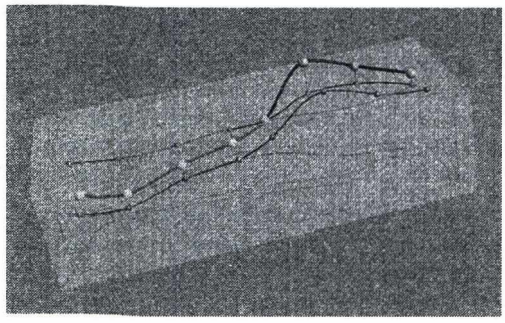
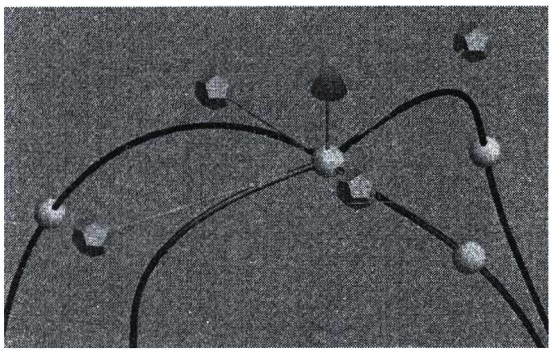


Figure 4: *Snapping a 3D Curve to approximate two guiding curves*

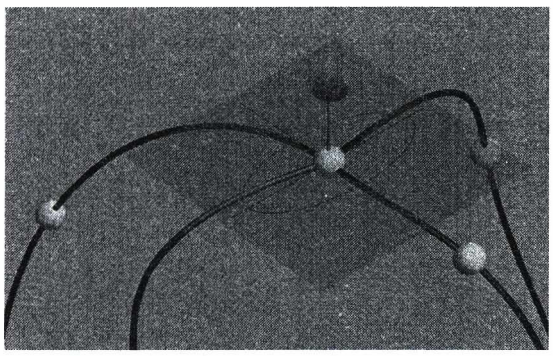
4. Surface Interpolation of 3D Curve Networks

4.1. Continuity Constraints at the Common Vertices

At this point we have a curve network of B-spline curves sharing common vertices. The shape of the curves can be adjusted by moving their markers and setting the end-point tangents. Every *loop* of curves will be filled by an N-sided surface patch. In order to assure smooth transitions it is necessary that the patches have the same normal vectors along their common border curves. This also implies that at the corner points (where three or more patches join) a common tangent plane is defined forcing the tangents of all related curves into this plane. In practice, this means that once two curves define a tangent plane at a common end point, all the additional curves ending there will automatically match this tangent plane. The normal vector at this point can turn into a design tool, i.e. if needed, it can be freely adjusted, and the curve tangents will be dragged accordingly (see Figure 5(a)).



(a) Common tangent plane



(b) Dupin indicatrix

Figure 5: *Continuity constraints*

In order to assure higher level ($G2$) continuity at the corner points, the curvature of the joining curves can also be made compatible by slightly modifying their first and second control points. These small modifications can be hardly noticed concerning the shape of the curves, but the resulting surface will be much smoother. Details of the related simple optimization algorithm can be found in [5]. Figure 5(b) shows the Dupin indicatrix[1] at a corner point, assuring common curvature there.

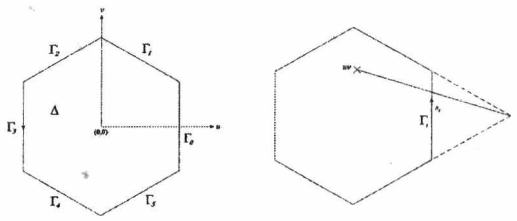
In special cases, when sharp edges are needed, these continuity constraints can be overridden by the user.

4.2. N-sided Patches

Having a curve network with the above properties, appropriate *cross-derivative functions* can be calculated along every patch boundary, which define the cross-tangents and spread over towards the interior of the N-sided patch. The cross-derivative functions are computed by purely geometric (i.e. non-parametric) quantities along the common boundary curve. This assures that once these functions are interpolated by the corresponding two patches, they will also smoothly join each other.

For each curve loop we have to satisfy positional and cross derivative constraints. In Sketches a scheme proposed by Gregory, Charrot, and Plowmann[2, 3] has been applied, though their concept has been significantly enhanced for better internal curvature distribution and to provide fast response times. In order to understand how Gregory patches work we briefly summarize the fundamentals, as follows.

This N-sided patch is created by mapping an N-sided regular polygon in the (u,v) domain plane to the 3D space. Let the center of the polygon be the origin and let the vertices lie on the unit circle. The edges of the polygon are denoted by Γ_i, $0 \leq i < N$; see Figure 6(a).



(a) Polygonal domain (b) Linear projector

Figure 6: *Polygonal domain and the linear projector of the corner interpolant*

The edges of the polygon are mapped to the N border curves denoted by C_i. Let us define so-called *corner interpolants* $(Q_i, i = 1,\ldots,N)$, that interpolate not only the C_i and

C_{i-1} curves, but also satisfy constraints implied by their associated cross-derivative functions D_i and D_{i-1}. Every (u,v) domain point defines a point on the i^{th} corner interpolant $Q_i(u,v)$. To get the N-sided patch, we weight these using special weighting functions:

$$P(u,v) = \sum_{i=0}^{N-1} \alpha_i(u,v) Q_i(u,v) \tag{1}$$

Corner interpolants

The i^{th} corner interpolant (Q_i) can be described by two local variables (s_i,t_i). The main problem is how to compute these local coordinates from a given (u,v) domain coordinate pair. We can use a *linear projector*: let us take the i^{th} edge and determine the intersection point of the extensions of the two neighboring edges. The intersection of Γ_i and the line passing through this point and (u,v) will determine the s_i parameter:

$$s_i = P^i(u,v) = \frac{1 - u\cos 2(i-1)\theta - v\sin 2(i-1)\theta}{2 - 2\cos 2\theta(u\cos 2i\theta + v\sin 2i\theta)} \tag{2}$$

where $\theta = \frac{\pi}{N}$; see Figure 6(b). It follows from the definition of the corner interpolants, that $t_i = 1 - s_{i-1}$ and $t_{i+1} = 1 - s_i$.

Thus Q_i is defined in the local (s_i,t_i) parameter space, and it interpolates not only the $C_i(s_i)$ and $C_{i-1}(s_{i-1}) = C_{i-1}(t_i)$ border curves, but also the corresponding $D_i(s_i)$ and $D_{i-1}(t_i)$ cross-derivative functions. We also assume that the cross-derivative functions are compatible, so there exists a unique twist vector (i.e. mixed partial derivative) at the corner point, denoted by T_i.

Then the equation of the corner interpolant is

$$\begin{aligned} Q_i(s_i,t_i) = {} & C_i(s_i) + C_{i-1}(s_{i-1}) - C_i(0) + t_i D_i(s_i) \\ & + s_i D_{i-1}(t_i) - t_i D_i(0) - s_i D_{i-1}(1) - s_i t_i T_i \end{aligned} \tag{3}$$

Note: these corner interpolants can be generalized to interpolate higher degree cross-derivatives as well.

Weighting functions

The weighting functions form a convex combination, i.e. they are non-negative and $\sum \alpha_i = 1, \forall (u,v)$.

In order to have a zero effect of the i^{th} corner interpolants along the other $N-2$ edges, the following must hold:

$$\alpha_i = \frac{\partial \alpha_i}{\partial u} = \frac{\partial \alpha_i}{\partial v} = 0, \quad \forall (u,v) \in \Gamma_j, j \neq i, i-1 \tag{4}$$

There are several ways to define such weighting functions, one option is a simple formula proposed by Charrot and Gregory[2].

$$\alpha_i = \frac{\beta_i}{\sum_{j=0}^{N-1} \beta_j}, \quad \beta_i = \prod_{j \neq i, i+1} s_j^2 \tag{5}$$

Using these equations the corner interpolants and the weighting functions can be evaluated for every (u,v) point of the domain and by means of Equation 1 the complete N-sided patch can be calculated. Performing this for the whole curve network a complete surface model of smoothly connected patches can be automatically built up, as was illustrated earlier in Figure 2.

4.3. Visualization

There are several ways to render the resulting surfaces and to verify their quality. These visual modes help to notice if continuity conditions are not satisfied or the curvature is not evenly distributed. These issues are particularly important in aesthetic design. Figures 7 and 8 show objects with slicing and isophotes to verify $G1$ and $G2$ continuity, respectively. In Figures 9 and 10 a curvature map and an environment map are shown for global quality verification.

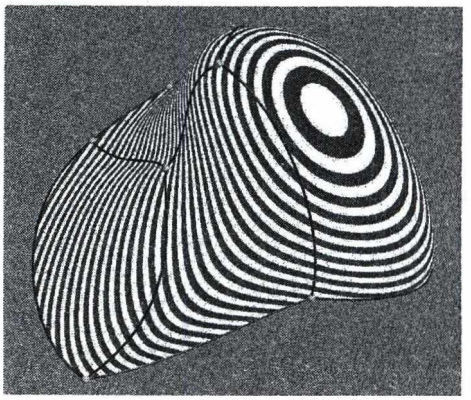


Figure 7: *Slicing*

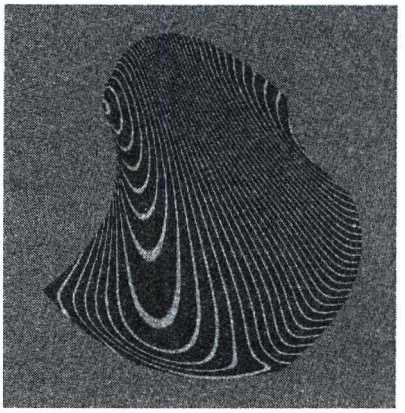


Figure 8: *Isophotes*

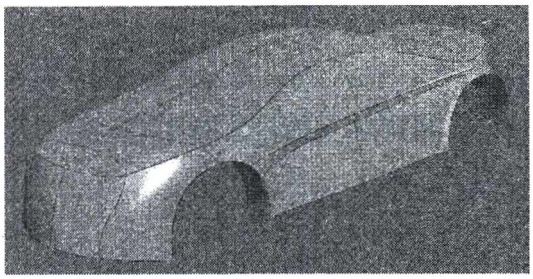


Figure 9: *Mean curvature map*

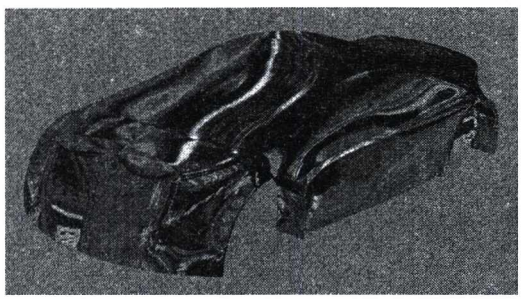


Figure 10: *Environment map*

5. Conclusion

A curve based method to create digital models of 3D objects has been described. The main difficulty is that the input is only a collection of hand-drawn sketches, which in mathematical sense only "loosely" define a free-form object. To obtain a consistent 3D structure a special tool, called construction box was introduced. Our method creates a 3D curve network, which is suitable to automatically span a complex surface, composed of general N-sided patches. The enhanced patch formulation provides internal smoothness, smooth connections and computational efficiency for real-time modifications. Algorithmic details were given and some encouraging results were shown, generated by a prototype styling system called Sketches. Future research issues will include global fairing of these type of patch complexes and developing further tools to adjust the interior of patches.

Acknowledgements

This research was conducted by Cadmus Consulting, Ltd. and later by Geomagic Hungary Ltd., Budapest. The project was supported by the Ministry of Education, grant number: OMFB-00567/2003. The Sketches prototype system is the result of a collective effort by a larger group; authors acknowledge important contributions by Zoltán Kovács, Volker Weiss and Bence Kodaj, and the group from the University of Industrial Design, Budapest led by Attila Bárkányi that tested and evaluated the system.

References

1. G. Farin: *Curves and Surfaces for CAGD, A practical guide*, 5th Edition, Academic Press, 2002 4
2. P. Charrot, J. Gregory: A pentagonal surface patch for CAGD, *Computer Aided Geometric Design*, Vol 1, 1984, pp 87–94 4, 5
3. D. Plowman, P. Charrot: A Practical Implementation of Vertex Blend Surfaces using an n-sided Patch, In: *Proc. of the Mathematics of Surfaces VI*, Ed.: G. Mullineux, Clarendon Press, Oxford, 1996, pp 67–78 4
4. M. Sonka, V. Hlavac, R. Boyle: *Image Processing, Analysis, and Machine Vision* 2nd Edition, Brooks/Cole Publishing Company, 1999 2, 3
5. T. Várady, T. Hermann: Best Fit Surface Curvature at Vertices of Topologically Irregular Curve Networks, In: *Proc. of the Mathematics of Surfaces VI*, Ed.: G. Mullineux, Clarendon Press, Oxford, 1996, pp 411–428 4

A general construction for barycentric coordinates in 3D polyhedra

Géza Kós,[1†]

[1] Computer and Automation Research Institute, Budapest; Department of Analysis, Loránd Eötvös Universitry, Budapest
kosgeza@sztaki.hu

Abstract

We prove a general formula which expresses a point inside a polyhedron with triangular faces as affine combination of the vertices. The free parameters of the formula allow to generate all possible weight sets; as a special case it provides an explicite formula for Floater's mean value coordinates in 3D and reproduces the discrete harmonic and Warren's coordinates.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computational Geometry and Object Modelling]: Hierarchy and geometric transformations

1. Introduction

In computational geometry, various applications, like finite element analysis or surface parametrisation, need constructions of barycentric coordinates in polygons or polyhedra.

Given a polygon or polyhedron $\mathcal{P}$ with vertices $v_1,\dots,v_n$. The goal is to construct real valued *coordinate* functions $\lambda_1,\dots,\lambda_n$ which map the interior (or just the kernel) of $\mathcal{P}$ to real numbers such that $\sum_i \lambda_i(v) = 1$ and $\sum_i \lambda_i(v)v = v$ for each interior/kernel point v. In some cases $\mathcal{P}$ is assumed to be convex; in other cases it is required that the coordinate functions satisfy $\lambda_i(v) > 0$ as well.

Most constructions define some weight functions w_i with the property

$$\sum_{i=1}^{n} w_i(v)\cdot(v_i - v) = 0$$

then normalise these functions to get the final coordinate functions,

$$\lambda_i(v) = \frac{w_i(v)}{w_1(v)+\dots+w_n(v)}.$$

Of course, for the normalisation it is required that the denominator does not vanish.

† Bolyai etc.

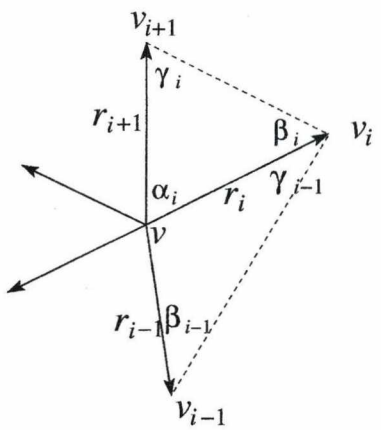


Figure 1: *Basic notations in 2D*

In the plane, the most commonly used constructions are the *Discrete Harmonic* coordinates [3], *Wachspress'* coordinates [1] and recently Floater's *Mean Value* coordinates [4]. Using the notations shown in Fig. 1, the Discrete Harmonic coordinates can be defined as

$$w_i(v) = \cot\beta_{i-1} + \cot\gamma_i; \qquad (1)$$

Wachspress' coordinates are

$$w_i(v) = \frac{1}{r_i^2}\left(\cot\beta_i + \cot\gamma_{i-1}\right); \tag{2}$$

finally, the Mean Value coordinates are defined as

$$w_i(v) = \frac{1}{r_i}\left(\tan\frac{\alpha_{i-1}}{2} + \tan\frac{\alpha_i}{2}\right). \tag{3}$$

All the three constructions have generalisations to higher dimensions. In this paper we consider the 3D case only.

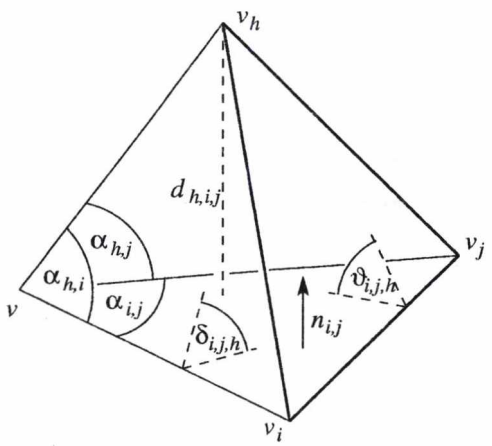


Figure 2: *Notations in 3D*

In 3D, the vertices of the polyhedron does not have a natural order which could be defined by their indices. Thus, we use the set $\mathcal{E}$ of edges and the set $\mathcal{T}$ of triangular faces. (Both Discrete Harmonic and Mean Value coordinates require triangular faces.) Edges are represented as pairs of vertex indices and triangles are represented as triplets.

For each edge $(i,j) \in \mathcal{E}$, denote the angle $\angle(v_i, v, v_j)$ by $\alpha_{i,j}$. For each face $(i,j,h) \in \mathcal{T}$, denote the dihedral angle between faces (v, v_i, v_j) and (v, v_i, v_h) by $\delta_{i,j,h}$; the angle between faces (v, v_i, v_j) and (v_j, v_i, v_h) by $\vartheta_{i,j,h}$. Finally, let the unit vector of triangle (v, v_i, v_j) be $n_{i,j}$ and denote the distance between vertex v_h and plane (v, v_i, v_j) by $d_{h,i,j}$ (see Fig. 1).

In 3D, the Discrete Harmonic coordinates can be computed as

$$w_i(v) = \sum_{j,h:(i,j,h)\in\mathcal{T}} |v_j - v_h| \cdot \cot\vartheta_{j,h,i}. \tag{4}$$

The Mean Value coordinates [7] are

$$w_i(v) = \sum_{j,h:(i,j,h)\in\mathcal{T}} \frac{\alpha_{j,h} - \alpha_{i,j}\cos\delta_{j,h,i} - \alpha_{h,i}\cos\delta_{h,i,j}}{2d_{i,j,h}}. \tag{5}$$

This construction has been found independently also by J. Warren [9].

The Wachspress' coordinates are generalised to convex polyhedra of any dimension by Warren et al. [2]. His construction allows faces with more than three sides but have the constraint that only three faces meet at each vertex. Denoting the normal vectors of the three faces sharing vertex v_i by $m_{i,1}$, $m_{i,2}$ and $m_{i,3}$ such that they point inside, Warren's coordinates are defined as

$$w_i(v) = \frac{\left|\det(m_{i,1}, m_{i,2}, m_{i,3})\right|}{\left((v-v_i)m_{i,1}\right)\cdot\left((v-v_i)m_{i,2}\right)\cdot\left((v-v_i)m_{i,3}\right)}. \tag{6}$$

Note that the constraint can be removed easily if vertices of higher degree are considered as degenerate edges. At the same time, faces with more than three edges can be divided into triangles by diagonals, without changing the coordinate functions.

Though formulae (1–6) look slightly different, we demonstrate that they are closely related to each other. We construct a general family of weight functions which contains them all as special cases. The difference between the formulae can be interpreted also as different simplified forms of the same equation, with different parameter sets.

2. The integral of normal vectors on surfaces

The following well-known fact will be our main tool.

Theorem 1. Let S be a closed, oriented, piecewise smooth surface of dimension $N-1$ in the space $\mathbb{R}^N$ and assume that its area is finite. Then the integral of normal vectors on the entire surface, $\int_S dS$ is 0.

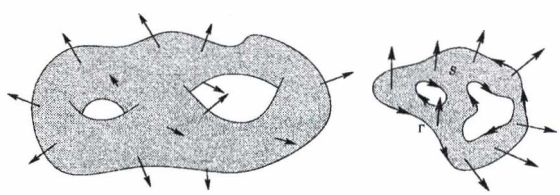


Figure 3: *Integral of normal vectors of a surface*

The theorem is a direct corollary of the general Stokes' theorem, but various attractive proofs exist which use more elementary tools. Now we show two of such proofs; the first one works only in 2D, the second one works in any dimension.

Proof in 2D. Consider S as a parametric curve $[0,1] \to \mathbb{R}^2$. For any vector x, let $x^\perp$ be the same vector, rotated by 90 degrees. Then

$$\int_S dS = \int_S \left(dS(x)\right)^\perp = \left(\int_S dS(x)\right)^\perp = 0^\perp = 0. \qquad \square$$

Proof in any dimension. Take an arbitrary unit vector u and a hyperplane U which is perpendicular to u. Consider an arbitrary surface piece of area vector dS, and project it to the

plane U. The signed area of the projection is exactly $u \cdot dS$ (see Fig. 4).

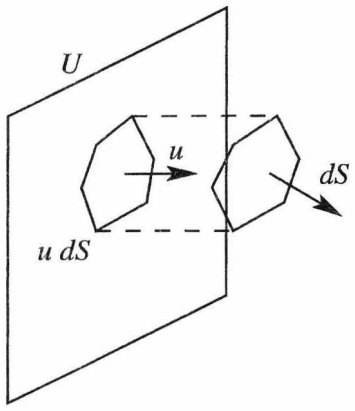


Figure 4: *Project area to plane U*

Projecting the entire surface S to the hyperplane, every point of U is covered the same number with positive and negative sign. Therefore, the integral of all projections is zero, $\int_S u \cdot dS = u \cdot \int_S dS = 0$. This is true for all unit vector u which implies $\int_S dS = 0$. □

Theorem 2. Let S be an oriented, piecewise smooth surface of finite area. The integral of area vectors on S depends only on the boundary. In other words, if S_1 and S_2 are two surfaces with the same boundary then $\int_{S_1} dS = \int_{S_2} dS$.

Proof. Invert the orientation of S_2 and glue the two surfaces together to obtain a closed surface S_{1+2}. Then $\int_{S_1} dS - \int_{S_2} dS = \int_{S_{1+2}} dS = 0$. □

3. The general formula in the plane

In this section we introduce the general formula for weights in the 2D case. (The results are introduced already in [6] in a little more computational way; now we focus on more intuitive geometric approaches.)

3.1. Generalising the Mean Value coordinates

To construct the formula of the Mean Value Coordinates, Floater [4] computes the integral of unit vectors in the angle domain $\angle(v_i, v, v_{i+1})$ and expresses it as a linear combination of $v_i - v$ and $v_{i+1} - v$ (see Fig. 5/a). The sum of all such expressions — the integral of all unit vectors — is always zero and the resulting weights are always positive.

