REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT DATE	3. DATES COVERED *(From - To)*
01/22/2001	Final Report	1/1/97 -- 12/31/99

4. TITLE AND SUBTITLE
Volumetric Representation and Manipulation of Geometric Models

5a. CONTRACT NUMBER

5b. GRANT NUMBER
N00014-97-0223

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Turk, Greg

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Georgia Tech Research Corporation
Georgia Institute of Technology
Atlanta GA 30332-0420

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research, Ballston Centre Tower One
800 North Quincy Street
Arlington VA 22217-5660

10. SPONSOR/MONITOR'S ACRONYM(S)
ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This is the final report of the work that was funded by the ONR contract number N00014-97-1-0223. The goal of this research project was to investigate new methods of representing and manipulating three-dimensional geometric models using volumetric techniques. Three sub-areas were particular targets for these investigations: 1) explore ways of extending the kinds of models that can be represented volumetrically, 2)create nultiresolution models using volume techniques, and 3) perform shape transformation using a volumetric framework.

15. SUBJECT TERMS
3D Geometric Models

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Greg Turk
U	U	U	UU	73	19b. TELEPONE NUMBER *(Include area code)* 404/894-7508

Interior/Exterior Classification of Polygonal Models

F.S. Nooruddin and Greg Turk

GVU Center, College of Computing, Georgia Institute of Technology

Abstract

We present an algorithm for automatically classifying the interior and exterior parts of a polygonal model. The need for visualizing the interiors of objects frequently arises in medical visualization and CAD modeling. The goal of such visualizations is to display the model in a way that the human observer can easily understand the relationship between the different parts of the surface. While there exist excellent methods for visualizing surfaces that are inside one another (nested surfaces), the determination of which parts of the surface are interior is currently done manually.

Our automatic method for interior classification takes a sampling approach using a collection of direction vectors. Polygons are said to be interior to the model if they are not visible in any of these viewing directions from a point outside the model. Once we have identified polygons as being inside or outside the model, these can be textured or have different opacities applied to them so that the whole model can be rendered in a more comprehensible manner. An additional consideration for some models is that they may have holes or tunnels running through them that are connected to the exterior surface. Although an external observer can see into these holes, it is often desirable to mark the walls of such tunnels as being part of the interior of a model. In order to allow this modified classification of the interior, we use morphological operators to close all the holes of the model. An input model is used together with its closed version to provide a better classification of the portions of the original model.

Keywords: Visibility, Surface Classification, Rendering, Interior Surfaces

1 Introduction

In this paper we present a method for determining the interior and the exterior portions of a given polygonal model. Our motivation is that for many visualization applications it is desirable to display surfaces in such a way that a human observer can clearly see the relationships between different parts of the object. While excellent techniques exist for displaying nested surfaces [4, 5, 6, 9], the determination of which surfaces are exterior to the model and which are interior is typically not an automated process. Usually this classification is done either by hand or by connected component analysis. For many models in application areas such as medicine or industrial design, connected component analysis is not helpful because different parts of the model may be connected to each other by holes, tubes or thin structures. Our method overcomes this limitation of connected component analysis.

For some applications, the nature of the data can give clues as to whether a portion of a surface should be considered interior. For example, with medical data from CT or MRI, the volume densities may be used to help classify different tissue types. Unfortunately, such approaches fail when a user wishes to visualize the interior structure of an organ that is relatively uniform in tissue type such as the chambers of the heart, the alveoli of the lungs or the folds and interior cavities of the brain. Likewise, some CAD data may be tagged with different part identifiers or surface properties. In some cases, however, these tags have been lost or the model that is being visualized is from actual captured data for parts inspection such as CT. In these cases, once again we require an automated method of identifying interior parts without the aid of pre-specified tags.

In this paper we present a new technique that classifies each polygon of a given model as being either interior or exterior to the surface. We can then use different rendering styles to show off these different portions of a surface. Our method uses a collection of viewing directions in order to classify each point as being a part of the exterior or the interior of the model. When processing models with holes, we first use volumetric morphology to obtain a closed version of the input model. The closed model is used in conjunction with the original model to provide a better classification for the surfaces of the original model.

The remainder of the paper is organized as follows. In Section 2 we review existing methods for viewing interiors of models. In Section 3 we present our definition of visibility. In Section 4 we describe how to generate a "deep" depth buffer for each viewpoint. In Section 5 we describe how we use the depth buffers to classify a polygon as being exterior or interior to the model. In Section 6 we describe the use of morphology to close holes in models to get improved results. In Section 7 we present the results of our method. Section 8 provides a summary and describes possible future work.

2 Previous Work

There have been several techniques in graphics and visualization that are tailored towards viewing the interior of surfaces, including wireframe drawing, clipping and volume rendering. We will briefly review each of these below.

Wireframe drawing of polygonal models has been used in computer graphics at least since the 1950's. Until fast polygon rasterization hardware came along in the 1980's, wireframe drawings on raster or calligraphic displays was the viewing method of choice for rapid examination of 3D ob-

e-mail: {fsn,turk}@cc.gatech.edu

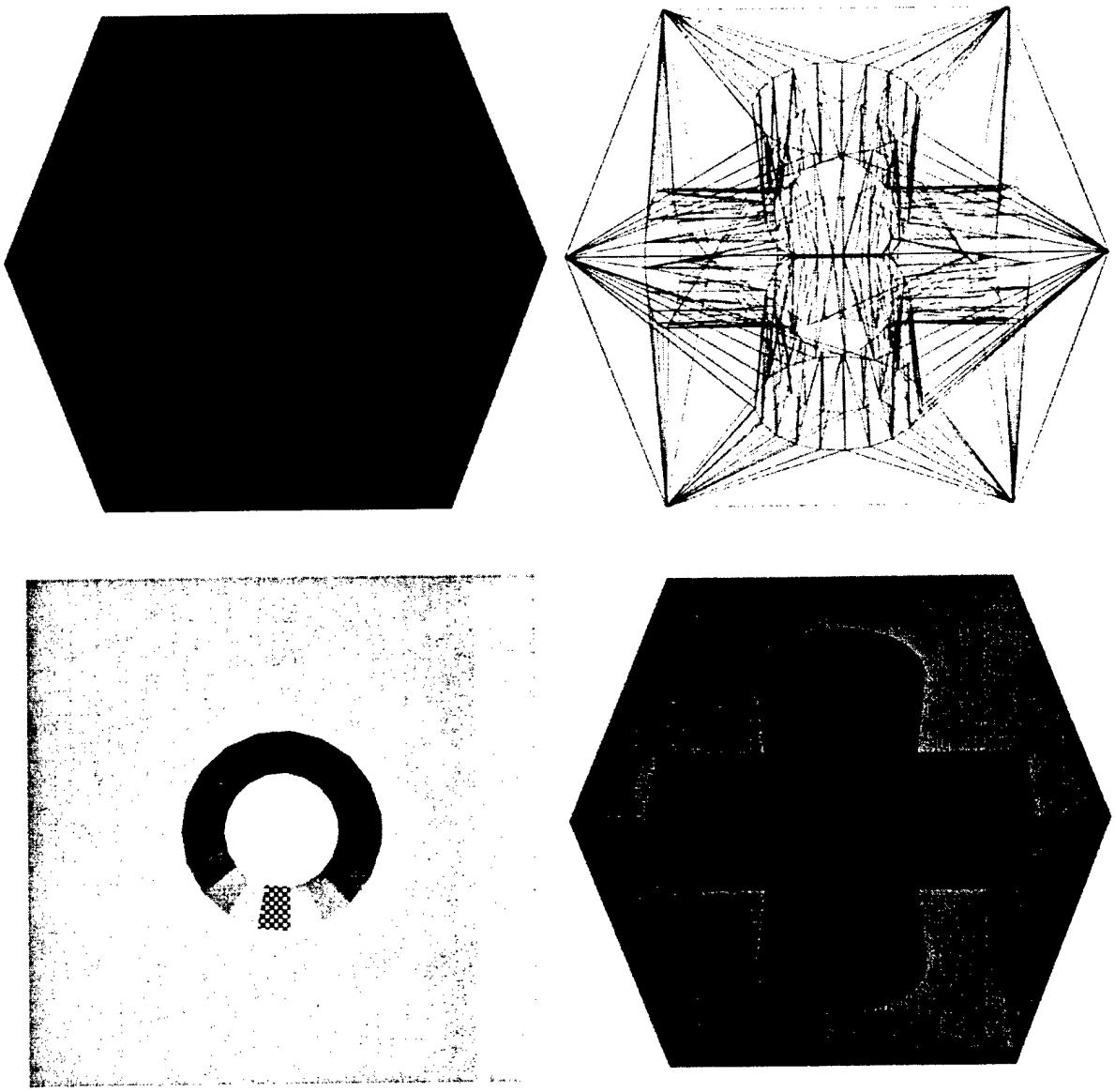


Figure 1: Top left: Cube with three holes, rendered as an opaque surface. Top right: wireframe rendering of the same cube. Bottom left: Another view of opaque cube with an internal polygon highlighted. Bottom right: Cube with a translucent external surface and an opaque interior surface.

jects. At present, wireframe drawings of models are sometimes used to show some of the interior detail of simple models. Unfortunately there is considerable ambiguity in viewing such a wireframe model due to lack of occlusion between lines at different depths. Moreover, polygon size greatly affects the understanding of such images. An inner surface that is tiled with just a few large polygons may be difficult to see because the surface is represented only by a small number of lines. Figure 1 (top right) shows a wireframe drawing of a cube with holes.

Clipping planes and solids are often used in visualization to allow an observer to see into a portion of a model. Rossignac et al. demonstrated a variety of solid clipping techniques for CAD model inspection, including methods for cross-hatching those portions of a model that have been sliced [10]. Cutting planes, half-planes and other shapes have been used in volume rendering to see into the interiors of objects. Medical data has been the application area where this has been put to use the most extensively [12]. Unfortunately, creating clipping planes or clipping volumes that are tailored for a given object is a user-intensive task.

Volume rendering techniques allow some portions of a model to be made translucent or entirely transparent, allowing an observer to see interior parts of a model [2, 6]. Segmentation of a volume model, whether it is done automatically or by hand, allows regions to be tagged as translucent or opaque for better comprehension of interior surfaces. Automatic segmentation is very useful if the model to be visualized is composed of different materials with differing voxel densities, as is common with medical data. In many cases,

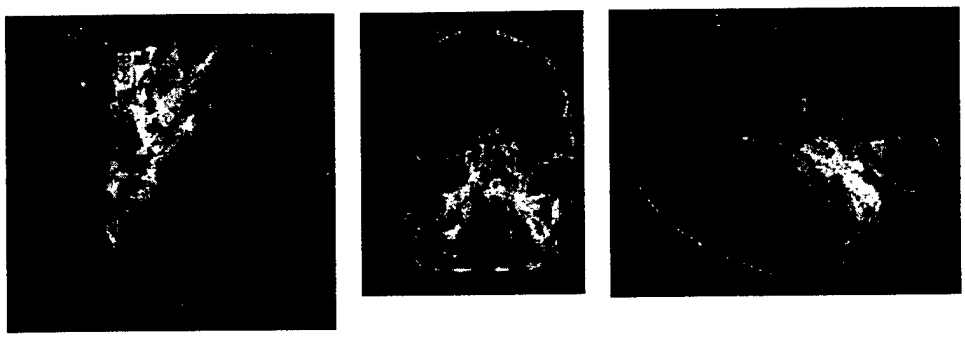


Figure 2: Gray-scale representation of three of the depth buffers created for a skull dataset. Brighter pixels are locations where more depth samples are present in the buffer.

however, such as the turbine blade or the skull of Figure 6, there is no way to separate out the interior structure based on density values.

There are several published methods for viewing multiple nested surfaces in order to show the context of one surface inside another. These methods render interior surfaces as opaque, but place a pattern on a translucent exterior surface in order to better understand the shape of this partially transparent surface. Interrante has used both strokes and line integral convolution to decorate translucent exterior surfaces [4, 5]. Rheingans used re-triangulation of a given polygonal model in order to create regular patterns such as disks on a partially transparent exterior surface [9]. Given an automatic technique for identifying interior and exterior portions of a surface, any of these techniques could be put to better use to show the relationships between these different portions of a surface.

3 Defining Visibility

Our first task is to define what we mean for a portion of a surface S to be exterior or interior. These definitions will depend on the visibility of the part of the surface in question. Intuitively, we will say that a point $\mathbf{p}$ is on the exterior of a surface if there is some sufficiently distant viewpoint $\mathbf{q}$ from which $\mathbf{p}$ is visible. We say "sufficiently distant viewpoint" because there is always some position $\mathbf{q}$ from which a given point will be visible, and we wish to disallow those viewpoints that are too close (or perhaps inside) the surface.

To define exterior, we will make use of the concept of the *extremal points* of a surface along a line. A line L that passes through two points $\mathbf{p}_1$ and $\mathbf{p}_2$ can be defined as the set of points $L = \{\mathbf{p}_1 + t(\mathbf{p}_2 - \mathbf{p}_1) : t \in \Re\}$. Consider the set of points of the surface along a line, $T = S \cap L$. Any point $\mathbf{p}$ in this set can be written as $\mathbf{p} = \mathbf{p}_1 + t(\mathbf{p}_2 - \mathbf{p}_1)$ for some t, which can allow us to define a function $t(\mathbf{p}) = (\mathbf{p} - \mathbf{p}_1)/(\mathbf{p}_2 - \mathbf{p}_1)$. We will say that a point $\mathbf{p}$ in T is an *extremal point* if the value of $t(\mathbf{p})$ takes on either the minimum or maximum value for all points in T. It is now easy to define interior and exterior. We define a point $\mathbf{p}$ on S to be *exterior* if it is an extremal point for some line L. We will say $\mathbf{p}$ on S is *interior* if it is not an extremal point for any line L.

It is a compute-intensive task to determine whether there exists a line for which a given point is extremal. For our own work, we have found it sufficient to consider a collection of lines that are parallel to one of a small set of directions V. The elements of V are vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ that are evenly distributed over a hemisphere. We then consider whether a given point $\mathbf{p}$ is extremal along any of the lines $L = \mathbf{p} + t\mathbf{v}$ for $\mathbf{v} \in V$. In our method, we classify a point $\mathbf{p}$ on a surface S as *exterior* if it is an extremal point along a line that passes through $\mathbf{p}$ along at least one of the directions in V. We will classify a point as *interior* if it is not exterior. In the following sections we will describe a fast algorithm for performing this classification task.

4 Generating Depth Buffers

Given a set of viewing vectors $V = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$, we seek a way to mark which polygons are visible along each viewing direction $\mathbf{v}_i$. A polygon p is classified as being inside the model if all the rays cast along $\mathbf{v}_i$ strike another surface before hitting p. In order to do this process efficiently for large polygonal models, we use polygon scan-conversion with an orthographic projection to create "deep" depth buffers. Our "deep" depth buffers are similar in structure to the Layered Depth Images presented in [11]. In that work, each pixel in a LDI kept a list of depth samples representing depth samples further and further along a single line of sight. Similarly in our case, a depth buffer is a 2D array of pixels. Each "deep" pixel in the depth buffer contains not only the depth sample for the nearest polygon, but a sorted list of depth samples. Each depth sample in the list represents the intersection of a ray shot through the pixel with a polygon of the model. By using an orthographic projection, we are effectively sampling the model along a large collection of rays that are parallel to a given direction $\mathbf{v}_i$.

5 Classification

5.1 Using Depth Buffers

Once a depth buffer has been generated for each view, we are ready to classify all the polygons of the input model. To this end, we take a polygon p from the input model, and check how many views from which it is visible. For each view $\mathbf{v}_k$ we select the depth buffer d_k. For each vertex of p, we find the depth pixel $d_k(i, j)$ onto which the vertex projects. The vertex is marked exterior if it is either the first or the last surface that the ray passing through $d_k(i, j)$ intersects. This corresponds to testing whether the vertex generated the first or last depth sample in the sorted list of depth samples stored at $d_k(i, j)$. The closest intersection to the vertex of p is reported back as the depth sample generated by that vertex. The polygon p is marked visible if all of its vertices are visible, and it is marked invisible otherwise.

We keep track of the number of depth buffers that declare p to be on the exterior and the number that classify p as being on the interior of the model. Figure 2 shows three out of the 40 depth buffers we used to classify the polygons of a skull model. When assigning the final classification to p, we require that at least m buffers classify it as being visible for the algorithm to declare that p is visible. This threshold value m is a user defined parameter in our implementation.

The reason a voting scheme is necessary to make the final visibility classification is that often all the depth buffers will not agree on the classification of a polygon. This can happen in several cases. The most common case is that of the input model containing tunnels. The polygons along the wall of a tunnel will be visible from a viewpoint that looks down the tunnel. This case in shown in Figure 1 (bottom left) where the marked polygon is visible from the current viewpoint, although it is embedded deep in the interior of the model. In this case, viewing directions orthogonal to the current viewpoint will vote that the highlighted polygon is interior to the model.

The other case where the depth buffers disagree on whether a polygon is visible or not is that of complex surfaces. Such models have polygons on the exterior surface that lie in highly occluded parts of the model. A large number of viewpoints must be used to adequately sample all the parts of the exterior surface. Polygons that lie in occluded parts of the model are only visible from a small number of viewing directions. Therefore we should expect that most of the depth buffers will vote for classifying such polygons as being invisible, while a small number of depth buffers will vote to make these polygons visible.

The classification threshold represents a trade-off between the two cases outlined above. If set too low, most polygons that lie on the interior of tubes embedded in the model will incorrectly be marked visible. However, polygons that lie on the exterior surface but in highly occluded parts of the model will be correctly marked visible. If it is set too high, then the algorithm will misclassify highly occluded polygons on the exterior surface of the model as being invisible. Polygons inside tubes will be marked invisible as expected. To alleviate this difficulty in choosing the correct classification threshold, we use morphology to close holes and collapse tunnels in complex models. This preprocessing step allows us to use the classification threshold solely to classify highly occluded polygons on surface the of a complex surface.

It is also possible to use the depth buffers to assign opacity values to the polygons of the model. These opacity values can then be used to render the model. For example, if 40 viewing directions are used to do the classification, then polygons visible from all 40 directions can be made almost transparent. Polygons that were not visible from any directions can be marked opaque. The opacity of polygons visible from a certain number of viewing directions can have an opacity value assigned to them based on the number of directions from which they are visible. The Skull and Motor models in Figure 6 were rendered using this scheme.

5.2 Processing Large Triangles

As described in Section 5.1, we project each of the vertices of a polygon p into all the depth buffers to determine the visibility of p. If p is a large polygon, then there is a good chance that the visibility will vary across p. Figure 1 (bottom left) shows this case. The highlighted polygon runs all the way across the tunnel. The solution that we employ to deal with this problem is to break up large triangles into smaller ones. The decision of whether to subdivide a triangle is based on an edge-length tolerance that can be specified by the user. The smaller the edge-length tolerance, the greater the number of triangles that will be generated. And as smaller triangles result in better classification, the final result will benefit from a small edge-length threshold. In practise, the edge-length threshold is constrained by the largest file size on the machine being used, or by memory considerations.

We iterate over all the triangles in the input model, subdividing them until the largest edge in the model is below the user specified threshold. Once a triangle t has been subdivided into a n smaller triangles $\{t_0, t_1, \cdots, t_n\}$, we process each of newly created triangles in the same way that a triangle belonging to the original model would be treated. For each t_i, we project its three vertices into all the depth buffers. Once all the depth buffers have voted on the visibility of t_i, we use the threshold to decide on the final classification of t_i.

6 Morphology

We use volumetric morphology to close holes and tunnels in complex models. Because our input models are polygonal, we require a robust method of converting a polygonal model to a volumetric representation. The technique that we use to voxelize the input polygonal model is described in [8]. To perform volumetric morphology, we use a 3D version of Danielsson's Euclidean 2D distance map algorithm [1]. The input to the distance map algorithm is a binary volume. Such a volume has voxels with the value 0 if they are not on the surface of the object, and 1 otherwise. The output of the distance map algorithm is a volume with the same dimensions as the input volume. This distance map volume contains, for each voxel, the distance to the nearest voxel on the surface. As expected, the distance map value of surface voxels is zero.

Given the distance map and a closing distance d, we can perform a volumetric *closing* of the volume. A closing operation consists of a *dilation* followed by an *erosion*. A dilation will convert all the background voxels within a distance d of the surface into surface voxels. The erosion operator converts all the surface voxels within distance d of any background voxel into background voxels. A closing operation will close holes and eliminate thin tunnels. Figure 3 shows the underside of both the original and a morphologically closed version of a turbine blade. The size of tunnels and holes in the volumetric description depends on the voxel resolution. In our work, we have found that the model should be voxelized at approximatley 10 times the closing distance.

After performing a closing on the volumetric description of the model, we apply the Marching Cubes algorithm [7] to the volume to obtain an isosurface. This isosurface, which is a closed version of the input model, is used to generate another set of depth buffers from all the viewpoints. The depth buffers from the original and closed models are used together to classify all the polygons in the original model.

When classifying a polygon p in this case, we project p into both the original depth buffers and the closed depth buffers. We first check to see if p was near the exterior surface in the original depth buffer. As before, we project p into the depth buffer being processed. If p is too far away from both the minimum and maximum depth values, then it is marked as interior to the surface and the next depth buffer is considered. The maximum distance that p is allowed to

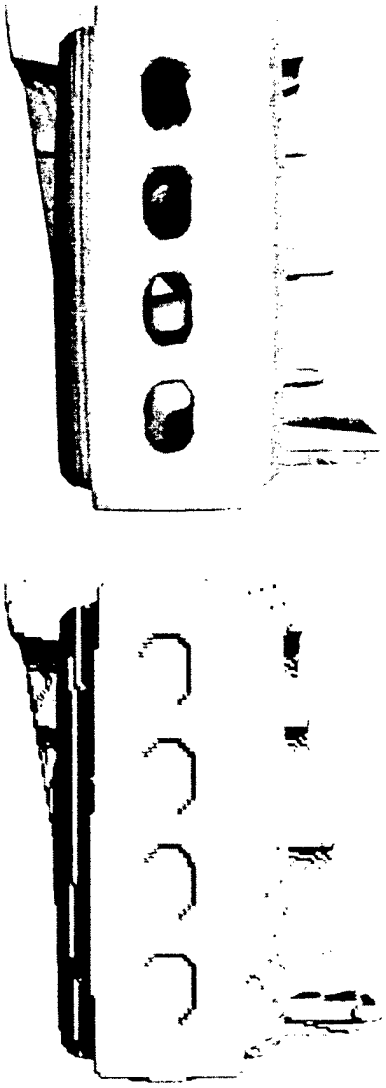


Figure 3: View of the four large holes in the base of the turbine blade (top) and the morphologically closed holes of the same model (bottom).

deviate from the minimum and maximum depth values is a user controlled parameter in our implementation. If both a morphologically closed and the original model are used to determine the visibility of the polygons, then this distance is equal to the closing distance used to produce the closed version of the model. If morphology is not used, then this distance should be set to the depth buffer resolution.

When p is determined to lie close to the exterior surface in the original model, we need to ensure that it is not part of a tunnel in the original model. To do this, we project p into the closed depth buffer and examine the list of depth values at the depth pixel that p projects to. If p was inside a tunnel in the original model, then it will not be close to any depth sample values in the closed model's depth buffers. This is due to the fact that the closed model does not have any tunnels, and therefore there will be no depth samples in its depth buffers at locations where the tunnel existed in the original model.

7 Results

In this section we describe the results of applying our visibility classification to several polygonal models. All the images of Figure 6 are reproduced in the colorplate. For all of the results in this paper we used 40 viewing directions to create 40 depth buffers. These 40 directions were taken from the face normals of an icosahedron whose triangles were subdivided into four smaller triangles and whose vertices were then projected onto the surface of a sphere. Only face normals from half this model were used for the viewing directions since opposite faces define the same lines.

Figure 1 shows a model of a cube that has three holes punched through it. Rendering the surface as opaque does not show details of the geometry of how the holes meet in the interior of the model. The lower right of this figure shows a rendering of this cube based on our algorithm's classification of interior and exterior portions of the model. The exterior surface has been rendered partially transparent, and the interior is opaque. In this image, the interior structure is clearly evident. For this relatively simple model it would have been possible to come up with a mathematical expression to classify the interior and exterior. The other models we discuss below are far too complex to create any classification using such a mathematical expression.

Figure 4 (left) shows an opaque version of a turbine blade model that was created from CT data of an actual blade. The right portion of this figure shows only those polygons that have been classified as interior by our algorithm. Figure 6 (top row) shows more views of the same turbine blade model. These images were created using classification based on 40 viewing directions and the morphologically closed version of the object that is shown in the bottom of Figure 3. The detailed interior structure of the turbine blade is clearly revealed in these images. There are four large holes in its base that merge pairwise into two passageways. Each passageway snakes up, down, up, and then each meets up with many tiny holes on the left and right sides of the blade. In addition, the top right image shows a small tube (in the lower right of this image) that extends from the back to the side of the blade. This tube was previously unknown to us, even though we have used this turbine blade as a test model for several years, often rotating it interactively on a high-end graphics workstation. This indicates to us that images based on interior/exterior classification show promise for exploratory visualization of data.

Figure 6 (bottom row) shows a car engine model that was created by hand. This model has interesting topology in the sense that there are a number of parts connected by tubes and thin structures. In addition, this model has a large number of degeneracies such as T-joints, zero-area faces and non-manifold vertices. In this case, we did not perform a binary interior/exterior classification of the polygons of the motor model. Instead each polygon was assigned an opacity value based on the number of buffers that it was visible from. Again, 40 views and a morphologically closed version of the model were used to perform the opacity assignment. Polygons that were seen from more than 60% of the depth buffers were made almost transparent. And polygons that were visible from less than 10% of the buffers were made opaque. Other polygons were assigned opacities that varied linearly between these two limits. The closeup of the front part of the model shows quite clearly the gears and thin pipes in the

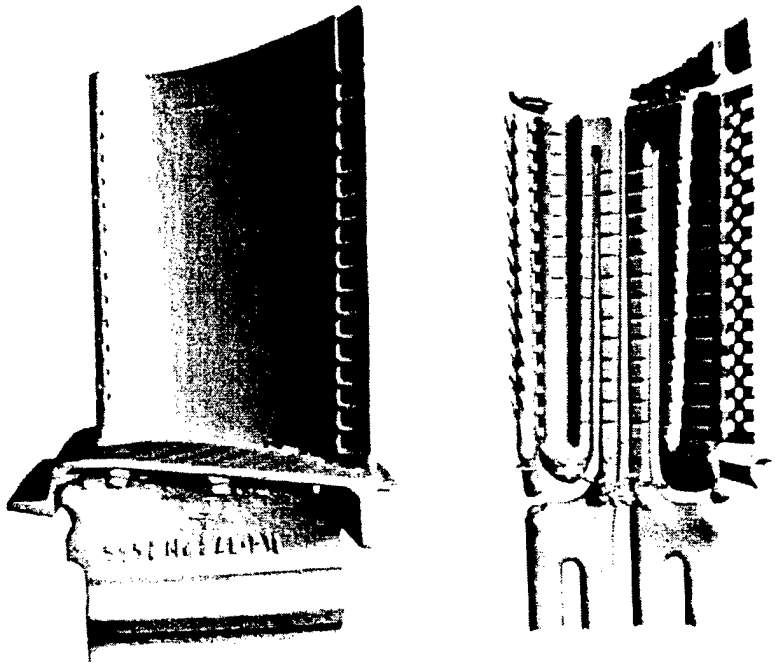


Figure 4: Opaque rendering of turbine blade (left), and just the interior surfaces (right) as classified by the method of this paper.

interior of the motor. The side view of the motor shows the cam-shaft of the engine in the lower part of the model.