The idea of the generalisation is the following. Replace the unit circle with a closed curve C which surrounds v. For each index i, denote by p_i the intersection point of C and the half line (v, v_i). (In case of more intersection points, choose one arbitrarily.) For each index i, compute the integral of normal vectors on the curve segment (p_i, p_{i+1}) and express

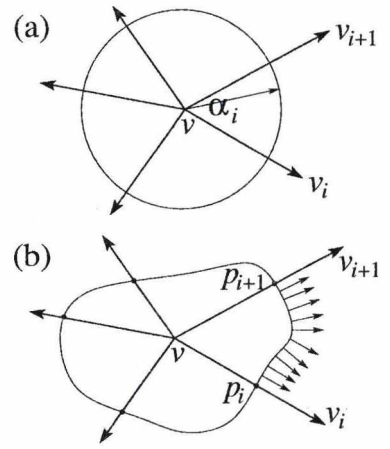


Figure 5: *Generalising the Mean Value coordinates. (a) Integrate unit vectors; (b) integrate normal vectors of a closed curve*

it as a linear combination of $v_i - v$ and $v_{i+1} - v$ (Fig. 5/b). By Theorem 1, the sum of all integrals is zero. Summing up the linear combinations for all i we obtain some real numbers $w_1, \ldots, w_n$ such that $\sum w_i(v_i - v) = 0$.

By Theorem 2, the curve segments can be replaced arbitrarily, only points p_i must be preserved. As an opposite way, we can choose points p_i first then connect them arbitrarily. Moreover, it is not required that p_i lies on the half line (v, v_i); it can be placed anywhere on line (v, v_i).

To place point p_i we introduce a real parameter, the signed distance between p_i and v, which is denoted by d_i. The distance is positive on the half line (v, v_i) and negative in the opposite direction. These distances will be the free parameters of the general formula.

To construct curves or polygons which connect the points $p_1, \ldots, p_n$, several reasonable ways exist. One of them is shown in Fig. 6 which walks around the star of centre v and points p_i. We use this curve to prove our general weight formula. Denote by n_i the unit vector which is perpendicular to $v_i - v$, in counter-clockwise direction. The integral of normal vectors (pointing outside) in the curve segment (p_i, p_{i+1}) is

$$-\int_{p_i}^{p_{i+1}} dC = d_i n_i - d_{i+1} n_{i+1} =$$

$$= d_i \frac{\frac{v_{i+1}-v}{r_{i+1}} - \cos\alpha_i \cdot \frac{v_i - v}{r_i}}{\sin\alpha_i} + d_{i+1} \frac{\frac{v_i - v}{r_i} - \cos\alpha_i \cdot \frac{v_{i+1}-v}{r_{i+1}}}{\sin\alpha_i} =$$

$$= \left(\frac{d_{i+1}}{\sin\alpha_i} - d_i \cot\alpha_i\right) \frac{v_i - v}{r_i} + \left(\frac{d_i}{\sin\alpha_i} - d_{i+1}\cot\alpha_i\right) \frac{v_{i+1}-v}{r_{i+1}}.$$

Summing up these expressions, we obtain the general formula.

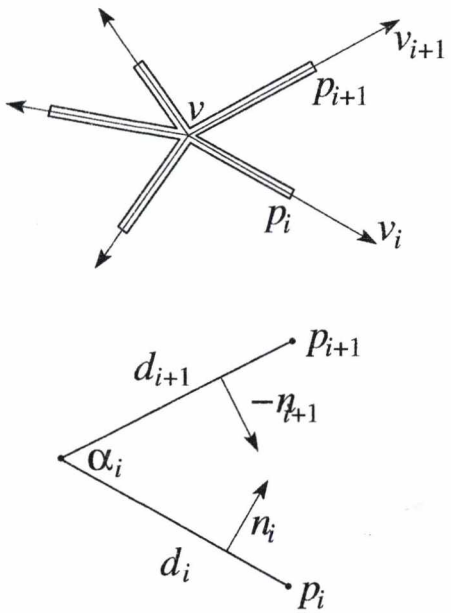


Figure 6: *Connect points p_i by walking around the star*

Theorem 3. Let $d_1, \ldots, d_n$ be arbitrary real numbers and define

$$w_i = \frac{1}{r_i}\left(\frac{d_{i+1}}{\sin\alpha_i} - d_i\cot\alpha_i - d_i\cot\alpha_{i-1} + \frac{d_{i-1}}{\sin\alpha_{i-1}}\right) \quad (7)$$

for all i. Then $\sum_{i=1}^{n} w_i(v_i - v) = 0$.

An alternative connecting curve is shown in Fig. 7. For each index i, put a line through p_i which is perpendicular to $v_i - v$ and denote the intersection point of the ith and $(i+1)$th line by q_i. Then the integral of normal vectors on the polygon segment (p_i, p_{i+1}) is

$$-\int_{p_i}^{p_{i+1}} dC = (p_i - q_i)^{\perp} + (q_i - p_{i+1})^{\perp} =$$

$$= \frac{(q_i - p_i)n_i}{r_i}(v_i - v) + \frac{(p_{i+1} - q_i)n_{i+1}}{r_{i+1}}(v_{i+1} - v).$$

(The numerators are the signed distances between q_i and the vertices p_i and p_{i+1}, respectively.) By summing up, we obtain the alternative formula for the same weights:

$$w_i = \frac{(q_i - q_{i-1})n_i}{r_i} = \frac{\pm|q_i - q_{i-1}|}{r_i}. \quad (8)$$

This result shows that the weights are positive if and only if the polygon $(q_1, \ldots, q_n)$ is not self-intersecting.

In practical applications, the parameters d_i are functions of v which can depend on v_i and have a geometric meaning as well. To generate formulae (1-5), d_i will be a very simple function of r_i.

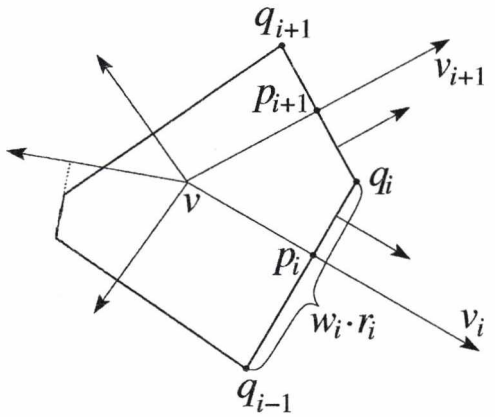


Figure 7: *Put perpendicular lines at points p_i*

3.2. Three special cases

Equation 8 is very useful in some special cases. The right angles and cyclic quadrilaterals enable attractive elementary geometric tools to avoid long computation. In this section we show three important special cases.

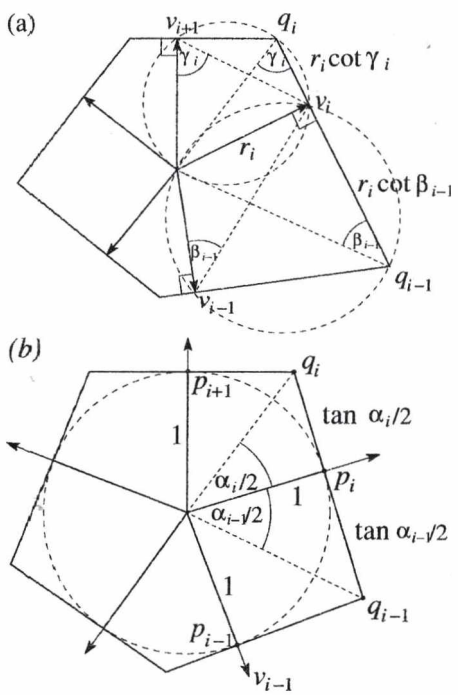


Figure 8: *(a) Discrete Harmonic coordinates: $d_i = r_i$. (b) Mean Value coordinates: $d_i = 1$*

The first example is $d_i = r_i$. As it can be read directly from Fig. 8/a, $r_i w_i = (q_i - q_{i-1})n_i = r_i(\cot\beta_{i-1} + \cot\gamma_i)$ Thus we

obtain the Discrete Harmonic coordinates, $w_i = \cot\beta_{i-1} + \cot\gamma_i$.

The second example is Floater's Mean Value coordinates [4]. Choosing $d_i = 1$, we have $|q_i - q_{i-1}| = \tan\frac{\alpha_{i-1}}{2} + \tan\frac{\alpha_i}{2}$ (see Fig. 8/b). Hence, this choice of parameters provides $w_i = \frac{1}{r_i}\left(\tan\frac{\alpha_{i-1}}{2} + \tan\frac{\alpha_i}{2}\right)$.

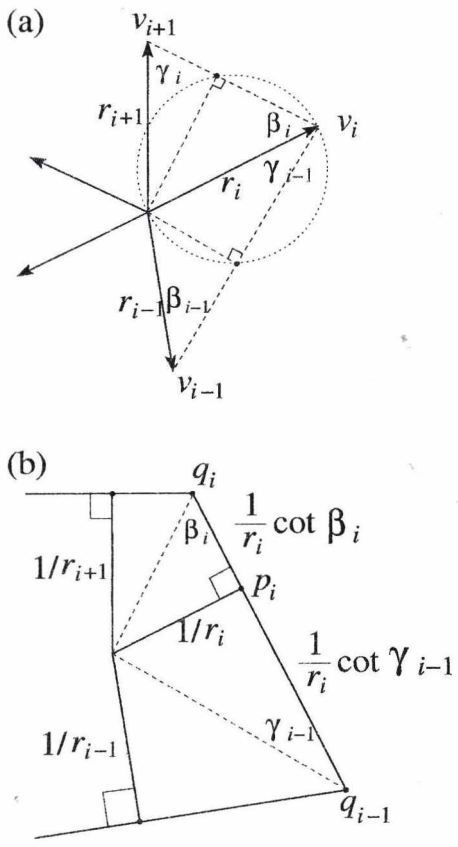


Figure 9: *Wachspress' coordinates: $d_i = 1/r_i$. (a) original; (b) inverted configuration*

The last example is Wachspress' coordinates [1]. Choose $d_i = 1/r_i$. Point p_i is obtained from v_i by inversion (see Fig. 9) and points q_i and q_{i-1} are the inverses of the footpoints of v to the lines (v_i, v_{i+1}) and (v_i, v_{i-1}), respectively. Then $(q_i - q_{i-1})n_i = \frac{1}{r_i}(\cot\beta_i + \cot\gamma_{i-1})$ and we obtain Wachspress' formula, $w_i = \frac{1}{r_i^2}(\cot\beta_i + \cot\gamma_{i-1})$.

3.3. Generality of the formula

Using the alternative construction (Fig. 7), the parameters $d_1, \ldots, d_n$ can be reconstructed easily from the weights.

Theorem 4. Let $w_1, \ldots, w_n$ be real numbers such that $\sum_{i=1}^n w_i(v_i - v) = 0$. Then there exist real numbers $d_1 \ldots, d_n$ for which Theorem 3 generates the weights w_i.

Proof. Construct a closed polygon $(q_1, \ldots, q_n)$ where $q_i - q_{i-1} = w_i \cdot (v_i - v)^{\perp}$. For each i, let p_i be the intersection of lines (q_{i-1}, q_i) and (v, v_i). Finally, choose $d_i = \frac{(p_i - v)(v_i - v)}{r_i}$. Starting from the parameters d_i, the construction builds the same points q_i and the same weights w_i. □

Note that the polygon $(q_1, \ldots, q_n)$ can be translated arbitrarily, thus our freedom is of degree 2. This is consistent to the number of variables and conditions: we have n free parameters while the system of all weight sets form a linear space of dimension $n - 2$.

4. The general formula in 3D

To generalise Theorem 3 to 3D polyhedra, let us start with the Mean Value coordinates again. For an arbitrary face $(i, j, h) \in \mathcal{T}$, consider the spherical triangle spanned by the directions $v_i - v$, $v_j - v$ and $v_h - v$. Take the integral of unit vectors in the spherical triangle and express it as a linear combination of vectors $v_i - v$, $v_j - v$ and $v_h - v$. Since the integral vector lies inside the trihedron spanned by these vectors, the three coefficients will be all positive. Summing up the coefficients for all faces, we obtain a linear combination of vectors $v_i - v$ with positive coefficients. On the other hand, the sum is the integral of all unit vectors which is 0.

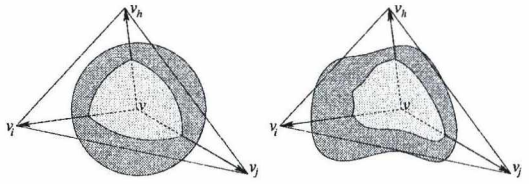


Figure 10: *Idea and generalisation in 3D*

Though the idea can be transformed easily, the computation becomes harder. Computing the integral of unit vectors on a spherical triangle needs more thoughts. A solution to this problem has been shown in [7] where the spherical triangle was replaced with the three bounding circle slices. Before going into details, let us introduce the generalisation.

Replace the unit sphere with an oriented, closed surface $\mathcal{S}$ which contains v in is interior (see Fig. 10). Replace the spherical triangle with the intersection of $\mathcal{S}$ and the trihedron bounded by the angle domains (v_i, v, v_j), (v_j, v, v_h) and (v_h, v, v_i) — denote it by $T_{i,j,h}$ and take the integral of normal vectors on this curved triangle. (For the simplicity, assume that $T_{i,j,h}$ is really triangle shaped.) The sum of all such integrals is the integral of normal vectors on $\mathcal{S}$ which is zero by Theorem 1. Therefore, we get a set of weights w_i such that $\sum w_i(v_i - v) = 0$.

For each edge $(i, j) \in \mathcal{E}$, consider the intersection of the interior of $\mathcal{S}$ and the angle domain (v_i, v, v_j). Denote the area of this "slice" by $A_{i,j}$ (see Figure 11. By Theorem 2, the

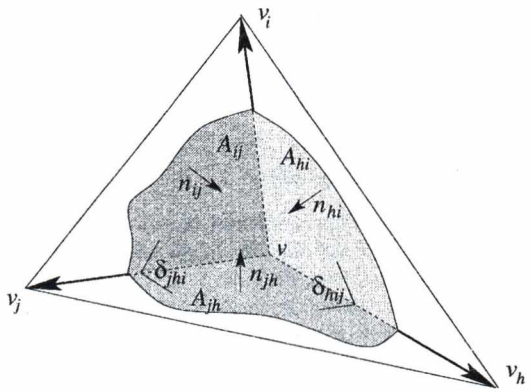


Figure 11: *Replace surface with the three "slices"*

integral of normal vectors is the same on $T_{i,j,h}$ and on the union of the three slices. Hence,

$$\int_{T_{i,j,h}} dS = A_{j,h}n_{j,h} + A_{h,i}n_{h,i} + A_{i,j}n_{i,j}.$$

We want to express this as $x\cdot(v_i-v)+y\cdot(v_j-v)+z\cdot(v_h-v)$ with some real numbers x,y,z. Taking dot-product with $n_{j,h}$,

$$\left(A_{j,h}n_{j,h} + A_{h,i}n_{h,i} + A_{i,j}n_{i,j}\right)\cdot n_{j,h} =$$

$$= \left(x\cdot(v_i-v)+y\cdot(v_j-v)+z\cdot(v_h-v)\right)\cdot n_{j,h},$$

$$A_{j,h} - A_{h,i}\cos\delta_{h,i,j} - A_{i,j}\cos\delta_{j,h,i} = x\cdot d_i.$$

Therefore,

$$x = \frac{A_{j,h} - A_{h,i}\cos\delta_{h,i,j} - A_{i,j}\cos\delta_{j,h,i}}{d_i}. \qquad (9)$$

In the final formula, the coefficient of $(v_i - v)$ is the sum of all such expressions.

Theorem 5. For each $(i,j)\in\mathcal{E}$, let $A_{i,j}$ be an arbitrary real number such that $A_{j,i} = A_{i,j}$ and define

$$w_i = \sum_{j,h:\ (i,j,h)\in\mathcal{T}} \frac{A_{j,h} - A_{h,i}\cos\delta_{h,i,j} - A_{i,j}\cos\delta_{j,h,i}}{d_i}. \qquad (10)$$

Then $\sum_i w_i(v_i - v) = 0$.

Proof. For each triangle $(i,j,h)\in\mathcal{T}$, take the vector $A_{i,j}n_{i,j} + A_{j,h}n_{j,h} + A_{h,i}n_{h,i}$ and express it as a linear combination of $(v_i - v)$, $(v_j - v)$ and $(v_h - v)$; the same formula (9) is obtained for the coefficient of $v_i - v$. Therefore, (10) is the sum of coefficients of $(v_i - v)$ for all triangles. On the other hand, for each edge (i,j), vectors $A_{i,j}n_{i,j}$ and $A_{j,i}n_{j,i}$ eliminate each other and the sum is always zero. □

Note that for n vertices, the weight sets form a linear space of dimension $n-3$ while the number of free parameters is larger, $3n-6$.

Theorem 6. Let $\omega_1,\dots,\omega_n$ be real numbers such that $\sum\omega_i(v_i - v) = 0$. Then the parameters $A_{i,j}$ in Theorem 5 can be chosen such that (10) generates these weights, $w_i = \omega_i$ for all i.

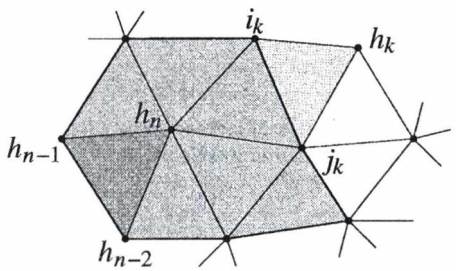


Figure 12: *Construct sequences (i_k), (j_k) and (h_k)*

Proof. Construct sequences of vertex indices, $i_1,\dots,i_{n-3}$, $j_1,\dots,j_{n-3}$ and $h_1,\dots,h_n$ with the following properties:

- $(h_{n-2},h_{n-1},h_n)\in\mathcal{T}$ is a face;
- For each $1\le k\le n-3$, $(i_k,j_k,h_k)\in\mathcal{T}$ is a face such that $i_k,j_k\in\{h_{k+1},\dots,h_n\}$ but $h_k\notin\{h_{k+1},\dots,h_n\}$.

The construction can be done inductively (see Fig. 12). First, select an arbitrary face (v_{n-2},v_{n-1},v_n). If $1\le k\le n-3$ and $h_{k+1},\dots,h_n$ are already defined, consider all faces spanned by vertices $v_{h_{k+1}},\dots,v_{h_n}$. This is not the complete polyhedron, so there exists at least one triangle which has two vertices among $v_{h_{k+1}},\dots,v_{h_n}$ and one vertex outside. Take such a triangle and set i_k, j_k and h_k to its vertex indices, according to the conditions.

We will use only parameters A_{i_k,j_k} $(k=1,\dots,n-3)$; set the remaining parameters to 0. The parameter A_{i_k,j_k} affects four vertices, the vertices of the two triangles which share the edge (i_k,j_k). These four vertices are v_{i_k}, v_{j_k} and v_{h_k}, and a fourth one in the set $\{v_{k+1},\dots,v_n\}$. It is important to remark that the coefficient of A_{i_k,j_k} in (10) is $1/d_{h_k}$ which is always positive. In other words, the matrix of the linear transform

$$\left(A_{i_1,j_1},\dots,A_{i_{n-3},j_{n-3}}\right)\mapsto\left(w_{h_1},\dots,w_{h_{n-3}}\right)$$

is a triangle matrix and the diagonal elements are all positive. Therefore, the parameters A_{i_k,j_k} can be chosen to obtain the requested weights $\omega_{h_1},\dots,\omega_{h_{n-3}}$.

Now we have a set of parameters $A_{i,j}$ such that $w_{h_k} = \omega_{h_k}$ for $k = 1,\dots,n-3$. To complete the proof, the same must be shown for $n-2\le k\le n$. Since

$$\sum_{k=1}^{n} w_{h_k}(v_{h_k} - v) = \sum_{k=1}^{n}\omega_{h_k}(v_{h_k} - v) = 0$$

and the first $n-3$ terms are the same, we have

$$\sum_{k=n-2}^{n}\left(\omega_{h_k} - w_{h_k}\right)(v_{h_k} - v) = 0.$$

But the vectors $v_{h_{n-2}} - v$, $v_{h_{n-1}} - v$ and $v_{h_n} - v$ are linearly independent, so this implies $\omega_{h_k} - w_{h_k} = 0$ for $k\ge n-2$ as well. □

4.1. Mean Value Coordinates

Again, we show three special choices for the parameters $A_{i,j}$ which generate known weight constructions. The first is Mean Value coordinates. The choice is of course the area of the circle slice:

$$A_{i,j} = \frac{1}{2}\alpha_{i,j}.$$

Substituting these parameters, we obtain the explicit formula for the Mean Value coordinates:

$$w_i = \sum_{j,h:\ (i,j,h)\in\mathcal{T}} \frac{\alpha_{j,h} - \alpha_{h,i}\cos\delta_{h,i,j} - \alpha_{i,j}\cos\delta_{j,h,i}}{2d_i}. \quad (11)$$

Though from this form it is not obvious, from the construction we already know that all fractions in (11) are positive. Of course, this can be proved directly, too. As it was shown in [5], if we project the spherical triangle to the plane (v, v_j, v_h), the area of the projection — which is positive — will be exactly $\alpha_{j,h} - \alpha_{h,i}\cos\delta_{h,i,j} - \alpha_{i,j}\cos\delta_{j,h,i}$.

4.2. Discrete Harmonic Coordinates

Similarly to the 2D case, let us choose $\mathcal{S}$ to be the polyhedron formed by the faces in $\mathcal{T}$. In this case

$$A_{i,j} = \frac{1}{2}|v_j - v_i| \cdot |v_h - v_i| \cdot \sin\alpha_{i,j}.$$

To compute the coefficients, consider an arbitrary face (i,j,h) and its area vector m (see Fig. 13).