Figure 6 (middle row) shows a dataset created from 124 CT slices of a human skull. Figure 5 shows two opaque renderings of the skull. 40 views and a morphologically closed version of the model were used to perform the opacity assignment. Again we did not do a binary interior/exterior classification of the polygons of the skull model, but instead assigned them opacities based on the number of directions from which they were visible. Turning the exterior surface transparent clearly reveals the two large sinus cavities on either side of the nose. Other detail is evident in these images, including a thin hollow tube on the forehead near the location where the plates of the skull join in the first two years after birth, and another small tube at the base of the jaw. The rightmost image of the back of the skull shows the *interior* suture lines, which are the interior versions of the more familiar suture lines on top of the skull that are visible from the outside.

Table 1 list the timing results of the different portions of the classification algorithm. In all cases, classification is performed in just a few minutes. Because our algorithm is proposed as a preprocessing step to a visualization method, the classification only needs to be done once for a given polygonal model.

8 Summary and Future Work

We believe that this paper makes two contributions to the field of visualization. First, we have identified a new problem, namely the problem of classifying exterior and interior portions of a polygonal model. The solution to this problem has potential applications in a number of areas in visualization. Second, we have proposed a specific solution to this problem – creating depth buffers of a model in several directions and classifying polygons based on whether they are occluded in each of these directions. We further enhanced this method by closing holes using a morphological operator. We use classification information to make interior surfaces opaque and exterior surfaces partially transparent in order to visualize the relation between these portions of a model. This new approach gave results that revealed several features of the models that we did not know about prior to running the interior/exterior classification.

As with most first attempts to solve a new computational problem, the approach that we describe here is not ideal in all cases. High frequency variations in the surface can cause some small portions of a surface to be classified as interior even though a human observer would label them as exterior. Perhaps surface smoothing or some form of neighborhood information could be used to eliminate this problem.

Although our current implementation is quite fast, it may be possible to accelerate the method using hardware rendering rather than software scan-conversion to create the depth buffers. Finally, the results of our classification should be used in conjunction with a method that adds texture to exterior surfaces to better understand their shape.

9 Acknowledgements

We are grateful to Will Schroeder, Ken Martin and Bill Lorensen for the use of the turbine blade data that is included in the CD-ROM of their book *The Visualization Toolkit*. We thank Bruce Teeter and Bill Lorensen of General Electric and Terry Yoo of the National Library of Medicine for the skull data.

References

[1] DANIELSSON, P. Euclidean Distance Mapping. *Computer Graphics and Image Processing, 14*, 1980, pp. 227-248.

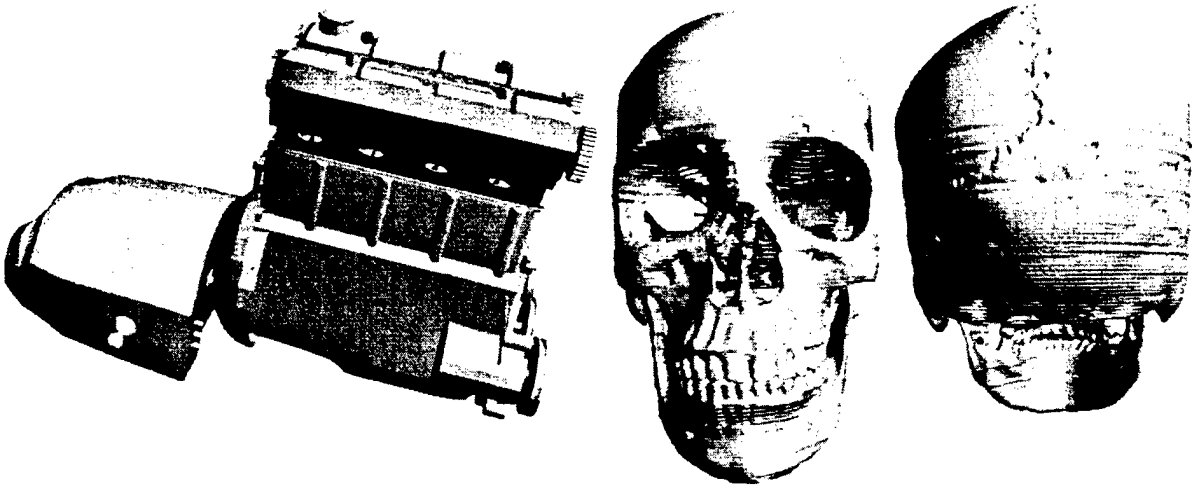


Figure 5: Opaque renderings of the Motor and Skull models. Notice the exterior suture lines on the back of the skull.

	Time to Process Models (minutes:seconds)					
	Model Size (# polygons)	Triangles Processed (Small Triangles Generated)	Depth Buffer Creation	Morphology	Classification	Total
Blade	50,000	285,070 (375,994)	1:29	1:09	8:44	11:22
Skull	200,000	216,625 (29,371)	6:00	0:53	6:01	9:51
Motor	140,113	219,698 (122,219)	2:57	0:52	6:09	9:59
Cube	578	1,730 (1,820)	0:27	*	0:42	1:09

Table 1: This table shows the timing information for each stage of the classification process. All the timing measurements were taken on a SGI Octane with 256 Mb of Ram.

[2] DREBIN, ROBERT A., LOREN CARPENTER, and PAT HANRAHAN Volume Rendering, *Computer Graphics*, Vol. 22, No. 4 (SIGGRAPH 88), August 1988, pp. 65-74.

[3] EL-SANA, J. and A. VARSHNEY Controlled Simplification of Genus for Polygonal Models, *Proceedings of IEEE Visualization '97*, Oct. 19 - 24, Phoenix, Arizona, pp. 403-412.

[4] INTERRANTE, VICTORIA, HENRY FUCHS and STEPHEN PIZER, Enhancing Transparent Skin Surfaces with Ridge and Valley Lines, *Proceedings of IEEE Visualization '95*, October 29 - November 3, Atlanta, Georgia, pp. 52-59.

[5] INTERRANTE, VICTORIA, Illustrating Surface Shape in Volume Data via Principal Direction- Driven 3D Line Integral Convolution, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 97), pp. 109-116.

[6] LEVOY, MARC, Display of Surfaces from Volume Data, *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May 1988, pp. 29-37.

[7] LORENSEN, W.E. and CLINE, H.E., Marching cubes: A high resolution 3-d surface construction algorithm, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 1987), pp. 163-169.

[8] NOORUDDIN, F.S. and GREG TURK, Simplification and Repair of Polygonal Models Using Volumetric Morphology, *Technical Report GIT-GVU-99-37, Georgia Institute of Technology*, 1999.

[9] RHEINGANS, PENNY, Opacity-modulated Triangular Textures for Irregular Surfaces, *Proceedings of IEEE Visualization '96*, San Francisco, California, Oct. 27 - Nov 1, 1996, pp. 219-225.

[10] JAREK ROSSIGNAC, ABE MEGHAD, and BENGT-OLAF SCHNEIDER, Interactive Inspection of Solids: Cross-sections and Interferences, *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 353-360.

[11] SHADE, JOHNATHAN, S. GORTLER, L. HE and R. SZELISKI, Layered Depth Images, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 98), pp. 231-242.

[12] TIEDE, U., K. H. HOHNE, M. BOMANS, A. POMMERT, M. RIEMER, and G. WIEBECKE, Investigation of Medical 3D-Rendering Algorithms, *IEEE Computer Graphics and Applications*, Vol. 10, No. 2, 1990, pp. 41-53.

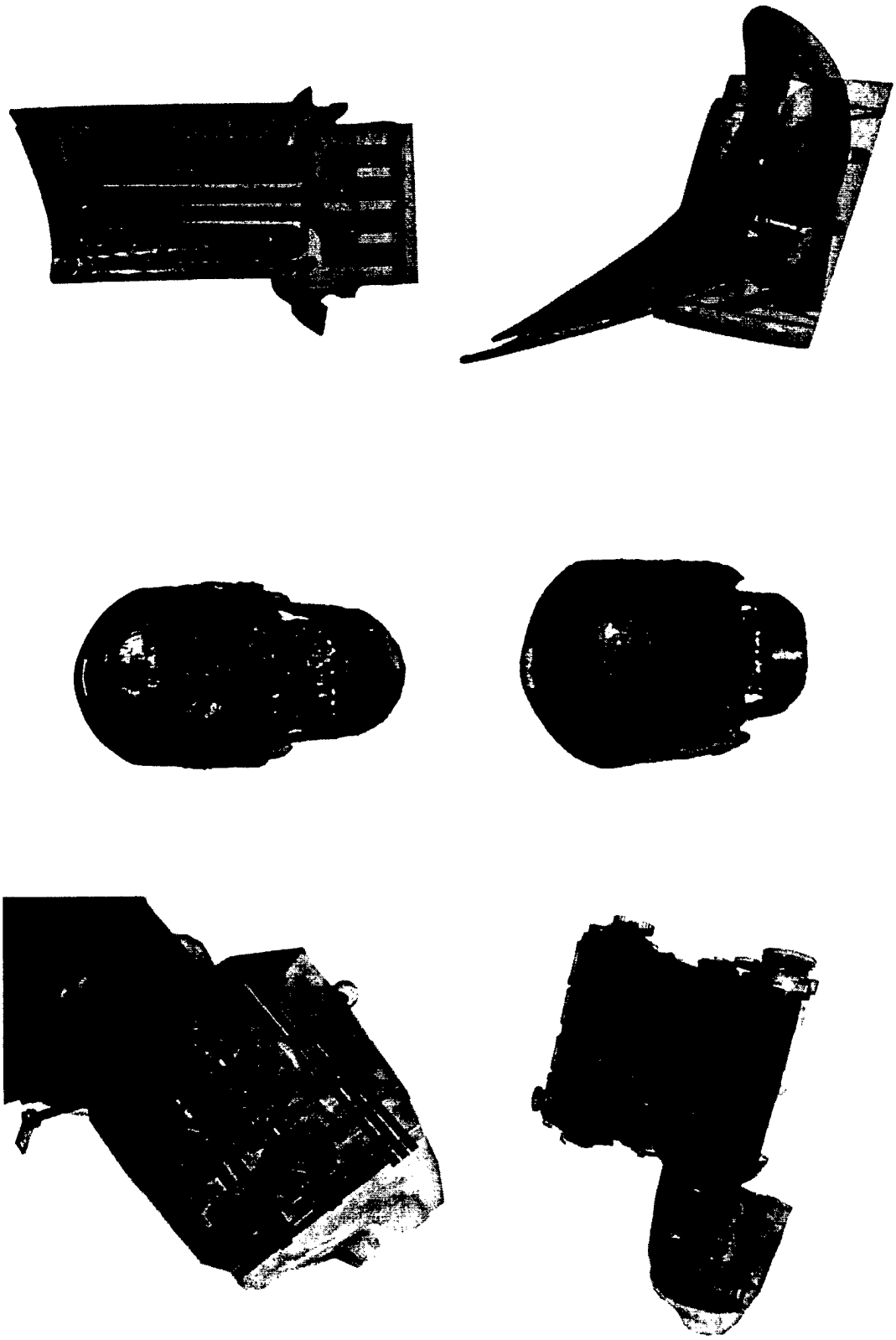


Figure 6: Results of applying our algorithm to the Skull, Motor and Turbine Blade models. A binary classification was performed on the polygons of the blade model. The polygons of the Skull and Motor models had their opacities modulated based on the number of directions from which they were visible.

Simplification and Repair of Polygonal Models Using Volumetric Techniques

F.S. Nooruddin and Greg Turk

I. Abstract

Two important tools for manipulating polygonal models are simplification and repair, and we present voxel-based methods for performing both of these tasks. We describe a method for converting polygonal models to a volumetric representation in a way that handles models with holes, double walls and intersecting parts. This allows us to perform polygon model repair simply by converting a model to and from the volumetric domain. We also describe a new topology-altering simplification method that is based on 3D morphological operators. Visually unimportant features such as tubes and holes may be eliminated from a model by the *open* and *close* morphological operators. Our simplification approach accepts polygonal models as input, scan converts these to create a volumetric description, performs topology modification and then converts the results back to polygons. We then apply a topology-preserving polygon simplification technique to produce a final model. Our simplification method produces results that are everywhere manifold.

II. Introduction

We are in the midst of an explosion in the production of very large geometric models. Advances in many technical areas are fueling this trend: remote sensing, medical scanning, scientific computing, CAD. Remote sensing devices such as synthetic aperture radar produce enormous terrain datasets. Medical sensing technology such as MRI, CT and PET scanners produce large volume datasets that lead to the creation of large isosurfaces. Scientific computing for applications such as structural analysis, synthetic wind tunnels and weather prediction result in large datasets that may vary over time. Finally, computer-aided design is used routinely for large tasks in architecture and mechanical design. Polygon representations of CAD models may run into the hundreds of thousands of polygons. We require robust methods for manipulating such large models. Two important tools are repair of models that are non-manifold and simplification of models. *Repair* is the process of taking a model that may have undesirable features such as cracks or self-intersections and creating a new model similar to the original but that has none of its flaws. *Simplification* of a polygonal model produces another model that has much the same appearance as the original but has many fewer polygons. Our paper addresses both of these tasks.

Many algorithms and applications require well-behaved polygon models as input. T-joints, cracks, holes, double walls, and more than two polygons meeting at an edge are just a few of the possible degeneracies that are often disallowed by various algorithms. Unfortunately, it is all too common to find polygonal models that have such problems. Applications that may require "clean" models include finite element analysis, radiosity, shape transformation, surface smoothing, calculation of moments of inertia, automatic model simplification, and stereolithography. Several approaches to polygonal model repair have been presented in the graphics literature. Unfortunately, most of these proposed methods are complex to program and some do not scale well as the polygon count increases. We present a method of scan-converting polygons into a voxel representation that yields a simple yet effective solution to polygon repair. The same voxelization process is also an important step in our simplification method.

Much work has been published recently in the area of automatic simplification of polygonal models, and yet there are still many problems that need to be addressed. One of the important

issues is the elimination of unnecessary fine details such as small holes or thin struts— a task that implies making changes to the topology of a model. Many of the earlier published simplification methods made an effort to preserve the topology of the original model. It eventually became evident, however, that topology is often a limiting factor in the simplification of a given object. Consider a box with 100 tiny holes punched all the way through it. A simplification method that preserves topology must retain at least three polygons to represent each hole, and thus will retain at least 300 polygons, yet the model can be fairly well represented using just six faces. This problem has led several researchers to relax the restriction on topology preservation in order to remove small features such as small holes or thin bars and pipes. One important issue in topology simplification is whether a user may specify the exact size of the features to be removed from a model. A second issue is whether the simplification method produces manifold surfaces. Additional issues include the simplicity of programming, the memory requirements and the computational cost, and these are important regardless of the treatment of topology.

We have pursued a volumetric approach to geometric simplification. There are several reasons for this choice. First, volume models have none of the topological ambiguities that a polygonal model may have. For example, it is possible for a polygonal model to contain three or more polygons that share an edge— a non-manifold situation. Purists may argue that such models should never be created in the first place, but the fact is that models with non-manifold surfaces are only too common. We feel it is necessary to handle these common cases, and we do this during the step that converts polygonal models to a volumetric representation. A second reason for working in the volume domain is that we then have access to a wide array of techniques that have been developed for image processing, since volumes have the same regular structure as images but in one higher dimension. Finally, there are dozens of polygon-based methods for performing simplification, and in contrast there have been relatively few proposed methods that make use of a volumetric representation.

Figure 1 shows a schematic diagram of our simplification pipeline. There are four stages in our simplification method: voxelization, 3D morphology, isosurface extraction and triangle count reduction. If the goal of the user is only to repair a polygonal model, then the morphological operations are not performed.

The remainder of this paper is organized as follows: in Section 3, we present a brief literature review of polygon simplification, voxelization and model repair. In Section 4 we describe our new method for converting a polygonal model into a volumetric representation. In Section 5, we describe volumetric morphology and show how it is used to simplify the topology of an object. In Section 6 we discuss isosurface extraction and topology preserving triangle count reduction for producing the final model. Section 7 presents the results of our approach when used on a variety of models, discusses these results and gives timing information. Section 8 summarizes the characteristics of our approach and describes possible future work.

III. Related Work

In this section we review previous work in simplification and model repair. Because conversion of polygons into voxels is an important step in our approach, we also review related work in voxelization.

A. Simplification

A large number of approaches to geometric simplification have been published in the graphics literature. Rather than attempting to cover all of them, we will concentrate our attention on those simplification methods that allow the topology of a model to be changed.

Rossignac and Borrel created one of the earliest methods of performing polygonal simplification that allows topological changes [27]. Their approach is to group the vertices of a model into clusters that fall within the cubes formed by a uniform spatial subdivision. Those vertices that fall within one cell are merged into a single vertex, and the degenerate triangles that are created

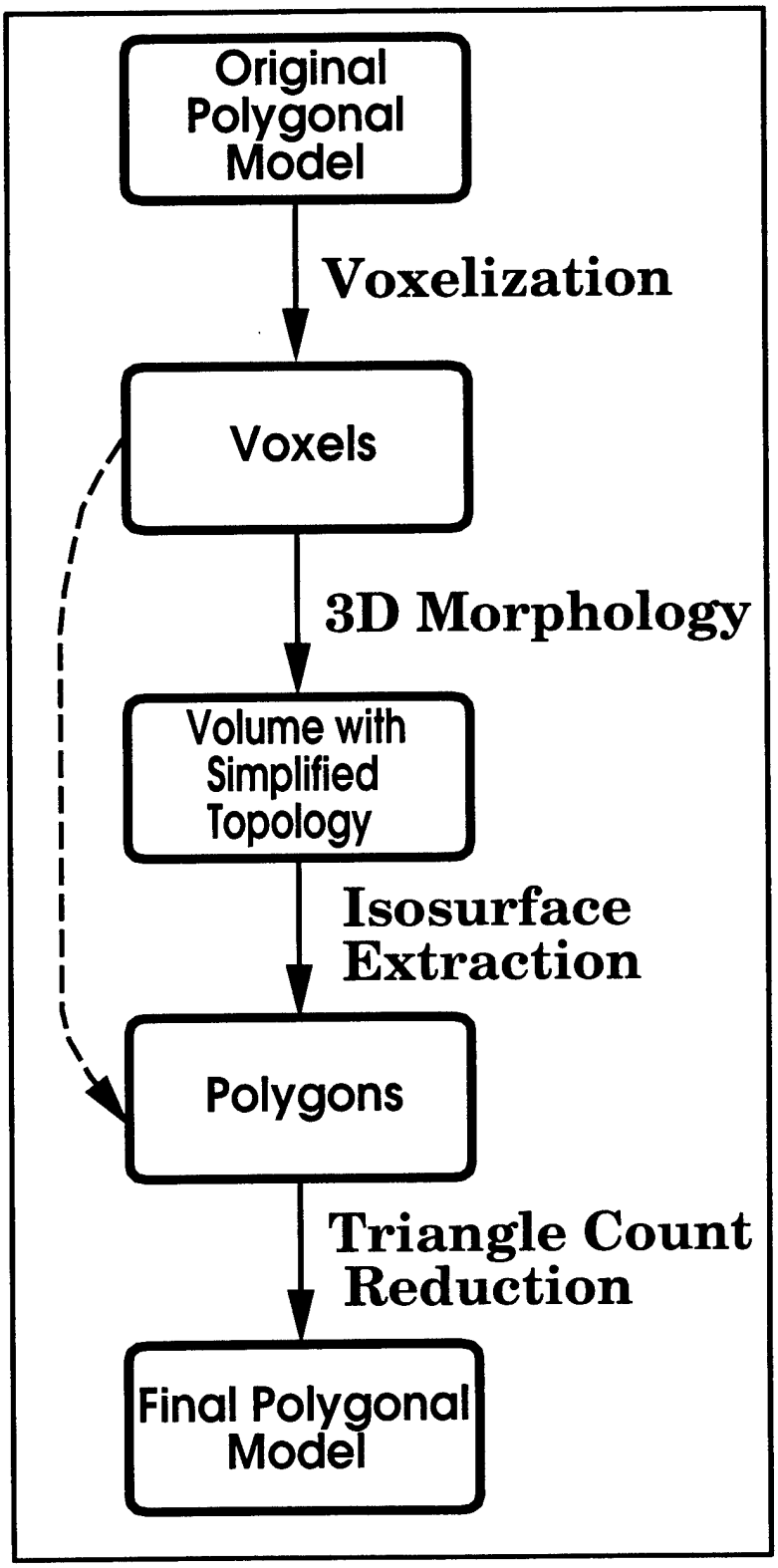


Fig. 1. The simplification pipeline. Dotted arrow shows the path used for model repair.

by this are removed from the model. More recently, Low and Tan have enhanced this approach by making the vertex clustering independent of the position of the model in 3D, and they also select the position of the new vertices using new heuristics [22]. Luebke and Erikson also used such a vertex clustering scheme in their view-dependent simplification approach [23]. Due to the dynamic nature of view-dependent simplification, they used a tree data structure in which to store a hierarchy of potential vertex clusters.

Schroeder and co-workers created one of the earliest polygonal simplification algorithms that successively removes vertices near relatively flat regions [29]. The original algorithm preserved topology, but in more recent work, Schroeder extended this method to allow topological changes [30]. When no more vertices can be removed from the model due to topological restrictions of the algorithm, the method splits apart the polygons adjacent to a vertex. This allows greater freedom in vertex removal, and thus allows the model to be further simplified. This newer algorithm also tracks error bounds at vertices, allowing bounds to be put on the amount of error incurred during simplification.

Garland and Heckbert demonstrated a topology-modifying simplification algorithm based on a generalization of the edge collapse operator [8]. An edge collapse replaces two vertices that share an edge with a single vertex, removing two triangles in the process. Their more general *vertex pair contraction* operator merges together any two vertices, regardless of whether they are joined by an edge or not. Garland and Heckbert use a quadric error metric to determine the best vertex pair contraction during simplification. Popovic and Hoppe take a similar approach to simplification, also using vertex pair contraction to reduce a model's complexity [28]. They use a cost function for a contraction that includes a measure of distance to the original surface as well as a term that penalizes contractions that would merge vertices which have different material properties.

El-Sana and Varshney use an approach that is inspired by *alpha-hulls* (a distance-controlled portion of the Delaunay triangulation) to identify small holes and protrusions that can be removed from a model [5]. Sharp edges are marked as candidates that are likely to surround a hole. Then an *alpha-prism* is used to determine whether candidate hole is small enough to be filled. Identified holes have their associated polygons removed and the boundary edges that are created are filled using triangulation. They use the same process to identify and remove thin structures that protrude from a model.

Quite a different approach is taken by He et al. to perform topology-modifying simplifications of models[14]. They convert models into the volumetric domain, perform low-pass filtering, and then use isosurface extraction to produce a new polygonal model. Low-pass filtering of the volume model eliminates fine details such as thin tubes and surfaces, and also closes small holes in the model. Unfortunately, low-pass filtering does not offer strict control over the topological changes that are to be made to an object. For instance, a hole of radius r might be filled if the hole is in the middle of an otherwise unbroken surface. A hole of the exact same size, however, can help create a larger hole if it is near one or more additional holes. In addition, large, thin surfaces of a model that should be retained can be accidentally eliminated by low-pass filtering. Despite these shortcomings, the volumetric filtering method has much to recommend it. Inspired by this approach, we created the new volumetric simplification method that we present in this paper.

B. Voxelization

Converting a polygonal model into a volume is an integral step in our method, thus we briefly review previous techniques that convert polygonal models into volumes. Wang and Kaufman use a method that samples and filters the voxels in 3D space to produce alias-free 3D volume models [33]. They place an appropriately shaped filter at the voxel centers and filter the geometric primitives (e.g. polygons) that lie inside the region of support of the filter kernel, and this produces the final density value for the voxel. This paper does not address how to determine

whether a point is interior to a collection of of polygons, an issue that needs to be addressed if a solid rather than a thin-shelled model is to be created.

Huang et al. describe *separability* and *minimality* as two desirable features of a discrete representation [16]. If a discrete surface is thick enough to prevent ray penetration it is said to meet the separability condition. If it contains only those voxels that are indispensable for separability then it also satisfies the minimality condition. They use bounding spheres around vertices, bounding cylinders around edges and bounding planes around each edge of each polygon to produce surfaces that meet both the separability and minimality conditions. The volumetric representations produced by this method are thin-shelled.

Schroeder and Lorensen create volumetric models by calculating a distance map from the input polygonal model [31]. Using this distance map they find the closest polygon to a given voxel and use the polygon's normal to classify the voxel as interior or exterior. They then use a distance threshold to obtain an isosurface from this distance map. They use the resulting offset surface to generate swept surfaces for the purpose of path planning for object assembly.

C. Model Repair

There are several different approaches that have been taken towards repairing polygonal models, including user-guided repair, crack identification and filling, and creating manifold connectivity.

Several interactive systems have been proposed for fixing errors in polygonal models such as cracks and T-joints. Two such systems that used manual intervention to repair architectural models are described in [7] and [18]. Morvan and Fadel proposed a virtual environment in which to perform user-directed repair for layered manufacturing [25]. Interactive techniques for model repair becomes unattractive as the size of the models becomes large.

A number of other model repair methods have concentrated on automatic crack identification and filling. Bohn and Wozny use Jordan curve construction and local hole filling to fix models with cracks [3]. Barequet and Sharir describe a method for crack finding and filling by triangulation [2], and Barequet and Kumar improve upon this method by sometimes shifting vertices to eliminate cracks [1]. Murali and Funkhouser create a BSP-tree representation of a model and then construct and solve a linear system of equations in order to determine which cells of the BSP-tree are solid or non-solid [26].

A third approach to model repair is presented by Gueziec et al. [13]. The goal of their repair method is to produce models that are everywhere manifold (perhaps with boundaries), and they are not concerned with eliminating cracks or self-intersections. Their method separates edges between polygons and then selectively stitches together some of these edges in a manner that avoids non-manifold configurations. This method operates entirely upon the connectivity between polygons and does not examine the 3D positions of the vertices.

All of the repair methods described above operate directly upon a polygon or half-space description of a given model. The method of model repair that we present in this paper is unique in that we convert a model into voxels in order to perform repair.

Now that we have reviewed the related work, we will describe the components of our model manipulation pipeline for simplification and repair.

IV. Polygons to Voxels

In order to use morphological operators to simplify topology, we must first voxelize the given polygonal model. In this section we present two new methods of voxelization, the *parity-count* and the *ray-stabbing* methods. At the end of this section we describe how voxelization provides a simple method for performing model repair.

A voxel representation of a model is a regular grid of cells, in our case a rectilinear grid, in which each cell (voxel) contains a density value in the range of zero to one. In this paper we will use a voxel-value of zero to represent a portion of un-occupied space and a value of one to

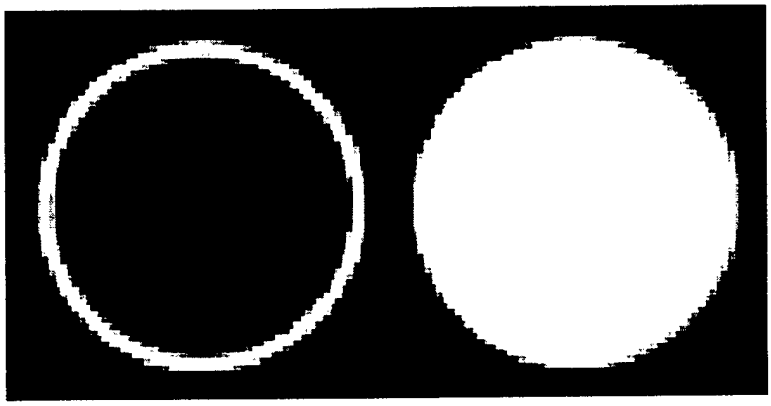


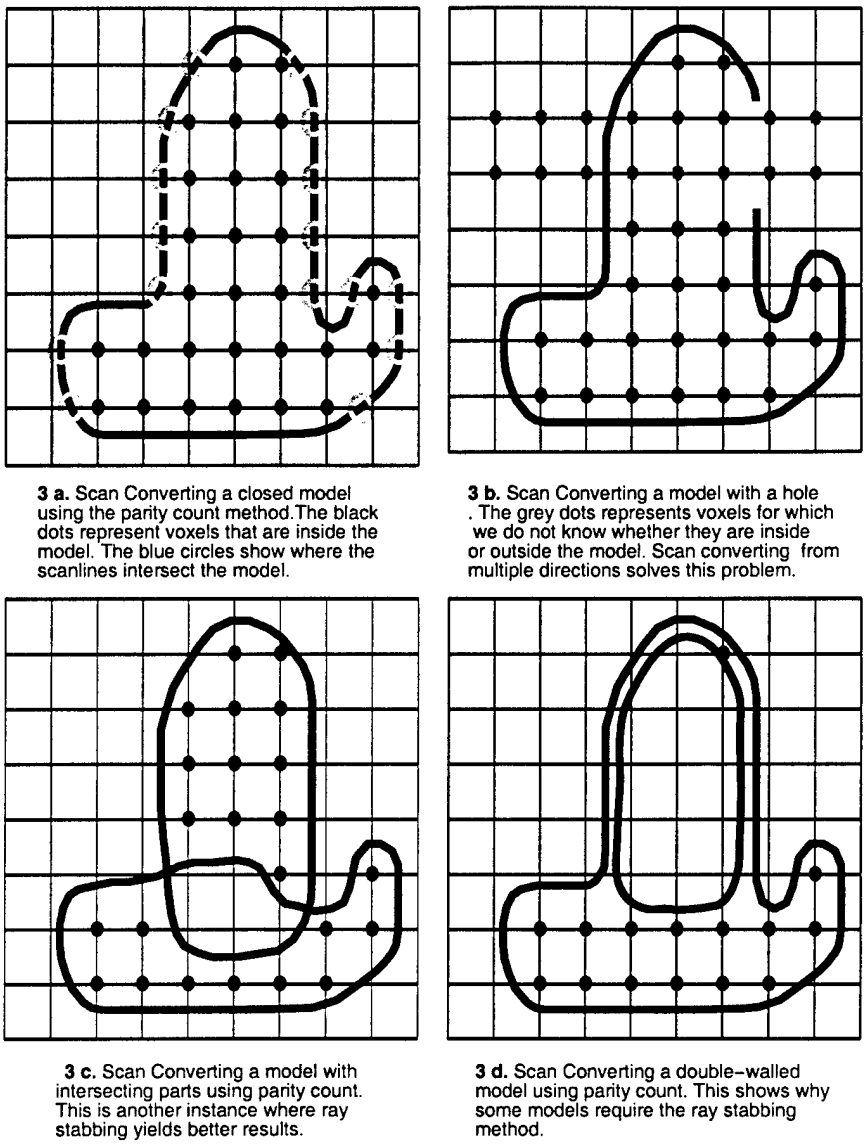
Fig. 2. (a) Slice through a thin-shelled volumetric representation of a sphere, (b) Slice through a solid volumetric representation of a sphere.

represent a voxel that is entirely interior to our model. Values between zero and one represent voxels that are near the surface of an object.

As described above, there are several published methods for performing voxelization of polygons [16],[31],[33]. Unfortunately, none of the published techniques are satisfactory for our needs. For our purposes, the voxel representation should not be thin-shelled. A thin-shelled voxelization of polygons is one in which only voxels that are near a polygon of the original model have a non-zero voxel value. Thin-shelled voxelization is performed by finding the distance between a given voxel and the nearest polygon [19, 33]. A thin-shelled representation of a sphere, for instance, would contain non-zero voxels only near the sphere's surface. Such a sphere would have a large region of zero-valued voxels inside its boundary. Figure 2 (a) shows a slice through such a thin-shelled sphere model. Performing isosurface extraction on such a model would produce a polygonal model that had two surfaces that are very near one other. In contrast, the voxel models that we use have voxel values of one in the interior of the object so that isosurface extraction yields a single surface. Figure 2(b) shows a slice through such a voxel model of a sphere.

A. Parity Count

To produce voxel models with true interiors, the exterior/interior classification of a voxel must take into account non-local aspects of the polygonal model. We will first discuss our *parity count* method of voxel classification when used on manifold polygonal models that are water-tight (have no cracks or boundaries). For such models, we classify a voxel V by counting the number of times that a ray with its origin at the center of V intersects polygons of the model. An odd number of intersections means that V is interior to the model, and an even number means it is outside. This is simply the 3D extension to the parity count method of determining whether a point is interior to a polygon in 2D. Note that for manifold models the direction of the ray is unimportant, and we can take advantage of this to speed up the voxel classification. In essence, we cast many parallel rays through the polygonal model, and each one of these rays classifies all of the voxels along the ray. For an $N \times N \times N$ volume, we need to cast only $N \times N$ rays, with each ray passing through N voxel centers. Instead of using ray-tracing, however, we actually use orthographic projection and polygon scan-conversion to create a "deep" z-buffer. Each pixel in the z-buffer retains not just the nearest polygon, but a linked list of depth samples. Each of these depth samples records an intersection with one polygon. Thus a "deep" pixel represents one of the parallel rays that has been cast through the model. Each voxel behind a given pixel can be rapidly classified by counting how many depth samples are behind or in front of the voxel center. Figure 3(a) shows a 2D representation of this process. In this figure, each blue circle represents a depth sample along a ray. Polygon scan-conversion takes advantage of incremental



3 a. Scan Converting a closed model using the parity count method.The black dots represent voxels that are inside the model. The blue circles show where the scanlines intersect the model.

3 b. Scan Converting a model with a hole . The grey dots represents voxels for which we do not know whether they are inside or outside the model. Scan converting from multiple directions solves this problem.

3 c. Scan Converting a model with intersecting parts using parity count. This is another instance where ray stabbing yields better results.

3 d. Scan Converting a double-walled model using parity count. This shows why some models require the ray stabbing method.

Fig. 3. Scan-converting polygonal models with a variety of degeneracies.

calculations, so this process is much faster than a ray-tracing approach would be.

Although the parity count method works well for manifold models, many polygonal models have various degeneracies that require us to modify the voxelization process. One common problem is for a model to have small cracks or holes in the surface. The Stanford Bunny model, for example, has several holes on its base, and the Utah Teapot contains a hole at the tip of the spout. Figure 3(b) illustrates the problem. To voxelize such models, we extend the parity count method by using k different directions of orthographic projection and by scan-converting the model once for each direction. Each of the k projections votes on the classification of a voxel (interior or exterior), and the majority vote is the voxel's final classification. For water-tight models, all of the votes will agree. This is not the case, however, for models that have a crack through which a ray may pass. Rays that pass through a single crack or hole will have an odd number of depth samples, and these rays are marked as invalid and do not vote. It can happen on rare occasions that one ray will pass through two cracks, and this will cause the ray to improperly classify many of the voxels. The majority voting between the directions of projection overrules the voting of such rays. Typically we perform three orthographic projections, one in each of the major axis directions. For troublesome models, we project in 13 directions, three along the

major axes and 10 directions that are described by the surface normals of an icosahedron. By choosing an odd number of projection directions we avoid having many ties in voting. Voting ties can still occur due to invalid rays, and we mark such voxels as being exterior to the model.

Figures 4(a) and (d) are two views of the Stanford Bunny polygonal model, and (d) shows the large holes in its base. Parts (b) and (e) show the result of using a single orthographic projection for the parity count voxelization method. The holes in (b) and (e) are the result of the algorithm classifying invalid columns of voxels (an odd number of ray intersections) as being exterior to the model. Using 13 projections creates a water-tight model, shown in parts (c) and (f) of Figure 4. This repaired model has none of the holes that were in the original model. In addition to the bunny model, both the turbine blade and the chair models (Figures 6 and 9) were voxelized using the parity count method.

We note that Lorensen and Schroeder also have converted polygonal models to voxel models that have true interiors [31]. They find the closest polygon to a given voxel and classify the voxel based on the polygon's normal. Their method is tolerant of models with small cracks, but it would produce poor results for polygonal models that are double-walled or that have intersecting surfaces. We handle such models using our ray stabbing approach.

B. Ray Stabbing

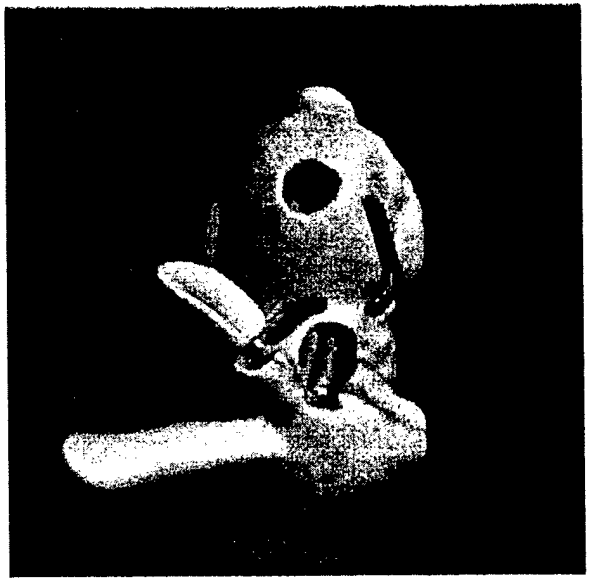
Unfortunately, cracks and holes are not the only kind of troublesome degeneracies in polygonal models. One other common problem is to have a model that is composed of several interpenetrating sub-parts. This is often found in articulated figures of humans and animals, where each limb or limb segment is a separate closed surface. For instance, an upper arm might be placed so that portions of its surface are inside the torso. This is not a problem if we are just rendering such a model. The parity count method, however, would incorrectly classify the overlapped portions of the arm and torso as being outside of the model. Figure 3(c) shows an example of two objects intersecting in this manner. Another common problem is to have a polygonal model in which there is more than one polygon at or near the same location in space. This is often the case for mechanical models where two sub-components are made exactly adjacent, but where the shared surface is represented by polygons from both sub-components. Double walls in building models are a similar problem. Such redundant polygons may cause problems for the parity count method as well. Figure 3(d) shows that the parity count method would create an empty interior for such a model. To voxelize this kind of model, we have created the *ray-stabbing* method of voxel classification.

The ray-stabbing method also makes use of orthographic projections of a polygonal model. It differs from the parity count method, however, in the way it interprets the depth samples of a ray. The ray stabbing method only retains the first and last depth sample along each ray. In effect, each ray only keeps those points of intersection where the ray first stabs the surface of the model. By keeping both the first and last depth sample, this is equivalent to stabbing the surface from two directions at once, at no extra cost. A voxel is classified by a ray to be interior if the voxel lies between these two extreme depth samples, otherwise it is classified as an exterior voxel. For a single direction of projection, this can cause some voxels to be mis-classified as being interior to the surface. To avoid this, we perform several projection in different directions. If *any* of the projections classify a voxel as exterior, it is given an exterior final classification. Only those voxels that are classified as interior for *all* projections are given the final classification of interior. Although reasonable voxel models result from three projections, we typically perform 13 projections for the ray-stabbing approach. Both the Al Capone and motor models of Figures 7 and 8 were voxelized using the ray stabbing method.

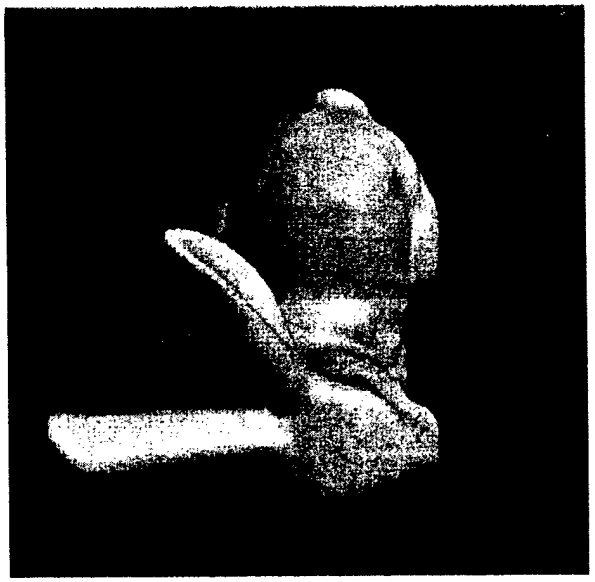
C. Polygonal Model Repair

We perform polygon repair by converting a model to a volumetric representation and then converting it back to polygons using isosurface extraction. This produces an everywhere man-

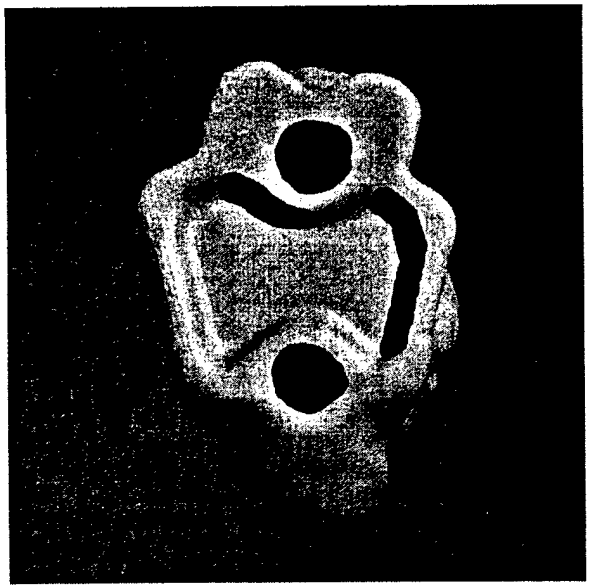
8 a. Original Bunny Model (Top View)
69,451 Faces



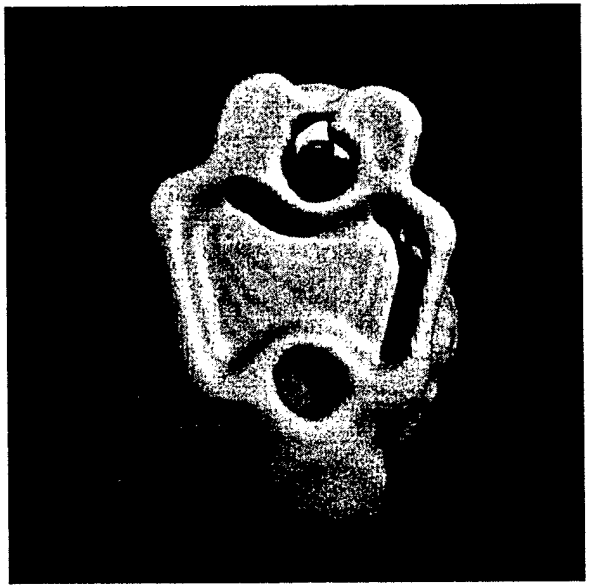
8 b. Results of Parity count using one scanning direction (Top View)
134,920 Faces



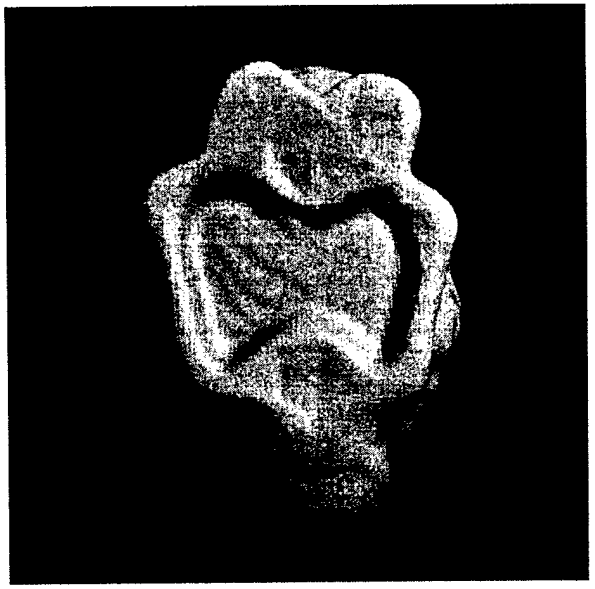
8 c. Results of Parity count using 13 scanning directions (Top View)
101,536 Faces



8 d. Bunny Model (Bottom View)
69,451 Faces



8 e. Results of Parity count using one scanning direction (Bottom View)
134,920 Faces



8 f. Results of Parity count using 13 scanning directions (Bottom View)
101,536 Faces

Fig. 4.

Types of Degeneracies		
	Parity Count	Ray Stabbing
Fixes T-Joints	Y	Y
Fixes Cracks / Holes	Y	N
Retains Interior Detail	Y	N
Merges Interpenetrating Surfaces	N	Y
Fixes Non-manifold edges and vertices	Y	Y

Table I. This table summarizes the types of degeneracies that the ray-stabbing and parity-count methods are able to fix.

ifold polygonal model that is free of holes, cracks, T-joints, double walls and interpenetrating polygons. The number of polygons produced by the conversion to and from the voxel domain is a function of the resolution of the voxel representation. If the polygon count for the model should be small, we reduce the number of polygons using standard polygon-based simplification. Our polygon repair method uses the same basic pipeline of operations as our simplification approach, but we skip the volumetric morphology step, as indicated by the dotted line shows in Figure 1. Figure 7 shows an example of polygon repair of a model with a number of interpenetrating parts, and Figure 4 illustrates repair of a model with several large holes.

The final results of polygon repair are significantly improved if proper sampling and anti-aliasing are performed during voxelization. To do so, we use supersampling and filtering in our implementation. We have implemented several filter kernels, and our best results are from a Gaussian filter kernel with a radius of two voxels and a standard deviation of 0.7 voxels. For an excellent survey of filter kernels for volume anti-aliasing, see [24]. We typically use $3 \times 3 \times 3$ supersampling to achieve high quality results, but using $2 \times 2 \times 2$ often produces results that are quite acceptable.

It is left to the user to choose between the parity-count method and the ray-stabbing method when repairing a given model. However, we can offer some guidelines on which method to use based on the types of degeneracies that are present in the model to be repaired. If the model does not have any interpenetrating parts, then the parity-count method should be used. As shown in Table I, the parity count method is able to repair most of the commonly occuring degeneracies in polygonal models such as non-manifold vertices and edges, T-joints etc. In the case that the model does have interpenetrating parts, then the ray-stabbing method should be used to eliminate them. Unfortunatly, neither the ray-stabbing nor the parity-count method can deal with models that have small cracks *and* interpenetrating parts. One possibility that exists in dealing with models like this is that the user can apply a polygon based model repair method such as that presented in [13] to remove the cracks from the model. Once that has been done, the ray-stabbing method can be then used to eliminate the interpenetrating parts of the model.

V. Morphological Operations

The morphological operators constitute the heart of our topology simplification algorithm. These operators are well suited to simplify the topology of objects because they present a clean and efficient way to remove small features, close holes and join disconnected components of a model. In addition, *openings* and *closings* provide precise tolerances so that the user can specify the size of the feature to be removed or the size of the hole to be closed. Finally, because these operations are done in the volume domain, we are able to recover a manifold mesh after the topology simplification has taken place.

The first step in using morphological operators is the calculation of a distance map. Given a binary volume that is classified into *feature* and *non-feature* voxels, a distance map associates with each voxel the distance to the nearest feature voxel. Feature voxels are those that are inside the object and non-feature voxels are those that lie outside the object. Feature voxels

have a distance map value of zero. We used Danielsson's algorithm [4] to calculate the distance map on our volumes. Specifically, we chose to implement the 4SED (four-point sequential Euclidean distance mapping) algorithm proposed by Danielsson. This algorithm is fast, but it is known to give slightly incorrect distances in some situations due to the fact that only the four immediate neighbors of a pixel contribute to its distance value. However, Danielsson reports that the absolute error in this case is less than 0.29 pixel units, which is quite acceptable for our purposes. When more accuracy is necessary, the user may simply use a finer voxel grid.

Below, we explain how the algorithm works on 2D images, after which we give a brief outline of how this is extended to 3D in order to create distance maps for volumes. Danielsson's 2D algorithm produces a floating-point distance map that contains one scalar value per entry. During the calculation of the distance map, the distances at each pixel are represented as two-dimensional integer vectors. For a given pixel, its distance map vector gives the integer distance in the x and y directions to the nearest feature pixel. The final step in the algorithm involves calculating the magnitude of these vectors, yielding the final scalar floating-point distance map.

The 2D algorithm starts by assigning a distance of zero to all feature pixels and a value of MAXVAL to the non-feature pixels. After the distance map is initialized, the image is scanned from bottom to top (the j direction). A pixel's distance map value changes if its distance map value is greater than that of its neighbors. Thus distance map values propagate from the sources of change (the feature pixels) to the non-feature pixels. For every j scan of the image, new values are propagated left, right and from the row of pixels below. This bottom-to-top scan only propagates information about a given feature pixel horizontally and upward. The image is then scanned a second time, from top to bottom, so that the distance values are propagated downward as well.

First loop of Danielsson's algorithm (sweeping from bottom-to-top)

for j = 1 to dy -1

Examine pixels below the current row
for i = 0 to dx - 1
if mag($\overrightarrow{D(i,j)}$) $\leq mag(\overrightarrow{D(i,j-1)} + \overrightarrow{<0,1>})$
$\overrightarrow{D(i,j)} = \overrightarrow{D(i,j-1)} + \overrightarrow{<0,1>}$

Examine pixels to the left of each pixel in a row
for i = 0 to dx - 1
if mag($\overrightarrow{D(i,j)}$) $\leq mag(\overrightarrow{D(i-1,j)} + \overrightarrow{<1,0>})$
$\overrightarrow{D(i,j)} = \overrightarrow{D(i-1,j)} + \overrightarrow{<1,0>}$

Examine pixels to the right of each pixel in a row
for i = dx - 2 downto 0
if mag($\overrightarrow{D(i,j)}$) $\leq mag(\overrightarrow{D(i+1,j)} + \overrightarrow{<1,0>})$
$\overrightarrow{D(i,j)} = \overrightarrow{D(i+1,j)} + \overrightarrow{<1,0>}$

The above pseudo-code is that of the bottom-to-top scan of an image. The second loop (top-to-bottom scan of the image) of Danielsson's 2D algorithm is similar to the loop shown above. The extension of this algorithm to 3D involves applying Danielsson's algorithm on a slice by slice basis to the volume. There are two passes done through the volume: one each in the forward and backward directions in the k dimension. For each of these passes, the distance map for each slice is calculated as described above. In 3D, a voxel's distance map value is calculated from the distance map values of its six immediate neighbors.

The two atomic morphological operations are *erosion* and *dilation*. They take as input the

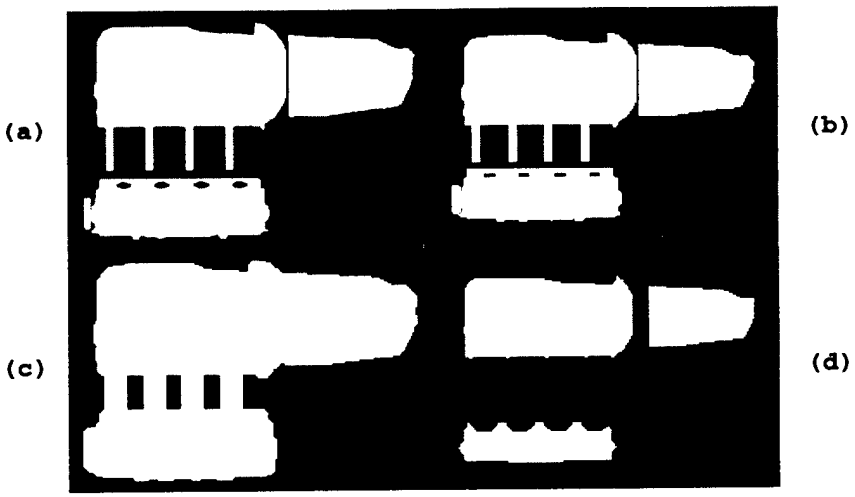


Fig. 5. (a) Slice through voxelized motor, (b) distance map, (c) dilation, (d) erosion.

volume, the distance map and an erosion/dilation distance. For dilation, we look through the distance map, and any non-feature voxel that has distance less than or equal to the threshold is turned into a feature voxel. Erosion is the complement of dilation. In this case, we negate the volume (i.e. a feature voxel becomes non-feature and vice versa), calculate the distance map, and then perform a dilation. After this, the volume is negated again to obtain the final result. These basic morphological operations are commonly used in image processing [17].

Figure 5 shows the results of applying erosion and dilation to a 2D image. Figure 5 (a) shows the original image. This is a slice of the volumetric description of the motor model. Figure 5 (b) shows a colorized distance map in which the colors indicate the distance of a pixel from the surface. Near the surface the blue color indicates a small distance, while the red color indicates a large distance. The colors cycle at greater distances. Using this distance map, we performed dilation and erosion on the image. Figure 5 (c) shows the result of performing dilation on the input image. It demonstrates how dilation will close small holes and join previously unconnected parts of the input image. The result of performing erosion is shown in figure 5 (d). Erosion eliminates thin structures and increases the distance separating two unconnected parts of the image.

While useful by themselves, erosion and dilation are usually used in conjunction with each other. The reason is that if they are used in isolation, then they increase (in the case of a dilation) or decrease (in the case of an erosion) the bounds of the volume. When an erosion is done followed by a dilation, it is called an *opening*. This is due to the fact that this operation will widen holes, eliminate small features and disconnect parts of the model that are connected by thin structures. The complement of this operation is a *closing*, which is a dilation followed by an erosion. This will close holes and connect previously disconnected parts of the model. There is a connection between the resolution at which a polygon model is scan-converted, and the distance parameter that is used by the morphological operators to simplify the volumetric description of the input polygonal model. The size of features such as tunnels and thin-structures grows larger in the volumetric description as the scan-conversion resolution is increased. Therefore the distance parameter used by the morphological operators to eliminate such features must also be increased. In our experiments, we have found that the polygon model should be scan-converted at approximatley 10 times the distance value used by the morphological operators. Notice that if erosion or an opening is performed on a thin-shelled volume model, the erosion will completely destroy the surface. This is another reason we require that our voxelization process not produce a thin-shelled volumetric representation.

VI. Polygonal Model Creation and Triangle Count Reduction

Now that we have seen how to remove small features using volumetric morphology, we turn our attention to converting the model back to polygons. There are two steps involved in this: isosurface extraction and polygon simplification.

A. Isosurface Extraction

To create a manifold polygonal model, we extract an isosurface from our volumetric representation of the model. We do this using the a modified version of the standard Marching cubes algorithm [21]. This algorithm works by examining the eight adjacent voxels at the corners of a cube. Using a threshold value, the corners of this cube are classified as being either inside or outside the surface. This classification scheme yields 256 possible configurations. A lookup table is used to generate triangles within a cube based on the configuration of its corners. The Marching Cubes algorithm can produce up to 11 triangles from each cube. The original algorithm proposed in [21] produces ill-formed isosurfaces in some cases. We use the modifications to Marching Cubes proposed in [?] to extract isosurfaces that are everywhere manifold.

As shown in Figure 1, there are two possible paths through our simplification pipeline: one where volumetric morphology is performed, and the other where we extract the isosurface directly after scan converting the input polygonal model into a volumetric representation. We omit the morphology stage if our goal is either to repair a polygonal model or to eliminate its interior detail. If no morphological operations are performed, then the resulting isosurface is smooth. This results from the fact that during scan conversion we use supersampling to obtain voxel values that vary between 0 and 1. On the other hand, because morphological operations act on binary volumes, the isosurfaces extracted after volume morphology have voxelization artifacts. We use Taubin's smoothing technique to reduce these artifacts [32]. Taubin uses a low-pass filter over the position of the vertices to create a new surface that is smoother than the original. One of the design goals of this smoothing method was that it be able to reduce the voxelization artifacts that are introduced during the voxelization stage. Gortler et al. use the same method in The Lumigraph to smooth polygonal models that they produce via isosurface extraction [12].