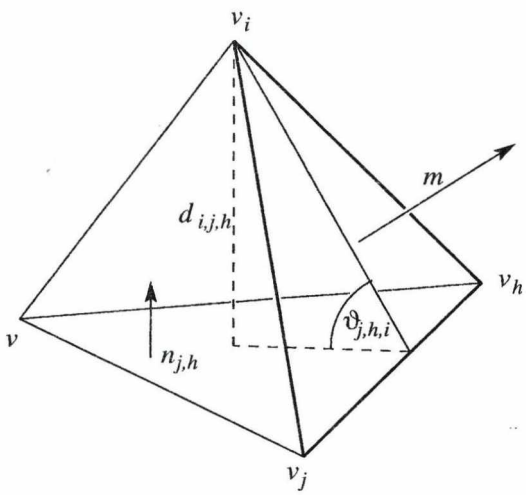


Figure 13: *Construct discrete harmonic coordinates*

The area of triangle (v_i, v_j, v_h) is $|m| = \frac{1}{2}|v_h - v_j| \cdot \frac{d_{i,j,h}}{\sin\vartheta_{j,h,i}}$. We need real numbers x, y, z such that $m = x(v_i - v) + y(v_j - v) + z(v_h - v)$. Taking dot-product with $n_{j,h}$,

$$x \cdot d_{h,i,j} = m \cdot n_{j,h} = |m| \cdot \cos\vartheta_{j,h,i} = \frac{1}{2} \cdot |v_j - v_h| \cdot d_{h,j,i} \cdot \cot\vartheta_{j,h,i}.$$

Hence, $x = \frac{1}{2}|v_j - v_h|\cot\vartheta_{j,h,i}$ and

$$w_i = \frac{1}{2} \sum_{j,h:\ (i,j,h)\in\mathcal{T}} |v_j - v_h|\cot\vartheta_{j,h,i}.$$

4.3. Warren's coordinates

To reproduce Warren's weights, consider a convex polyhedron where the faces may have more than 3 sides but only 3 faces meet at a vertex. Let v_i be an arbitrary vertex. Let $m_{i,j}$ $(j = 1,2,3)$ be the normal vectors of the three faces containing v_i. Denote by $f_{i,j}$ and $e_{i,j}$ the foot-points of v to the three faces and the edges starting from v_i. These points lie on the sphere of diameter (v, v_i) (see Fig. 14/a).

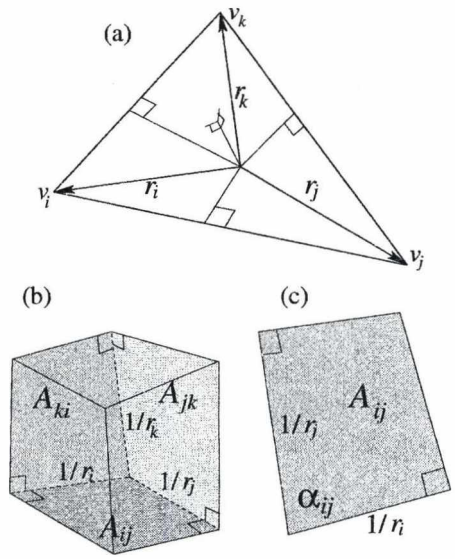


Figure 14: *Reproducing Warren's coordinates. (a) Foot-points; (b) inverted points and face of dual polyhedron; (c) slice of dual polyhedron*

Apply an inversion from point v and denote the inverted points by $e'_{i,j}$, $f'_{i,j}$ and v'_i (Fig. 14/b). These points lie in a plane which is perpendicular to (v, v_i) and points $e_{i,j}$ lie on the sides of the triangle $(f_{i,1}, f_{i,2}, f_{i,3})$. Constructing this triangle for all vertices, we obtain a dual polyhedron which has vertices instead of faces — the inverted foot-points $f'_{i,j}$ — and triangular faces instead of the vertices — each face is perpendicular to the corresponding line (v, v_i).

The volume of tetrahedron $(v, f'_{i,1}, f'_{i,2}, f'_{i,3})$ is

$$\mathrm{vol}(v, f'_{i,1}, f'_{i,2}, f'_{i,3}) = \frac{1}{6}\det(f_{i,1} - v, f_{i,2} - v, f_{i,3} - v) =$$

$$= \frac{\det(m_{i,1}, m_{i,2}, m_{i,3})}{6 \cdot |f_{i,1} - v| \cdot |f_{i,2} - v| \cdot |f_{i,3} - v|} =).$$

which is exactly $\frac{1}{6}$ of Warren's weight. On the other hand,

$$\mathrm{vol}(v, f'_{i,1}, f'_{i,2}, f'_{i,3}) = \frac{1}{3}\mathrm{area}(f'_{i,1}, f'_{i,2}, f'_{i,3}) \cdot |v'_i - v| =$$

$$= \frac{\mathrm{area}(f'_{i,1}, f'_{i,2}, f'_{i,3})}{3r_i}.$$

If we choose surface $\mathcal{S}$ to be the dual polyhedron, then $w_i(v_i - v)$ is the area vector of face $(f'_{i,1}, (f'_{i,2}, (f'_{i,3})$ and $w_i|v_i - v| = w_i r_i$ is the area. The dual polyhedron produces the halves of Warren's weights.

At this point constraints can be changed to our case. Allow vertices of degree more than three — replace tetrahedra $(v_i, f'_{i,1}, f'_{i,2}, f'_{i,3})$ by pyramids — and subdivide faces having more than 3 sides into triangles. To generate Warren's weights, use the slices of the dual polyhedron, shown in Fig. 14/c. (Note that in some cases this quadrilateral is self-intersecting and the area is the difference between the areas of the two bounded regions.) As an easy computation shows, the area can be expressed as

$$A_{i,j} = \frac{1}{r_i r_j \sin\alpha_{i,j}} - \left(\frac{1}{2r_i^2} + \frac{1}{2r_j^2}\right)\cot\alpha_{i,j}.$$

5. Summary and final remarks

We introduced a general family of weights in convex 3D polyhedra with the property $\sum w_i(v_i - v) = 0$. The free parameters allows to generate all possible weight sets. As special cases, we reproduced the Discrete Harmonic coordinates, Floater's Mean Value coordinates, Wachspress' coordinates and Warren's coordinates.

Due to space limitations, many subtopics, like continuity and extension problems have not been addressed. In the following paragraphs some results in these topics are listed which will go to separate papers.

The idea of the general formula, the statements and proofs of Theorems 5 and 6 and the special parameter settings for the three known constructions can be changed to work in any dimension. For the general formula in N-dimension, faces and edges will be replaced by $(N-1)$ and $(N-2)$-dimensional sub-faces and there will be a parameter for each $(N-2)$-dimensional face of $\mathcal{P}$.

The general constructions, Theorem 3 and Theorem 5 can be extended outside the kernel. In the plane, the sign of angle α_i must match the orientation of triangle (v, v_i, v_{i+1}). In 3D, the sign of $d_{i,j,h}$ must match the orientation of tetrahedron (v, v_i, v_j, v_h). However, the formula is singular on the extensions of the sides/faces of $\mathcal{P}$.

For the planar case Hormann [8] proved that Mean Value coordinates can be extended to the entire plane. (The proof works in more general cases as well when $\mathcal{P}$ is replaced by a set of closed polygons.) If the parameters are restricted to be continuous single variable functions, $d_i = f(r_i)$, the only construction which can be extended continuously to the entire plane is the Mean Value coordinates.

Hormann's result can be generalised to 3D — and higher dimensions as well. The recent paper of Ju, Schaefer and Warren considers the properties of the Mean Value coordinates in 3D. On the other hand, if the parameters $A_{i,j}$ are restricted to depend only on the geometric properties of triangle (v, v_i, v_j), there exists no more such weight construction.

References

1. E. L. Wachspress. A rational finite element basis. *Academic Press*, 1975.
2. J. Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, **6**(2):97–108, 1996.
3. A. Iserles. A first course in numerical analysis of differential equations. *Cambridge University Press*, 1996.
4. M. S. Floater. Mean value coordinates. *Computer-Aided Geometric Design*, **20**(1):19–27, 2003.
5. Solutions to advanced problems. *Mathematical and Physical Journal for Secondary Schools (KöMaL)*, **2004**(2):54–57, 2004
6. M. S. Floater, K. Hormann, G. Kós. A general construction of barycentric coordinates over convex polygons. *Advances in Computational Mathematics*, accepted in 2004
7. M. S. Floater, G. Kós, M. Reimers. Mean value coordinates in 3D. *Computer-Aided Geometric Design* **22**(7):623–631, 2005
8. K. Hormann. Barycentric Coordinates for Arbitrary Polygons in the Plane. *Institute of Computer Science, Clausthal University of Technology*, Technical Report IfI-05-05, 2005
9. T. Ju, S. Schaefer, J. Warren, Mean value coordinates for closed triangular meshes, *ACM Transactions on Graphics* **24**(3):561–566, 2005

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Properties and Applications of Neighborhood Sequences

András Hajdu[1], Lajos Hajdu[2†] and Tamás Tóth[1]

[1] Faculty of Informatics, University of Debrecen, H-4010 Debrecen, P.O. Box 12.
[2] Institute of Mathematics, University of Debrecen, H-4010 Debrecen, P.O. Box 12.

Abstract

The wide-range used distance measurement in digital image processing is the Euclidean norm nowadays. This is defined over the set of the real numbers that is why it does not suit the digital approach, perfectly. The use of neighborhood sequences is an alternative to the L_p metrics and because they are based on the set of integers it is more suitable for digital thinking. In this paper we summarize some properties of neighborhood sequences and their applications in image processing methods.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Image Processing and Computer Vision]: Segmentation I.5.3 [Pattern Recognition]: Clustering G.2.0 [Discrete Mathematics]: General

1. Introduction

The wide-range used distance measurement in digital image processing is the Euclidean (L_2) norm nowadays. As it is defined over the set of the real numbers, it only rarely produces a round result and that is why it is not too capable to the digital approach. The use of neighborhood sequences can be an alternative to the set of L_p metrics [2, 4, 5, 6, 7, 8, 9, 11, 18]. Neighborhood sequences are based on integers and they suit perfectly the digital approach.

In this paper we summarize some theoretical results on neighborhood sequences, like their geometric properties and certain approximating problems. We also illustrate some applications using neighborhood sequences. We show that most image processing methods that use distance functions are able to handle neighborhood sequences to measure distance. Moreover, the set of neighborhood sequences can make the image processing more flexible, because the same method can produce different results with only changing the sequence.

The structure of the paper is as follows. We introduce our notation in Section 2. In Section 3 we summarize the geometric properties of neighborhood sequences [11]. In Section 4 we discuss how the Euclidean metric can be approximated by neighborhood sequences [12]. Finally, in Section 5 we review classical image processing methods using neighborhood sequences [14].

2. Basic concepts and notation

Rosenfeld and Pfaltz [18] introduced the concept of octagonal distances by mixing the 4- and 8-neighboring relations in 2D. In [19] Yamashita and Ibaraki introduced the concept of general periodic neighborhood sequences in $\mathbb{Z}^n$. In this section we recall some definition and notation from Das et al. [5] and from Fazekas et al. [9] to define neighborhood sequences.

2.1. Periodic neighborhood sequences

Let $n \in \mathbb{N}$, and let $p, q \in \mathbb{Z}^n$ be two points in $\mathbb{Z}^n$. Denote by $\mathrm{Pr}_i(p)$ the ith coordinate of p. Let $m \in \mathbb{N}$, $0 \leq m \leq n$. the points p and q are called m-neighbors if the following two conditions hold:

- $|\mathrm{Pr}_i(p) - \mathrm{Pr}_i(q)| \leq 1$ for all $1 \leq i \leq n$,
- $\sum_{i=1}^{n} |\mathrm{Pr}_i(p) - \mathrm{Pr}_i(q)| \leq m$.

The sequence $B = (b(i))_{i=1}^{\infty}$, where $b(i) \in \{1, \ldots, n\}$ for all $i \in \mathbb{N}$ is called a neighborhood sequence in nD. B is called periodic if for some $l \in \mathbb{N}, b(i+l) = b(i)$ for all $i \in \mathbb{N}$. The brief notation of a periodic neighborhood sequence with a period l is $B = \{b(1), b(2), \ldots, b(l)\}$. If the period length is 1, the neighborhood sequence is called constant. In further

† Research was supported in part by the OTKA grants F043090, T042985 and T048791, and by the János Bolyai Research Fellowship of the Hungarian Academy of Sciences.

investigations we denote the set of n-dimensional periodic neighborhood sequences by S_n.

2.2. Distance measurement

Let p and q be two points in $\mathbb{Z}^n$ and $B \in S_n$. The point sequence $p = p_0, p_1, p_2, \ldots, p_m = q$, where p_{i-1} and p_i are $b(i)$ neighbors, called a B-path from p to q of length m. The B-distance $d(p,q;B)$ between p and q is the length of a shortest B-path between them.

A natural algorithm to measure distance by neighborhood sequences is presented in [9]. It is summarized as follows.

Let p and q two points in $\mathbb{Z}^n$, and $B = \{b(1), b(2), \ldots, b(l)\}$ an n-dimensional neighborhood sequence.

1. Let $x = (x(1), x(2), \ldots, x(n))$ be the non-ascending ordering of $|\mathrm{Pr}_i(p) - \mathrm{Pr}_i(q)|$ for $0 \le i \le n$.
2. In every step the first $b(i)$ coordinates of x should be decreased by 1.
3. The coordinates of the new x are rearranged into non-ascending order.
4. The second and third steps should are repeated until every coordinate of x becomes 0.

The distance function generated by neighborhood sequences are not metrics in general, as the triangle inequality is not always satisfied. This is shown by the following example. Let $p = (0,0), q = (1,1), r = (2,2) \in \mathbb{Z}^2$ and $B = \{2,1\}$ be a two-dimensional periodic neighborhood sequence. Then we have $d(p,q;B) = d(q,r;B) = 1$, however, $d(p,r;B) = 3$; see Figure 1.

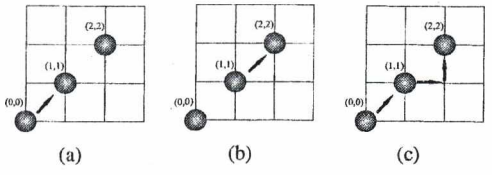


Figure 1: *Example for a neighborhood sequence which does not generate metric; (a) $d(p,q;B) = 1$, (b) $d(q,r;B) = 1$, (c) $d(p,r;B) = 3$.*

3. Geometric properties of neighborhood sequences

In this section we give an account of the results of [11]. A. Hajdu investigated in [11] the way neighborhood sequences spread in the digital space from a starting point of $\mathbb{Z}^n$. The spreading is translation invariant, so we may choose the origin $\mathbf{O}$ as the starting point. Let B be a neighborhood sequence in nD, and for $k \in \mathbb{N}$ let

$$B_k = \{p \in \mathbb{Z}^n : d(\mathbf{O}, p; B) \le k\}.$$

So B_k is the region occupied by B after k steps. Let $H(B_k)$ be the convex hull of B_k in $\mathbb{R}^n$.

Let $B \in S_n$ and $k \in \mathbb{N}$. If a point $p \in \mathbb{Z}^n$ with coordinates $(p_1, p_2, \ldots, p_n)$ belongs to B_k, then the coordinates $(\delta_1 p_{i_1}, \delta_2 p_{i_2}, \ldots, \delta_n p_{i_n})$ also belong to B_k, where $\delta_j = \pm 1 (j = 1, \ldots, n)$, and $(i_1, i_2, \ldots, i_n)$ is an arbitrary permutation of $(1, 2, \ldots, n)$.

Let $k(i)$ denote the number of i values among the first k elements of $B \in S_n$.

The vertices of B_k are exactly those points, whose coordinates are the permutation of the values

$$(\delta_1 q_1, \delta_2 q_2, \ldots, \delta_n q_n) = \left(\delta_1 \sum_{i=1}^{n} k(i), \delta_2 \sum_{i=2}^{n} k(i), , \ldots, \delta_n k(n)\right).$$

A. Hajdu performed an investigation in $\mathbb{Z}^2$ and $\mathbb{Z}^3$, see [11]. The analysis could be done in higher dimension and also in other kind of digital spaces (that are based on hexagonal or triangular neighborhood relation).

3.1. The 2D case

Das and Chatterji showed that for every 2D neighborhood sequence B, $H(B_k)$ is always an octagon, see [7]. They calculated the coordinates of the vertices of the octagon, and also its sides based on the ratio of 1 and 2 values among the first k elements of B.

The well-known cityblock and chessboard motions can be represented with constant neighborhood sequences $\{1\}$ and $\{2\}$, respectively. In Figure 2 we show the octagons occupied by these neighborhood sequences, and also by $\{12\}$.

Let B be a 2D neighborhood sequence, $x(k)$ the length of the horizontal-, and $y(k)$ the length of the inclined sides of the octagon $H(B_k)$, illustrated in Figure 2. Moreover, denote by $P_{2D}(k)$ the perimeter and by $V_{2D}(k)$ the area of the octagon, respectively. Because of symmetry the length of the horizontal and vertical sides are equal, and this holds for the inclined sides.

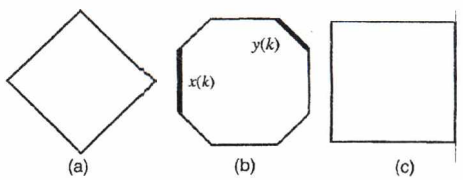


Figure 2: *Octagons occupied by 2D neighborhood sequences; (a) $B = \{1\}$, (b) $B = \{2\}$, (c) $B = \{12\}$; from [11].*

Using the above notation the following relations hold:

- $x(k) = 2k(2)$;
- $y(k) = \sqrt{2}k(1)$;
- $P_{2D}(k) = 4(\sqrt{2}k(1) + 2k(2))$;
- $V_{2D}(k) = 2(k(1)^2 + 4k(1)k(2) + 2k(2)^2)$.

For proofs see [11].

3.2. The 3D case

In 3D one can see that if $B \in S_3$ and $k \in \mathbb{N}$, then the region $H(B_k)$ is a polyhedron with at most 48 vertices, 72 edges and 26 faces. The vertices can be obtained by permuting the coordinates $(\delta_1(k(1)+k(2)+k(3)), \delta_2(k(2)+k(3)), \delta_3 k(3))$.

In our next examples we show the polyhedra obtained with neighborhood sequences containing one, two or three types of elements, see Figures 3, 4 and 5, respectively.

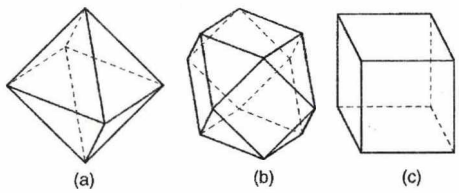


Figure 3: *Polyhedra occupied by constant 3D neighborhood sequences; (a) $B = \{1\}$, (b) $B = \{2\}$, (c) $B = \{3\}$; from* [11].

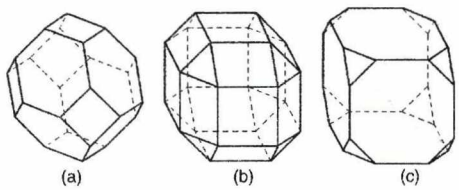


Figure 4: *Polyhedra occupied by 3D neighborhood sequences; (a) $B = \{12\}$, (b) $B = \{13\}$, (c) $B = \{23\}$; from* [11].

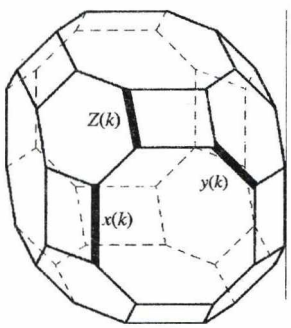


Figure 5: *Polyhedron occupied by the 3D neighborhood sequence $B = \{123\}$; from* [11].

By symmetry, the sides of $H(B_k)$ for a $B \in S_3$ can have at most three different lengths. They are indicated by $x(k)$, $y(k)$ and $z(k)$, respectively, as it is shown in Figure 5. The length of sides, the surface $P_{3D}(k)$ and the volume $V_{3D}(k)$ are computed as follows:

- $x(k) = 2k(3)$;
- $y(k) = \sqrt{2}k(2)$;
- $z(k) = \sqrt{2}k(1)$;
- $P_{3D}(k) = 6P_{oct} + 12P_{rec} + 8P_{hex}$, where
  $P_{oct} = 2(k(2)^2 + 4(k2)k(3) + 2(k(3)^2)$,
  $P_{rec} = 2\sqrt{2}k(1)k(3)$,
  $P_{hex} = \frac{\sqrt{3}}{2}(k(1)^2 + 4k(1)k(2) + k(2)^2)$;
- $V_{3D}(k) = 6\frac{P_{oct}H_{oct}}{3} + 12\frac{P_{rec}H_{rec}}{3} + 8\frac{P_{hex}H_{hex}}{3}$, where
  $H_{oct} = k(1) + k(2) + k(3)$,
  $H_{rec} = \frac{\sqrt{2}}{2}(k(1) + k(2) + k(3))$, $H_{rec} = \frac{\sqrt{3}}{2}(k(1) + k(2) + k(3))$.

For proofs cf. again [11].

4. Approximation of the Euclidean measurement

In this section we summarize some results from [12] about the approximation of the Euclidean distance L_2 in $\mathbb{Z}^2$ with digital distances generated by neighborhood sequences. We distinguish two types of approximation:

- finding a digital metric $d(A)$ which approximates L_2 best,
- approximating L_2 from below.

To handle these problems, we compare the regions occupied by a neighborhood sequence $A \in S_2$ with the Euclidean disks. We use the notation introduced in Section 3. Further, let

$$O_k = \{q \in \mathbb{Z}^2 : |q| \leq k\}$$

and

$$G_k = \{q \in \mathbb{R}^2 : |q| \leq k\}$$

denote the disks of radius k in $\mathbb{Z}^2$ and $\mathbb{R}^2$, respectively. The sets A_k and O_k are called the k-disks of the distances $d(A)$ and L_2, respectively.

To decide how a digital neighborhood sequence approximates the Euclidean distance L_2 on $\mathbb{Z}^n$, we compare the k-disks A_k and O_k. A natural error function could be the number of integer points in the symmetrical difference $A_k \bigtriangledown O_k$. However, there is no exact formula for the number of integer points inside O_k. So we compare $H(A_k)$ and G_k, and choose A to minimize the area of $H(A_k) \bigtriangledown O_k$. As it turns out, for every $k \in \mathbb{N}$ the same $A \in S_2$ can be chosen to minimize this area. This A can be regarded as the one that approximates L_2 best.