B. Triangle Count Reduction

The isosurface we extract usually has simpler topology than the input model. In addition, because the Marching cubes algorithm considers cubes in isolation, it frequently over tessellates the surface. These two properties of the isosurface allow us to drastically reduce its triangle count without degrading the model's quality. To achieve this end, we use Garland and Heckbert's polygon-based simplification method that is guided by a quadric error measure [8]. We use their method because it is efficient and produces high quality results. Garland and Heckbert use the planes passing through a vertex to estimate the amount of error introduced by an edge collapse. As discussed in the Section 3, their simplification process is based on a generalized form of the edge collapse operation called vertex pair contraction. A vertex merge may join together two vertices that do not share an edge, altering the topology of a model. Since we have already performed topological modifications using volumetric morphology, we only allow the merging of vertices that share an edge when using their simplification method. One consequence of using this edge-collapse based simplification method to reduce the triangle count of the isosurface is that as this technique does not gaurentee the preservation of the volume of a model, the simplified/repaired meshes are smaller in size than the original model.

When volume morphology is used to simplify the topology of an object, the resulting volume typically has no small holes, interior details, or tunnels. Thus all the polygons of the resulting isosurface can be used to represent the exterior surface of the object. This enables us to reduce the triangle count of a given model to much lower levels than would have been possible with the

original model.

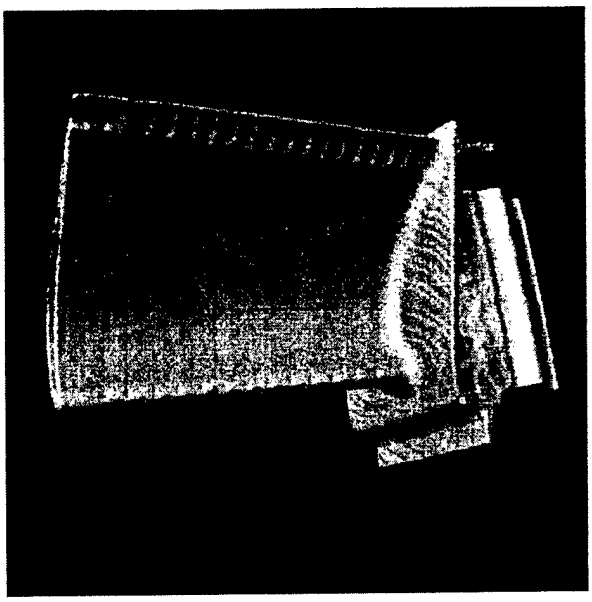
In order to maintain the color of a given model, we record the color values of the polygonal model during voxelization. These color values are associated with *surface* voxels. Surface voxels are those that intersect a polygon of the input model. During scan conversion, when we detect the intersection of a polygon with a voxel we record the color of the polygon and associate it with the voxel that the polygon intersects. These color values are then carried through the rest of the simplification pipeline. During the morphology stage, we process a volumetric representation of an object that has color by dividing the morphological operations into two steps. In the first step, a distance map is calculated based on the density values. This distance map is then used to perform either the opening or closing operation. After the morphological operations have been performed on the density values, a second distance map is calculated using the color voxels. This distance map is used to find the color of the nearest surface voxel to each voxel in the volume. The density volume and the color volume are combined to form the new volumetric representation of the input polygonal model. The next step in the simplification pipeline is isosurface extraction. Here, we extend the Marching Cubes algorithm to take into account color when generating triangles. In our version of the Marching Cubes algorithm, we simply interpolate the color values associated with voxels when we are generating triangles within a cube. These interpolated color values are then associated with the triangles that are generated. The final step in our simplification algorithm is triangle count reduction. In this step, we use Garland and Heckbert's newer simplification method that preserves the color of the original surface [9]. This extension to their previous algorithm simplifies meshes that have color associated with the vertices by adding the red, green and blue color coordinates to the quadric error measure.

VII. Results

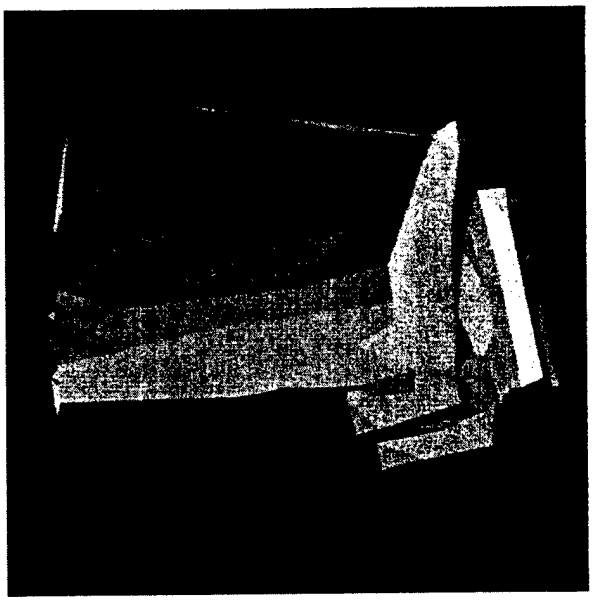
Figures 6-9 show the results of our algorithm on four models. The model of Figure 6 is a CT scan of a turbine blade. This model poses a challenge to most simplification algorithms due to its size, fine interior detail and complex topology (small holes, thin tunnels etc.) Many methods cannot simplify this model beyond a certain point because of its complex topology. Part (a) of Figure 6 shows the original model, part (b) is a volume rendering of the voxelized model, and part (c) shows the model after morphological closing and isosurface extraction. Parts (d) and (f) show the surface after the polygon reduction stage. For comparison to part (d), the effect of using Garland and Heckbert's quadric simplification method alone is shown in part (e). As can be seem from the images, our approach retains more details than Qslim (Garland and Heckbert's method) alone for the same number of faces in the simplified model. We note that there are many parameters for Garland and Heckbert's method. For fairness, we use the same choice of parameters to produce both parts (d) and (e). (Recall that Qslim is a component of our simplification pipeline.)

Figure 7 demonstrates our approach on a model of Al Capone. This hand-constructed model has fifteen interpenetrating parts. The head, arms and legs continue into the torso region, as can be seen by the darker regions in the wireframe rendering of part (d). In addition, this model also has color, which is an attribute that we preserve during the simplification and repair. For this model we do not perform any morphology on the model, but instead recover an isosurface after the model has been scan-converted. The resulting isosurface has no intersecting parts and is everywhere manifold. This guarantees that the different body parts will not become disjoint from one another when the object is simplified, as shown in parts (c) and (f). After the isosurface was extracted, we reduced its triangle count to match that of the input model. The resulting surface can be seen in parts (b) and (e) of the figure.

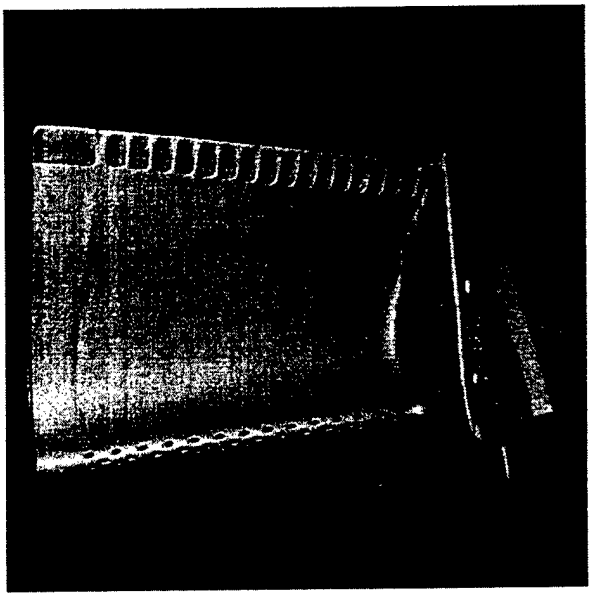
Figure 8 demonstrates simplification of a car motor. This model contains many degeneracies such as T-joints, zero-area faces and non-manifold vertices. In addition, the motor model has interesting topology in that there are a number of parts connected by thin structures and a



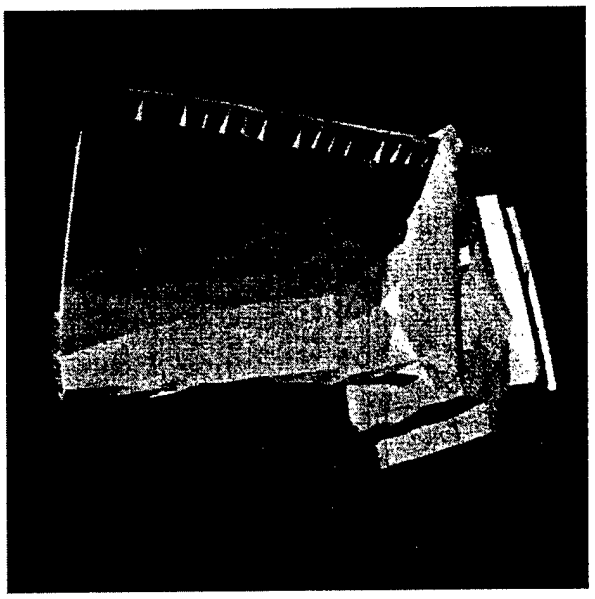
4 c. Isosurface after Morphological closing 578,098 Faces



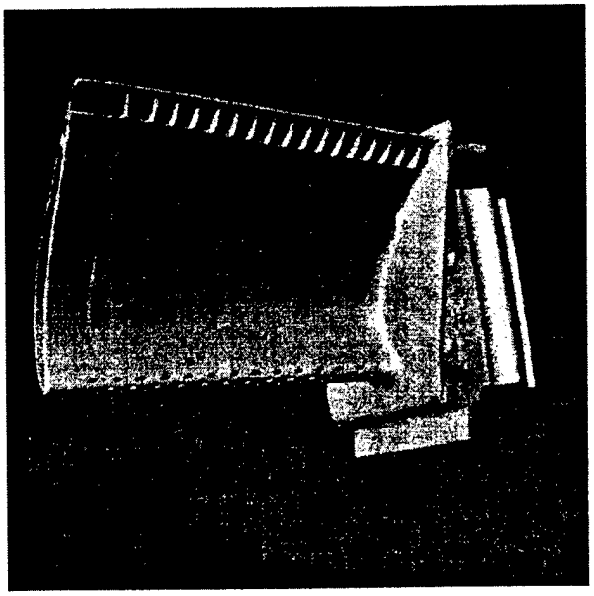
4 f. Simplified Model 500 Faces



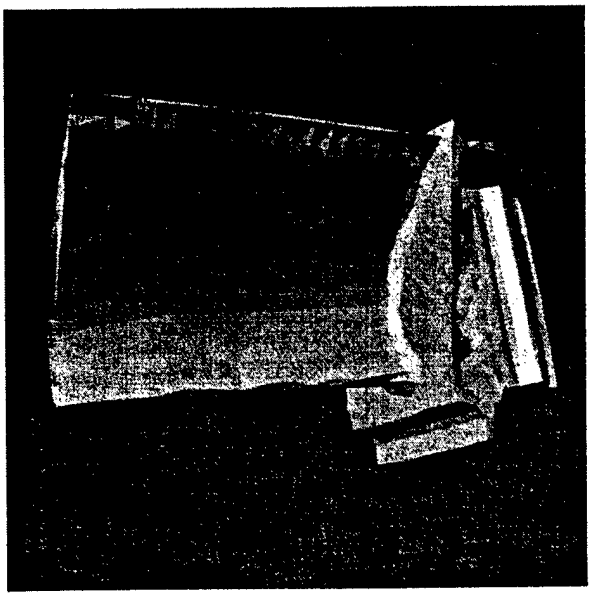
4 b. Voxelized Model (Volume rendered) 643x300x382 Voxels



4 e. Model Simplified by Qslim 2,200 Faces



4 a. Origina Turbine Blade Model 1,726,892 Faces



4 d. Simplified Model 2,200 Faces

Fig. 6.

5 a. Original Al Capone Model
13,476 Faces

5 b. Repaired Model
13,476 Faces

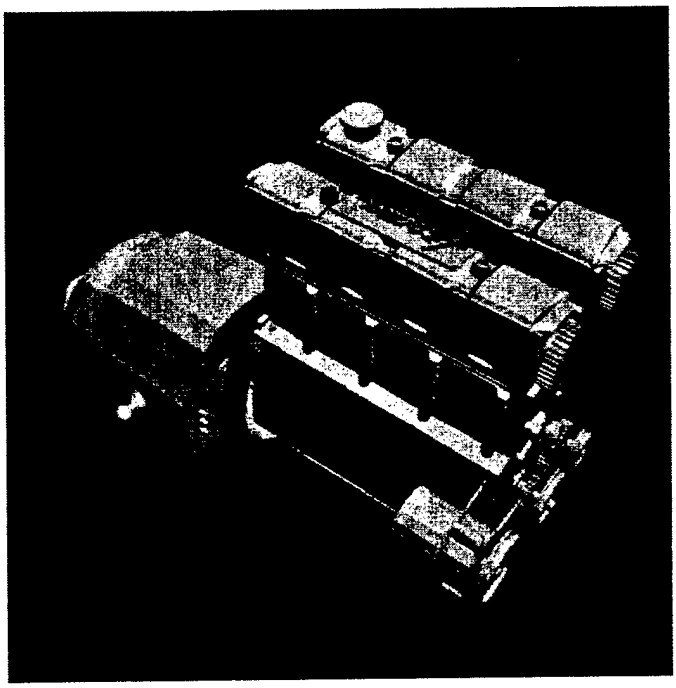
5 c. Simplified Model
1,721 Faces

5 d. Wireframe of Original Model (shows intersecting parts)
13,476 Faces

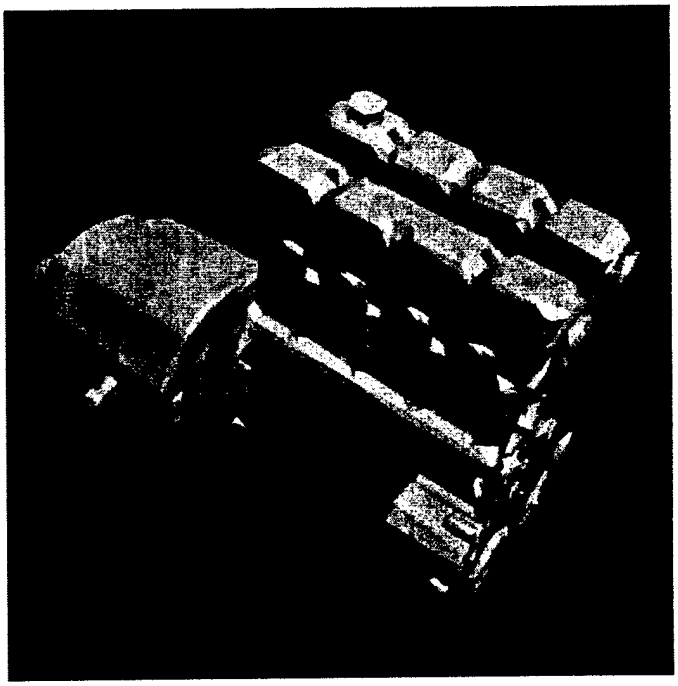
5 e. Wireframe of Repaired Model (shows manifold surface)
13,476 Faces

5 f. Simplified Model
860 Faces

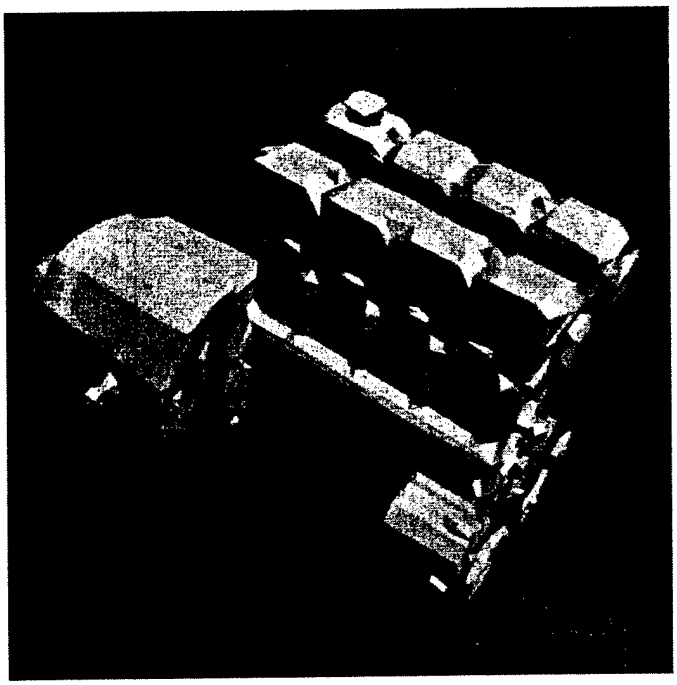
Fig. 7.



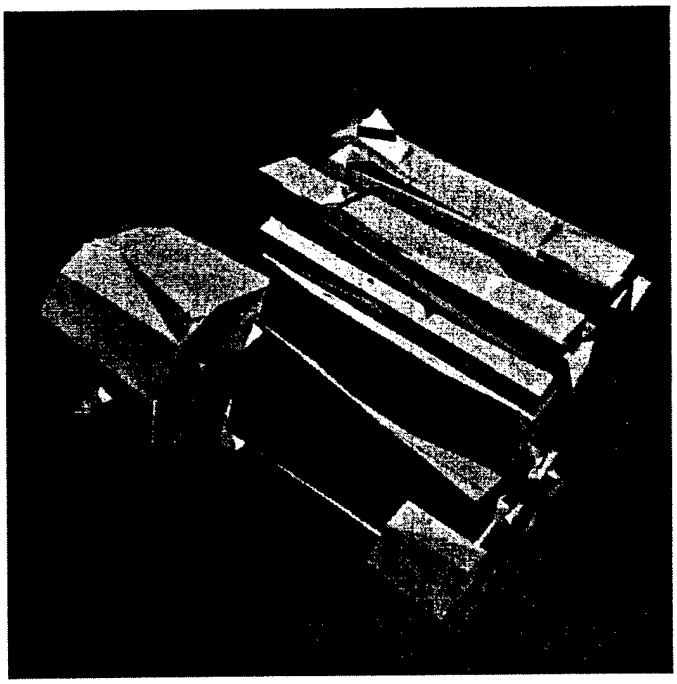
6 a. Original Motor Model
140,113 Faces



6 b. Simplified Model
5,000 Faces



6 c. Simplified Model
3,300 Faces



6 d. Model Simplifiedby Qslim
3,300 Faces

Fig. 8.

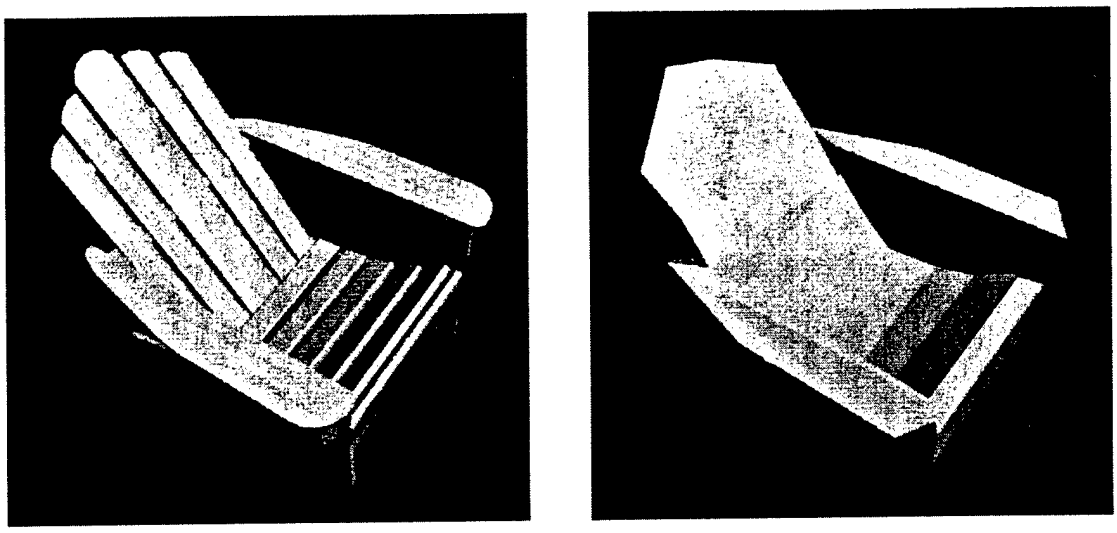


Fig. 9. (a) Original Chair Model (3,261 faces), (b) Simplified Model (170 faces).

lot of interior detail. Again in the comparison with Qslim, our method retains more detail at comparable levels of complexity.

Figure 9 shows simplification of a model chair. This model demonstrates the topology modification capabilities of our algorithm. Our approach closes the holes that are present in the back and the seat of the model. We do not know of any other simplification algorithms that would smoothly join the slats of the chair into a single component and produce a manifold surface.

The results shown in these figures illustrate the improvement that our morphological operators make when used in conjunction with Garland and Heckbert's polygon-based simplification approach. We believe that similar benefits will result if our morphological technique is used in conjunction with any other polygon-based method. Many polygon-based simplification methods are affected during the later stages of simplification by small features that were present in the original model, even if those features have been removed by the simplification process. The memory of such small features are often used in distance measures that help guide the simplification process, such as are used in [8, 15]. Other simplification methods such as [20, 31] do not measure distances based on the original geometry, but are still influenced by the original topological features such as small holes. The benefit of the morphological stage of our pipeline is that it produces models in which the small features are completely absent. When morphological changes are performed before polygonal simplification, the polygon-based simplification stage never needs to be concerned with the small features in any way. The polygon-based method is never penalized for creating a surface that is distant from the small features because it never has knowledge of these small features. This results in better simplified models.

Table II shows the timing results for each stage of the simplification pipeline. Table III contains the sizes of the different representations of a model as it moves through the pipeline. To collect timing statistics that reflect all of the stages of the pipeline, we performed an extreme amount of simplification on all of the models– simplifying each model down to two hundred faces. (Note that these 200 face models are not the models shown in the figures.) When the input model contains a large number of faces, then voxelization is the most time consuming stage of the pipeline. This is evident from the timing results of the turbine blade model where voxelization accounted for 78% of the total time required to simplify the model. In other cases, most of the stages in the simplification pipeline took about the same amount of time. By way of comparison, we note that it took Qslim 4:08 minutes to simplify the blade down to 200

Simplification Pipeline Timing (minutes)						
	Voxelization	Morphology	Isosurface Extraction	Smoothing	Triangle Count Reduction	Total
Blade	19.06 (78 %)	1.03 (4 %)	0.76 (3 %)	1.9 (8 %)	1.6 (7 %)	24.35 (100 %)
Motor	2.6 (29 %)	*	0.63 (7 %)	3.63 (41 %)	2.02 (23 %)	8.88 (100 %)
Al	9.7 (47 %)	*	1.7 (8 %)	3.4 (17 %)	5.7 (28 %)	20.46 (100 %)
Chair	0.9 (25 %)	0.9 (25 %)	0.4 (11 %)	0.9 (25 %)	0.5 (14 %)	3.6 (100 %)

Table II. This table shows the timing information for each stage of our simplification pipeline. All the timing measurements were taken on a 4 processor SGI Onyx with 1 Gigabyte of main memory.

Dimensions of Models				
	Original Model (# faces)	Volume Representation	Isosurface (# faces)	Simplified Model (# faces)
Blade	1,729,892	268x128x161	578,098	200
Motor	140,113	386x256x197	177,158	200
Al	13,476	107x256x276	489,552	200
Chair	1,087	153x128x150	32,750	200

Table III. This table contains the sizes of the polygonal models as they move through the simplification pipeline. Table II above, gives the timing information for these model sizes.

polygons and 15 seconds to simplify the motor model. As reported in [20], Qslim is one of the fastest published simplification algorithms. Therefore it is not surprising to see it perform so well on these models. However, we believe that the increased quality of the simplified models that are obtained by performing the topology modification in the volumetric domain versus running Qslim on the original polygonal models justifies the additional time required by our simplification pipeline.

VIII. Conclusion and Future Work

In this paper we have introduced a new surface simplification technique that makes use of morphological operations in the volume domain to simplify the topology of an object. Specific advantages of the simplification method include:

- Performs controlled topology modification, allowing extreme simplification.
- Accepts arbitrary collections of polygons as input.
- Produces manifold meshes as output.
- Preserves surface attributes such as color.

A benefit of converting the input polygonal models into volumes is that we can repair a number of degeneracies in polygonal models. This model repair method is simple to program and it produces clean models that are everywhere manifold.

There are several possible avenues for future research. The erosion operator eliminates thin surfaces, thus large thin parts of the model can be eliminated resulting in a large perceptual error. For this reason we always perform dilation before erosion (which together are an opening), but we are investigating possible solutions to this issue. One are of future research would be to extend the morphological operators so that the distance parameter would vary in different parts of the surface. This would allow models that have thin structures to be processed by the erosion operator. Another future direction would be to extend other 2D image processing techniques into 3D, possibly resulting in other new and useful methods of manipulating volumetric models.

IX. Acknowledgements

We thank the many people at Georgia Tech who have helped and encouraged us. This work was funded by ONR grant N00014-97-1-0223.

References

[1] Barequet, G. and Kumar, S. Repairing CAD Models. In *Proceedings of IEEE Visualization '97*, October 19-24, 1997, pp. 363–370.

[2] Barequet, G. and Sharir, M. Filling Gaps in the Boundary of a Polyhedron. *Computer Aided Geometric Design*, Vol. 12, No. 2, 1995, pp. 207–229.

[3] Bohn, J.H. and Wozny, M.J. A Topology-Based Approach for Shell-Closure. In *Geometric Modeling for Product Realization*, Edited by P.R. Wilson et al, North-Holland, 1993, pp. 297–319.

[4] Danielsson, P. Euclidean Distance Mapping. *Computer Graphics and Image Processing*, Vol. 14, 1980.

[5] El-Sana, J. and Varshney, A. Controlled Simplification of Genus for Polygonal Models. In *Proceedings of the IEEE Visualization.*, August, 1997.

[6] El-Sana, J. and Varshney, A. Topology Simplification for Polygonal Virtual Environments. In *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 2, June 1998, pp 133 - 144.

[7] Erikson, C. Error Correction of a Large Architectural Model: The Henderson County Courthouse. Technical Report TR95-013, Dept. of Computer Science, University of North Carolina at Chapel Hill, 1995.

[8] Garland, M. and Heckbert, P. S. Surface Simplification using Quadric Error Metrics. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 209–216.

[9] Garland, M. and Heckbert, P. S. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *IEEE Visualization '98* Proceedings, October 1998, pp. 263–269.

[10] Gelder, Allen Van and Wilhelms, Jane Topological Considerations in Isosurface Generation *IEEE Transactions on Graphics*, Vol. 13, No. 4, November 1994, pp. 337-375.

[11] Shade, Jonathan, Gortler, Steven. J, He, Li-wei , and Szeliski, Richard Layered Depth Images Proceedings of SIGGRAPH 98. In *Computer Graphics* Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pp. 209–216.

[12] Gortler, S.J., Grzeszczuk, R., Szeliski, R., and Cohen. M.F. The Lumigraph. In *SIGGRAPH '96 Proc.*, August, 1996,pp. 43–54.

[13] Gueziec, A., Taubin, G., Lazarus, F., and Horn. W. Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching. In *Proceedings IEEE Visualization 1998.*, Research Triangle Park, North Carolina, October 18-23, 1998, pp. 383–390.

[14] He, L. Hong, Kaufman, A.E., Varshney, A. and Wang, S. Voxel-Based Object Simplification, *IEEE Visualization '95 Proceedings*, October 1995.

[15] Hoppe, H. Progressive Meshes. Proceedings of SIGGRAPH 96. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 99–108.

[16] Huang, J., Yagel, R., Filippov, V. and Kurzion, Y. An Accurate Method for Voxelizing Polygonal Meshes. In *Symposium of Volume Visualization*, 1998.

[17] Jain, A. *Fundamentals of Digital Image Processing*, Englewood Cliffs, New Jersey, Prentice-Hall Inc., 1989.

[18] Khorramabdi, D. A Walk Through the Planned CS Building. Technical Report UCB/CSD 91/652, Computer Science Department, University of California at Berkeley, 1991.

[19] Levoy, M. A Hybrid Ray Tracer for Rendering Polygon and Volume Data *IEEE Computer Graphics and Applications*, Vol. 10, No. 2, March, 1990.

[20] Lindstrom, P. and Turk, G. Evaluation of Memoryless Simplification. In *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, No. 2, April-June 1999, pp 98 - 115.

[21] Lorensen, W.E. and Cline, H.E. Marching cubes: A high resolution 3-d surface construction algorithm. Proceedings of SIGGRAPH 87. In *Computer Graphics*, July, 1987.

[22] Low, Kok-Lim and Tan, Tiow-Seng Model Simplification using Vertex-Clustering. In *Interactive 3D Graphics*, Providence, Rhode Island, April 27-30, 1997, pp. 75–81.

[23] Luebke, D. and Erikson, C. View-Dependent Simplification of Arbitrary Polygonal Environments. Proceedings of SIGGRAPH 97. In *Computer Graphics*, August 1997.

[24] Marschner, S.R. and Lobb, R.J. An Evaluation of Reconstruction Filters for Volume Rendering. In *IEEE Proceedings of Visualization '94*, Washington, D.C., October 17-21, 1994, pp. 100–107.

[25] Morvan, S.M. and Fadel, G.M. IVECS: An Interactive Virtual Environment for the Correction of .STL files. In *Conference on Virtual Design*, University of California at Irvine, August 1996.

[26] Murali, T. M. and Funkhouser, T. Consistent Solid and Boundary Representations from Arbitrary Polygonal Data. In *Proceedings 1997 Symposium on Interactive 3D Graphics*, Providence, Rhode Island, April 27-30, 1997, pp. 155–162.

[27] Rossignac, J. and Borrel, P. Multi-resolution 3D approximations for rendering complex scenes. *Modeling in Computer Graphics: Methods and Applications*, June, 1993.

[28] POPOVIC, J. and HOPPE, H. Progressive Simplicial Complexes. Proceedings of SIGGRAPH 97. In *Computer Graphics*, August 1997.

[29] SCHROEDER, W. J., ZARGE, J. A., and LORENSEN, W. E. Decimation of Triangle Meshes. Proceedings of SIGGRAPH 92. In *Computer Graphics*, July 1992, pp. 65–70.

[30] SCHROEDER, W. J. A Topology Modifying Progressive Decimation Algorithm. In *IEEE Visualization '97* Proceedings, October 1997, pp. 205–212.

[31] SCHROEDER, W. J. and LORENSEN, W. E. Implicit Modeling of Swept Surfaces and Volumes, In *Proceedings of Visualization '94*, Washington, D.C., October 17-21, 1994, pp. 40–45.

[32] TAUBIN, G. A Signal Processing Approach To Fair Surface Design. Proceedings of SIGGRAPH 95. In *Computer Graphics*, July 1995, pp. 351-358.

[33] WANG, S. and KAUFMAN, A.E. Volume Sampled Voxelization of Geometric Primitives, *IEEE Visualization '93 Proceedings, IEEE Computer Society Press* October, 1993.

Volumetric Representation and Manipulation of Geometric Models

Principal Investigator: Greg Turk, Georgia Institute of Technology

I. Introduction and Project Goals

This is the final report of the work that was funded by the ONR contract number N00014-97-1-0223.

The goal of this research project was to investigate new methods of representing and manipulating three-dimensional geometric models using volumetric techniques. Three sub-areas were particular targets for these investigations: 1) explore ways of extending the kinds of models that can be represented volumetrically, 2) create multiresolution models using volume techniques, and 3) perform shape transformation using a volumetric framework. Several application areas that are important to ONR should benefit from this research, including:

- Flight simulators
- Walkthroughs of large buildings and vehicles
- Comparison between model organs and possibly damaged tissue
- Reconstruction of medical data from CT and MRI slices

The remainder of this report details the research successes in each of the three sub-areas, as well as additional products of the project that were not foreseen in the original proposal.

II. Representing Models as Volumes

One of the earliest aspects of the research into volumetric methods that we explored was the issue of what 3D models could be represented as volumes. In particular, many 3D models do not originally come in a volumetric form, but rather are represented by other methods such as polygons. Over the course of our investigations, we came up with two distinct methods of converting polygonal models into volumetric representations.

Our first method of converting polygonal models to volumetric models is based on creating an implicit function from the surface points on a model. First, a relatively sparse set of constraint points are identified on the polygonal surface, along with some surface normal constraints. These constraints are then used to create an implicit function using radial basis functions that naturally minimize the curvature of the function being created. The implicit surface of this function is a first approximation to the original model. Places where the two models differ are identified, and more constraints are placed at these locations. Another implicit surfaces is created, this

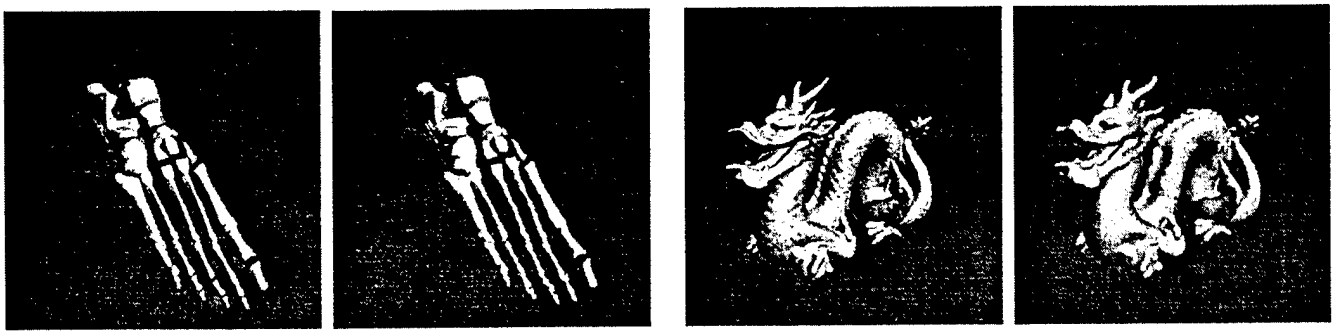
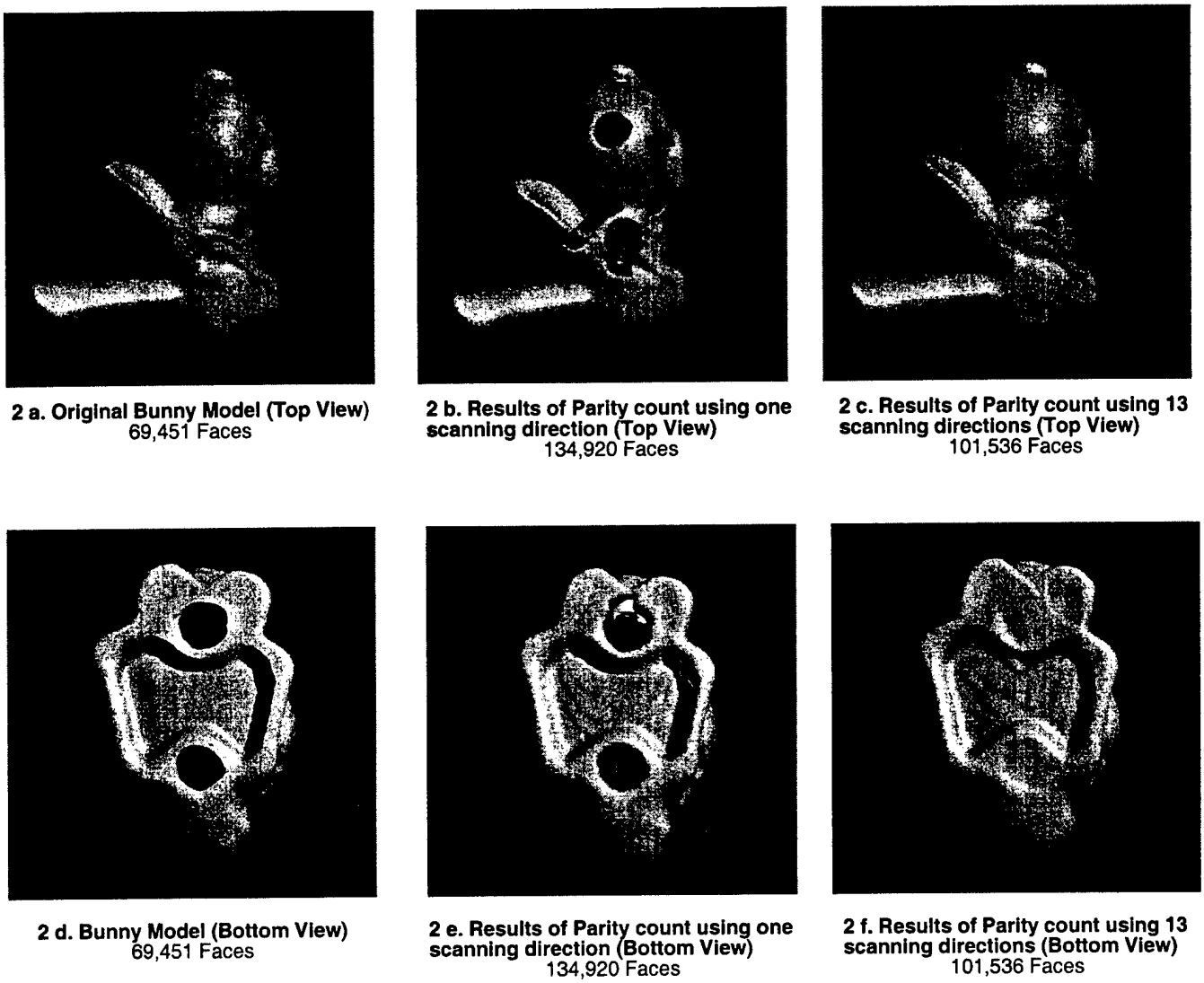


Fig. 1. Conversion from polygons to implicit volume models.



2 a. Original Bunny Model (Top View) 69,451 Faces

2 b. Results of Parity count using one scanning direction (Top View) 134,920 Faces

2 c. Results of Parity count using 13 scanning directions (Top View) 101,536 Faces

2 d. Bunny Model (Bottom View) 69,451 Faces

2 e. Results of Parity count using one scanning direction (Bottom View) 134,920 Faces

2 f. Results of Parity count using 13 scanning directions (Bottom View) 101,536 Faces

Fig. 2. Converting polygonal models with holes to a volumetric model.

time that is a better match to the original polygonal data. This process repeats until the user-supplied tolerance values are met. Figure 1 (far left) shows an original model of the bones in a human's foot, and the second image in this figure shows the implicit surface representation of the same model. The third image in this figure is a much more detailed polygonal model, and the far right image is the implicit representation of this surface. The input model is far more complex (over one million polygons) than previous methods have been able to use in creating an implicit volumetric representations from polygons. This method is described in [4], and is a technical report and is also currently in review for the IEEE Transactions on Visualization and Computer Graphics.

Our second method of converting polygon models to voxels was based on the idea of counting the number of intersections that a ray makes with the surface of a model. If a ray from a given point $\mathbf{P}$ intersects the model's surface an even number of times, then $\mathbf{P}$ is probably outside the object. If there are an odd number of intersections, $\mathbf{P}$ is likely to be inside. Using polygon scan conversion, we in effect cast many rays from many directions through the model, and collect together votes on whether a given point is likely to be interior. This method allows us to avoid problems such as cracks and holes in the model that cause other voxelization methods to fail. For example, the bunny model shown in Figure 2a and 2d has a number of holes in the bottom.

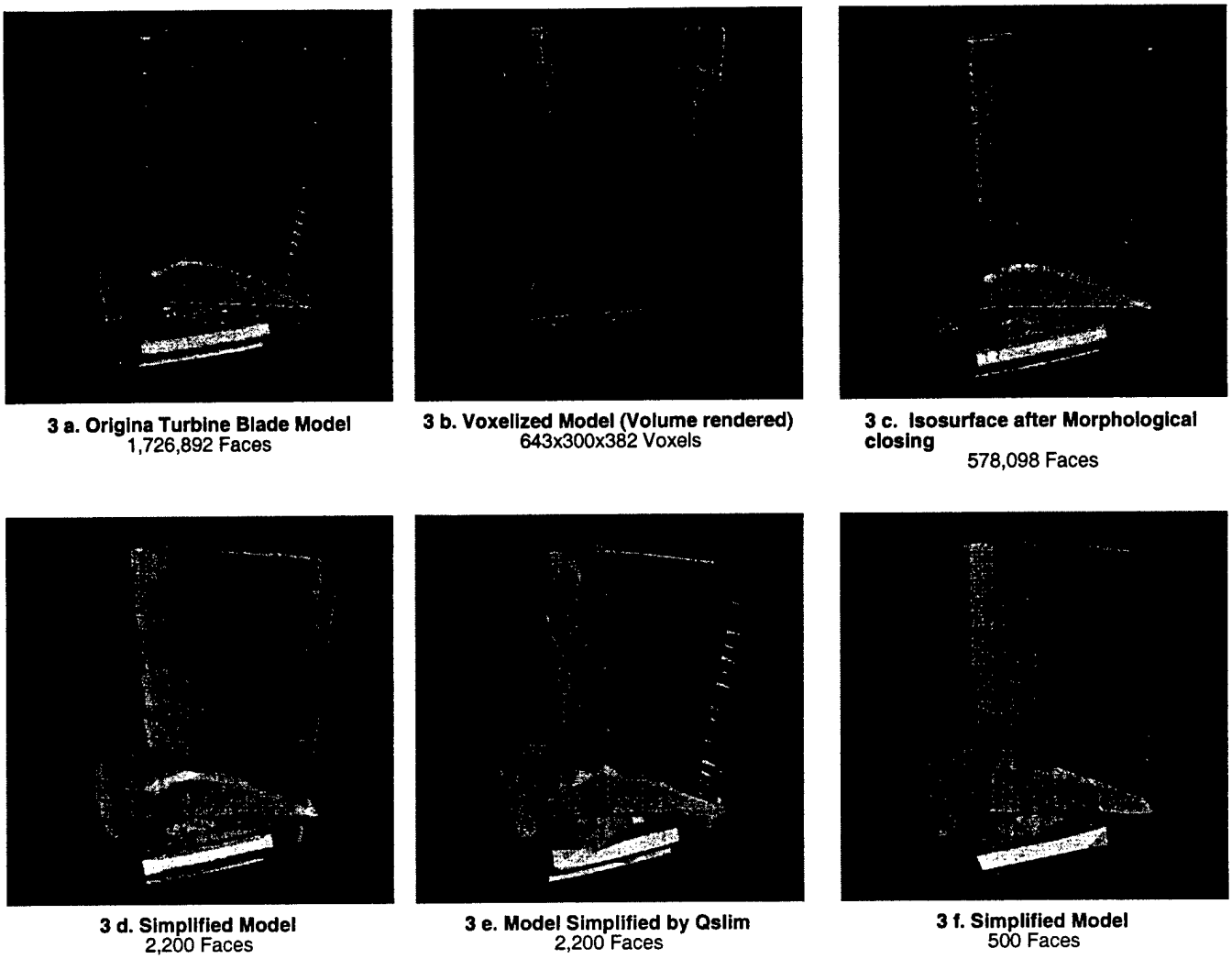


Fig. 3. Simplification of an engine turbine blade model.

If we use information from just a single direction, then these holes appear in the volumetric version of the model (Figure 2b and 2e). If, on the other hand, we collect ray intersection information from several directions, the holes are disregarded and the final model (Figures 2c and 2f) have no holes. This approach was described in [2], which is accepted for publication in the IEEE Transactions on Visualization and Computer Graphics.

The two main areas that stand to benefit from these new techniques are medical diagnosis and mechanical design. The implicit volumetric models are far smaller than the polygonal models, and medical data on wounded personnel could be rapidly transmitted to a medical expert at a remote location. Many CAD models of mechanical components are currently represented in polygonal form. Instead of painstakingly converting these models to volumetric representations by hand, our methods could be used to convert these models directly. Because of the robustness of our conversion method, even models that have significant numbers of geometric degeneracies (t-junctions, holes, cracks, interpenetrating parts) can be converted to volumes.

III. Multiresolution Models

There are many published methods of creating multiresolution models, but most or all of these methods begin to break down at very low resolutions. When a complex model is being viewed from a distance (e.g. when simulating a fly-over a convoy of trucks), the exact details of each model is not important, but the frame-rate of the simulator is critical. We have created a

level-of-detail algorithm that produces very good models even at extremely low polygon counts. Our method relies on volumetric operations, and in particular on 3D morphology.

Our approach is to convert a given model into a volumetric representation (using methods we described earlier), and then to perform volume morphology on these models. The morphological operation that we use is the "close" operator, which closes up small holes and fuses together objects that are extremely near to one another. These small topological changes are not visible when the model is far from the viewer, but they allow these objects to be greatly simplified. Once the morphological operation has been performed, the model is then converted back into polygons (using marching cubes), and then is simplified using traditional polygon simplification methods. The topological simplification that the closing operator changes the nature of the models so that much more aggressive simplification may be performed and still result in visually high-fidelity models. Figure 3 illustrates this form of simplification on a turbine blade model that begins with approximately 1.7 million polygons. Even the 500 polygon model retains much of the shape of the original object. This method is described in [2].

IV. Shape Transformation

The goal of shape transformation is, given two shapes, to produce a sequence of shapes that smoothly transitions from the first shape to the second one. Our approach to shape transformation was to construct a four-dimensional volumetric function that encapsulated the entire sequence of shapes. Two of the 3D slices of this 4D function (at $w = 0$ and $w = 1$) would produce the given shapes, and intermediate slices ($0 \leq w \leq 1$) give intermediate shapes in the sequence. The manner in which we create this 4D function is to use a higher-dimensional generalization of thin-plate interpolation, a method usually used to solve 2D scattered data interpolation problems. The constraints given to this interpolation approach come from the

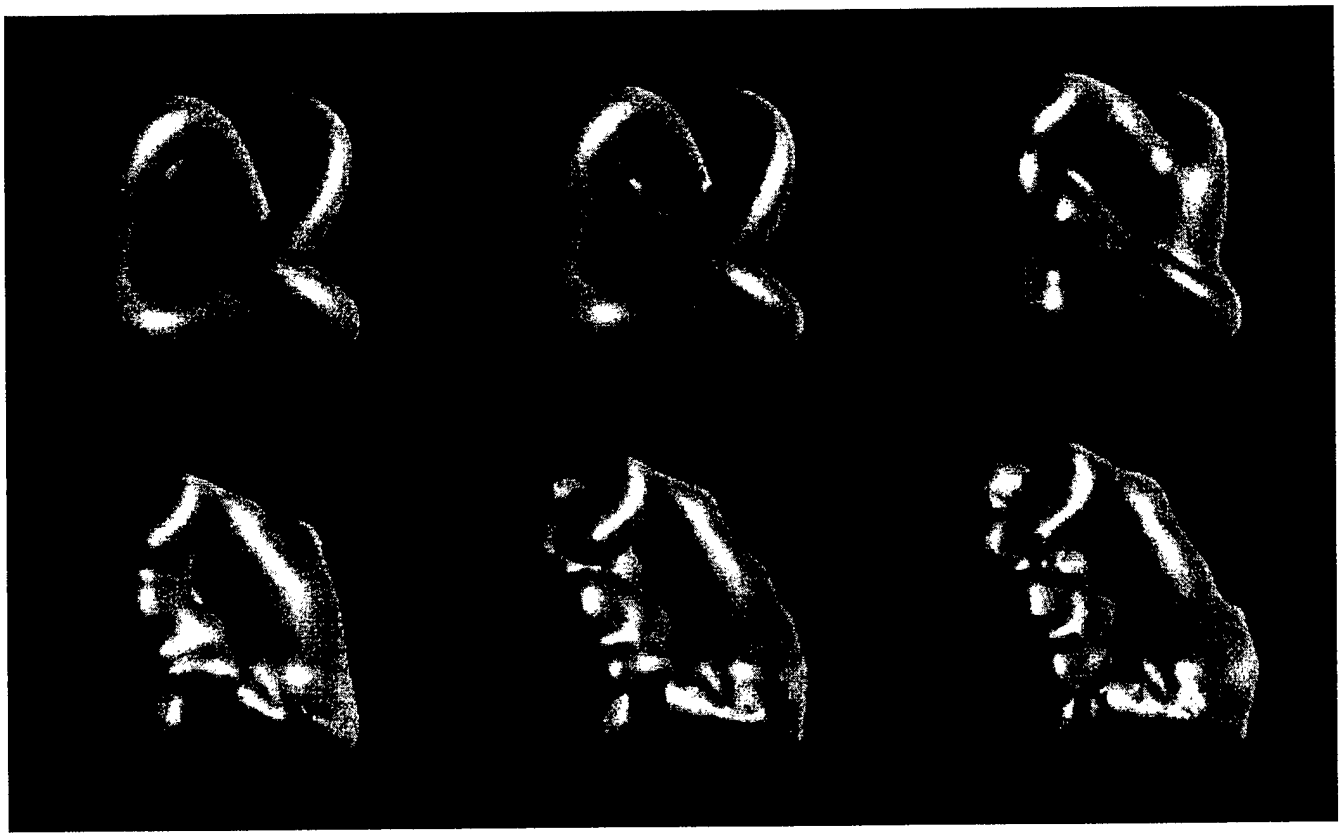


Fig. 4. Shape transformation between knot and human fist, showing that topology changes are easy to perform.

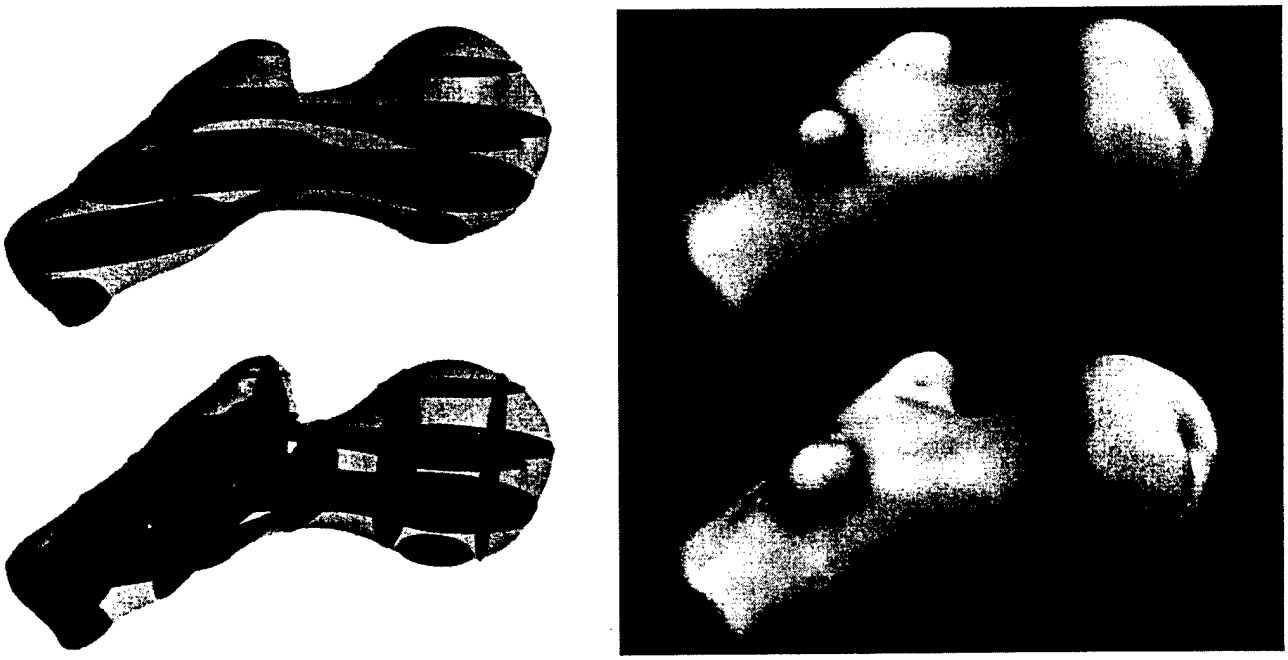


Fig. 5. Surface reconstruction of hip joint from parallel (top) and non-parallel slices (bottom).

vertices and the surface normals of the two given objects. This method produces results such as that shown in Figure 4, in which a shape transformation sequence is shown between models with different topologies. Our method was published in the SIGGRAPH 1999 conference proceedings [3].

Shape transformation has several important applications, including creation of smooth joins between surfaces for mechanical design and also for the creation of surfaces from CT and MRI medical data. The 2D shape interpolation problem is just another form of contour interpolation from 2D slices. Previous methods of contour interpolation require that the slices be in planes that are all parallel to one another. Our own method allows the planes to be non-parallel and even allows the planes to intersect one another. Figure 5 shows an example of this, where the slices are of a person's hip bone. This opens up the possibility of acquiring medical data in a much less restricted environment, such as a hand-held scanning device that may be taken to remote locations.

V. Visualization of Hidden Surfaces

In addition to the three sub-areas described above, our research also yielded a new visualization technique that we did not anticipate at the start of our research. We have found that volumetric techniques that are similar to those we used to convert polygons to voxels could also be used to identify those portions of a model that are hidden from view. This provides an automatic way in which to tag hidden surfaces, allowing the outer surfaces to be marked as translucent in order to show the interior of an object. Our technique consists of converting a polygonal model into a volumetric representation, morphologically closing holes and then identifying interior and exterior portions of the surface using ray voting. Figure 6 illustrates this technique with a motor. The left portion of this figure shows an opaque view of the model, and the right image shows the visualization of this model using our interior/exterior classification method. This new method was published in the Visualization 2000 conference proceedings [1]. This automatic method of visualization is particularly applicable to examining medical data and mechanical parts for CAD.