L. Hajdu and A. Hajdu performed several types of approximation in [12]:

Problem 1. Find a neighborhood sequence $A^{(1)} \in S_2$ such that for every $B \in S_2$ and $k \in \mathbb{N}$

$$\text{Area}(H(A_k^{(1)}) \bigtriangledown G_k) \leq \text{Area}(H(B_k) \bigtriangledown G_k).$$

Problem 2. Find a neighborhood sequence $A^{(2)} \in S_2$ such that $H(A_k^{(2)}) \supseteq G_k$ for every $k \in \mathbb{N}$ and $B \in S_2$, $H(B_k) \supseteq G_k$ implies that

$$\text{Area}(H(A_k^{(2)}) \setminus G_k) \leq \text{Area}(H(B_k) \setminus G_k).$$

Figure 6 illustrates Problems 1 and 2.

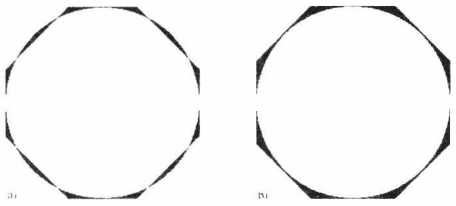


Figure 6: *The error of approximation; (a) general case, (b) when* $H(A_k) \supseteq G_k$.

Problem 3. Find a neighborhood sequence $A^{(3)} \in S_2$ such that $A^{(3)} \supseteq O_k$ for every $k \in \mathbb{N}$, and if $B \in S_2$ with $B_k \supseteq O_k$, then $B_k \supseteq A_k^{(3)}$.

For any $x \in \mathbb{R}$, let $\lfloor x \rfloor$ denote the largest integer which is less than or equal to x, and $\lceil x \rceil$ the smallest integer which is greater than or equal to x. The next result gives a solution to Problem 2.

Theorem. Let $A^{(2)} = (a^{(2)}(i))_{i=1}^{\infty}$ be the unique neighborhood sequence for which $k(2) = \lceil k(\sqrt{2}-1) \rceil$ $(k \in \mathbb{N})$, that is

$$a^{(2)}(i) = \lceil i(\sqrt{2}-1) \rceil + \lceil (i-1)(\sqrt{2}-1) \rceil + 1 \quad (i \in \mathbb{N}).$$

Then $H(A_k^{(2)}) \supseteq G_k$ for every $k \in \mathbb{N}$, and $B \in S_2$, $H(B_k) \supseteq G_k$ implies that

$$\text{Area}(H(A_k^{(2)}) \setminus G_k) \leq \text{Area}(H(B_k) \setminus G_k).$$

For the proof see [12].

The following result solves Problem 3.

Theorem. Let $A^{(3)} = (a^{(3)}(i))_{i=1}^{\infty}$ be the unique neighborhood sequence for which $k(2) = \lfloor k(\sqrt{2}-1) \rfloor$ $(k \in \mathbb{N})$, that is

$$a^{(3)}(i) = \lfloor i(\sqrt{2}-1) \rfloor - \lfloor (i-1)(\sqrt{2}-1) \rfloor + 1 \quad (i \in \mathbb{N}).$$

Then for every $k \in \mathbb{N}$, $O_k \subseteq H(A_k^{(3)})$. Moreover, if $B \in S_2$ such that and $O_k \subseteq H(B_k)$ for some $k \in \mathbb{N}$, then $A_k^{(3)} \subseteq B_k$. For the proof cf. again [12].

The distance function $d(A^{(3)})$ is the best one to approximate L_2 from below; that is for any $B \in S_2$, if

$$d(q, r; B) \leq L_2(q, r) \quad \text{for any} \quad q, r \in \mathbb{Z}^2.$$

then

$$d(q, r; B) \leq d(q, r; A^{(3)}) \quad \text{for any} \quad q, r \in \mathbb{Z}^2.$$

Because the solution of Problem 1 would need the introduction of further complicated notation, we omit the details. For the solution see [12].

5. Applications in image processing

In Section 5.1 we investigate some classical digital image processing methods in which neighborhood sequences can be used, while in Section 5.2 we introduce a new approach to image database retrieval. This section summarizes [14, 16].

5.1. Indexing and segmenting color images

In image processing we usually work with images with three color components as red, green and blue; this is called RGB color representation. Every component is an integer in $[0, 255]$. We use the 24-bit RGB cube, that is the domain between black$= (0, 0, 0)$ and white$= (255, 255, 255)$. We consider the points in this domain as colors. Thus we can measure distance between colors with neighborhood sequences. The investigated methods are the following:

- the fuzziness method,
- region growing,
- clustering.

Furthermore, we present a simple tool, the fuzziness histogram, which is to make the choice between neighborhood sequences easier.

The fuzziness method. The procedure selects those pixels which are within a given distance k to one or more initially fixed seed colors. This method is implemented in Adobe Photoshop, where it is referred to as the "Fuzziness" option [1]. The result of the fuzziness method highly depends on the chosen seed color(s), threshold and neighborhood sequence, see Figure 7.

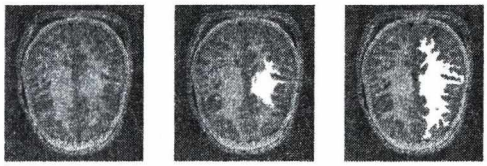
(a) (b) (c) (d)

Figure 7: *Fuzziness from the initial seed colors; (a) Original, (b)* $B = \{1112\}$, *(c)* $B = \{311\}$, *(d)* $B = \{3\}$.

Region growing. To obtain a connected region, we can insert a distance function used in the fuzziness method into a region growing algorithm. With this method we can get the connected region of the pixels within distance k from the starting pixel color, see Figure 8.

The condition of connectedness can be satisfied by using arbitrary neighborhoods.

Clustering. We recall an algorithm for indexing images based on cluster analysis [10]. In this method the elements of the RGB cube are classified into clusters using a suitable metric. In our investigations we used digital distance functions generated by neighborhood sequences, see Figure 9.



(a) (b) (c)

Figure 8: *Region growing of a medical picture; (a) Original image, (b) region growing with $B = \{1\}$, (c) $B = \{3\}$; the bound k for the color distance was 70 in both cases.*

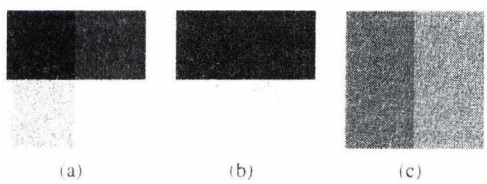
(a) (b) (c)

Figure 9: *Clustering into 6 colors; (a) Original, (b) $B = \{12\}$, (c) $B = \{23\}$.*

In this example we used a so called K-means clustering method. Of course, several other algorithms are known to solve this problem.

We illustrate the importance of choosing the suitable neighborhood sequence by the following example. Let $c_1 = (0,0,0)$, $c_2 = (15,15,255)$, $c_3 = (240,240,0)$ and $c_4 = (255,255,255)$ be four colors. We want to merge the two pair of nearest colors in one step, using different distance measurements. During this step we reduce the number of colors to two.

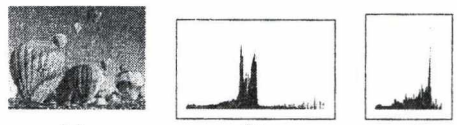
Using the constant neighborhood sequence $A = \{1\}$, the two pairs are (c_1, c_2) and (c_3, c_4), both with distance 285. If we use the neighborhood sequence $B = \{3\}$, the nearest colors and distances are $d(c_1, c_3) = d(c_2, c_4) = 240$. We present Figure 10 to illustrate this problem.



(a) (b) (c)

Figure 10: *Effect of different distance measurement; (a) Original, (b) using $A = \{1\}$, (c) using $B = \{3\}$.*

Fuzziness histogram. We present a tool that is to give a guideline to help with finding the optimal neighborhood sequences and threshold values for the above methods. This tool is based on the distance-histogram of the image. The kth column of the histogram illustrates the amount of the pixels inside distance k from the one or more seed color(s). The shape of the histogram highly depends on the chosen distance function; a "faster" distance function results a shorter histogram, but reasonable differences may occur with respect to the modality, as well, see Figure 11.

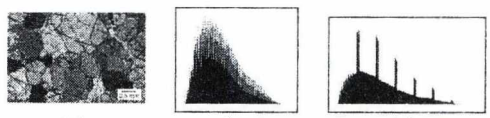


(a) (b) (c)

Figure 11: *Fuzziness histograms for the marked initial seed color, using different neighborhood sequences as distant functions; (a) Original image, (b) $B = \{1\}$, (c) $B = \{3\}$.*

The difference between two such histograms can be measured by suitable histogram measures [3]. The first mode of the histogram may play a very important role in region growing, as we do not care about colors far from the seed(s).

Global histogram. In [13] another type of histogram was proposed, which depends on only the chosen distance function and the image. The kth column of the histogram shows the number of chosen pixel pairs whose colors have distance k, see Figure 12. (In case of c) in the figure we used the periodic neighborhood sequence $B = \{b(1), \ldots, b(50)\} = \{3^{10}1^{40}\}$ with $b(i) = 3$ for $1 \leq i \leq 10$ and $b(i) = 1$ for $11 \leq i \leq 50$.) Fuzziness and global histograms are produced similarly, but in the latter case the first node has no particular importance.



(a) (b) (c)

Figure 12: *Global histograms for using different neighborhood sequences as distant functions (a) Original, (b) $B = \{123\}$, (c) $B = \{3^{10}1^{40}\}$.*

5.2. Image retrieval by neighborhood sequences

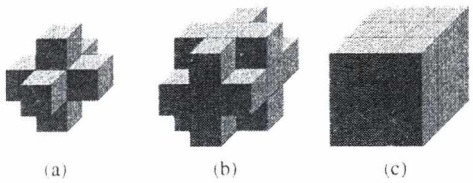
We can consider neighborhood sequences based on very general neighborhoods, see [19]. In [15,16] we introduced some restricted families having natural background for moving in the space. The first family contains the following neighborhoods:

$N_1 = \{\mathbf{O}, (0,0,\pm 1), (0,\pm 1,0), (\pm 1,0,0)\}$,

$N_2 = N_1 \cup \{(0,\pm 1,\pm 1), (\pm 1,0,\pm 1), (\pm 1,\pm 1,0)\}$ and

$N_3 = N_2 \cup \{(\pm 1,\pm 1,\pm 1)\}$.

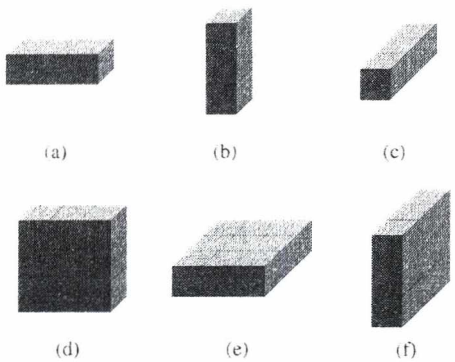
Note that these neighborhoods are based on the well known 6-, 18- and 26 neighborhood, respectively. We denote this classic subset of neighborhood sequences by CNS_3, see Figure 13.



(a) (b) (c)

Figure 13: *Neighborhoods used for CNS_3; (a) N_1, (d) N_2, (c) N_3.*

The next family of neighborhood sequences consist the followings:
$N_x = \{\mathbf{O}, (\pm 1, 0, 0)\}$, $N_y = \{\mathbf{O}, (0, \pm 1, 0)\}$, $N_z = \{\mathbf{O}, (0, 0, \pm 1)\}$,
$N_{xy} = N_x \cup N_y \cup \{(\pm 1, \pm 1, 0)\}$,
$N_{xz} = N_x \cup N_z \cup \{(\pm 1, 0, \pm 1)\}$,
$N_{yz} = N_y \cup N_z \cup \{(0, \pm 1, \pm 1)\}$ and
$N_{xyz} = N_{xy} \cup N_{xz} \cup N_{yz} \cup \{(\pm 1, \pm 1, \pm 1)\}$.
Each of these neighborhoods spans a 1D, 2D or 3D subspace of $\mathbb{Z}^3$, respectively, thus the set of the sequences generated by them is denoted by SNS_3, see Figure 14. With the sequences of these neighborhoods we can explicitly prescribe which coordinate(s) are allowed to change at a step, while CNS_3 sequences let as prescribe the number of the changeable coordinates only. Note that $CNS_3 \subset SNS_3$, nor $SNS_3 \subset CNS_3$ and $P_{xyz} = P_3$.



(a) (b) (c)

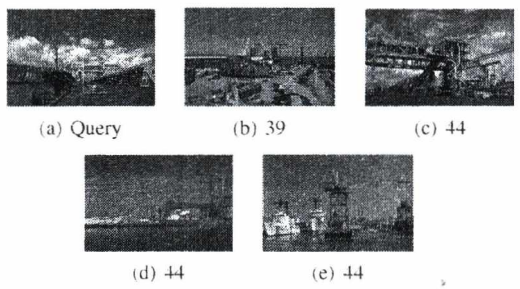
(d) (e) (f)

Figure 14: *Neighborhood used for SNS_3; (a) N_x, (b) N_y, (c) N_z, (d) N_{xy}, (e) N_{xz}, (f) N_{yz} (for $N_x yz$ see Figure 13(c).*

The third family of neighborhood sequences is the mixture of the CNS_3 and SNS_3 so the capable neighborhoods are the followings: $P_1, P_2, P_3, P_x, P_y, P_z, P_{xy}, P_{xz}, P_{yz}, P_{xyz}$. The set of the mixed sequences is denoted by MNS_3. Note that the definition of the three sets is extendable in arbitrary finite dimensions.

In image database retrieval we usually have three features: color, shape and texture, denoted by c, s and t to extract feature vectors (c, s, t). The neighborhood notation N_c, N_s, N_t will be used instead of N_x, N_y, N_z and similarly to other SNS neighborhoods. With (c, s, t) feature vectors we can find images similar to the query image in databases.

E.g. we want to select such images that are quite close in color and texture to the input image. The most important features should be achieved within the least steps, while non-important features should need more steps. Applying these considerations, a possible neighborhood sequence answer is $N = [N_{ct}^3][N_s^{40}]$. In this case we allow 3 steps in the c and t directions first, then s can be changed for 40 steps. The periodicity of N guarantees that we do not exclude vectors having larger values than 3 in either their c or t coordinates, though, they will be reached only after applying more periods. See Figure 15 for the matches ranked by their distance from **O**.



(a) Query (b) 39 (c) 44

(d) 44 (e) 44

Figure 15: *Query result for $N = [N_{ct}^3][N_s^{40}]$; (a) query image, (b-e) retrieved images and their norm.*

6. Conclusion

In this paper we summarized some results about neighborhood sequences. We introduced the basic concepts and presented some geometric properties of the polyhedra occupied by neighborhood sequences in $\mathbb{Z}^2$ and $\mathbb{Z}^3$; [11]. The question of approximating the Euclidean distance by digital metrics was also considered; [12]. We illustrated the use of neighborhood sequences in some digital image processing methods; [14]. We also indicated that such sequences can be useful in applications based on feature vectors; [16].

References

1. Adobe Photoshop Users Guide.
2. G. Borgefors: Distance transformations in arbitrary dimensions. *Comput. Vision Graphics Image Process.* **27** (1984), 321-345.
3. S.H. Cha and S.N. Srihari, On measuring the distance between histograms, Pattern Recognition 35 (2002), 1355-1370.
4. P.E. Danielsson: 3D octagonal metrics, *Eighth Scandinavian Conf. Image Process.*, 1993, pp. 727-736.

5. P.P. Das, P.P. Chakrabarti and B.N. Chatterji: Distance functions in digital geometry, *Inform. Sci.* **42** (1987), 113-136.

6. P.P. Das, P.P. Chakrabarti and B.N. Chatterji: Generalised distances in digital geometry, *Inform. Sci.* **42** (1987), 51-67.

7. P.P. Das and B.N. Chatterji: Octagonal distances for digital pictures, *Inform. Sci.* **50** (1990), 123-150.

8. A. Fazekas: Lattice of distances based on 3D-neighbourhood sequences, *Acta Math. Acad. Paedagog. Nyházi.* **15** (1999), 55-60.

9. A. Fazekas, A. Hajdu and L. Hajdu: Lattice of generalized neighbourhood sequences in nD and ∞D, *Publ. Math. Debrecen* **60** (2002), 405-427.

10. R. C. Gonzalez and R. E. Woods, Digital image processing, Addison-Wesley, Reading, MA, 1992.

11. A. Hajdu: Geometry of neighbourhood sequences, *Pattern Recognition Lett.* **24/15** (2003), 2597-2606.

12. A. Hajdu and L. Hajdu: Approximating the Euclidean distance using non-periodic neighbourhood sequences, *Discrete Math.* **283/1-3** (2004), 101-111.

13. A. Hajdu, J. Kormos, B. Nagy and Z. Zörgő: Choosing appropriate distance measurement in digital image segmentation, *Annales Univ. Sci. Budapest. Sect Comp.* **24** (2004), 193-208.

14. A. Hajdu, B. Nagy and Z. Zörgő: Indexing and segmenting colour images using neighbourhood sequences, *IEEE International Conference of Image Processing* (2003), Barcelona, Spain, I/957-960.

15. A. Hajdu, T. Tóth, K. Veréb: Noval approach for comparing similarity vectors in image retrieval, OAGM-HACIPPR 2005, Veszprém.

16. A. Hajdu, T. Tóth, K. Veréb: Flexible image retrieval using neighborhood sequences, *Pattern Recognition Lett.* (2005), submitted.

17. B. Nagy: Characterization of digital circles in triangular grid, *Pattern Recognition Letters* **25/11** (2004), 1231-1242.

18. A. Rosenfeld and J.L. Pfaltz: Distance functions on digital pictures, *Pattern Recognition* **1** (1968), 33-61.

19. M. Yamashita and T. Ibaraki: Distances defined by neighbourhood sequences, *Pattern Recognition* **19** (1986), 237-246.

Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Transformations of the triangular grid

Benedek Nagy

Faculty of Informatics, University of Debrecen, Hungary
Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain

Abstract

This paper is about transformations of the triangular grid. These linear transformations are widely used in square and cubic grids, where the space is described by independent coordinate values. The hexagonal and the triangular grids both belong to the basic (regular) grids, and they have more and more applications in Image Processing, Computer Graphics and in other fields of Computer Sciences. In our description – preserving the symmetry of the grid – we use 3 dependent coordinate values to refer the points of the triangular grid whose sum can be 0 and 1. Several applications are based on the transformations of the grid. In this paper basic transformations of the grid are provided and described in mathematical way. Isometric transformations, such as translations, rotations and mirror images are detailed by using the symmetric coordinate frame. An image-storing method for images on this grid is also presented.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: E.1 [Data Structures]: Arrays I.4.10 [Image Processing and Computer Vision]: Image Representation

Keywords: Digital geometry, Geometric transformation, Triangular Grid

1. Introduction

The digital geometry is an important part of Computer Graphics and Digital Image Processing. The classical digital geometry started by a paper of Rosenfeld and Pfaltz[9], where the two possible neighborhood relations on the square grid are defined. The theory is well developed for $\mathbb{Z}^n$, one of the basic books is written by Voss[13]. Nowadays, in many applications[1, 4] it is worth to consider other grids than the square one. With processing power of computers and capabilities of graphics devices increasing rapidly, the time is ripe to consider using non-square sampling for Image Processing, Computer Vision and Computer Graphics in earnest. The hexagonal and triangular based models have some advantages comparing them to the square based models[12]. The triangular grid is used for instance in surface construction, in Computer Graphics[14]. Some classical drawing algorithms for these grids are already known: algorithm generating digital straight lines by Freeman[3] and drawing digital circles on a triangular grid by Shimizu[11]. The theoretical study of non-square based structures is of interest as well, because it provides unified treatments of large classes of models for computer images[10]. The hexagonal grid and its dual grid have some nice properties – for instance, they have more symmetries (6 possible direction of axes) than the square grid (only 4 possible symmetry axes) – and they are regular grids, therefore it is not too hard to handle them. Moreover there are three types of neighbors in the triangular grid, which give more flexibility in applications. Before applying these grids a good mathematical description is required.

The Cartesian coordinate system fits very well to the square (cubic) lattice, the description is well-known, widely used and it is symmetric on the roles of the coordinates. The hexagonal lattice is described, for instance by Snyder et al[12]. In several papers, both in theoretical ones[7] and in applications[2] as well, there are two independent coordinate values that are used to refer to the points of the hexagonal lattice. Sometimes it is worth to use more values, even if they are dependent, to have a system which is more simple to use. Her[5] described the hexagonal grid by three co-

ordinate values in a symmetric way. Each hexagon is addressed by a zero-sum coordinate triplet. The transformations of the grid are analyzed using the symmetric coordinate frame by Her[6]. For the rectangular case the dual of the grid is identical with the original, therefore the Cartesian system can be used for both of them. It is not the case of the hexagonal grid; its dual is not identical with the original one, it is the triangular grid. This grid is also used in some applications (see Deutsch[2]), usually described by two coordinate values in a complex way. We want to describe this grid in a simple elegant way. This grid is not identical with the original hexagonal lattice, therefore the coordinate system given by Her[5] does not fit to the triangular grid. A symmetric coordinate frame describing the triangular grid will be used here based on Nagy[8]. Three coordinate values with sum 0 or 1 are used to refer to a point. There are three kinds of neighborhood relations in this grid which allow more flexibility in applications as well.

In this paper we are describing several linear transformations of the grid using the symmetric coordinate system for the triangular grid. In applications usually the isometric transformations play basic roles. For this reason we describe them. We would like to show that using the symmetric coordinate frame it is easy to handle the grid, and the way is open for the applications not only on the square and the hexagonal lattices, but on this grid as well. This paper presents a framework for drawing pictures, processing images on the triangular grid. It concentrates on storage and transformation of images.

After the introduction the structure of this paper is as follows.

In the second section we give notation and preliminaries, such as the coordinate system. In the third section of this paper we show some properties of the triangular grid using the symmetric coordinate frame with the help of lanes and half-planes. After this, we will consider different types of linear transformations of the grid and we will analyze what coordinate values the image-points of the original ones have. Then we present a practical method for storing images on the analyzed grid. In the last section we summarize our results.

2. Basic notation and concepts

In this section we recall some definitions and notation from the literature mentioned earlier concerning neighborhood relations and the coordinate system.