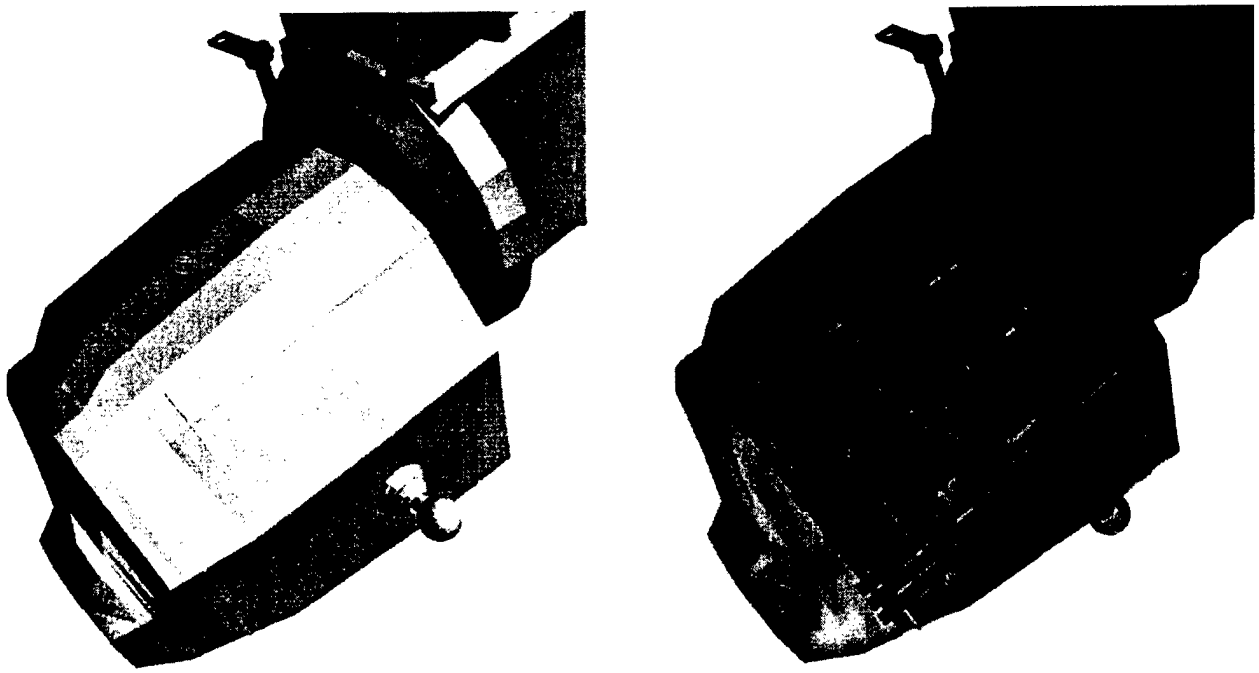


Fig. 6. Visualization of hidden surfaces in a motor.

VI. Conclusion

Our research in volumetric representations has led to several new computer graphics techniques that are likely to be useful for a number of applications. Particular applications that are of interest to ONR include visualization and surface reconstruction for medicine, design and visualization of CAD models, and simplification of models for flight and walkthrough simulators.

References

[1] Nooruddin, F. S. and Greg Turk "Interior/Exterior Classification of Polygonal Models," *Visualization 2000 Conference Proceedings*, Salt Lake City, Utah, October 2000.

[2] Nooruddin, F. S. and Greg Turk "Simplification and Repair of Polygonal Models Using Volumetric Techniques," to appear, *IEEE Transactions on Visualization and Computer Graphics.*

[3] Turk, Greg and James O'Brien, "Shape Transformation Using Variational Implicit Functions," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 99), August 1999, pp. 335–342.

[4] Yngve, Gary and Greg Turk "Robust Creation of Implicit Surfaces from Polygonal Meshes," in review, *IEEE Transactions on Visualization and Computer Graphics.*

Shape Transformation Using Variational Implicit Functions

Greg Turk James F. O'Brien

Georgia Institute of Technology

Abstract

Traditionally, shape transformation using implicit functions is performed in two distinct steps: 1) creating two implicit functions, and 2) interpolating between these two functions. We present a new shape transformation method that combines these two tasks into a single step. We create a transformation between two N-dimensional objects by casting this as a scattered data interpolation problem in $N+1$ dimensions. For the case of 2D shapes, we place all of our data constraints within two planes, one for each shape. These planes are placed parallel to one another in 3D. Zero-valued constraints specify the locations of shape boundaries and positive-valued constraints are placed along the normal direction in towards the center of the shape. We then invoke a variational interpolation technique (the 3D generalization of thin-plate interpolation), and this yields a single implicit function in 3D. Intermediate shapes are simply the zero-valued contours of 2D slices through this 3D function. Shape transformation between 3D shapes can be performed similarly by solving a 4D interpolation problem. To our knowledge, ours is the first shape transformation method to unify the tasks of implicit function creation and interpolation. The transformations produced by this method appear smooth and natural, even between objects of differing topologies. If desired, one or more additional shapes may be introduced that influence the intermediate shapes in a sequence. Our method can also reconstruct surfaces from multiple slices that are not restricted to being parallel to one another.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surfaces and object representations

Keywords: Shape transformation, shape morphing, contour interpolation, implicit surfaces, thin-plate techniques.

1 Introduction

The shape transformation problem can be stated as follows: Given two shapes A and B, construct a sequence of intermediate shapes so that adjacent pairs in the sequence are geometrically close to one another. Playing the resulting sequence of shapes as an animation would show object A deforming into object B. Sequences of 2D shapes can be thought of as slices through a 3D surface, as shown in Figure 1. Shape transformation can be performed between objects of any dimension, although 2D and 3D shapes are by far the most common cases. Shape transformation has applications in medicine, computer aided design, and special effects creation. We give an overview of these three applications below.

One important application of shape transformation in medicine is contour interpolation. Non-invasive imaging techniques often collect data about a patient's internal anatomy in "slices" of a particular size such as 512×512 samples. Usually many fewer slices are taken along the third dimension so that a resulting volume might, for example, be sampled at $512 \times 512 \times 30$ resolution. To reconstruct a 3D model of a particular organ, the samples are segmented to create shapes (contours) within the slices. Intermediate shapes are then created between slices in the sparsely sampled dimension. The reconstructed 3D object is formed by stacking together the original and the interpolated contours. This is an example of 2D shape transformation.

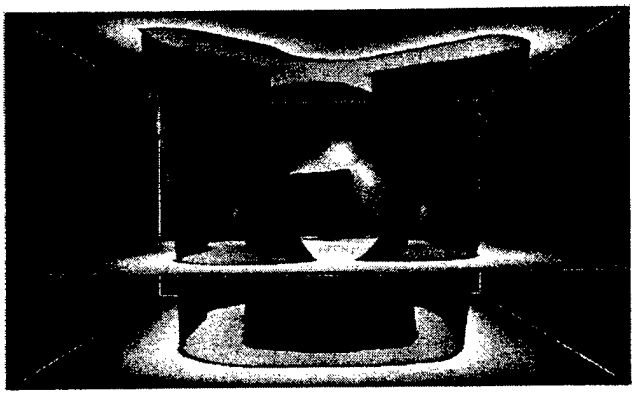


Figure 1: Visualization of transformation between X and O shapes. Top and bottom planes contain constraints for the two shapes. Translucent surface is the isosurface of a 3D variational implicit function, and slices through it give intermediate shapes.

Shape transformation can also be a useful tool in computer aided geometric design. Consider the problem of creating a join between two metal parts with different cross-sections. It is important for the connecting surface to be smooth because those places with sharp ridges or creases are the locations that are most likely to form cracks. The intermediate surface joining the two parts can be created using shape transformation, much in the same way as with contour interpolation for medical imaging. Because of the smoothness properties of variational interpolation methods, we consider them a natural tool to explore for shape transformation in CAD.

Finally, animated shape transformations have been used to create dramatic special effects for feature films and commercials. One of the best-known examples of shape transformation is in the film *Terminator 2*. In this film, a cyborg policeman undergoes a number of transformations from an amorphous and highly reflective surface to various destination shapes. 2D image morphing would not have accurately modeled the reflection of the environment off the surface of the deforming cyborg, hence tailor-made 3D shape transformation programs were used for these effects [9].

In this paper we use variational interpolation in a new way to produce high-quality shape transformations that may be used for any of the previously mentioned applications. Our method allows a user to control the transformation in several ways, and it is general enough to produce transformations between shapes of any topology.

2 Previous Work

Most shape transformation techniques can be placed into one of two categories: parametric correspondence methods and implicit

turk@cc.gatech.edu, job@acm.org.

function interpolation. Parametric methods are typically faster to compute and require less memory because they operate on a lower-dimensional representation of an object than do implicit function methods. Unfortunately, transforming between objects of different topologies is considerably more difficult with parametric methods. Parametric approaches also can suffer from problems with self-intersecting surfaces, but this is never a problem with implicit function methods. Techniques that use implicit function interpolation gracefully handle changes in topology between objects and do not create self-intersecting surfaces.

A parametric correspondence approach to shape transformation attempts to find a "reasonable" correspondence between pairs of locations on the boundaries of the two shapes. Intermediate shapes are then created by computing interpolated positions between the corresponding pairs of points. Many shape transformation techniques have been created that follow the parametric correspondence approach. One early application of this approach is the method of contour interpolation described by Fuchs, Kedem and Uselton [10]. Their method attempts to find an "optimal" (minimum-area) triangular tiling that connects two contours using dynamic programming. Many subsequent techniques followed this approach of defining a quality measure for a particular correspondence between contours and then invoking an optimization procedure [22, 25]. There have been fewer examples of using parametric correspondence for 3D shape transformation. One quite successful 3D parametric method is the work of Kent et al. [17]. The key to their approach is to subdivide the polygons of the two models in a manner that creates a correspondence between the vertices of the two models. More recently, Gregory and co-workers created a similar method that also allows a user to specify region correspondences between meshes to better control a transformation [12].

An entirely different approach to shape transformation is to create an implicit function for each shape and then to smoothly interpolate between these two functions. A shape is defined by an implicit function, $f(\mathbf{x})$, as the set of all points $\mathbf{x}$ such that $f(\mathbf{x}) = 0$. For contour interpolation in 2D, the implicit function can be thought of as a height field over a two-dimensional domain, and the boundary of a shape is the one-dimensional curve defined by all the points that have the same elevation value of zero. An implicit function in 3D is a function that yields a scalar value at every point in 3D. The shape described by such a function is given by those places in 3D whose function value is zero (the isosurface).

One commonly used implicit function is the *inside/outside function* or *characteristic function*. This function takes on only two values over the entire domain. The two values that are typically used are zero to represent locations that are outside and one to signify positions that are inside the given shape. Given a powerful enough interpolation technique, the characteristic function can be used for creating shape transformations. Hughes presented a successful example of this approach by transforming characteristic functions into the frequency domain and performing interpolation on the frequency representations of the shapes [15]. Kaul and Rossignac found that smooth intermediate shapes can be generated by using weighted Minkowski sums to interpolate between characteristic functions [16]. They later created a generalization of this technique that can use several intermediate shapes to control the interpolation between objects [24]. Using a wavelet decomposition of a characteristic function allowed He and colleagues to create intermediates between quite complex 3D objects [13].

A more informative implicit function can provide excellent intermediate shapes even if a simple interpolation technique is used. In particular, the *signed distance function* (sometimes called the *distance transform*) is an implicit function that gives very plausible intermediate shapes even when used with simple linear interpolation of the function values of the two shapes. The value of the signed distance function at a point $\mathbf{x}$ inside a given shape is just the Euclidean distance between $\mathbf{x}$ and the nearest point on the boundary of the shape. For a point $\mathbf{x}$ that is outside the shape, the signed distance function takes on the negative of the distance from $\mathbf{x}$ to the closest point on the boundary.

Several researchers have used the signed distance function to interpolate between 2D contours [19, 14]. The distance function for each given shape is represented as a regular 2D grid of values, and an intermediate implicit function is created by linear interpolation between corresponding grid values of the two implicit functions. Each intermediate shape is given by the zero iso-contour of this interpolated implicit function. In contrast to the global interpolation methods described above (frequency domain, wavelets, Minkowski sum), this interpolation is entirely local in nature. Nevertheless, the shape transformations that are created by this method are quite good. In essence, the information that the signed distance function encodes (distance to nearest boundary) is enough to make up for the purely local method of interpolation. Payne and Toga were the first to transform three dimensional shapes using this approach [23]. Cohen-Or and colleagues gave additional control to this same approach by combining it with a warping technique in order to produce shape transformations of 3D objects [7].

Our approach to shape transformation combines the two steps of building implicit functions and interpolating between them. To our knowledge, it is the only method to do so. The remainder of this paper describes how variational interpolation can be used to simultaneously solve these two tasks.

3 Variational Interpolation

Our approach relies on *scattered data interpolation* to solve the shape transformation problem. The problem of scattered interpolation is to create a smooth function that passes through a given set of data points. The two-dimensional version of this problem can be stated as follows: Given a collection of k constraint points $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$ that are scattered in the plane, together with scalar height values at each of these points $\{h_1, h_2, \ldots, h_k\}$, construct a smooth surface that matches each of these heights at the given locations. We can think of this solution surface as a scalar-valued function $f(\mathbf{x})$ so that $f(\mathbf{c}_i) = h_i$, for $1 \leq i \leq k$.

One common approach to solving scattered data problems is to use variational techniques (from the calculus of variations). This approach begins with an energy that measures the quality of an interpolating function and then finds the single function that matches the given data points and that minimizes this energy measure. For two-dimensional problems, thin-plate interpolation is the variational solution when using the following energy function E:

$$E = \int_{\Omega} f_{xx}^2(\mathbf{x}) + 2f_{xy}^2(\mathbf{x}) + f_{yy}^2(\mathbf{x}) \qquad (1)$$

The notation f_{xx} means the second partial derivative in the x direction, and the other two terms are similar partial derivatives, one of them mixed. The above energy function is basically a measure of the aggregate squared curvature of $f(\mathbf{x})$ over the region of interest Ω. Any creases or pinches in a surface will result in a larger value of E. A smooth surface that has no such regions of high curvature will have a lower value of E. The thin-plate solution to an interpolation problem is the function $f(\mathbf{x})$ that satisfies all of the constraints and that has the smallest possible value of E.

The scattered data interpolation problem can be formulated in any number of dimensions. When the given points $\mathbf{c}_i$ are positions in N-dimensions rather than in 2D, this is called the N-dimensional scattered data interpolation problem. There are appropriate generalizations to the energy function and to thin-plate interpolation for other dimensions. In this paper we will perform interpolation in two, three, four and five dimensions. Because the term *thin-plate* is only meaningful for 2D problems, we will use *variational interpolation* to mean the generalization of thin-plate techniques to any number of dimensions.

The scattered data interpolation task as formulated above is a variational problem where the desired solution is a function, $f(\mathbf{x})$, that will minimize equation 1 subject to the interpolation constraints $f(\mathbf{c}_i) = h_i$. Equation 1 can be solved using weighted sums of the

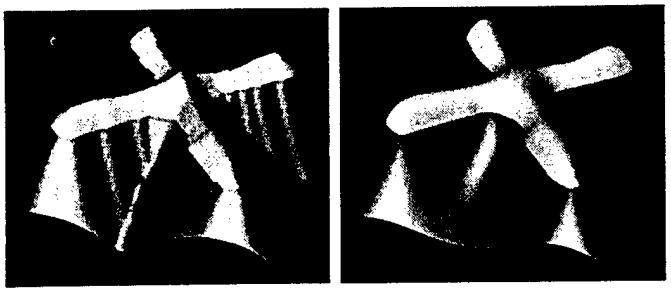


Figure 2: Implicit functions for an X shape. Left shows the signed distance function, and right shows the smoother variational implicit function.

radial basis function $\phi(\mathbf{x}) = |\mathbf{x}|^2 \log(|\mathbf{x}|)$. The family of variational problems that includes equation 1 was studied by Duchon [8].

Using the appropriate radial basis function, we can then express the interpolation function as

$$f(\mathbf{x}) = \sum_{j=1}^{n} d_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}) \tag{2}$$

In the above equation, $\mathbf{c}_j$ are the locations of the constraints, the d_j are the weights, and $P(\mathbf{x})$ is a degree one polynomial that accounts for the linear and constant portions of f. Because the thin-plate radial basis function naturally minimizes equation 1, determining the weights, d_j, and the coefficients of $P(\mathbf{x})$ so that the interpolation constraints are satisfied will yield the desired solution that minimizes equation 1 subject to the constraints. Furthermore, the solution will be an exact analytic solution, and is not subject to approximation and discretization errors that may occur when using finite element or finite difference methods.

To solve for the set of d_j that will satisfy the interpolation constraints $h_i = f(\mathbf{c}_i)$, we can substitute the right side of equation 2 for $f(\mathbf{c}_i)$, which gives:

$$h_i = \sum_{j=1}^{k} d_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}) \tag{3}$$

Since this equation is linear with respect to the unknowns, d_j and the coefficients of $P(\mathbf{x})$, it can be formulated as a linear system. For interpolation in 3D, let $\mathbf{c}_i = (c_i^x, c_i^y, c_i^z)$ and let $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$. Then this linear system can be written as follows:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \cdots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \cdots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \cdots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \cdots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The above system is symmetric and positive semi-definite, so there will always be a unique solution for the d_j and p_j [11]. For systems with up to a few thousand constraints, the system can be solved directly with a technique such as symmetric LU decomposition. We used symmetric LU decomposition to solve this system for all of the examples shown in this paper.

Using the tools of variational interpolation we can now turn our attention to creating implicit functions for shape transformation.

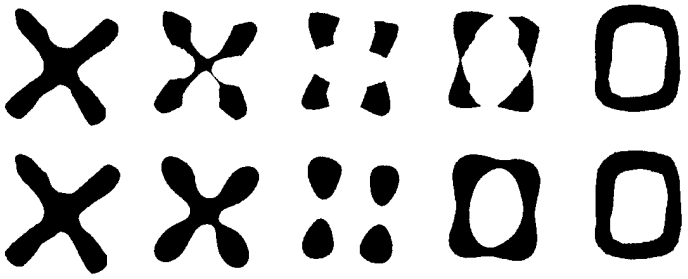


Figure 3: Upper row is a shape transformation created using the signed distance transform. Lower row is the sequence generated using a single variational implicit function.

4 Smooth Implicit Function Creation

In this section we will lay down the groundwork for shape transformation by discussing the creation of smooth implicit functions for a single shape. In particular, we will use variational interpolation of scattered constraints to construct implicit functions. Later we will generalize this to create functions that perform shape transformation.

Let us first examine the signed distance transformation because it is commonly used for shape transformation. The left half of Figure 2 shows a height field representation of the signed distance function of an X shape. The figure shows sharp ridges (the *medial axis*) that run down the middle of the height field. Ridges appear in the middle of shapes where the points are equally distant from two or more boundary points of the original shape. The values of a signed distance function decrease as one moves away from the ridge towards the boundaries. Figure 3, top row, shows a shape interpolation sequence between an X and an O shape that was created by linear interpolation between two signed distance functions. Note the pinched portions of some of the intermediate shapes. These sharp features are not isolated problems, but instead persist over many intermediate shapes. The cause of these pinches are the sharp ridges of signed distance functions. In many applications such artifacts are undesirable. In medical reconstruction, for example, these pinches are a poor estimate of shape because most biological structures have smooth surfaces. Because of this, we seek implicit functions that are continuous and that have a continuous first derivative.

4.1 Variational Implicit Functions in 2D

We can create smooth implicit functions for a given shape using variational interpolation. This can be done both for 2D and 3D shapes, although we will begin by discussing the 2D case. In this approach, we create a closed 2D curve by describing a number of locations through which the curve will pass and also specifying a number of points that should be interior to the curve. We call the given points on the curve the *boundary constraints*. The boundary constraints are locations at which we require our implicit function to take on the value of zero. Paired with each boundary constraint is a *normal constraint*, which is a location at which the implicit function is required to take on the value one. (Actually, any positive value could be used.) The locations of the normal constraints should be towards the interior of the desired curve, and also the line passing through the normal constraint and its paired boundary constraint should be parallel to the desired normal to the curve. The collection of boundary and normal constraints are passed along to a variational interpolation routine as the scattered constraints to be interpolated. The function that is returned is an implicit function that describes our curve. The curve will exactly pass through our boundary constraints.

Figure 4 (left) illustrates eight such pairs of constraints in the plane, with the boundary constraints shown as circles and the normal constraints as plus signs. When we invoke variational interpo-

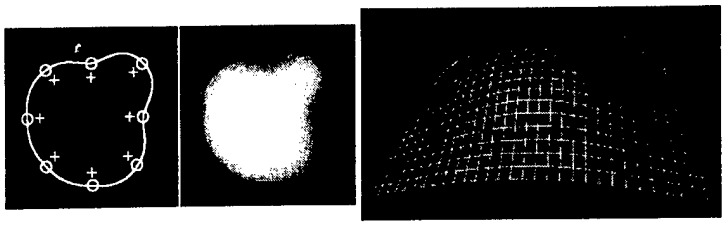


Figure 4: At left are pairs of boundary and normal constraints (circles and pluses). The middle image uses intensity to show the resulting variational implicit function, and the right image shows the function as a height field.

lation with such constraints, the result is a function that takes on the value of zero exactly at our zero-value constraints and that is positive in the direction of our normal constraints (towards the interior of the shape). The closed curve passing through the zero-value constraints in Figure 4 (middle) is the iso-contour of the implicit function created by this method. Figure 4 (right) shows the resulting implicit function rendered as a height field. Given enough suitably-placed boundary constraints we can define any closed shape. We call an implicit function that is created in this manner a *variational implicit function*. This new technique for creating implicit functions also show promise for surface modeling, a topic that is explored in [27].

We now turn our attention to defining boundary and normal constraints for a given 2D shape. Assume that a given shape is represented as a gray-scale image. White pixels represent the interior of a shape, black pixels will be outside the shape, and pixels with intermediate gray values lie on the boundary of the shape. Let m be the middle gray value of our image's gray scale range. Our goal is to create constraints between any two adjacent pixels where one pixel's value is less than m and the other's value is greater. Identifying these locations is the 2D analog of finding the vertex locations in a 3D marching cubes algorithm [21].

We traverse the entire gray-scale image and examine the east and south neighbor of each pixel $I(x,y)$. If $I(x,y) < m$ and either neighbor has a value greater than m, we create a boundary constraint at a point along the line segment joining the pixel centers. A boundary constraint is also created if $I(x,y) > m$ and either neighbor takes on a value less than m. The value of the constraint is zero, and we set the position of the constraint at the location between the two pixels where the image would take on the value of m if we assume linear interpolation of pixel values. Next, we estimate the gradient of the gray scale image using linear interpolation of pixel values and central differencing. We then create a normal constraint a short distance away from the zero crossing in the direction of the gradient. We have found that a distance of a pixel's width between the boundary and normal constraints works well in practice. Figure 2 (right) shows the implicit function for an X shape that was created using variational interpolation from such constraints. It is smooth and free of sharp ridges.

4.2 Variational Implicit Functions in 3D

We can create implicit functions for 3D surfaces using variational interpolation in much the same way as for 2D shapes. Specifically, we can derive 3D constraints from the vertex positions and surface normals of a polygonal representation of an object. Let (x,y,z) and (n_x,n_y,n_z) be the position and the surface normal at a vertex, respectively. Then a boundary constraint is placed at (x,y,z) and a normal constraint is placed at $(x-kn_x, y-kn_y, z-kn_z)$, where k is some small value. We use a value of $k = 0.01$ for models that fit within a unit cube for the results shown in this paper. All of the 3D models that we transform in this paper were constructed by building an implicit function in this manner. Note that we can use this method to build an implicit function whenever we have a collection of points and normals— polygon connectivity is not necessary.

Now that we can construct smooth implicit functions for both two- and three-dimensional shapes, we turn our attention to shape transformation. It would be possible to create variational implicit functions for each of two given shapes and then linearly interpolate between these functions to create a shape transformation sequence. Instead, however, we will examine an even better way of performing shape transformation by generalizing the implicit function building methods of this section.

5 Unifying Function Creation and Interpolation

The key to our shape transformation approach is to represent the entire sequence of shapes with a single implicit function. To do so, we need to work in one higher dimension than the given shapes. For 2D shapes, we will construct an implicit function in 3D that represents our two given shapes in two distinct parallel planes. This is actually simple to achieve now that we know how to use scattered data interpolation to create an implicit function.

5.1 Two-Dimensional Shape Transformation

Given two shapes in the plane, assume that we have created a set of boundary and normal constraints for each shape, as described in Section 4. Instead of using each set of constraints separately to create two different 2D implicit functions, we will embed all of the constraints in 3D. We do this by adding a third coordinate value to the location of each boundary and normal constraint. For those constraints for the first shape, we set the new coordinate t for all constraints to $t = 0$. For the second shape, all of the new coordinate values are set to $t = t_{max}$ (some non-zero value). Although we have added a third dimension to the locations of the constraints, the values that are to be interpolated remain unchanged for all constraints.

Once we have placed the constraints of both shapes into 3D, we invoke 3D variational interpolation to create a single scalar-valued function over $\mathbf{R}^3$. If we take a slice of this function in the plane $t = 0$, we find an implicit function that takes on the value zero exactly at the boundary constraints for our first shape. In this plane, our function describes the first shape. Similarly, in the plane $t = t_{max}$ this function gives the second shape. Parallel slices at locations between these two planes ($0 < t < t_{max}$) represent the shapes of our shape transformation sequence. Figure 1 illustrates that the collection of intermediate shapes are all just slices through a surface in 3D that is created by variational interpolation.

Figure 3 (bottom) shows the sequence of shapes created using this variational approach to shape transformation. Topology changes (e.g. the addition or removal of holes) come "for free", without any human guidance or algorithmic complications. Notice that all of the intermediate shapes have smooth boundaries, without pinches. Sharp features can arise only momentarily when there is a change in topology such as when two parts join. Figure 5 shows two more shape transformations that use this approach and that also incorporate warping. Warping is an another degree of control that may be added to any shape transformation technique, and is in fact

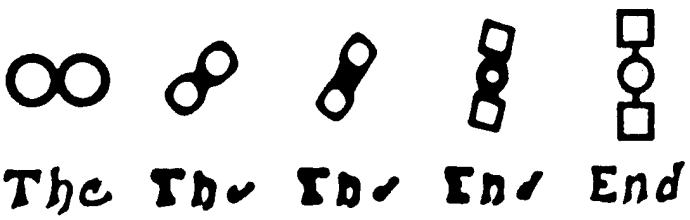


Figure 5: Two shape transformation sequences (using the variational implicit technique) that incorporate warping.

an orthogonal issue to those of implicit function creation and interpolation. Although it is not a focus of our research, for completeness we briefly describe warping in the appendix.

Why has this implicit function building method not been tried using other ways of creating implicit functions? Why not, for example, build a signed distance function in one higher dimension? Because a *complete* description of an object's boundary is required in order to build a signed distance function. When we embed our two shapes into a higher dimension, we only know a *piece* of the boundary of our desired higher-dimensional shape, namely the cross-sections that match the two given objects. In contrast, a complete boundary representation is *not* required when using variational interpolation to create an implicit function. Variational interpolation creates plausible function values in regions where we have no information, and especially in the "unknown" region between the two planes that contain all of our constraints. This plausibility of values comes from the smooth nature of the functions that are created by the variational approach.

5.2 Three-Dimensional Shape Transformation

Just as we create a 3D function to create a transformation between 2D shapes, we can move to 4D in order to create a sequence between 3D shapes. We perform shape interpolation between two 3D objects using boundary and normal constraints for each shape. We place the constraints from two 3D objects into four dimensional space, just as we placed constraints from 2D contours into 3D. Similar to contour interpolation, the constraints are separated from one another in the fourth dimension by some specified distance. We place all the constraints from the first object at $t = 0$, and the constraints from the second object are placed at $t = t_{max}$, where t_{max} is the given separation distance. We then create a 4D implicit function using variational interpolation. An intermediate shape between the two given shapes is found by extracting the isosurface of a 3D "slice" (actually a volume) of the resulting 4D function.

Figure 6 shows two 3D shape transformation sequence that were constructed using this method. To extract these surfaces we use code published by Bloomenthal that begins at a seed location on the surface of a model and only evaluates the implicit function at points near previously visited locations [4]. This is far more efficient than sampling an entire volume of the implicit function and then extracting an isosurface from the volume. The matrix solution for the transformation sequence of Figure 6 (left) required 13.5 minutes, and each isosurface in the sequence took approximately 2.3 minutes to generate on an SGI Indigo2 with a 195 MHz R10000 processor. Figure 6 (right) shows a transformation between 3D shapes that used warping to align features.

6 Surface Reconstruction from Contours

So far we have only considered shape transformation between pairs of objects. In medical reconstruction, however, it is often necessary to create a surface from a large number of parallel 2D slices. Can't we just perform shape interpolation between pairs of slices and stack the results together to create one surface in 3D? Although this method will create a continuous surface, it is almost certain to produce a shape that has surface normal discontinuities at the planes of the original slices. In the plane of slice i, the surface created between slice pairs $i-1$ and i will usually not agree in surface normal with the surface created between slices i and $i+1$. Nearly all contour interpolation methods consider only pairs of contours at any one time, and thus suffer from such discontinuities. (A notable exception is [1]).

To avoid discontinuities in surface normal, we must use information about more than just two slices at a given time. We can accomplish this using a generalization of the variational approach to shape transformation. Assume that we begin with k sets of constraints, one set for each 2D data slice. Instead of considering the contours in pairs, we place the constraints for all of the k slices into

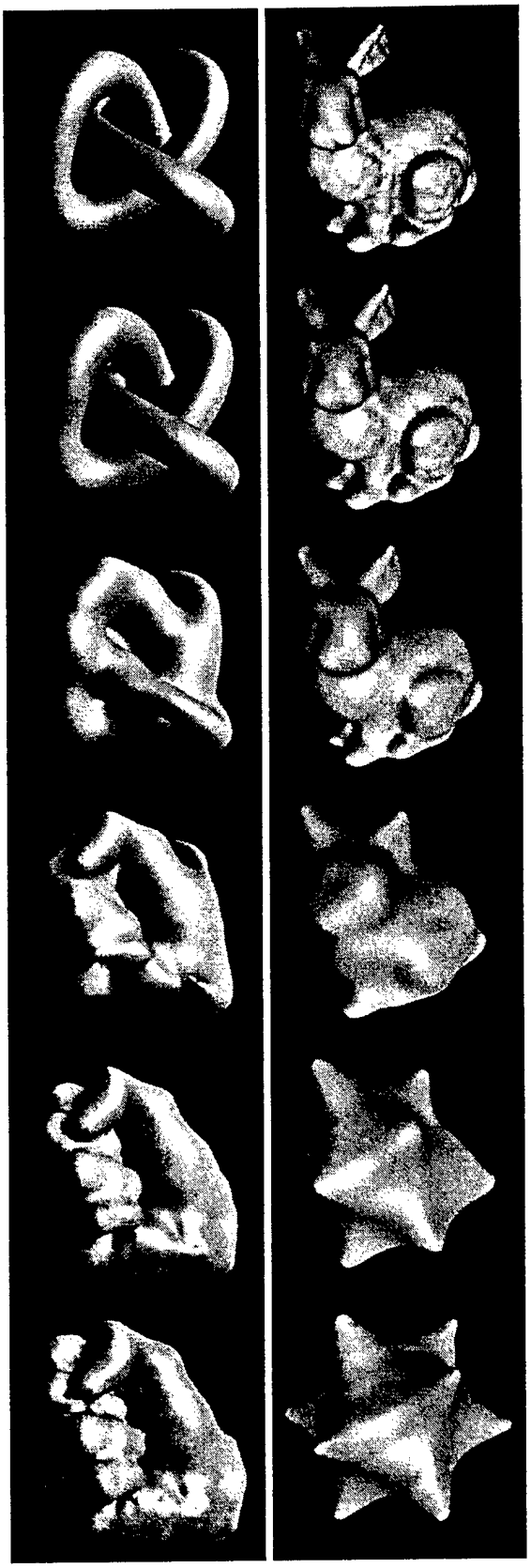


Figure 6: 3D shape transformation sequences.

3D simultaneously. Specifically, the constraints of slice i are placed in the plane $z = si$, where s is the spacing between planes. Once the constraints from *all* slices have been placed in 3D, we invoke

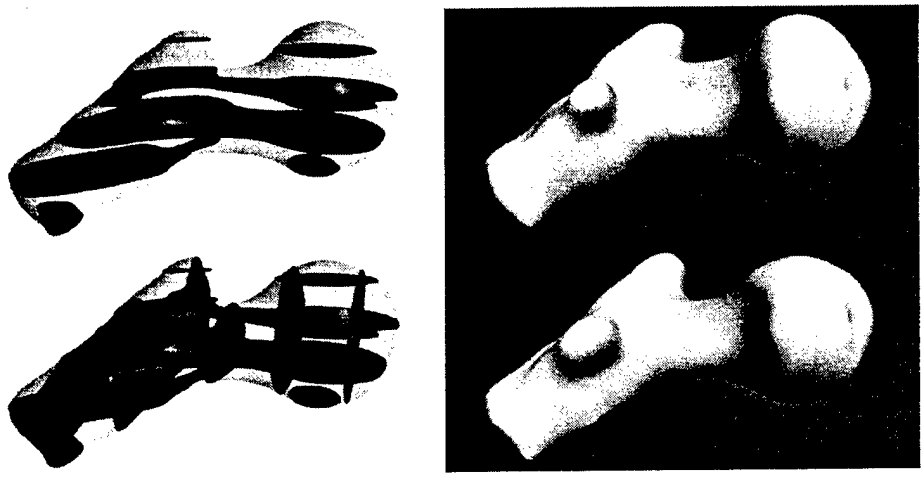


Figure 7: Reconstruction of hip joint from contours. Top row shows the five parallel slices used and the final surface. Bottom row shows intersecting contours and the more detailed surface that is created.

variational interpolation *once* to create a single implicit function in 3D. The zero-valued isosurface exactly passes through each contour of the data. Due to the smooth nature of variational interpolation, the gradient of the implicit function is everywhere continuous. This means that surface normal discontinuities are rare, appearing in pathological situations when the gradient vanishes such as when two features just barely touch. Figure 7 (top row) illustrates the result of this contour interpolation approach. The hip joint reconstruction in the upper right was created from the five slices shown at the upper left.

A side benefit of using the variational implicit function method is that it produces smoothly rounded caps on the ends of surfaces. Notice that in Figure 7 (top left) that the reconstructed surface extends beyond the constraints in the positive and negative z direction (the direction of slice stacking). This "rounding off" of the ends is a natural side effect of variational interpolation, and need not be explicitly specified.

6.1 Non-Parallel Contours

In the previous section, we only considered placing constraints within planes that are all parallel to one another. There is nothing special about any particular set of planes, however, once we are specifying constraints in 3D. We can mix together constraints that are taken from planes at any angle whatsoever, so long as we know the relative positions of the planes (and thus the constraints). Most contour interpolation procedures cannot integrate data taken from slices in several directions, but the variational approach allows complete freedom in this regard. Figure 7 (lower row) shows several contours that are placed perpendicular to one another, and the result of using variational interpolation on the group of constraints from these contours.

6.2 Between-Contour Spacing

Up to this point we have not discussed the separating distance s between the slices that contain the contour data. This separating distance has a concrete meaning in medical shape reconstruction from 2D contours. Here we know the actual 3D separation between the contours from the data capture process. This "natural" distance is the separating distance s that should be used when reconstructing the surface using variational interpolation. Upon reflection, it is odd that some contour interpolation methods do not make use of the data capture distance between slices. In some cases a medical technician will deliberately vary the spacing between data slices in order to capture more data in a particular region of interest. Using variational interpolation, we may incorporate this information about varying separation distances into the surface reconstruction process.

For both special effects production and for computer aided design, the distance between the separating planes can be thought of as a control knob for the artist or designer. If the distance is small, only pairs of features from the two shapes that are very close to one another will be preserved through all the intermediate shapes. If the separation distance is large, the intermediate shape is guided by more global properties of the two shapes. In some sense, the separating distance specifies whether the shape transformation is local or global in nature. The separation distance is just one control knob for the user, and in the next section we will explore another user control.

7 Influence Shapes

In this section we present a method of controlling shape transformation by introducing an *influence shape*. The idea to use additional objects as controls for shape transformation was introduced by Rossignac and Kaul [24]. Such intermediate shape control can be performed in a natural way using variational interpolation. The key is to step into a still higher dimension when performing shape transformation.

Recall that to create a transformation sequence between two given shapes we added one new dimension, called t earlier. We can think of the two shapes as being two points that are separated along the t dimension, and these two points are connected by a line segment that joins the two points along this dimension. If we begin with three shapes, however, we can in effect place them at the three points of a triangle. In order to do so we need not just one additional dimension but two, call them s and t.

As an example, we may begin with three different 3D shapes A, B and C. To each constraint that describes one of the shapes, we can add two new coordinates, s and t. Constraints from shape A at (x,y,z) are placed at $(x,y,z,0,0)$, constraints from shape B are placed at $(x,y,z,1,0)$ and shape C constraints are placed at $(x,y,z,1/2,1/2)$. Variational interpolation based on these 5-dimensional constraints results in a 5D implicit function. Three-dimensional slices of this function along the s-dimension between 0 and 1 are simply shape sequences between shapes A and B when the t-dimension value is fixed at zero. If, however, the t-dimension value is allowed to become positive as s varies from 0 to 1, then the intermediate shapes will take on some of the characteristics of shape C. In fact, the 5D implicit function actually captures an entire family of shapes that are various blends between the three shapes. Figure 8 illustrates some members of such a family of shapes.

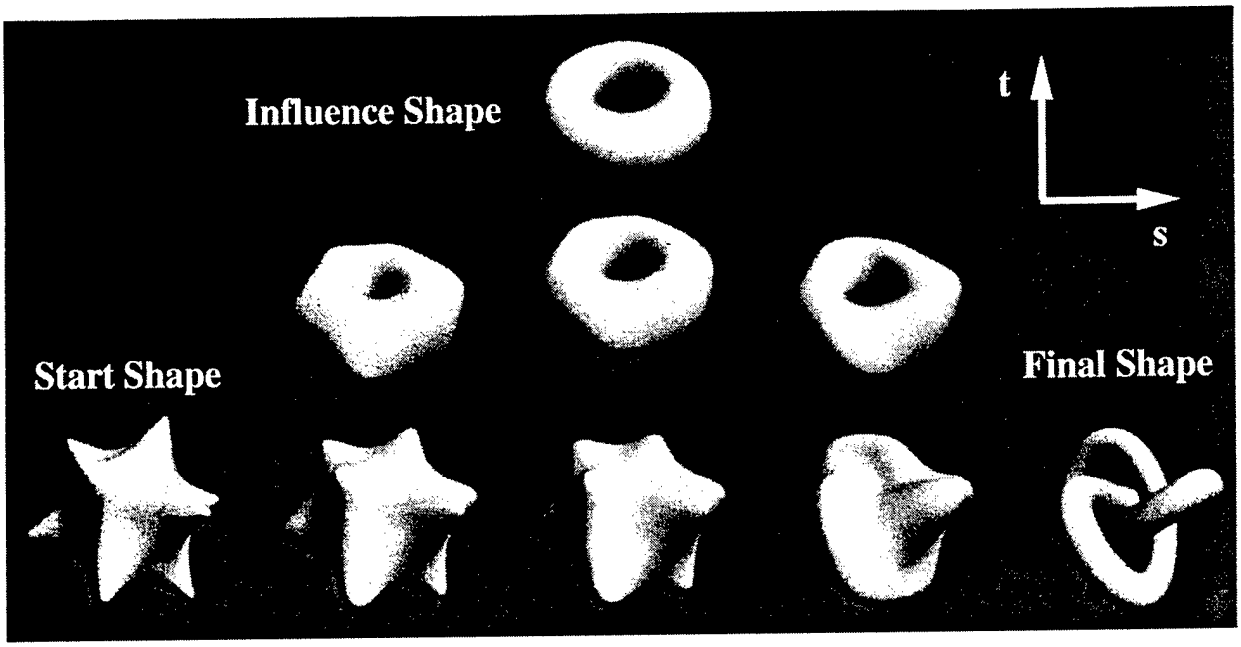


Figure 8: Sequence between star and knot can be influenced by a torus (the influence shape) if the path passes near the torus in the five-dimensional space.

There is no reason to stop at three shapes. It is possible to place four shapes at the corners of a quadrilateral, five shapes around a pentagon, and so on. If we wish to use four shapes, then placing the constraints at the corners of a quadrilateral using two additional dimensions would not allow us to produce a shape that was arbitrary mixtures between the shapes. In order to do so, we can place the constraints in yet a higher dimension, in effect placing the four shapes at the corners of a tetrahedron in $N+3$ dimensions, where N is the dimension of the given shapes.

There are two related themes that guide our technique for shape transformation. The first is that shape transformation should be thought of as a shape-creation problem in a higher dimension. The second theme is that better shape transformation sequences are produced when all of the problem constraints are solved simultaneously— in our case by using variational interpolation. Influence shapes are the result of taking these ideas to an extreme.

8 Conclusions and Future Work

Our new approach uses variational interpolation to produce one implicit function that describes an entire sequence of shapes. Specific characteristics of this approach include:

- Smooth intermediate shapes
- Shape transformation in any number of dimensions
- Analytic solutions that are free of polygon and voxel artifacts
- Continuous surface normals for contour interpolation
- Contour slices may be at any orientation, even intersecting

This approach provides two new controls for creating shape transformation sequences:

- Separation distance gives local/global interpolation tradeoff
- May use influence shapes to control a transformation

The approach we have presented in this paper re-formulates the shape interpolation problem as an interpolation problem in one higher dimension. In essence, we treat the "time" dimension just like another spatial dimension. We have found that using the variational interpolation method produces excellent results, but the mathematical literature abounds with other interpolation methods. An exciting avenue for future work is to investigate what other interpolation techniques can also be used to create implicit functions for shape transformation. Another issue is whether shape transformation methods can be made fast enough to allow a user interactive control. Finally, how might surface properties such as color and texture be carried through intermediate objects?

9 Acknowledgements

This work owes a good deal to Andrew Glassner for getting us interested in the shape transformation problem. We thank our colleagues and the anonymous reviewers for their helpful suggestions. This work was funded by ONR grant N00014-97-1-0223.

References

[1] Barequet, Gill, Daniel Shapiro and Ayellet Tal, "History Consideration in Reconstructing Polyhedral Surfaces from Parallel Slices," *Proceedings of Visualization '96*, San Francisco, California, Oct. 27 – Nov. 1, 1996, pp. 149–156.

[2] Barr, Alan H., "Global and Local Deformations of Solid Primitives," *Computer Graphics*, Vol. 18, No. 3 (SIGGRAPH 84), pp. 21–30.

[3] Beier, Thaddeus and Shawn Neely, "Feature-Based Image Metamorphosis," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 35–42.

[4] Bloomenthal, Jules, "An Implicit Surface Polygonizer," in *Graphics Gems IV*, edited by Paul S. Heckbert, Academic Press, 1994, pp. 324–349.

[5] Bookstein, Fred L., "Principal Warps: Thin Plate Splines and the Decomposition of Deformations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 6, June 1989, pp. 567–585.

[6] Celniker, George and Dave Gossard, "Deformable Curve and Surface Finite-Elements for Free-Form Shape Design," *Computer Graphics*, Vol. 25, No. 4 (SIGGRAPH 91), July 1991, pp. 257–266.

[7] Cohen-Or, Daniel, David Levin and Amira Solomovici, "Three Dimensional Distance Field Metamorphosis," *ACM Transactions on Graphics*, 1997.

[8] Duchon, Jean, "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," in *Constructive Theory of Functions of Several Variables*, Lecture Notes in Mathematics, edited by A. Dolb and B. Eckmann, Springer-Verlag, 1977, pp. 85–100.

[9] Duncan, Jody, "A Once and Future War," *Cinefex*, No. 47 (entire issue devoted to the film Terminator 2), August 1991, pp. 4–59.

[10] Fuchs, H., Z. M. Kedem and S. P. Uselton, "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, Vol. 20, No. 10, October 1977, pp. 693–702.

[11] Golub, Gene H. and Charles F. Ban Loan, *Matrix Computations*, John Hopkins University Press, 1996.

[12] Gregory, Arthur, Andrei State, Ming C. Lin, Dinesh Manocha, Mark A. Livingston, "Feature-based Surface Decomposition for Correspondence and Morphing between Polyhedra", *Proceedings of Computer Animation*, Philadelphia, PA., 1998.

[13] He, Taosong, Sidney Wang and Arie Kaufman, "Wavelet- Based Volume Morphing," *Proceedings of Visualization '94*, Washington, D. C., edited by Daniel Bergeron and Arie Kaufman, October 17-21, 1994, pp. 85–92.

[14] Herman, Gabor T., Jingsheng Zheng and Carolyn A. Bucholtz, "Shape-Based Interpolation," *IEEE Computer Graphics and Applications*, Vol. 12, No. 3 (May 1992), pp. 69–79.

[15] Hugues, John F., "Scheduled Fourier Volume Morphing," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 43–46.

[16] Kaul, Anil and Jarek Rossignac, "Solid- Interpolating Deformations: Construction and animation of PIPs," *Proceedings of Eurographics '91*, Vienna, Austria, 2-6 Sept. 1991, pp. 493–505.

[17] Kent, James R., Wayne E. Carlson and Richard E. Parent, "Shape Transformation for Polyhedral Objects," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 47–54.

[18] Lerios, Apostolos, Chase Garfinkle and Marc Levoy, "Feature-Based Volume Metamorphosis," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 95), pp. 449–456.

[19] Levin, David, "Multidimensional Reconstruction by Set-valued Approximation," *IMA J. Numerical Analysis*, Vol. 6, 1986, pp. 173–184.

[20] Litwinowicz, Peter and Lance Williams, "Animating Images with Drawings," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 94), pp. 409–412.

[21] Lorenson, William and Harvey E. Cline, "Marching Cubes: A High Resolution 3-D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4 (SIGGRAPH 87), July 1987, pp. 163–169.

[22] Meyers, David and Shelley Skinner, "Surfaces From Contours: The Correspondence and Branching Problems," *Proceedings of Graphics Interface '91*, Calgary, Alberta, 3-7 June 1991, pp. 246–254.

[23] Payne, Bradley A. and Arthur W. Toga, "Distance Field Manipulation of Surface Models," *IEEE Computer Graphics and Applications*, Vol. 12, No. 1, January 1992, pp. 65–71.

[24] Rossignac, Jarek and Anil Kaul, "AGRELs and BIPs: Metamorphosis as a Bezier Curve in the Space of Polyhedra," *Proceedings of Eurographics '94*, Oslo, Norway, Sept. 12–16, 1994, pp. 179–184.

[25] Sederberg, Thomas W. and Eugene Greenwood, "A Physically Based Approach to 2-D Shape Blending," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 25–34.

[26] Sederberg, Thomas W. and Scott R. Parry, "Free-Form Deformations of Solid Geometric Models," *Computer Graphics*, Vol. 20, No. 4 (SIGGRAPH 86), pp. 151–160.

[27] Turk, Greg and James F. O'Brien, "Variational Implicit Surfaces," Tech Report GIT-GVU-99-15, Georgia Institute of Technology, May 1999, 9 pages.

[28] Wolberg, George, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, California 1990.

10 Appendix: Warping

Warping is a commonly used method of providing user control of shape interpolation. Although warping is not a focus of our research, for the sake of completeness we describe below how warping may be used together with our shape transformation technique. Research on warping (sometimes called deformation) include [2, 26, 28, 3, 18, 7].

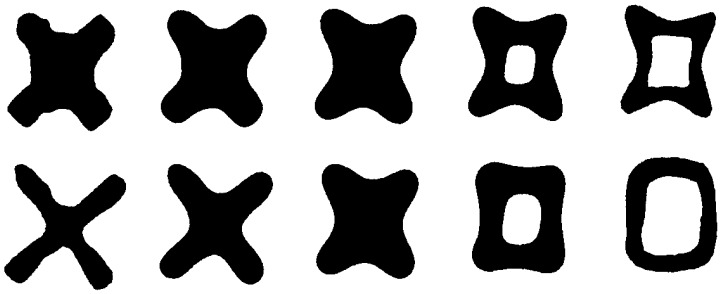


Figure 9: The extreme left and right shapes in the top row have been warped before creating the upper shape transformation sequence. The lower row is an un-warped version of this sequence that gives the final transformation from an X to O.

For symmetry, we choose to warp each shape "half-way" to the other shape. Given a set of user-supplied corresponding points between two shapes A and B, we construct two displacement warp functions $\mathbf{w}_A$ and $\mathbf{w}_B$. The function $\mathbf{w}_A$ specifies what values to add to locations on shape A in order to warp it half-way to shape B, and the warping function $\mathbf{w}_B$ warps B half of the way to A.

In what follows, we will describe the warping process for two-dimensional shapes. Let $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k\}$ be a set of points on shape A, and let $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k\}$ be the corresponding points on B. We construct the two functions w_A and w_B such that $w_A(\mathbf{a}_i) = (\mathbf{b}_i - \mathbf{a}_i)/2$ and $w_B(\mathbf{b}_i) = (\mathbf{a}_i - \mathbf{b}_i)/2$ hold for all i. Constructing these functions is another example of scattered data interpolation which we can solve using variational techniques. For 2D shapes, if $\mathbf{a}_i = (a_i^x, a_i^y)$ and $\mathbf{b}_i = (b_i^x, b_i^y)$, then the x component w_A^x of the displacement warp $\mathbf{w}_A$ has k constraints at the positions $\mathbf{a}_i$ with values $(b_i^x - a_i^x)/2$. We invoke variational interpolation to satisfy these constraints, and do the same to construct the y component of the warp. The function $\mathbf{w}_B$ is constructed similarly. This is not a new technique, and researchers who use thin-plate techniques to perform shape warping include [5, 20] and others.

In order to combine warping with shape transformation, we use these functions to displace all of the boundary constraints of the given shapes. These displaced boundary constraints are embedded in 3D (as described in Section 5) and then variational interpolation is used to create the implicit function that describes the entire shape transformation sequence. The result of this process is a three-dimensional implicit function, each slice of which is an intermediate shape between two warped shapes. The top row of Figure 9 shows such warped intermediate shapes. We can think of the two "ends" of this implicit function (at $t = 0$ and $t = t_{max}$) as being warped versions of our original shapes. In order to match the two original shapes, the surface of this 3D implicit function needs to be unwarped. To simplify the equations, assume that the value of t_{max} is 2. If $t \leq 1$ the unwarping function $u(x,y,t)$ is:

$$u(x,y,t) = (x + (1-t)w_A^x(x,y), y + (1-t)w_A^y(x,y), t) \qquad (4)$$

If $t > 1$ then the unwarping function is:

$$u(x,y,t) = (x + (t-1)w_B^x(x,y), y + (t-1)w_B^y(x,y), t) \qquad (5)$$

At the extreme of $t = 0$, the warp $u(x,y,t)$ un-does the warping we used for the first shape. At $t = 2$, the function $u(x,y,t)$ reverses the warping used for the second shape. When $t = 1$ (the middle shape in the sequence), no warp is performed. The bottom sequence of shapes in Figure 9 shows the result of the entire shape transformation process that includes warping. Both sequences in Figure 5 were created using warping in addition to shape transformation.

Although we have described the warping process for 2D shapes, the same method may be used for shape transformation between 3D shapes. For Figure 6 (right), warping was used to align the bunny ears to the points of the star.

Robust Creation of Implicit Surfaces from Polygonal Meshes

Gary Yngve
University of Washington
gyngve@cs.washington.edu

Greg Turk
Georgia Institute of Technology
turk@cc.gatech.edu

Abstract

Implicit surfaces are used for a number of tasks in computer graphics, including modeling soft or organic objects, morphing, collision detection, and constructive solid geometry. Although operating on implicit surfaces is usually straightforward, creating them is not — interactive techniques are impractical for complex models, and automatic techniques have been largely unexplored. We introduce a practical method for creating implicit surfaces from polygonal models that produces high-quality results for complex surfaces. Whereas much previous work in implicit surfaces has been done with primitives such as "blobbies," we use implicit surfaces based on a variational interpolation technique (the three-dimensional generalization of thin-plate interpolation). Given a polygonal mesh, we convert the data to a volumetric representation to use as a guide for creating the implicit surface iteratively. We begin by seeding the surface with a number of constraint points through which the surface must pass. Additional constraints are then added to specify new points on the surface. The resulting intermediate surface is evaluated by error metrics, and this error guides the placement of subsequent constraints. We have applied our method successfully to a variety of polygonal meshes and consider it to be robust.

Keywords

Geometric Modeling, Surface Representations, Implicit Surfaces.

I. Introduction

The task of constructing smooth surfaces is ubiquitous throughout computer graphics. Often parametric surfaces are the choice representation because of the capabilities of many commercial modeling packages. Once constructed, the parametric surfaces are then used in a variety of graphics algorithms, ranging from ray-tracing to morphing. However, many of these graphics algorithms have more elegant solutions when used with implicit surfaces. For implicit surfaces to become more widely used, however, they must become easier to create. We approach this issue by introducing a new method to convert polygonal surfaces to smooth implicit surfaces automatically.

Because points can be evaluated easily as being inside or outside an implicit surface, many applications that are challenging for parametric surfaces (including polygonal meshes) become simple when implicit surfaces are used. Boolean CSG operations (union, intersection, subtraction) reduce to simply examining the signs of the implicit functions. Operations on implicit surfaces that may cause the genus of the surface to change have simple implementations because the operations affect every point in space — on the isosurface, inside, and outside. Shape morphing can be performed simply by interpolating between two implicit functions, and the two shapes can be of arbitrary manifold topology [1], [2], [3]. Implicit surfaces can collide and deform [4], [5]; the resulting fusions and separations are handled automatically. Often in graphics, implicit functions are created by summing many infinitely differentiable functions, yielding surfaces that are smooth and seamless. The forms that they can represent are useful for modeling organic shapes and some classes of machine parts that require blends and fillets.

Although implicit surfaces have many benefits, they can be difficult to model. Most parametric surface representations use basis functions with finite support, and thus give the user an easy way to perform local control of the surface shape. In contrast, the bases which are used as primitives for implicit surfaces can often have non-obvious influences on surface position. Modeling with "blobbies" [6] suffers from this problem because each blobby primitive only indirectly influences the position of the isosurface. We note that the work of Witkin and Heckbert is aimed at overcoming this difficulty [7].

In our approach, rather than using the more traditional blobby primitives approach to implicit surface creation, we instead use variational implicit surfaces. This form of implicit surface allows a user to specify locations that the surface will exactly interpolate; this property allows more direct control over surface creation. As we will describe more fully later, solving a set of linear equations will guarantee that the surface interpolates a given set of constraint points. In addition to this interpolating property, variational implicit functions are smooth when their basis functions are chosen to satisfy an energy functional related to the desired degree of smoothness. Our approach to creating these surfaces is to add new constraints iteratively until the model is a close approximation to the input polygonal

mesh. Figure 1 shows twenty-three iterations of our algorithm while creating a frog model.

We focus on the creation of implicit models from polygonal meshes because of the large number of existing high-quality polygonal meshes. Having a robust automatic conversion procedure from meshes to implicits should provide a pathway towards creating a large library of implicit surfaces. With such a technique, all of the interactive modeling tools for creating polygonal meshes can then be used to create implicit surfaces. This ability would mean that we can avoid having to create special-purpose modeling programs for implicit surfaces. Because implicit models are so much more compact in terms of storage, converting from polygons to implicits can also be viewed as a compression scheme.