As Fig. 1 shows, there are three types of neighbors[2] among the triangles of the grid (we will call the triangles points). The figure displays a triangle with its

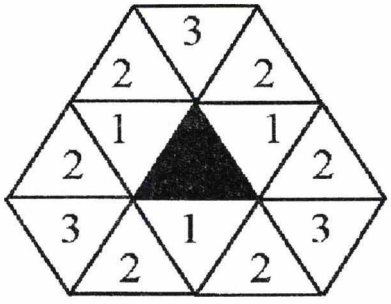


Figure 1: *Types of neighbors of triangles*

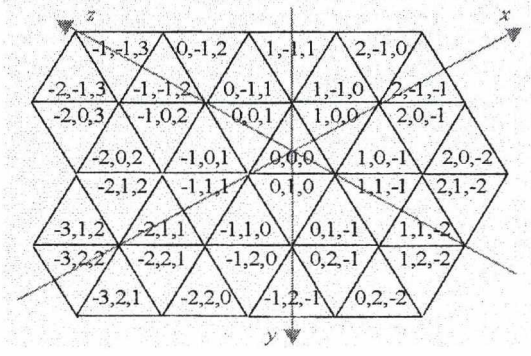


Figure 2: *Coordinate values of the triangular grid*

12 neighbors. Only the 1-neighbors have a common side, the 2- and 3-neighbors are corner-neighbors having only a common vertex. These relations are symmetric: if a point P is an m-neighbor ($m = 1, 2, 3$) of a point Q, then Q is an m-neighbor of P.

Now, for the mathematical description coordinate values are introduced[8] by the following method.

Let us fix a triangle as the Origin putting the triplet $(0, 0, 0)$ to it. This triangle has three symmetry axes. Let them be the (lines and directions of the) coordinate axes: x, y and z (in angle $\frac{2}{3}\pi$ by clockwise direction), respectively. Now let the coordinate triplets of the neighbors be $(1, 0, 0)$ (at the direction x), $(0, 1, 0)$ (at y) and $(0, 0, 1)$ (at direction z). When a step is taken through on a side of a triangle of the grid (i.e. to a 1-neighbor) parallel to an axis then the respective value of the triplet changes. Going to the direction of the axis it is increasing by 1, while going in the opposite direction it is decreasing by 1. Using the iterative steps all points get their respective triplets from their neighbors.

Fig. 2 shows a part of the grid with coordinate axes and the associated coordinate triplets.

With the assigned coordinate values we will handle the grid in a simple mathematical way. In this way every triangle has a unique triplet which exactly shows its place.

Using the coordinate values one can write the neighborhood relations in the following formal form. The points $P(P_x, P_y, P_z)$ and $Q(Q_x, Q_y, Q_z)$ of the triangular grid are m-neighbors ($m = 1, 2, 3$), if the following two conditions hold:

1. $|P_i - Q_i| \leq 1$, for $i \in \{x, y, z\}$,
2. $|P_x - Q_x| + |P_y - Q_y| + |P_z - Q_z| = m$.

We note that the triangular grid contains exactly those points which have sum of coordinate value 0 or 1. We call them *even* (shape $\triangle$ in Fig. 2, as the Origin) and *odd* points (opposite shape ∇), respectively. So the two possible shapes of the triangles represent the two possible *parities* of the points.

We will use the following definition of *lanes*, which is very helpful for our purpose. The sequence of points, for which a coordinate value remains constant, forms a lane. For instance, the lane containing the points having the third coordinate exactly 1 (i.e. $P(P_x, P_y, 1)$) will be referred as lane $z = 1$. Two lanes are *parallel* if their fixed values correspond to the same coordinate axis.

3. Properties of the grid

In this section we describe some properties of the grid with the assigned coordinate-values. We will use some of them later.

Observe, that an axis goes through on a triangle, if and only if two of the coordinates of the triangle have the same value.

Now, we are going to detail some properties of the lanes.

3.1. About the lanes

Remark 1 Every lane is orthogonal to one of the coordinate axes, that is the axis for which the coordinate value is fixed.

A lane has exactly one triangle meeting the coordinate axis, which is orthogonal to it, and two triangles (points) meeting each non-orthogonal axis. The lane $z = 1$ contains the point $(0, 0, 1)$ meeting the axis z, and two-two 1-neighbor points meeting the other two axes, namely $(-2, 1, 1)$ and $(-1, 1, 1)$ with x and the points $(1, -1, 1)$, $(1, -2, 1)$ with the axis y.

Two non-parallel lanes must intersect each other. If two lanes are not parallel then they have exactly two common points at their intersection (those triangles are 1-neighbors).

As we stated before, if a point meets an axis, then its two coordinate-values regarding the another two axes are the same. For instance every point on the axis x has the same second and third coordinate value. It means that the point is on the lanes for which the same value is fixed on axes y and z respectively. For example, the lanes $z = 1$ and $y = 1$ intersect each-other. They are orthogonal to the axes z and y, respectively. The two points where they meet are on the axis x: $(-2, 1, 1)$ and $(-1, 1, 1)$.

The next subsection is about some connected infinite parts of the grid.

3.2. On half-planes in the grid

With the coordinate axes and values we have half-planes where a coordinate is (non-)positive/negative. (We use the term half-plane as a set of points which belong to a Euclidean half-plane.) One can check on Fig. 2 that, for instance the points for which the value of the first coordinate is non-negative form a half-plane containing all the points $P(P_x, P_y, P_z)$ having $P_x \geq 0$. The intersection of two half-plane regions can be a third or a sixth of the plane. One can obtain a third of the grid, where a coordinate value (let it be, for instance, P_x) is positive and another one (let us say, for instance, P_y) is negative. The intersection of two half-planes where the coordinate values (let them be P_x and P_y) have the same sign (they are both positive or negative) determines a sixth of the grid. Since we have only points with two kinds of coordinate sum, in these cases the third coordinate must have opposite sign than these two ones. Therefore, the points for which a value (let us say at coordinate x) is positive and the two others are negative, are occupying a connected sixth of the plane, starting from the Origin and covering the axis for which the value is positive. Similar statement holds for a negative and two positive values.

The points between the positive part of an axis a and the negative part of another axis b (where a and b are any two of the axis x, y, z) have coordinate triplets with greatest value at the place according to the axis a and lowest value at the place according to b. As an example, one can check that the points $P(P_x, P_y, P_z)$ with $P_x \geq P_y \geq P_z$ form a connected sixth of the plane.

In the following sections we describe in a detailed way the transformations of the grid, such as, for example, translations, mirroring and rotations.

4. Translations

A translation which maps a point to a point with the same parity is an isometric transformation. If a translation maps a point to a point with the opposite parity, then the map-grid differs from the original one. This happens because the triangular grid is not a lattice. So, there is a grid-vector $\underline{v}(v_x, v_y, v_z)$ connecting two points of the grid such that the grid translated by $\underline{v}$ does not cover the original grid.

We are detailing only the first type of translations. Remember, that we are using only integer coordinates.

Proposition 1 A translation with vector $\underline{v}(v_x, v_y, v_z)$ maps the grid to itself if and only if $v_x + v_y + v_z = 0$.

Watching the neighbors which have the same parity, one can see that there are 3 basic translations (orthogonal directions of coordinate axes), they are: $\underline{x}(0, 1, -1)$ to orthogonal direction of axis x, $\underline{y}(-1, 0, 1)$ to direction according to y and $\underline{z}(1, -1, 0)$ orthogonal to direction z. One can check that the translation with vector $\underline{z} - \underline{y}$ transforms the grid to direction x by the translation unit of the grid in this direction (it is $\sqrt{3}$ times the length of a side of a grid-triangle). All translations can be given by using the basic vectors finitely many times. Moreover, our space is two dimensional, therefore we have the following statement.

Proposition 2 Hence any two of the basic vectors are independent, but altogether they form a linear dependent set, two basic vectors are enough to describe all translations. We will use vectors $\underline{x}(0, 1, -1)$ and $\underline{y}(-1, 0, 1)$. Each translation (given by a vector $\underline{v}$) has a unique linear decomposition to translations by $\underline{x}$ and $\underline{y}$.

Theorem 1 Let $\underline{v}(v_x, v_y, v_z)$ be a vector of a translation which maps the grid to itself. Then the image of every point can be calculated in the following way. Let $P(P_x, P_y, P_z)$ be a point of the grid, then its image is

5. Mirror images

In this section we are describing isometric transformations such as axial and central mirroring.

5.1. Axial mirroring

First we are analyzing the special cases, when the symmetry axis is the same as a coordinate axis.

Theorem 2 Using axis a (a can be any of the three coordinate axes x, y, z) as a symmetry axis a point P has mirror image P'_a such that the coordinate value according to the axis a is the same for the points P and P'_a; and the other two coordinate-values are interchanged.

For instance the point $P(1, 2, -3)$ has image $P'_x(1, -3, 2)$ mirroring it to the axis x.

The lanes orthogonal to the symmetry axis are transformed to themselves. A lane which is not orthogonal to the symmetry axis has image exactly that lane which has the same common points with the symmetry axis, i.e. it has the same value fixed but in other coordinate.

Proposition 3 Let P be a point. The points obtained by permutating the coordinate-values of P exactly the same points as the mirror images of P obtained by mirroring it to some of the coordinate axes.

As a consequence of the previous theorem we claim the following fact.

Corollary 1 A point has the same parity as its symmetric pairs (using axial mirroring to the coordinate axes).

Lemma 1 Every point has an image in the sixth $x \geq y \geq z$ of the grid.

It is a nice property of the assignment of coordinates to the grid, that a point and any of its symmetric pairs (mirroring the point to an axis) have the same coordinate values. So a point and its mirror images identical within a permutation of their coordinates.

Using our previous results from Section 4, we extend this result to the cases in which the symmetry axis is parallel to a coordinate axis.

So, let the symmetry axis be parallel to an axis, let us say, to x. Let $n \in \mathbb{Z}$ the parameter of the symmetry axis, i.e. the distance of the axes x and the symmetry axis in the following sense. Translating the grid with vector $n\underline{z}$ the image of the axis x is exactly the symmetry axis. (In this way all possible symmetry axes can be obtained which contains some of the vertices of the grid and it is parallel to the axis x.) Then the mirror image can be obtained by the translation (by $n\underline{z}$) of the mirror (using x as symmetry axis) of the translated (by $-n\underline{z}$) grid. In this way, we have the following proposition.

Proposition 4 Mirroring the grid to the axis parallel with x and parameter n, the image of a point $P(P_x, P_y, P_z)$ will be

$$P'_{nx}(P_x, P_z - n, P_y + n)$$

by a simple calculation.

Similar method works for other axes, using vector $\underline{x}$ and $\underline{y}$ for the translations when the symmetry axis is parallel to the axis y or z, respectively. Let the parameter of the symmetry axis be n. Therefore the images of $P(P_x, P_y, P_z)$ are

$$P'_{ny}(P_z + n, P_y, P_x - n)$$

and

$$P'_{nz}(P_y - n, P_x + n, P_z),$$

respectively.

Now we are considering the case when the symmetry axis is orthogonal to one of the coordinate axes. First we are dealing with special cases and after we will extend the results.

As a special case, for instance we will use the symmetry axis which is orthogonal to the axis x containing the side between $(0,0,0)$ and $(1,0,0)$. This line is between the lanes $x = 0$ and $x = 1$ such that these two lanes are mirror images of each others. The mirror image of the axis x is the axis x itself containing the points in inverse direction. (There is not any point of the grid which is image of itself.) The image of the axis y is a line parallel to the axis z in inverse direction, going through on the midpoint of the triangle $(1,0,0)$. The mirror image of the axis z can be obtained from the axis y by shifting it by the vector $(1,0,0)$ and consider it in opposite direction. So, we have the following formula.

Proposition 5 The mirror image of the point $P(P_x, P_y, P_z)$ using the symmetry axis between $(0,0,0)$ and $(1,0,0)$ is given as

$$P'_{x\perp}(-(P_x - 1), -P_z, -P_y).$$

Using another orthogonal symmetry axis to x we must use translations as well.

Theorem 3 Let the symmetry axis be orthogonal to x in the midpoint of the edge between $\left(n, -\left\lfloor \frac{n}{2} \right\rfloor, -\left\lfloor \frac{n}{2} \right\rfloor\right)$ and $\left(n+1, -\left\lfloor \frac{n+1}{2} \right\rfloor, -\left\lfloor \frac{n+1}{2} \right\rfloor\right)$ (we use the integer part function at the second and third coordinate values; we call n the parameter of the symmetry axis). Then the mirror image of a point $P(P_x, P_y, P_z)$ is given as

$$P'_{nx\perp}(2n + 1 - P_x, -n - P_z, -n - P_y).$$

Similarly, one can check the formulae for the cases when the symmetry axis is orthogonal to another coordinate axis. (One can use the vectors $\underline{x}$ or $\underline{y}$ for translations in case when the symmetry axis is orthogonal to the coordinate axis y or z, respectively.)

Proposition 6 The mirror images of a point $P(P_x, P_y, P_z)$, mirroring it to a symmetry axis orthogonal to y or z with parameter n are given as

$$P'_{ny\perp}(-n - P_z, 2n + 1 - P_y, -n - P_x)$$

and

$$P'_{nz\perp}(-n - P_y, -n - P_x, 2n + 1 - P_z).$$

Note, that these transformations change the parities of points.

It is a simple observation that a point can be transformed to any of its 12 neighbors by an axial mirroring.

Since all triangles have three directions of symmetry axes and there are three other directions of possible symmetry axes of the grid (containing sides of the triangles), we dealt all the possible symmetry axes of the grid. So, we finished with axial mirroring, let us see the central ones.

5.2. Central mirroring

There are two different possibilities for central mirroring. The center can be: a midpoint of an edge or a vertex of the grid (i.e. corner of triangles).

First we show the special case, when the center of the operation is the midpoint of the edge between $(0,0,0)$ and $(1,0,0)$. One can check, that the formula is the following.

Proposition 7 The mirror image of a point $P(P_x, P_y, P_z)$ by central mirroring to the center which is the middle-point of the edge between $(0,0,0)$ and $(1,0,0)$ is

$$P'_{c_x}(1 - P_x, -P_y, -P_z).$$

For the mirroring with center such that the edge containing the center of the transformation is not orthogonal to the axis x we use special cases the edges between $(0,0,0)$ and $(0,1,0)$ or $(0,0,1)$.

Proposition 8 The mirror image of a point $P(P_x, P_y, P_z)$ by central mirroring to the center in the middle between $(0,0,0)$ and $(0,1,0)$ is

$$P'_{c_y}(-P_x, 1 - P_y, -P_z);$$

by central mirroring to the center in the middle between $(0,0,0)$ and $(0,0,1)$ is

$$P'_{c_z}(-P_x, -P_y, 1 - P_z).$$

Let vector $\underline{v}$ be the parameter of the center, in the sense that translating the grid by $-\underline{v}$ the edge containing the center has mirror image between $(0,0,0)$ and one of the following three vertices (according to the direction of the edge): $(1,0,0)$, $(0,1,0)$, $(0,0,1)$.

Theorem 4 A center for mirroring is given. Let $\underline{v}(v_x, v_y, v_z)$ be its parameter vector. We denote the axis by a (it can be x, y or z) which is orthogonal to the side containing the center. The mirror image $P'_{\underline{v}c_a}$ of a point $P(P_x, P_y, P_z)$ is given by the following coordinate values: the value assigned to axis a is calculated

as $2v_a + 1 - P_a$, the other two values are $2v_b - P_b$, for each $b \in \{x, y, z\}$ with $a \neq b$.

Here we present a simple example for better understanding. Let the center of the transformation be the middle of the edge between $(-2, 0, 2)$ and $(-1, 0, 2)$. It is orthogonal to axis x and the parameter is $\underline{v}(-2, 0, 2)$. Let us compute the mirror of the point $P(-2, 0, 3)$. In our case $a = x$, therefore $P'_{(-2,0,2)c_x}$ has the following coordinate values:
$(2(-2) + 1 - (-2), 0 - 0, 2 \cdot 2 - 3) = (-1, 0, 1)$.

Now we are describing the case when the center of the mirroring is a vertex of the grid. Every vertex of the grid is a corner of six triangles, but three triangles are enough to have a unique address of the vertex. Let us consider, first, the center vertices at the corner of $(0, 0, 0)$, $(1, 0, 0)$, $(1, 0, -1)$, $(1, 1, -1)$, $(0, 1, -1)$ and $(0, 1, 0)$. One can compute that the following statement is true.

Proposition 9 The mirror image of a point $P(P_x, P_y, P_z)$ using central mirroring with center in a corner of triangle Origin which corresponds to direction $-z$ has the next coordinate values:

$$P'_c(1 - P_x, 1 - P_y, -1 - P_z).$$

Let us calculate the other cases, using translations. Let $\underline{v}(v_x, v_y, v_z)$ be the (parameter) vector which starts at the corner previously analyzed and ends at the center of the mirroring (another vertex of the grid). It is easy to check that $\underline{v}$ has integer coordinate values with zero sum. Then the transformation can be calculated in the following method.

Theorem 5 The mirror image of a point $P(P_x, P_y, P_z)$ using central mirroring at a corner with parameter $\underline{v}(v_x, v_y, v_z)$ is

$$P'_{\underline{v}c}(2v_x + 1 - P_x, 2v_y + 1 - P_y, 2v_z - 1 - P_z).$$

All these central mirroring operations change the parities of the points.

Note, that each central mirroring can be obtained by two axial mirrorings using orthogonal axes which are meeting at the center of the central mirroring. In the cases when the center of the transformation is on an edge one can use mirroring with parallel axis of the edge (exactly that one, which line is the extension of the edge) and mirroring with an axis orthogonal to the edge. For instance, the mirroring with center the midpoint of the edge between $(0, 0, 0)$ and $(1, 0, 0)$ one can use the mirroring to axes x and $x\perp$ (we used these indices for the given transformations). Similarly, when the center is a vertex of the grid, for example the one which is used in Proposition 9 we have: the axial mirroring with axes indexed by $x\perp$ and $-1x$.

Since the central mirroring changes the parity of the points, there is no this kind of mirroring to transform two points with the same parity to each other.

6. Rotations

In this paper we are using only rotations which map to the grid itself. Therefore the center of the rotation must be a kind of symmetry center of the grid. There are three possibilities:

- the center is a vertex of the grid,
- the center is a midpoint of an edge of the grid or
- the center is in the middle of one of the triangles.

In the latter case the rotation can be $2k\pi/3$ with $k \in \mathbb{Z}$. First, we will detail the special case when the center is in the Origin $(0, 0, 0)$. With simple computation we have:

Proposition 10 Rotating clock-wise direction the grid around the midpoint of the Origin with angle $2k\pi/3$, where $k = 3n + 1$ (for some $n \in \mathbb{Z}$) then the image of the point $P(P_x, P_y, P_z)$ is

$$P'_r(P_z, P_x, P_y).$$

With $k = 3n + 2$ (for some $n \in \mathbb{Z}$) the image of the point P is

$$P'_{rr}(P_y, P_z, P_x).$$

(In the case of $k = 3n$ the rotation does not change anything.)

Using center in another even point as the center of rotation, one can translate the grid, such that the center's image is the midpoint of $(0, 0, 0)$. The rotation can be made and after this the translation in backward direction. If a center is in an odd triangle then a mirroring is needed to have a transformation which maps the center into $(0, 0, 0)$. We showed how it is going in the previous section. So, we have:

Theorem 6 Let our transformation be a rotation around a center of the triangle $(V(v_x, v_y, v_z))$. (Let us say the vector $\underline{v}(v_x, v_y, v_z)$ is the parameter of the rotation.) Then the images of a point $P(P_x, P_y, P_z)$ are

$$P'_{\underline{v}r}(P_z - v_z + v_x, P_x - v_x + v_y, P_y - v_y + v_z)$$

(by $2\pi/3$ clock-wise direction) and

$$P'_{\underline{v}rr}(P_y - v_y + v_x, P_z - v_z + v_y, P_x - v_x + v_z)$$

(by $2\pi/3$ anti-clock-wise direction or by $4\pi/3$ clock-wise direction).

When the center is a midpoint of an edge of the grid, only the rotations with $k\pi$ ($k \in \mathbb{Z}$) transform the grid to itself, but these cases (for odd value of k) are

exactly the central mirroring case, therefore we skip it (they were detailed in the previous section). For even values of k the image of a point is exactly the same as the original one.

When the center of the rotation is a vertex of the grid, the rotation can be with angle $k\pi/3$ with $k \in \mathbb{Z}$. Here we are detailing the special case, when the the rotation center is a corner of triangles $(0,0,0),(1,0,0)$ and $(0,1,0)$ (and, of course, $(1,0,-1),(1,1,-1),(0,1,-1)$). Let us analyze what are the images of the different lanes.

Proposition 11 Rotate around the common corner of triangles $(0,0,0),(1,0,0)$ and $(0,1,0)$ clockwise direction by the angle $\pi/3$. Let $c \in \mathbb{Z}$ be a constant.
The lane $x = c$ has image lane $z = -c$.
The lane $y = c$ is mapped to the lane $x = 1 - c$.
Finally, a lane with $z = c$ has image $y = -c$.

Using the facts given above, one can compute the image of a point.

Proposition 12 The image of a point $P(P_x, P_y, P_z)$ rotating it by $\pi/3$ (clockwise) using the center at the corner of $(0,0,0)$ in the opposite direction of axis z as center:

$$P'_R(1 - P_y, -P_z, -P_x).$$

Note, that this rotation changes the parity of the points.

Using another angle ($k \neq 1$) one can repeat the above rotation. Note, that rotating by angle $(2k + 1)\pi, (k \in \mathbb{Z})$ (i.e. three times the rotation by angle $\pi/3$) the image is the same as it was with central mirroring with the same center; rotation by angle $2k\pi$ (i.e. rotation with six times by angle $\pi/3$) does not change anything.

When the center is in another hexagon, then first a translation must be used, as we used in the previous sections. After this the rotation can go and the transition which is the inverse of the original one. We can formulate it in the following way:

Theorem 7 Let us use the transformation, which rotates the grid by $k\pi/3$ ($k = 1, 2$) clock-wise direction around the center with parameter $\underline{v}(v_x, v_y, v_z)$. Let $P(P_x, P_y, P_z)$ be a point. The images of P are

$$P'_{\underline{v}R}(v_x + v_y + 1 - P_y, v_y + v_z - P_z, v_z + v_x - P_x)$$

and

$$P'_{\underline{v}RR}(v_x - v_z + 1 + P_z, v_y - v_x + P_x, v_z - v_y - 1 - P_y).$$

The rotations by larger angles can be obtained as the mirror images of the points P, $P'_{\underline{v}R}$ and $P'_{\underline{v}RR}$, by the same center.

7. Mixed transformations

In this section we are describing, which basic transformations are needed to have all detailed isometric transformations. As we showed in Section 4 two basic translations are enough to describe all translations. Allowing repetitions the translations by basic vectors $\underline{x}$ and $\underline{y}$ are enough. We need basic mirror operations, one can use them with indices x, $x\perp$ and $-1x$. We need rotations with indices r and R. Using the rotations more times one can obtain rotations with larger angles. The rotation with r interchange the coordinate axes, therefore one can do the mirroring to other axes as well. Finally, one can obtain the central mirroring by using two axial ones.

So, we can state, that all isometric transformations of the grid were computed.

8. Storing images of the triangular grid

In the previous sections of the paper some processing methods, namely the isometric transformations were described; now we give another practical aspect to Computer Graphic on the triangular grid.

In this section we show a method to store images of the used grid. The grid is described by coordinate triplets, but to use a real three dimensional storage-array is highly ineffective.

The number of data would be stored is the number of the elements of two two-dimensional arrays. So, there is a more rational way to store the data on the grid by using a three dimensional matrix, with two real dimensions (x and y) and only two possible values in the third dimension (denoted by $\overline{z}$). The points of the grid have triplets with sum 0 and 1. Therefore two given coordinate values (x, y) address exactly two points, an even (0-sum) and an odd (1-sum) point. The third 'dimension' of the array is refer to the parity of the addressed point. The actual third value of a stored point $(x, y, \overline{z})$ can be calculated in any time by the expression $z = \overline{z} - x - y$.

Using the storage system described above one can easily can write a program which transforms an image by any of the detailed transformations in an efficient way (all the transformations are in linear time computable).

Several well-known compression methods of images can also be used in a highly similar way than in the square case.

9. Conclusion

In this paper the mathematical background is presented to apply the triangular grid in Image Process-

ing and Computer Graphics. This grid has more symmetry than the square grid therefore one can get nicer results. The three types of neighbors give more flexibility in applications than the hexagonal or the square grids have. The triangular grid is described in a symmetric coordinate-frame.

The square grid can be stored as a matrix in the computer, using the coordinate system (that we used in this paper) the triangular grid can be stored as a three dimensional array (as a part of the cubic grid) or using much less space as two two-dimensional matrices (containing the odd and the even points, respectively). We recommend to use this grid in applications as well. To help to develop real applications the central transformations (rotation, mirroring, etc.) are provided for the Community. Similarly several axial coordinate-transformations are described. We presented formulae for any isometric transformations of the grid. These equations are the basic tools in many possible applications of this grid, such as Image Processing, Computer Graphics, etc. In some applications this grid may have nicer and better properties than the square-grid, it is a further research to discover and describe many of them (for instance the approximation of the Euclidean circles looks one of the candidates by symmetric reasons).

Acknowledgments

This research was supported by a grant from the Hungarian National Foundation for Scientific Research (OTKA F043090).

References

1. C.H. Chen, L.F. Pau and P.S.P. Wang (eds.), *Handbook of pattern recognition & computer vision*, (Handbooks in Science and Technology), Orlando, FL: Academic Press, 1986. 1
2. E.S. Deutsch, "Thinning algorithms on rectangular, hexagonal and triangular arrays," *Communications of the ACM* vol. 15, pp. 827-837, 1972. 1, 2
3. H. Freeman, "Algorithm for Generating a Digital Straight Line on a Triangular Grid," *IEEE Trans. Computers* vol. 28 pp. 150-152, 1979. 1
4. R.C. Gonzalez and R.E. Woods, *Digital image processing*, Reading, MA: Addison-Wesley, 1992. 1
5. I. Her, "A symmetrical coordinate frame on the hexagonal grid for computer graphics and vision", *ASME J. Mech. Design*, vol. 115, no. 3, pp. 447-449, Sept. 1993. 1, 2
6. I. Her, "Geometric transformations on the hexagonal grid", *IEEE Transaction on Image Processing*, vol. 4, no. 9, pp. 1213-1221, Sept. 1995. 2
7. E. Luczak and A. Rosenfeld, "Distance on a hexagonal grid," *IEEE Transactions on. Computers*, vol. C-25(5) pp. 532-533, 1976. 1
8. B. Nagy, "A symmetric coordinate system for the hexagonal networks", *Information Society 2004 - Theoretical Computer Science (IS04-TCS), ACM Slovenija conference*, Ljubljana, Slovenia, pp. 193-196, 2004. 2
9. A. Rosenfeld and J.L. Pfaltz, "Distance functions on digital pictures," *Pattern Recognition* vol. 1, pp. 33-61, 1968. 1
10. A. Rosenfeld, "Digital Geometry: Introduction and Bibliography", Technical Report CS-TR-140 / CITR-TR-1 (Digital Geometry Day 1997), Computer Science Department of The University of Auckland, CITR at Tamaki Campus, 1997. 1
11. K. Shimizu, "Algorithm for generating a digital circle on a triangular grid", *CGIP* vol. 15, pp. 401-402, 1981. 1
12. W.E. Snyder, H. Qi, and W.A. Sander, "A coordinate system for hexagonal pixels," *Prceedings of SPIE, Medical Imaging*, pp. 716-727, 1999. 1
13. K. Voss, *Discrete Images, Objects, and Functions in* $\mathbb{Z}^n$, Algorithms and Combinatorics; 11, Berlin, Heidelberg, New York: Springer-Verlag, 1993. 1
14. D. Zorin, "Smoothness of stationary subdivision on irregular meshes," *Constr. Approx.* vol. 16 no. 3, 359–397, 2000. 1

Interpolation by low degree Bézier-curves with different parameter sets

I. Juhász,[1†] M. Hoffmann[2]

[1] Department of Descriptive Geometry, University of Miskolc, Miskolc, Hungary
[2] Institute of Mathematics and Computer Science, Károly Eszterházy College, Eger, Hungary

Abstract

Given a sequence of points computation of the interpolating Bézier-curve is a standard method in CAGD. In this algorithm, however, the parameter values of the given points have to be defined in advance. Apart from the endpoints this parameterization heavily affects the shape of the Bézier-curve. The geometric influence of the changing parameter set is presented here for quadratic and cubic Bézier-curves. Choosing "best" parameterization from a geometric point of view is also discussed in this paper.

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

† This research is supported by the Hungarian National Research Fund research grant No.T 048523

Real-time Ray tracing with Image Coherence

Balázs Tóth†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapes/Hungary

Abstract

This paper presents a combination of ray-tracing acceleration techniques. This acceleration methods allow us to achieve interactive frame rates with the classic ray-tracing image synthesis method used originally for high-quality offline rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Ray-tracing

1. Introduction

Interactive rendering systems provide a powerful way to explore complex environments. Until recently the processing power of computers did not allow us to achieve high frame rates with ray-tracing based algorithms. The only interactive rendering methods were hardware accelerated polygonal rendering systems, which are less flexible and are poorer in providing sophisticated lighting effects.

Software-only methods are easy to modify and extend, which makes them a good candidate to experiment with various interaction and rendering methods. Nowadays an optimized ray tracer-based software rendering system can reach the performance of a polygonal algorithm.

The recursive ray-tracing algorithm is fairly easy to understand and implement, but it's powerful enough to examine the possibilities of the algorithmic and implementational optimizations. With the use of spatial subdivision algorithm and some optimization we can reach decent frame rates.

1.1. Recursive ray-tracing algorithm

The idea behind this algorithm is to simulate the path of light rays [7]. The most important part of the algorithm is the light-trace function. In this function we must determine the object the ray hits first. In the next step we calculate the direct contributions of the light sources with the diffuse and specular components of the hit object's material. In order to handle the reflective and translucent materials, we must determine the next object through the hit point. In this case we spawn new rays (the origins are the hit points and the directions are calculated using Snell's law) and trace them to determine these components.

The most time consuming part of this algorithm is the hit point search. If we want to make a high-resolution image, we have to trace million rays through the modell space. A naive ray-tracer calculates the intersection point with every object to find the closest. With the secondary and shadow rays the required computation will be enormous.

2. Subdivisional Acceleration Structure

To lower the number of intersection test we can use some acceleration structure. These structures are based on the repetitive subdivision of the modell space. With the use of these structures we can exclude large number of objects from the ray-object test, because we can determine group of objects which can't hit by the ray. For example, the objects behind the ray origin or out of the viewing frustrum.

2.1. KD-Tree implementation

The best subdivision method is based on a special data structure called kd-tree. My research is based on Vlastimil

† tbalazs@iit.bme.hu

Havran's thesis [2], who did an extensive study of available spatial subdivision schemes (regular grids, nested grids, octrees and kd-trees). He concluded that kd-trees beat others in most cases. It was also shown that the average number of intersection tests to find the closest intersection can be made as small as 2-3 independently of the number of objects [5,6].

The kd-tree is an axis-aligned Binary Space Partitioning tree. The space is partitioned by splitting it into two halves. The halves are processed recursively until every partition contains only one object. The most important difference compared to other schemes is that the position of the partitioning plane is axis aligned but not fixed. The use of the axis-aligned splitting planes has several advantages. Most importantly, it makes intersection test inexpensive with low memory footage of the tree [4].

2.1.1. Building the tree

To build the tree, we must determine the right positions of the splitting planes. The simplest method is to choose it in a way which ensures that the numbers of objects on both sides of the plane are roughly the same. This method is not the best because it doesn't produce empty nodes and the ray-tracer must check all objects in each node during the ray traversal.

Better trees can be constructed using a heuristic splitting rule. A good heuristic tries to isolate the empty spaces. In such nodes the traversal algorithm can travel through without expensive intersection tests. Such a heuristic is the Surface Area Heuristic. The basic idea is that the probability of a ray hitting object is related to it's surface area. The area of the node is

$$2 \times width \times length \times height.$$

The cost of traveling in a node is

$$Travel + Area \times ObjectsInTheNode \times IntersectionTest,$$

where *Travel* is a constant traversing cost in an empty node and the *Intersection Test* is a constant cost of ray-object intersection test. The splitting of the node produces two new nodes, so the splitting cost is calculated by summing the new node's costs. If we always use the less expensive splitting plane, we get a good kd-tree. To find the good splitting plane, we must test all the possible planes. There are many planes to choose from, but the number of the interesting positions is limited. The limitations are that the splitting plane must be axis aligned and must touch the border of at least one object.

Even with these constraints there are a large number of candidates. So building a kd-tree is a slow process, but with a static scene we must build the tree only once. The hit search with a good kd-tree is four times faster than the regular grid and many times faster than the naive test-with-all-objects method.

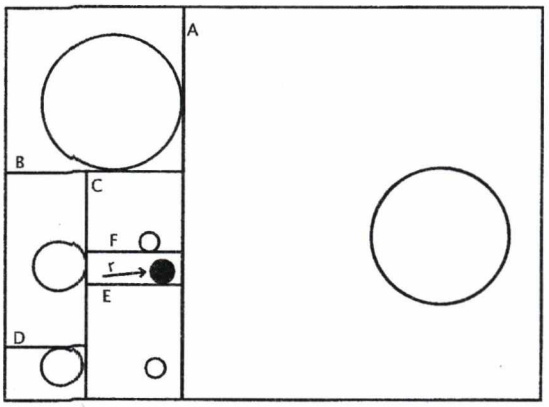


Figure 1: *Example of the subdivision*

2.1.2. Storing the tree

The kd-tree is built up from nodes that store the position of the splitting plane, a flag that indicates whether the node is a leaf node or not. If the node is not a leaf, then it must contain a pointer to its left child node.

If the node is a leaf node it contains a pointer to the list of objects in the node. These data members of the nodes can be stored in 8 bytes. We allocate the child nodes in pair at a 16 bytes boundary. With this allocation scheme we can save a pointer in each node, because the position of the right child node is right after the left node. The size of the node pair is 16 bytes, therefore in a single 64 kbytes cache line we can store four node pairs. When the traversal algorithm reads the left child node, the right child is loaded into the cache because of the behavior of the cache loading process. This improves cache performance if the series of the successive rays walk through the same nodes.

2.1.3. Traversing the tree

The ray traversal algorithm is a simple repetitive point-location search in the tree along the ray path. First we determine the point of the ray origin in the tree. If the node is not empty, we test the intersection of the objects with the ray and select the closest intersection point. If the node is empty or we didn't find an intersection, we determine the exit point of the node along the ray's direction. The exit point is slightly moved forward along the ray path to ensure that the next point-location search is in the next leaf and then the ray-traversal algorithm is called recursively. This recursion is terminated when a hit point is found or when the ray is out of the scene.

If the ray origin is out of the tree, we must determine the entry to the tree along with the ray in the first step.

The figure 2 is an example search of the location of ray

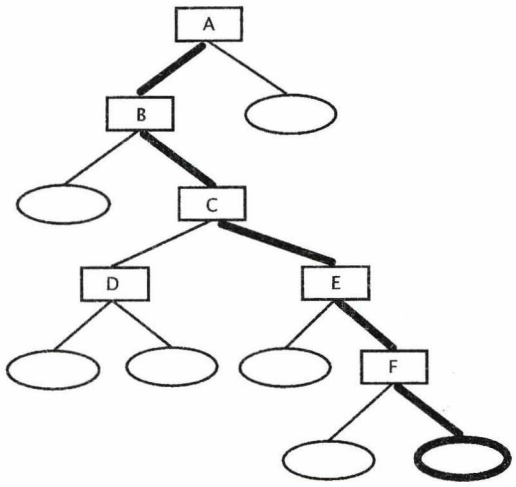


Figure 2: *Traversing the tree*

r in figure 1. The process starts at the root node of the tree. On every level we compare the origin of the ray with the position of the splitting plane. We choose the left or right child node by the comparison, and we go down the tree until we find a leaf node. In this leaf node will be the location of the ray. We test all the objects in this node, and if there is an intersection point (in the scene in figure 1 the ray r will intersect the black sphere) we finish the searching process. If there is no intersection we go to the next leaf, which is selected by the exit point of the ray from the node.

3. Ray grouping

Nowadays on modern CPUs the mathematical operations are faster than memory access. To exploit this computational power, we need to use the SIMD† capability of the CPU. With the SIMD extension we can handle four rays for the cost of one. This only possible if we carefully choose which rays to shoot together, as there is large spatial coherence in the geometry which can we exploit. This is especially true for the primary rays.

The proper selection of the groupped rays ensures to utilize the full power of the modern CPUs caching mechanism, because the performance gain correlates with the group size (Current SIMD implementations support only four simultaneous operations).

If we want to use ray grouping, we must deal with the situation when rays go through different paths in the tree. As the number of groupped rays increases, so increases the chance that these rays will diverge in the traversal process. We can deal with this problem if we use variable number rays and regroup them if necessary, or we can mark the diverging rays and disable them. All these methods has drawbacks, because the administration cost of the regrouping or marking the rays can easily amortize the performance gain of ray grouping.

3.1. Alternative entry points

To reduce the administration cost of the ray grouping, we can use alternative entry points to the acceleration tree as Alexander Reshetov et'al. [3] proposed. If we use ray group of four ray to represent and characterize a beam of rays as Assarsson and Möller defined [1], we can find alternative entry point for each ray beam and don't need to start at the root node of the tree for every ray. Ideally we might even find an optimal entry point, which is right at the leaf node. In this case we can find the first hit object in a single step, which reduces the traversing steps significantly.

To find an alternative entry point, we trace the corner rays as long as they walk through the same way in the tree. If at least one of the rays is diverging we can do two things. On the one hand, if the actual node is deep enough in the tree we can mark this node as the entry point for the rays in the beam. On the other hand, we can decompose the beam to more beams and search for individual entry points for every child beam. In the decomposition process we can split the beam to equal parts, or along the cells of the tree. The first method is faster but the second produces less beams along the searching process.

3.2. Tile based rendering

Ray grouping with alternative entry points tries to find a better entry point for the rays in the beam. If the beam represents a wide range, the common entry point will be in the higher region of the tree, so the benefit of this algorithm will be not too much unless we use the costly beam splitting. For primary rays, it is practical to split the viewing frustrum to equal sized sub-groups. The easiest way is to split the image plane into equal tiles. This tiles can be rendered independently, so they can assigned to a separate processor or machine if we use multi-processor or clustered system.

The tile based rendering is improve the cache performance too, because the chance of two rays - which are next each other - hit the same object is large. In this case the needed part of the acceleration tree and the geometrical information will be in the high speed cache memory and don't need to read from the slower main memory of the system.

† Single Instruction Multiple Data

3.3. Importance sampling

A recursive ray-tracer spawns new rays at every intersection point to determine the contributions of the light sources and other objects. The amount of the secondary ray's contributions is based on the scene setup and the attributes of the materials. Diffuse materials need far less secondary rays than the shiny, reflective surfaces.

The depth level of a secondary ray can be limited too. If the contribution of a secondary ray is negligible we can stop the ray spawning process before the depth reaches the maximum.

With these limitations we can reduce the count of the traced rays with a small decrease of quality.
In figure 3 the rays belong to white pixels are terminated before they reached the limit of the ray-tracing depth.

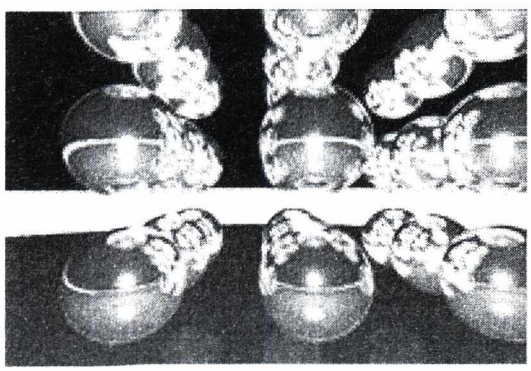


Figure 3: *Importance sampling*

3.4. Supersampling

A common way to improve the calculated picture's quality is supersampling. This means that the ray-tracer gets more samples through a pixel and calculates the average of them to get the final color. This results in an anti-aliased picture, but the supersampling process virtually enlarges the picture size, which requires more rays.

Supersampling is most important at the edges of objects. If supersampling is used only at the edges, overall quality does not reduce much, but the speed gain is huge. In 4 the white pixels are represent the region wherein we used supersampling.

To detect object switch we maintain a list of the objects that were visible in the previous line and a variable which stores the object of the previous pixel. With these two variables we can determine the object switching both in horizontal and vertical direction. If there was a switch we spawn more primary rays in that pixel to avoid aliasing. The origins of the primary rays are modified with a small random number to take advantage of the multiple samples.

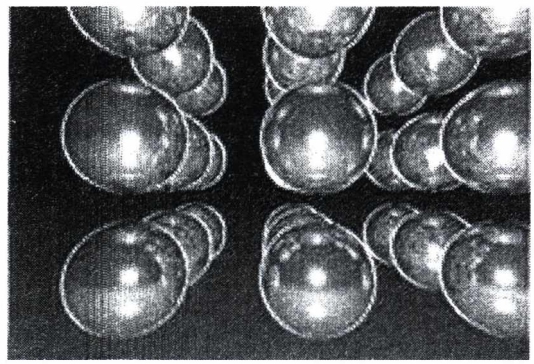


Figure 4: *Region of interest in supersampling*

4. Results and conclusions

The ray-tracer was tested on a system equipped with a Motorola G4 1.3GHz processor with 1 Gbyte ram. The operating system was Mac OSX. The program compilled with the GNU GCC v4.0. To measure performance we used test scenes with 10, 100, 1000 and 50000 spheres. The material of the spheres had diffuse, specular, reflective and refractive components. All of the measurements were with 640×480 picture size with $32bits$ color depth.

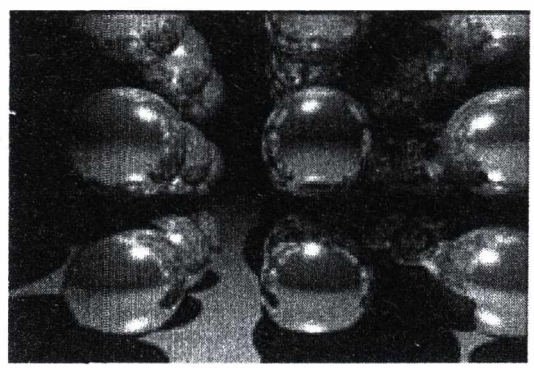


Figure 5: *Example scene*

Our tests showed that the most important optimization technique is the spatial subdivision. Using a KD-tree the rendering process is more than thousand times faster than the naive implementation in large scenes. With using alternative entry points and ray beams we can easily dubble the performance. The supersampling and importance sampling adds another ten percent performance boost. See in table 1.

Number of objects	10	1000	50000
Naive	2.2s	161.5s	-
KD-tree	0.1s	3.5s	6.3s
Alt. entry	0.05s	1.1s	3.3s
All	0.03s	0.8s	2.8s

Table 1: *Rendering times*

5. Future works

To improve the performance of the ray-tracer system we have several ways. The most important part of this work to extend the hit search algorithm with a caching mechanism to speed up the tracing process of the secondary and the shadow rays. Less important but holds out a promise to adapt the acceleration tree building process to the needs of the alternative entry point algorithm. Despite the tree works well with this implementation, the used heuristics is designed for individual rays and not for ray beams.

References

1. U. ASSARSSON and T. Moller. Optimized view frustrum culling algorithms for bounding boxes. In *Journal of Graphics Tools*, pages 9–22, 2000. 3

2. Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000. 2

3. Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, 2005. 3

4. Laszlo Szecsi. *An effective implementation of the k-D tree*, pages 315–326. Charles River Media, Inc., 2003. 2

5. L. Szirmay-Kalos, V. Havran, B. Benedek, and L. Szécsi. On the efficiency of ray-shooting acceleration schemes. In *Proc. Spring Conference on Computer Graphics (SCCG '2002)*, pages 97–106. Comenius University Press, 2002. 2

6. Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004. Available at http://www.mpi-sb.mpg.de/~wald/PhD/. 2

7. Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980. 1

A Real-Time Animation Engine for H-anim Characters

Zs. Ruttkay[1,2], R. Vanca[3]

[1]Information Technology Faculty, PPKE, Budapest, Hungary
[2]Dept. of Computer Science, University of Twente, The Netherlands
[3]Technical University of Budapest, Department of Control Engineering and Information Technology

Abstract

We report on ongoing work to develop a character animation engine which can be used to control the motion of embodied conversational agents (ECAs). The major characteristics of our envisioned engine are: real-time control allowing reactive and interactive behaviour; useable for any model defined according to the H-anim convention; interface for high-level behaviour control. In this paper we first outline the background of the work. Then we explain in detail the principles and implementation details of the low-level, DirectX based animation engine. Finally we sketch further extensions and potential applications.

Categories and Subject Descriptors (according to ACM CCS): 1.3.3 [Computer Graphics]: Animation

1. Introduction

An *Embodied Conversational Agents* (ECA) is a computer graphics model which looks like, more or less, as a person, and is capable to communicate by verbal and nonverbal modalities reminiscent of humans [3]. An ECA may act as an assistant, tutor, salesperson or psychologist, play in virtual theatres, stand for real people in teleconferencing or chat forums, translate written content to sign language for the hearing impaired, etc.

There is a need for a great variety of ECAs. One would not like to see the very same model in such different roles and applications. Besides the natural expectation to see a variety of virtual humans, it has been shown in scientific experiments that the design of an ECA have significant consequences on how people react to them [21]. This applies not only for the embodiment, but also for the subtle details of nonverbal communication of the ECA, such as hand gestures postures used [13]. Moreover, as there is not enough descriptive data on the characteristics of natural gesturing of people, an easy to tune virtual character should be used to evaluate and fine-tune the hand gestures of ECAs.

In spite of this quite natural expectation, the present practice falls short. One comes across the very same few models - the Greta model [10, 20] or the Yt model [2] - in many research groups and demo applications. If it is about commercial applications, then one has two choices:

Use a more or less task-tailored environment, with some different ECAs, with a given set of nonverbal gestures like certain facial expressions, pointing and beat hand gestures.

Use an expensive, general high-end animation tool like Maya or 3D Studio Max, requiring professional animator's skill and possibly some patches to be used together with house-in tools..

In the ECA research community, it has been stated that a general platform would be needed [8], which makes it possible:

to *author* ECAs with different embodiments;

to *animate* them, by re-using elements of an extensible nonverbal communication repertoire consisting of parameterized facial, hand and body gestures .

By the VRML/Web3D working group the H-Anim coding scheme was developed, to represent and animate synthetic characters [9]. While this effort is

highly appreciated by the research community, there has not been break-through of the H-anim format as a de-facto standard. Commercial companies have been using their own, individual formats, without practical support for H-anim exportation. The set of models is the same handful ones which were created a few years ago [2].

There are recent initiatives by some researchers to develop and share reusable modules as building blocks for ECAs, see the Mindmakers platform [16] and the GALA forum [7].

Much work has been done on providing computational models and animation environments for faithful and expressive hand gestures [1, 5, 10, 11]. While the principles of modelling characteristics of human gestures can be adopted, these works each have an implementation with one character to be controlled, in some own format.

Our objectives are motivated by the above outlined shortcomings. Namely, we set out to develop a body-animation engine, which:

- can be used with several EACs coded in some general body format;
- assumes the coding of animation is transparent and widely used format;
- provides fast animation, leaving time for the processing of eventual reasoning on a higher-level behaviour planning;
- assures absolute timing, not relative to the speed of the machine used for rendering the animation;
- provides good interface for eventual control of low-level details of the motion characteristics as well as to high-level behaviour control, such as scripted or intelligent, knowledge-based control.

Initially, we wanted to use the Greta body animation system [10], which does confirm to H-anim protocol. In Greta one may specify animation on a high level, in terms of marked up text, or on a body parameter level for each frame. Greta has its own scheduling and speech-alignment engine which bridges the gap between the two levels. We wanted to have control on an in-between level, that is on the level of gesture sequences, and in an interactive way. So we needed to produce our own engine for this purpose.

We also built on our earlier work on defining hand gestures and using those in the context of knowledge-base behavioural control an scripted control [22]. Particularly, in our earlier work we developed a framework where the ECA can be controlled marking up text to be uttered with gesture tags [18]. The essential shortcoming with the real-time absolute control of the low-level animation provided in Java and VRML urged us to develop an own, in all details controllable animation engine.

In this paper we give an account on ongoing work to achieve these goals. We first introduce the H-anim animation principle of characters, and explain how hand gestures are defined in terms of unit motions. Then we introduce our player in detail, and its implementation in DirectX, also providing performance figures. Then we show how the animation engine was coupled with certain modules of the Greta system, responsible for scheduling nonverbal signals accompanying speech. We also explain another possible high-level scripting interface. Finally, we provide some data on performance. In the closing section, we outline further work to be done and an envisioned application.

2. Real-time animation

2.1 H-anim character animation

An H-anim character consists of a hierarchical structure of *joints* like shoulder, elbow, wrist; and *body segments* like upper arm, lower arm, connected by the joints. The joints have 1,2 or 3 degree of rotational freedom. The position of each joint is given with respect to the parent joint's position, by a translation transformation. The structure is similar, more or less, to the structure of the human skeleton. A most detailed H-anim structure consists of 70 joints, models with less level of detail (LOA, Level Of Articulation) consist of only one joint, or the major joints but no finger joints. Joints have unique names.

The joints are used as abstract elements to be rotated. What is to be seen, are the body segments, driven by the joints. A body segment may have different shape, from separate cylinders (resulting in stick figures) to high-density single polygon meshes, resulting in near-realistic looking body geometry. Each body segment, or vertex of the mesh in case of single-mesh body, is associated with a joint. The transformation prescribed for a joint gets applied to the associated body vertices, and the visible body part gets rotated. Further more, if a joint gets moved, all its children joints, in the hierarchy, get moved too. Hence the actual position of a body segment (e.g. pointing finger tip) is determined by the cumulated effect of the rotations of the ancestor joints. According to the H-anim convention, joint rotations are given in Euler angles.

We are interested in motions of the ECA which have some (communicative or other) function, and thus a more or less strict morphology (unlike, e.g. a free dance movement performed.) That is, we are interested in hand- and body gestures which are performed in a similar form. E.g. to emphasize a piece of information, one uses so-called *beats* [4]: the hand moves down from the height of the shoulder to the height of the navel or lower, by rotating the lower arm. The hand shape may be pointing, or fist, or half bent, or open. When enumerating, similar beats are performed, but the hands show the appropriate number. A gymnastic exercise 'gesture' may involve the coordinated motion of several joints, may be distant in the joint hierarchy, involving e.g. arms and legs.

In our discussion, for simplicity we will use the term gesture for hand gestures. A *hand gesture* is coordinated motion of some joints which are used in more or less the same form, to express some meaning (in case of a chatting ECA), or to exercise some specific muscles (in case of a synthetic trainer) or perform some task (in case of an ECA as a cook or assembly instructor). However, much carry over to motions performed involving lower limbs and other joints. In this paper we concentrate on the morphology and performance of the gestures, about semantics, possible meanings and the origin of influences on the performance, see [14].

A simple gesture is defined by giving the start and end position (rotation) of all the joints involved. For the precise performance of the motion, it is also to be given how the joints rotate in time and space from the start to the end position. Linear interpolation results in robot-like motion. For more realistic motion dynamism, time curves have been suggested based on physiological and physical consideration, with ease-in and ease-out periods [11].

For our discussion, we will refer to a *start and an end configuration* for a simple gesture. We assume that these two, together with some interpolation scheme, define the gesture. Note that, for the time being, we do not deal with, explicitly, the *trajectory* of the motion, it is to be determined by the kind of interpolation used to rotate the joints (e.g. the usual slerp, that is spherical linear interpolation). If some motion path (e.g. a wave-like curve) needs to be followed which would not result from the interpolation, the gesture should be defined as compound, consisting of concatenated basic gestures. E.g. in case of a hand gesture of making a full circle, the start and end position are the same, so the interpolation would produce no motion at all. But one can break the whole circle into quarter circles, which can be well defined by the interpolated rotation of joints. If a path is to be followed which cannot be generated as a sequence of curves of basic gestures, the path should be approximated, which is beyond the scope of this paper.

As of *timing* of the motion, it is often expressed in terms of the timing of the start and/or end positions: e.g. a beat should start together with the utterance of the word. It may be also required that the end of the beat (the time of the end position reached) coincides with the end of the emphasized word or phrase. If moving for music or counting, the start times of repetitive motions should coincide with the beats of the music or beginning of the counting words. So we define, in general, the *morphology* of a basic gesture by a triple $<C_{Start}, C_{End}, Int>$, where the first two items are vectors defining (relative) joint rotations for each joint, and the third item tells what type of interpolation is to be used. For complex gestures, a sequence of positions and interpolations are to be given: $< C_0, C_1, C_2, \ldots C_n, Int_1, Int_2, \ldots, Int_n >$. The *actual duration* of the motion is to be given at animation time, the same unit motion may be performed at different tempo. Here we do not go into details of allowing default durations and prescribing constraints on durations, as characteristics of gestures.

2.2 The animation engine

We use DirectX's animation framework for our engine. We choose DirectX [5] (instead of, e.g. OpenGL) for the following of reasons:

- The .x file format for models used by DirectX is de-facto supported by most of the commercial character modelling tools (3D Studio Max, Maya, Poser).
- DirectX has a rich set of functions and classes for manipulation, detailed animation and rendering of models.
- These default functions can be over-defined, hence making it possible for specific, more sophisticated control.
- It assures absolute timing between key poses (to be discussed later).
- It runs fast, making best use of graphics cards.
- It is compatible with respect to different hw and CPUs, and everything can be done using software processing only, even rendering (using the DirectX reference rasterizer, hence a special graphics card is not a must).

The animation engine's task is to produce for each frame a set of joint rotations. It's input is a sequence of

C_i key positions corresponding to sequences of key positions in gestures (stored in an xml file in our so-called KFS format), with corresponding interpolation and timing information. The key positions are stored using a set of H-anim joint rotations represented in Euler angles. The engine loads the data, and builds an animation controller out of it. Our animation algorithm takes all the H-anim joints, and registers an SRT (Scale, Rotate, Translate) key for them. If a model doesn't have some of the joints, they simply wont be animated by the controller, because it does not find them, so level of articulation (LOA) is handled without problems. The animation controller has built in interpolation routines (of an interpolator class, which uses linear interpolation by default), which produces the interim positions, one for each frame, between the key positions. The animation is synced to a real time clock. That is, the in-between frame frequency is dynamically adjusted according to the resources available till the next prescribed key frame. This mechanisms is the basis for absolute time control. So as far as the key frames can be rendered on time, the interpolated frames will not cause any performance problem known with other systems (time lag, not smooth animation). The output of the animation engine is a new state of the model for the frame (a set of joint rotations).

As the animation facilities of DirectX are not well-known and documented, we explain the details. Animation is done using DirectX's built in animation controller and FrameHierarchy structures. We defined our own CAllocateHierarchy class to be able to store other data than the default transformations (we want to store cumulated transformations too).We use this class for the frame hierarchy building and destroying. At the mesh loading stage we load the hierarchy from the file, and the animation if there is one. Later we calculate the combined transformations for every joint, and we set up the mesh for software skinning. In the SetCamera() function we advance the animation using the AdvanceAnim() function of the animation controller (to tell the controller that we want to step to the next frame), and we update the frame matrices according to the new keyframe. At the render stage we either call the DrawSkeleton(), or the DrawMeshContainer() function. The latter draws every frame's mesh container, thus drawing the skinned mesh, the earlier one draws spheres in every jointlocation according to the actual joint transformation matrices. This is all done in the CRenderView class, in the LoadMesh(), SetCamera(), and OnRender() functions.

Finally, our rendering engine renders the new state to the screen. The rendering engine is based on DirectX 9. Below are the main steps:

Initialization steps (to be performed once):

DirectX Object creation.
DirectX Device creation.
Background creation (vertices and texture loading).
Effect loading and compiling.
Mesh loading and corresponding processing (normals computing, bounding sphere computing etc.).
Frame hierarchy creation, and Animation loading if one exists in the .x file.
Skeleton Creation from mesh data.
Matrix initialization (view, projection, world).

Rendering steps (performed in loop):

Making adjustments based on external viewing commands like rotating , zooming.
Setting the camera properties and animation update.
Drawing the background, and skeleton or mesh of the character based on user choice.

2.3 Performance details

Our player uses Direct3D 9 for rendering, its shaders and the built in effect framework. Because of compatibility issues, we have software vertex processing (can be switched to hardware at the DdirectX device creation function), and software skinning. (Hardware skinning is not implemented yet). We provide two effect files for the player, a Shader Model 2.0 compliant one, and a Shader Model 1.1 compliant one. Using the SM 1.1 version, the player is running on DirectX 8 compliant hardware, but one needs DirectX 9C installed for the program to run correctly. Our engine makes use of hardware acceleration when possible, mostly at the rendering stage, but we can also make use of hardware skinning in the future if its necessary for additional performance improvements.

Because of all the hardware acceleration that occurs, the engine is capable of around 500 Fps using a 4400 vertex model (see Fig. 2), and around 400 Fps using a 11400 vertex model on a Pentium4 2.4Ghz, 512MB Ram, and an Ati Radeon 9500Pro graphics card. This way we have a lot of resources left for e.g. own real time interpolation.

2.4 Coupling the player with high-level control of Greta and GESTYLE

The Greta system is a complete character animation environment, used in several research environments [20, 10]. It got the name from the unique female character which is animated. On the lowest level, both the face and body motion of Greta is expressed in terms of MPEG-4 parameters, and the original Greta player gets a sequence of such parameters, a set for each frame. Our own animation engine has specific characteristics with respect to the Greta engine, as reactive real-time control with interface on an intermediate level, and interface for any H-anim compliant model in .x format. To be able to use the high-level capabilities of Greta, we coupled our player with the user interface and motion scheduling modules of Greta. In this way, from a piece of text, marked up with tags indicating gestures to be performed, sequences of timed gestures are generated. The gestures occurring is mark-up tags should be from an earlier defined library of gestures. Eventual conflicts are resolved by the scheduler. The timing of the gestures is gained from the TTS engine Festival, which is an integrated module of Greta. The output is a sequence corresponding to timed unit motions stored in KFS format as discussed above, which is further processed by our H-anim Player. The general architecture is given in Fig. 1.

When running our animation engine in combination with Greta, one can drive any H-anim character to converse with the user, profiting form the communicative hand gestures available in Greta and the its markup control language.

Originally, the sampling of joint parameters for each frame was done by a Greta module, and the Greta player was driven by sets of parameters for each frame. To modify timing or other performance characteristics of the gestures, the marked up text had to be re-processed, and new parameter sequence generated accordingly for the player. Our player, getting input on a higher level and being able to process them at a high speed, makes real-time modifications possible, which involve changing the timing parameters one by one.

The interface for our player allows, in principle, to couple it with other high-level schedulers which can provide output. Particularly, in a next round we will couple the player with our GESTYLE markup language, which does not have an own player neither model to drive, but from marked up text input generates a timed sequence of gestures to be displayed by some player. Motion characteristics of gestures of the ECA as a permanent (e.g. trembling motion due to age) or temporal feature (in sad state, thus all gestures slowed down and performed with less energy) may be prescribed, and GESTYLE provides output accordingly, expressed in further parameters of gestures as noise, intensity, time dynamism etc. We plan to exploit the open animation functions and fast performance of the DirectX player to deal with these subtle details.

3 Discussion

3.1 Summary of our work

We have developed a character animation engine based on DirectX, which takes some timed key positions of joints according to H-anim standard, and interpolation scheme between them, and generates the animation in real-time, accordingly. The player is fast, can produce 500 fps. Thus there is time left to do further processing within response time. This opens possibilities for reactive animation and animation with subtle motion characteristics according to (changing) emotional state of the character.

3.2 Further issues

We are eager to use our engine with a variety of H-anim compliant bodies. Recently, and VizX3d [23] have improved their character modelling tool with respect to .x and H-anim exportation. Thus we hope to be able to use different characters for different applications and users, with the same platform. This asks for designing gesture libraries and principles of adaptation of them for H-anim characters with different geometry. Related to this, we wish to include *inverse kinematics* [12], which would be useful for defining key positions in terms of reference points on and around the body (e.g. touch with left hand right knee).

The major motivation for our engine was reactive animation. Ultimately, we would like to have an ECA which can generate motion on the fly, as reaction to perceived state of the user and other environmental signals. Particularly, we are interested in adjusting tempo and intensity of animation according to music, counting or the motion of a real person. We would like to develop a synthetic trainer, who can monitor the performance of the trained used, and fine-tune and even re-schedule his own motions according to the physical state and performance of the user.

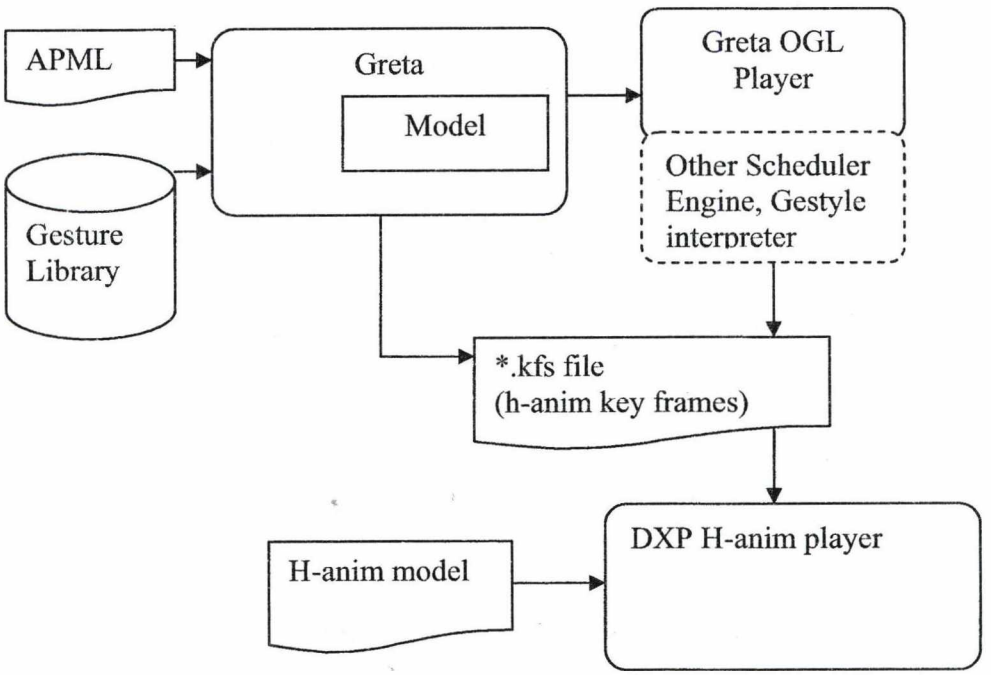


Figure 1: The architecture of our DXP H-anim player, with interfaces to the Greta or GESTYLE modules.

Figure 2: An H-anim character animated by the DVX engine. The model is the demo one provided with DirectX. SDK.

Acknowledgement

We thank Catherine Pelachaud for providing us her Greta system, and for discussions on its facilities. We are grateful for the help of Maurizio Mancini concerning format and programming details of Greta. The work was done while the first author was benefiting from the Szent-Györgyi Fellowship of the Hungarian Ministry of Education.

Bibliography

1. N. Badler, R. Bindiganavale, J. Allbeck, W. Schuler, L. Zhao, and M. Palmer. Parameterized action representation for virtual human agents. In: . Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.): Embodied Conversational Agents, MIT Press, Cambridge, MA. 2000.
2. Beitler, M. H-Anim 1.1 Compliant VRML97 Humanoid Models, http://www.cis.upenn.edu/~beitler/hanim/
3. J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.): Embodied Conversational Agents, MIT Press, Cambridge, MA. 2000.
4. J. Cassell, H. Vilhj´almsson, and T. Bickmore. BEAT: the behavior expression animation toolkit. in computer graphics proceedings, Annual Conference Series. ACM SIGGRAPH, 2001.
5. D. M. Chi, M. Costa, L. Zhao, N. I. Badler: The EMOTE model for effort and shape, Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, 2000. pp 173–182.
6. DirectX Graphics Documentation: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/dxgraphics.asp
7. GALA; http://hmi.ewi.utwente.nl/gala/
8. J. Gratch, J. Rickel, E. Andre, N. Badler, J. Cassell and E. Petajan. "Creating interactive virtual humans: Some assembly required." IEEE Intelligent Systems, 2002
9. H-anim: http://www.h-anim.org/
10. B. Hartmann, M. Mancini, C. Pelachaud: Formational parameters and adaptive prototype instantiation for MPEG-4 compliant gesture synthesis, Proc. of Computer Animation, Geneva, June 2002.
11. S. Kopp, I. Wachsmuth.: Planning and motion control: in lifelike gesture: A refined approach. In Proc. of Computer Animation, 2000. pp 92-97.
12. Y. Liu, N. Badler: Real-time reach planning for animated characters using hardware acceleration." Computer Animation and Social Agents, IEEE Computer Society, New Brunswick, NJ, May 2003, pp. 86-93.
13. J.-C., Martin, S. Buisine, S., Abrilian: 2D gestural and multimodal behavior of users interacting with embodied agents. Proc. of the workshop "Embodied Conversational Agents: Balanced Perception and Action", of AAMAS04, New York, USA, 2004.pp. 34-41.
14. D. McNeill: Hand and Mind: What Gestures Reveal about Thought. University of Chicago, 1992.
15. Microsoft DirectX: http://www.microsoft.com/windows/directx/default.aspx
16. Mindmakers: http://www.mindmakers.org/
17. MPEG-4: Overview of the MPEG-4 Standard. ISO/IEC JTC1/SC29/WG11 N4668, March 2002, http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm
18. H. Noot, Zs. Ruttkay: Style in gesture, In: A. Camurri, G. Volpe (Eds.), Gesture-Based Communication in Human-Computer Interaction, LNCS 2915, Springer-Verlag, 2004.
19. DirectX Vertex shader: http://msdn.microsoft.com/msdnmag/issues/01/06/Matrix/
20. S. Pasquariello, and C. Pelachaud, "Greta: a simple facial animation engine," Proc. 6th Online World Conf. Soft Computing in Industrial Applications, Springer-Verlag, 2001.
21. Zs. Ruttkay, C. Pelachaud (Eds): From Brows to Trust – Evaluationg ECAs, Kluwer, 2004.
22. Zs. Ruttkay, Z. Huang, A. Eliëns: The Conductor: Gestures for embodied agents with logic programming, Proc. of the 2nd Hungarian Computer Graphics Conference, Budapest, 2003. pp. 9-16.
23. VizX3d: www.vizx3d.com

Computer framework for organizing 3-dimensional graphical environment in image-guided planning and navigation

F. Pongrácz and Z. Bárdosi

Computer Automation and Research Institute, Budapest, Hungary

Abstract

Our goal is to develop a general-purpose interactive software module for run-time planning and execution of several (surgical) planning and navigational tasks. The software module has the capability to interactively create and modify the internal, coded representation of a virtual graphical environment which visualized and compared to real images of anatomical objects. The suitable organization of this 3-dimensional virtual environment is based on procedures known from data manipulation in tree structures of nodal topology. Using this module we are able to create and simulate complex models of several navigational and measurement tasks occuring in clinical field (movement analysis, registration of 3D space of motion trackers, frameless stereotactic navigation, sensor-based calibrations of surgical tools, trajectory tracking, etc.).

Keywords: 3D virtual environment, nodal topology, image-guided planning, motion tracking.

1. Introduction

We developed a framework to organize the 3-dimensional graphical environment in those applications which integrate large number of graphical elements, cameras and motion trackers. This environment is very common in the medical field of image-guided surgery planning and navigation which is getting less manageable with the increasing complexity and versatility of surgical tasks. Different 3D elements, supplemented by 6-degree of freedom floating sensors of an optical motion tracker and a variety of 3D cameras were organized into a hierarchial tree structure. The 3D cameras are referred as viewers which represent different diagnostic viewing and rendering of a volumetric model. The volumetric models are computed from CT (computer tomographic) or MR (magnetic resonance) image sequences [6]. The proposed hierarchial tree structure [for description of container class see 5] and its specialized functionality collectively shape up the virtual environment which is visualized and compared to the real images of body parts in patients.

Similar approach to design software architecture for surgical planning has been already described in [3] with some differences to our application. The system developed in [3] has the same nodal topology to build up the graphical environment but it does not integrate important elements into the topology (registration objects, sensor-based tool calibrations). On the other hand they are able to add video cameras very efficiently (for instance with special distorter functionalities). An open source toolkit is under development which uses a top-down approach from Problem Specification to Software Design but without clear strategy for a generic 3D topological organization of the problems [4]. The package is based on VTK and ITK open source libraries and called "The Image-Guided Software Toolkit" or IGSTK. Modularity is common in software for surgery planning where several graphical elements should be manipulated separately[8]. Others realised that standardized frameworks are needed to bring the actual computer-aided pre-operative planning scheme into the operating room [1,2]. We also accept this argument by creating one single software environment for both the preoperative planning process and the intra-operative intervention.

2. Navigation objects

The navigation tree is a hierarchical tree structure containing all the navigation objects existing in the 3D space of navigation. These objects can be of many types but their common properties are that they all have their local coordinate system. This coordinate system is defined by its position and orientation relative to parent's space giving to all objects of navigation space the 6 degrees of freedom. This nodal hierarchy gives the ability to efficiently determine the linear transformations between two navigation objects and this structure can be altered interactively. Several types of objects can be added to the navigation tree: volume and mesh objects can represent real objects (i.e. volumetric objects can represent the CT/MR scans of the patient while 3D meshes can create virtual versions of implants), geometrical features like points, lines, paths, coordinate systems, 3D models of tools can help to visualize the scene better. The transformation rules are defined between the local space and parent's space according to the floating axis Euler convention (**Figure 1**).

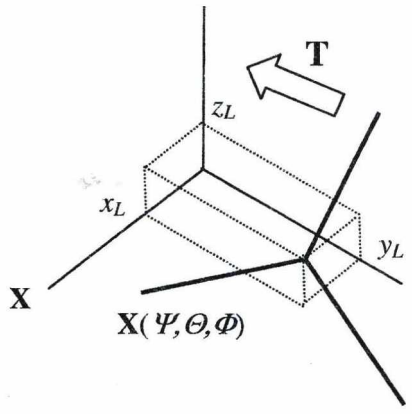


Figure 1. *3D local space of navigation object within its parent's space.*

Each navigation object stores the 3 translational values (x_L, y_L, z_L) and 3 Euler angles for orientation (Ψ= Azimuth, Θ = Elevation, Φ = Roll). Transformation (**T**) is calculated from local to parent's space with position and orientation values defined relative to the parent's space:

$$\mathbf{T}=\begin{bmatrix} \cos\Psi\cos\Theta & \cos\Theta\sin\Psi & -\sin\Theta & 0 \\ -\cos\Phi\sin\Psi+\sin\Phi\sin\Theta\cos\Psi & \cos\Phi\cos\Psi+\sin\Phi\sin\Theta\sin\Psi & \sin\Phi\cos\Theta & 0 \\ \sin\Psi\sin\Phi+\cos\Phi\sin\Theta\cos\Psi & -\sin\Phi\cos\Psi+\cos\Phi\sin\Theta\sin\Psi & \cos\Phi\cos\Theta & 0 \\ x_L & y_L & z_L & 1 \end{bmatrix} \quad Eq.1$$

The floating X axis orientation of the local space is calculated by sequential rotations with (Ψ,Θ,Φ). Each navigation object can be locally positioned and reoriented and in the same time appears as an element of nested transformations in the tree. An update of conversion between matrix (**T**) and (x_L, y_L, z_L, Ψ,Θ,Φ) values validates the user interaction. The conversion between the 3x3 rotation submatrix of **T** and the Euler angles is given by:

$$ce=\sqrt{(T_{11})^2+(T_{12})^2}$$

$$ce>0:\ \Psi=\arctan(T_{12}/T_{11})\quad \Theta=\arctan(-T_{13}/ce)\quad \Phi=\arctan(T_{23}/T_{33}) \qquad Eq.2$$

$$ce=0(T_{13}<0):\ \Psi=0\quad \Theta=\pi/2\quad \Phi=\arctan(T_{21}/T_{22})$$

$$ce=0(T_{13}\geq 0):\ \Psi=0\quad \Theta=-\pi/2\quad \Phi=\arctan(-T_{21}/T_{22})$$

The next navigation objects are derived from the ancestor object:

- **Volume navigation object** represents the CT or MR image scans and the related volumetric models. The resampled imaging data can be visualized with Slice Viewer objects or, after rendering, with Base Viewer objects. Unique feature of the Volume object is that it can be registered from a moveable reference space (sensor space attached to the volume) or from the global space of the motion tracker.
- **Slice Viewer object** gives a chance to view the image slices of the Volume object. Slice Viewer can be configured to look at the Volume object from predefined view directions (axial, lateral, frontal). This viewer may contain visible graphical elements with feature points for selection and manipulation. The Slice Viewer objects are located in the tree topology under their source volume (i.e. Volume object created from CT or MR data).
- **Base Viewer object** represents a generic camera which can be located with any position/orientation in the navigation environment. This viewer shows all navigation objects wich have the visibility flag on. It also displays the Volume object if the rendered surface data are available in 3D.
- **Coordinate System object** creates a coordinate system which can be translated and rotated about any axis by means of its projection in any Viewers.
- **Point, Line navigation objects** represent simple graphical elements which can be used for 3D measurements or for marking targets and directions in surgical planning.
- **Surface Model objects** help to add predefined surface mesh, as navigated object to the scene. This model can be the reconstruction of different implants or tools in standard surface formats (VTK, binary STL).
- **Drill navigation object** locates a drill in the environment which has a motion sensor attached and calibrated to the tip location and orientation. The configuration dialog controls the calculations which include SVD-based least-square estimations for tip offset and axis direction.
- **Mover navigation object** represents the sensor of an optical motion tracker. Movers can be added to any position in the tree but their global location and orientation remain always at the detected values (their local coordinates and angles are compensated for movement). The local objects within the space of a Mover are the navigated objects with adjustable translation and rotation parameters relative to the Mover.
- **Registration object** is added automatically to the tree after the succesful registration to Volume object has been found. The Registration object implements 4 points rigid body algorithm[7] and stores the marker positions and the registration matrix.

3. Navigation tree and transformation rules

Figure 2 summarizes the basic topology of the hierarchial tree structure for graphical planning without adding any elements of motion tracking.

3.1 Transformation between local spaces

The tree structure incorporates nested 3D coordinate spaces where the calculation of transformation matrices is based on relative location and orientation of two, topologically close parent-child spaces. Consecutive matrix multiplications determine the transformation from an arbitrary space to any other, topologically distant space. Because of branching in nodal structure, the multiple transformation is done through the global or World space or through the closest branching node if it is different from the World node:

$$\mathbf{T}_{i,j} = \mathbf{T}_{i,i-1} \times \mathbf{T}_{i-1,i-2} \times \ldots \times \mathbf{T}_{i-k,W} \times \mathbf{T}_{W,j-l} \times \mathbf{T}_{j-2,j-1} \times \mathbf{T}_{j-1,j} \quad Eq.3$$

The matrix elements from child to parent transformation ($\mathbf{T}_{i,i-1}$) are calculated according to *Eq.1* and indices *k* and *l* represent the number of intermediate nodes to the World node (denoted by *W*). The important functionality of tree topology is the easy handling of element relocation. However, in our case, two possible consequences of relocation should be considered: (A) the relocated tree element preserves the local position/orientation to its parent or (B) it keeps the global location/orientation relative to the World space. The second case is implemented by the Global Lock feature which is important in motion analysis. Motion path of navigated elements should be viewed relative to different reference spaces but keeping the global position and orientation unchanged. The element relocation after setting Global Lock to ON is driven by next equation:

$$\mathbf{T}_{i,j} = \left(\mathbf{T}_{W,i}\right)^{-1} \times \mathbf{T}_{W,j} \quad Eq.4$$

where $\mathbf{T}_{i,j}$ represents the new transformation of the relocated *i*th local space to its new parent, the *j*th space in the tree. *Eq.2* is used to get the local orientation parameters, the local translation coordinates are directly related to the matrix elements.

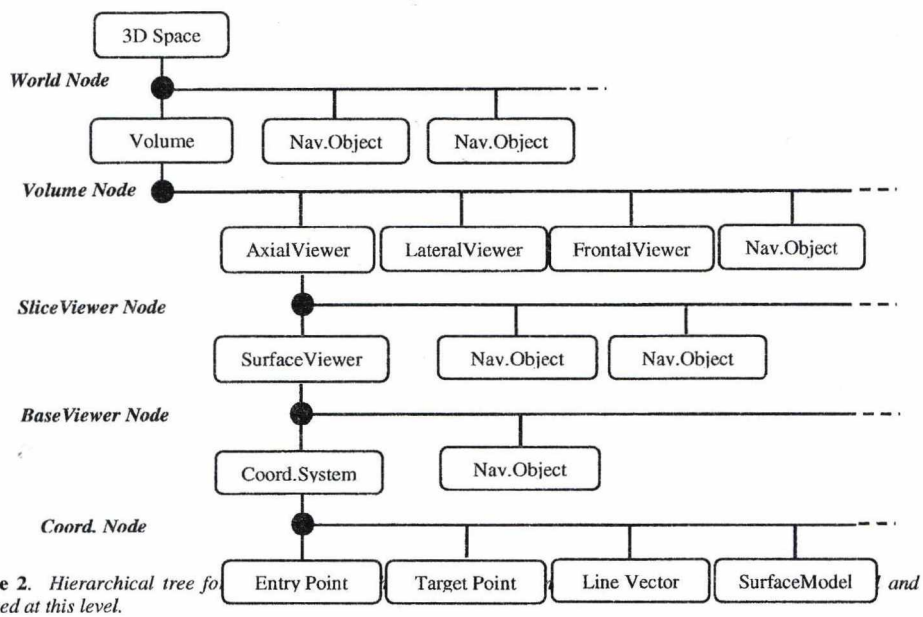


Figure 2. *Hierarchical tree fo and no registration involved at this level.*

3.2 Adding motion sensors (Movers) to the tree

Mover objects are the abstract representations of sensors of the optical motion tracker (Polaris, Northern Digital in this study). In the configuration dialog of Movers the user can select the actual sensor providing input data. Movers can be placed anywhere in the tree (**Figure 3**). However, all related sensors will be in Global Lock mode which needs a continuous update of their local position and orientation values as a function of their global parameters (see *Eq.4* and *Eq.2*).

Several types of Navigation objects can be attached to Movers: Volume, different graphical elements, local spaces, surface models etc. The attached sensor's space usually doesn't fit with the coordinate space of the Navigated object. Therefore, all cases need special registration (for Volume) or calibration procedures (in case of pointer, tools, moveable local spaces). Volume registration inserts a Registration object between the Volume Sensor (as reference Mover) and the registered Volume with optionally attached marker positions. The marker positions are transformed from the reference Mover's space with the registration matrix.

With the proposed approach one can easily switch between reference Movers that is important in clinical motion tracking and tool navigation. Reference Mover is a sensor which is attached to a moving object that permits 3D stereotactic analysis relative to this moving object. In our case (**Figure 3**) the Volume Sensor and the Local Sensor can be considered as reference Movers which are attached to different moveable body parts. The Global Lock feature permits the reference Mover switching by keeping the Global location and orientation unchanged for Navigation object.

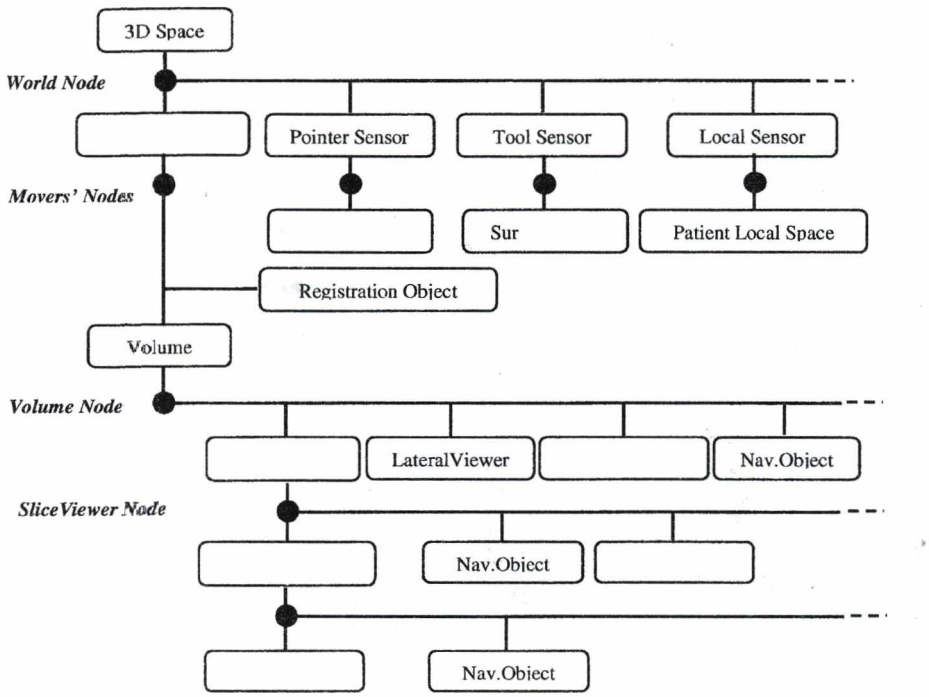


Figure 3. *Hierarchical tree for building up graphical environment in surgical navigation. Reference sensor and pointer are added for Volume registration. Tool sensors are included for surgical tool tracking and implant navigation. Local sensor with attached anatomical coordinate space can be used to follow body part's movement.*

4. Software implementation

The proposed navigation framework was implemented in a software performing several medical image processing tasks (viewing CT/MR data, image fusion, manipulation of polygonal surface mesh of vtk or binary stl formats and communication with optical motion tracker) (**Figure 4**). Following a panel structure the Navigation panel can utilize the output of the calculations performed in other modules. For instance the Image Fusion panel can provide for transformation matrix between two volumes of the same or different modalities. This way a second Volume object can be inserted into the Navigation tree without duplicating the effort for registration of the 3D space of a motion sensor.

During implementation several functionalities were added to the classes representing the Navigation objects. These objects have generic and special parameters which can be configured individually. The generic parameters are the local position/orientation values and the Global Lock and visibility flags. In some cases the Navigation object may have other embedded objects like the point markers in the Registration object. Similar embedded objects are the so called Features like the endpoints of a Line object which can be reached by hittests in Viewers. After adding a Navigation object to the tree its configuration and appearence (selection of a window slot for a Viewer, adding registration to Volume) can be initiated by context menus. Calibration procedures for drill and moveable local (anatomical) spaces can be initiated from their configuration dialog.

Important elements in the software are the Observer slots which can be defined for any Navigation object. These Observer slots numerically display certain 3D relationships between the selected object and the Global space or any other object in the tree. For now these relationships can be the relative coordinates, relative Euler angles and 3D distances.

5. Conclusion

A general purpose software framework has been developed which is usable to solve 3D graphical planning for surgical planning and navigation. The presented methodology adopts the known rules of nodal topology in hierarchical tree structures and converts those into the practice of software design in medical field.

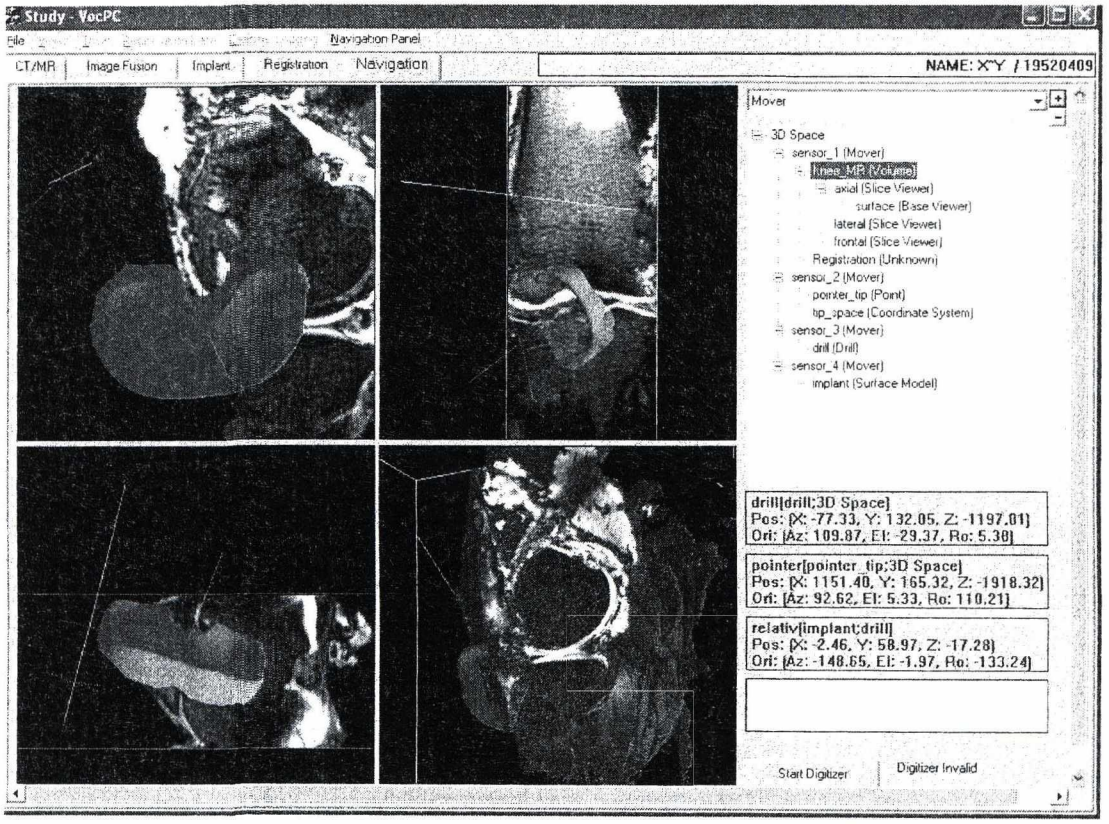


Figure 4. *Typical window of the program designed for solving surgical planning and navigation tasks. The original MR scan of cadaver knee (upper left Viewer) was reformatted and displayed in different Slice Viewers. The rendered isosurface is shown in the bottom right Base Viewer. The hierarchial topology of graphical elements was set up in a navigation tree (right, upper part of the window). A vtk surface representing an implant was added to the topology. The Observer slots are shown on the lower right part.*

Acknowledgement

This project was supported by the hungarian research grant NKFP/1B/0009/2002. The authors thank to G. Renner, G. Krakovits and Gy. Szántó for helpful discussions during this work.

References

1. E. Keeve, T. Jansen, Z. Kroll, L. Ritter, B. Rymon-Lipinski, R. Sader, HF. Zeilhofer and P. Zerfass. "JULIUS - An Extendable Software Framework for Surgical Planning and Image-Guided Navigation". *in Proceedings of MICCAI'2001*, Utrecht, The Netherlands, October 14-17, 2001.

2. P. Golland, R. Kikinis, C. Umans, M. Halle, M.E. Shenton, J.A. Richolt. "AnatomyBrowser: A Framework for Integration of Medical Information." *In Proceedings of MICCAI'98*, Cambridge, MA, 1998, pp.720-731.

3. W.Freysinger, M.J. Truppe, A.R. Gunkel, W.F. Thumfart. "A Full 3D-Navigation System in a Suitcase". *Computer Aided Surgery*, Vol.6, 2001, pp.85-93.

4. K. Cleary, L. Ibanez, S. Ranjan, B. Blake. "IGSTK: A Software Toolkit for Image-Guided Surgery Applications". *In Proceedings of CARS 2004*, Chicago, International Congress Series 1268, 2004, pp. 473-479.

5. K. Peeters. "tree.hh 2.0, an STL-like container class for n-ary trees, templated over the data stored at the nodes". http://www.damtp.cam.ac.uk/user/kp229/tree/, Open Source Under GNU General Public License.

6. F. Pongrácz, G.Renner, "Volume and surface models from CT/MR images." *In: Képfeldolgozók és Alakfelismerők IV. Konferenciája* (Eds. Gácsi, Z., Barkóczy, P., Sárközi, G.), Miskolc-Tapolca, 2004 jan.28-30. pp.244-251.

7. K.S.Arun, T.S.Huang, S.D.Blostein. "Least-Squares Fitting of Two 3-D Point Sets". *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.9,1987, pp.698-700.

8. K.Ollé, B. Erdőhelyi, A. Kuba, E. Varga and Cs. Halmai. "MedEdit: Mútéti tervezést segítő orvosi képfeldolgozó rendszer". *In: Képfeldolgozók és Alakfelismerők IV. Konferenciája* (Eds. Gácsi, Z., Barkóczy, P., Sárközi, G.), Miskolc-Tapolca, 2004 jan.28-30. pp.221-226.

ISBN 963-421-593-9