The rest of the paper will proceed as follows: We briefly discuss previous work in implicit-surface modeling in Section II. In Section III, we explain the variational implicit surface representation. Then

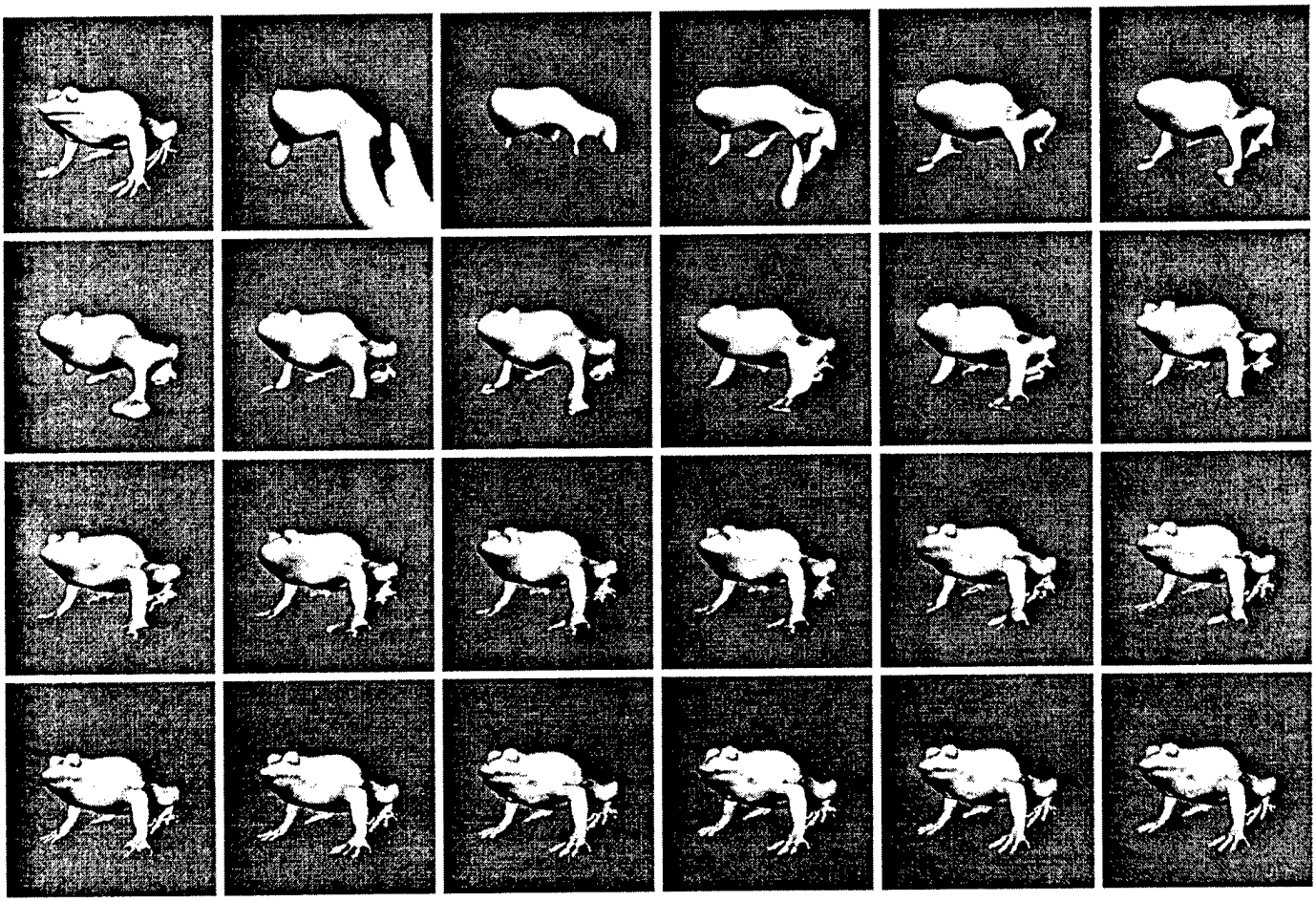


Fig. 1. A polygonal frog (top left) is converted to an implicit surface via our incremental improvement algorithm. The algorithm took 23 iterations to reach the final result. The results from each iteration are shown in successive images. The frog is a near fit after the first three rows; in the last row the toes get refined.

in Section IV, we introduce a set of tools that will be used by the algorithm. We present the algorithm in Section V, analyze the algorithm's parameters in Section VI, and show results in Section VII. Finally we conclude and discuss future work in Section VIII.

II. Previous Work

The very first implicit surfaces used in computer graphics were quadrics (degree-two polynomials of x, y, and z), such as spheres, ellipsoids, and cylinders [8]. Blinn generalized these implicit surfaces for the purpose of modeling molecules [6]. Basing his model on electron densities, he developed the blobby molecule model, which consists of Gaussian-like primitives blended together:

$$f_i(x) = A_i e^{b_i x^2 - c_i} \tag{1}$$

Each primitive is a radial basis function that can be tuned to control its size and blobbiness (its tendency to blend). This method and its variants [9] are widely used in the computer graphics community. As mentioned earlier, however, this form of implicit surface does not allow a user to directly specify points that the surface interpolates. The user must somehow estimate the location of the middle of the shape because this is where the centers of the primitives must be placed.

Another genre of implicit surfaces is the convolution surface [10]. These surfaces are created by convolving a skeleton shape (e.g. a collection of polygons) with a kernel sush as a Gaussian. The skeletons for the convolutions can actually be any form of geometry, including both 2D surfaces and solid objects. The resulting convolution surface is smooth. As with the blobby implicits, convolution surfaces do not allow a user to give specific points to be interpolated.

Interactive modeling techniques can be used to create implicit surfaces of modest complexity. One elegant method for interactive modeling was described by Witkin and Heckbert, in which they use particles to sample and control implicit surfaces [7]. Particles diffuse across the surface and are created and destroyed as necessary. They implemented their technique with blobby spheres and cylinders, and their technique is adaptable to variational implicit surfaces as well. However, for creating complicated

models, more automatic methods are needed.

Muraki developed a method to approximate range data by a blobby implicit surface [11]. Muraki's method incrementally adds primitives one at a time. At each iteration his algorithm picks a primitive, duplicates it, and then solves an optimization problem to minimize an energy functional. Because this requires is solving an optimization problem every iteration, the method is exceedingly slow — a model with 243 primitives took a few days to create on a Stardent Titan3000 2CPU. Bittar, Tsingos, and Gascuel addressed the modeling of an implicit surface from volume data [12]. They calculate a medial axis of the volume data as an aid to implicit function creation. They then use an optimization scheme based on Muraki's work to add primitives along the medial axis in substantially less time than Muraki's approach. However, the implicit surfaces that they generated with their method were small (the largest had only about 50 primitives). Both of these techniques produce results that provide a general fit of a model but lack high detail.

This brief summary barely scratches the surface of work on implicits in computer graphics. For an excellent overview of the area and more details on kinds of implicit surfaces, see the book by Bloomenthal et al. [13].

III. Variational Implicit Surfaces

In this section we give the equations that describe variational implicit surfaces and outline the algorithm that we use to create such surfaces from polygonal meshes.

A. Basic Formulation

Variational implicit surfaces are created by solving a scattered data interpolation problem [14]. The particular solution technique is based on ideas from the calculus of variations (solving an energy minimization problem). To create a variational implicit function, a user specifies a set of k constraint points $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$, along with a set of values $\{h_1, h_2, \ldots, h_k\}$ at the given constraint positions. The surfaces are controlled directly using three types of constraints. *Boundary constraints* are those

positions that are specified to take on the value zero, and the created implicit surface will exactly pass through these points. In addition, we can specify that certain points will be interior or exterior to the surface. *Interior constraints* are given positive values, and *exterior constraints* are given negative values. To create the appropriate implicit function, these constraints are handed to a sparse data interpolation routine that creates a function that exactly matches the given constraints.

The form of the function created by this technique is a weighted set of radial basis functions and a polynomial term. The weights of the basis function are found by solving a matrix equation (given below). We use the radial basis function

$$\phi(\mathbf{x}) = |\mathbf{x}|^3, \tag{2}$$

which minimizes the curvature functional

$$\int_{\mathbf{x}\in\Omega} \sum_{i,j} \left(\frac{\partial^2 \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_j} \right)^2 \mathrm{d}\mathbf{x}. \tag{3}$$

It is important to note that this functional does not represent curvature on the isosurface (the 2–manifold $f(x) = 0$ embedded in bounded region Ω), but rather the aggregate curvature of f over the entire region.

Using this radial basis function, the implicit function that we create has the form

$$f(\mathbf{x}) = \sum_{j=1}^{n} d_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}) \tag{4}$$

In the above equation, $\mathbf{c}_j$ are the locations of the constraints, the d_j are the weights, and $P(\mathbf{x})$ is a first-degree polynomial that accounts for the linear and constant portions of f.

To solve for the set of d_j that will satisfy the interpolation constraints $h_i = f(\mathbf{c}_i)$, we can substitute the right side of equation 4 for $f(\mathbf{c}_i)$, which gives:

$$h_i = \sum_{j=1}^{k} d_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}) \tag{5}$$

Because equation 5 is linear with respect to the unknowns, d_j and the coefficients of $P(\mathbf{x})$, it can be formulated as simple matrix equation. For interpolation in three dimensions, let $\mathbf{c}_i = (c_i^x, c_i^y, c_i^z)$ and let $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$. Then the linear system can be written as the following matrix equation:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \dots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \dots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \dots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

We used LU decomposition to solve this system for all of the examples shown in this paper. With the coefficients from the matrix solution (the d's and p's), evaluating the implicit function from Equation 4 becomes simple.

B. Outline of Our Approach

We will now examine how the constraints for a variational implicit surface can be derived from a polygonal model. This task is easy for models that are composed of polygons that are all nearly the same size. For such a polygonal model, we may use the vertices of the model as the positions of the boundary constraints. Similarly, we can create exterior constraints by moving out from each vertex in the normal direction. This basic technique was originally described in [14]. Unfortunately, most models are made of polygons that are widely varying in size, and for such models it is more difficult to create a variational implicit that faithfully matches a given polygonal model.

To produce high-quality implicit models from polygons, we have created an iterative method that repeatedly adds new constraints to a variational implicit representation in a manner that is guided

by a volumetric description of the model. To do so, we use a voxelization process to create the volumetric model from the polygons. The volumetric description of the given model acts as an ideal (but storage-intensive) implicit representation of the model that we can use to compare against the current variational implicit surface. In addition to the volumetric model, we also use a signed distance function to measure errors in the current iteration and to place new boundary, interior and exterior constraints. We repeatedly add new constraints until the implicit model is a near match to the original model. In the next section we will discuss the creation of the volumetric model, the signed distance function and the error metrics for evaluating the model. Later we describe in detail how they are used to define new constraints to make a variational implicit surface that is a close match to our original polygonal model.

IV. Volumes and Error Metrics

We choose to convert the polygonal model into a volumetric model due to its convenience for rapid evaluation of inside/outside queries. The disadvantages of a volumetric model are storage and computation costs.

A. Voxelization of a Polygonal Mesh

The process of converting a polygonal model into a volumetric representation is much like scan-converting a two-dimensional polygon into a set of pixels. One way to perform two-dimensional polygon scan conversion the is to find where a scanline intersects the edges of the polygon and then use a parity count of the number of such intersections to determine if a pixel lies inside the polygon. Similarly, to voxelize a polygonal mesh, we cast a ray through the mesh and find all the places where the ray intersects the surface of the mesh. Any point along the ray can be classified as inside or outside the polygonal model based on a simple parity count. To create a full volume, we cast a grid of parallel rays through the mesh and regularly sample the points along these rays. Each sample becomes one voxel. To minimize aliasing artifacts we perform supersampling and filtering so that the

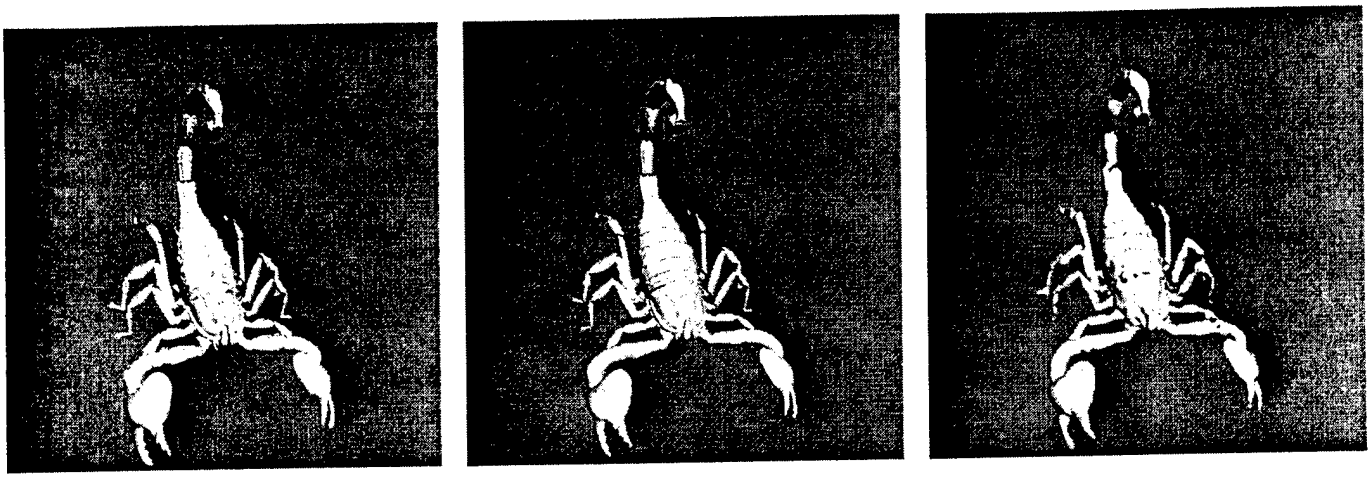


Fig. 2. Original polygonal model of a scorpion (left), intermediate volumetric representation (center), and final implicit surface (right) generated by our algorithm. The volumetric representation captures all but the finest detail, such as the hairs on the tail. Our algorithm refined the implicit surface using the volumetric model as the goal.

final densities vary continuously between zero and one. Further details of the voxelization process, including variations to handle troublesome meshes, can be found in [15]. For example, Figure 2 shows the original polygonal scorpion model on the left, the intermediate volumetric model in the center, and the final implicit representation generated by our algorithm on the right. The intermediate volumetric model captures all but the finest detail, such as the hairs on the tail, and serves as the goal surface for the surface evaluation and refinement. Note that if one has a volumetric representation of a given model, our method can be used directly to produce an implicit surface. Unfortunately most models do not originally come in a volumetric form, hence our need for conversion from polygons to voxels.

B. Signed Distance Transform

We use a signed distance transform to measure the error between our volumetric model and a given implicit representation. We use the voxelization of a given object as an inside-outside function of the object, and for this purpose we clamp all densities to either zero or one. A *distance transform* of an inside-outside function is the distance of a point to the nearest boundary (the transition regions between densities of zero and one). A *signed* distance transform negates the distances of those points that are outside the object.

Calculation of the signed distance transform for a large voxel volume can be expensive. Naively, the

running time for an $n \times n \times n$ volume is $O(n^6)$. We use a three-dimensional version of Danielsson's method for computing Euclidean distances [16] to achieve a running time of $O(n^3)$. This method requires making a small fixed number of sweeps through the entire volume, and at each voxel only a few neighbors are examined. The final result is a set of distances at each voxel that is a close approximation to the (signed) Euclidean distance to the nearest boundary.

C. Error Metric

To guide our surface creation method, we use a metric to evaluate how closely our current variational implicit surface matches the original data. Let ∂f_{curr} be the set of boundary voxels in the volumetric representation of the current implicit surface, and let ∂f_{goal} be the boundary voxels of the goal, that is, the volumetric representation of the original data. We want ∂f_{curr} to equal ∂f_{goal}, and we can measure the symmetric difference by the Hausdorff metric

$$H = \max \left[\max_{x \in \partial f_{goal}} \left(\min_{y \in \partial f_{curr}} ||x - y|| \right), \max_{x \in \partial f_{curr}} \left(\min_{y \in \partial f_{goal}} ||x - y|| \right) \right]. \tag{7}$$

The Hausdorff metric is zero if and only if ∂f_{curr} and ∂f_{goal} are identical. Furthermore, we can identify the voxels x farthest from the other surface and refine the surface by placing constraints at those locations. Using the signed distance functions for the goal and current surfaces as lookup tables, the new constraint location can be defined concisely and calculated efficiently as

$$C_{new} = \arg\max \left[\max_{x \in \partial f_{goal}} |sd_{curr}(x)|, \max_{x \in \partial f_{curr}} |sd_{goal}(x)| \right]. \tag{8}$$

The error metric has a two-fold purpose: to evaluate the attempted fit and to suggest where to refine the implicit representation further. Figure 3 illustrates the use of the metric as part of our algorithm on a two-dimensional teapot. Note that for 2D objects, the iso-valued set is one or more closed contours. In the black and white portions of this figure, black denotes interior (positive function values) and white denotes exterior (negative values). Images (a)-(e) show the refinement at the third

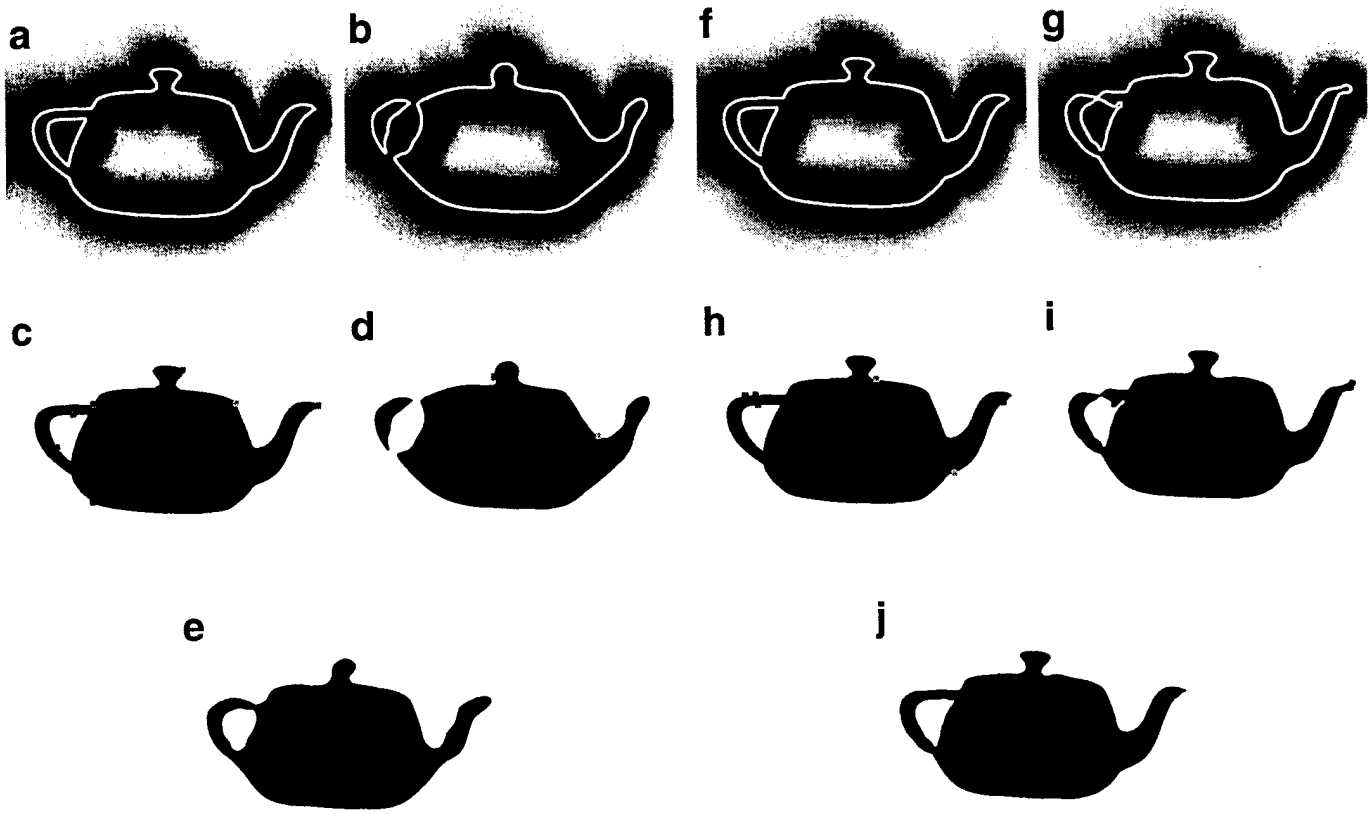


Fig. 3. Creating a two-dimensional implicit curve: To find the locations with greatest errors, we walk around the boundary of the goal curve and see how far it is from the boundary of the current curve. This query is done via a signed distance lookup, seen in image (a). Image (b) shows the symmetric counterpart, walking around the boundary of the goal curve and using the signed distance of the current curve. Locations of greatest errors become new constraints, shown in magenta. Image (c) shows the new boundary constraints and image (d) shows the new interior and exterior constraints. Image (e) shows the new curve after these refinements. Images (f)-(j) show the method applied to a later iteration of the algorithm.

iteration. Image (a) shows the signed distance function of the current implicit curve ($sd_{curr}(x)$) with the boundary of the goal (∂f_{goal}) overlaid. Similarly, image (b) shows $sd_{goal}(x)$ with ∂f_{curr} overlaid. Positive values of the signed distance are blue and negative red. As the magnitude increases, the colors go from dark to light. To find the locations to place new constraints, we simply walk around the overlaid boundaries in images (a) and (b) and choose points with the brightest background color (furthest distance from the other curve). The newly chosen constraints are shown as magenta dots in image (c) (boundary constraints) and image (d) (interior and exterior constraints). The implicit curve with these refinements is in image (e). Images (f)-(j) show the the respective information for

the refinement at the seventh iteration. After later iterations, the implicit curve becomes nearly indistinguishable from the original data.

V. Iterative Improvement of Model

We will now describe how the above tools allow us to model implicit surfaces. By using equation 8 to find candidate locations for more constraints, we can design an iterative algorithm to refine the implicit surface. Below is pseudocode for our algorithm.

Algorithm `MakeImplicitSurface(Volume` f_{goal}`, SDFunc` sd_{goal}`)`
Begin
  `Constraints = InitialConstraints(50)`
  Repeat
    f_{curr} `= MakeImplicit(Constraints)`
    sd_{curr} `= SignedDist(`f_{curr}`)`
    `GenerateCandidateConstraints()`
    Repeat
      `PruneCandidateList()`
      `NewCandidate = SelectHighestError()`
      `Constraints.add(NewCandidate)`
    Until `NoMoreCandidates`
  Until `DoneRefining`
End

First a number of initial constraints are chosen (in this case 50), a process which we discuss further in subsection V-A. Then, for each iteration, the following steps are taken: The implicit surface for the current set of constraints is generated by solving matrix equation 6. The resulting implicit function is evaluated over the volume to obtain f_{curr} (see subsection V-B), and the signed distance sd_{curr} of this function is also calculated. New candidate constraints are selected based on equation 8. Candidates may be pruned due to their proximity to other constraints or for other reasons, discussed further in subsection V-C. The candidates that are not pruned become new constraints, and the evaluation-refinement process repeats. The algorithm terminates when the implicit surface fits f_{goal} well enough,

and the criteria for termination is discussed in subsection V-D.

A. Initialization

The algorithm needs to start with an initial guess for the implicit surface before it can begin refining. We need to decide how many initial constraints to place — we want to create a reasonable first attempt, but we don't want to overwhelm the system with too many constraints. To get a good balance between these two extremes, we sample fifty boundary constraints from the points on the surface ∂f_{goal}. Three-dimensional Poisson-disk sampling ensures an approximately equal spacing of constraints. Constraints that are close to each other tend to have more influence on the rest of the system and can cause matrix equation 6 to be ill-conditioned. Other methods such as a point-repulsion technique might yield a more regular distribution of initial constraints across the surface, but we have found that our simple initialization technique is quite satisfactory.

In addition to choosing boundary constraints, we place non-zero constraints to indicate what portions of space are interior or exterior. It is essential to have these further specifications; otherwise, the isosurface could fit the goal exactly but the implicit function might be inside-out. These non-zero constraints are weighted by the actual signed distances; constraints more distant from the surface have larger magnitudes. For each of the boundary constraints that we have selected, we follow a path that follows the gradient of the signed distance function until we reach a local extremum. An interior or exterior constraint is then placed at this location. To decide whether to traverse the gradient uphill or downhill, we try both directions and pick the longer path. Shorter paths are usually in the direction that is locally convex. By selecting the longer path, our interior and exterior constraints then tend to "fan out" instead of getting clustered in ridges or valleys of the signed distance function. Because the implicit surface is smooth (locally planar), placing the interior or exterior constraints along the gradient of the signed distance function not only tells the implicit surface what direction is outside but also suggests the surface normal.

B. Implicit Function Evaluation

Given a current set of constraints, we solve the variational problem to obtain the basis-function weights for the corresponding implicit surface. Then the implicit surface is evaluated throughout the volume to find the boundary voxels ∂f_{curr} and the signed-distance function sd_{curr}. Once we classify the boundary voxels and then compute signed distance function, we can evaluate the error metric described in Section V.

Evaluating the implicit surface throughout the volume can be costly. Although surface-following isosurface-extraction techniques can reduce the evaluations of the implicit function by an order of magnitude, they make assumptions about topology; for example they may miss a detached portion of the surface. We wish ∂f_{curr} to capture all connected components, as they may indicate error in the current implicit surface. Because the radial basis functions we use have infinite support, a refinement in one location could create larger errors elsewhere. It is therefore especially critical to not overlook any regions in the evaluation. However, we do not want to evaluate 1000 radial basis functions over all the voxels in a $200 \times 200 \times 200$ volume, which would be computationally expensive. Our solution is to sample the volume finely in a thin shell around the goal boundary voxels and to sample coarsely elsewhere, then sampling more finely if we detect a boundary. First we sample the volume at coordinates that are all multiples of four. If any $4 \times 4 \times 4$ cube does not have its eight vertices entirely in the interior or exterior, we sample that cube voxel by voxel. Likewise, if the cube is within eight voxels of a boundary of f_{goal}, we sample it voxel by voxel. Otherwise, the $4 \times 4 \times 4$ cube is filled uniformly.

C. Refinement

Now that the boundary voxels can be evaluated by the metric, we can add constraints to refine the implicit surface further. We want to avoid adding constraints one at a time because performing an iteration per constraint would be quite costly. However, we also want to avoid having refinements

influencing each other and interfering. Likewise, making fine adjustments to regions of the surface could be ineffectual if coarse adjustments are made elsewhere on the surface because some adjustments can have a non-local effect.

We scan through ∂f_{curr} and ∂f_{goal} to find the voxel with the maximum error. In the event of a tie, we choose a boundary constraint over an interior or exterior constraint. The error for a voxel x in ∂f_{curr} is $|sd_{goal}(x)|$, and the error for a voxel y in ∂f_{goal} is $|sd_{curr}(y)|$. We will introduce the notation $|sd(x)|$ to represent both these cases. Searching for the maximum error is equivalent to walking along the overlaid boundaries in Figure 3 (a) and (b) and finding the largest magnitude (lightest background color).

We pick our new constraints from the boundary voxels ∂f_{curr} and ∂f_{goal}. Constraints added from ∂f_{goal} are boundary constraints. To prevent artifacts from the voxelization appearing, these constraints are actually placed at sub-voxel precision according to the densities of the voxels. Constraints from ∂f_{curr} are interior or exterior constraints, and take on the values given by the signed distance function of f_{goal}.

Not all candidate constraint locations are beneficial. We take our current set of candidate constraints and prune the list to avoid redundant or counterproductive constraints. To avoid having matrix equation 6 become ill-conditioned, we discard candidates less than one voxel from any pre-existing constraint. Boundary voxels with errors less than half the maximum error at the current iteration are too fine an adjustment, so those voxels are eliminated from the candidate list. This pruning results in only one constraint added per error so that no unnecessary constraints are placed. In the event that only one constraint did not fix the error, more constraints will be added there at later iterations. We eliminate any candidate that is within $2 \times sd(x)$ of a voxel x where a constraint was added on the current iteration. This distance restriction, along with the greedy approach of adding the constraints with greatest errors first, guarantees that for all i, the circles of radius $sd(x_i)$ centered at x_i will be disjoint.

D. Termination

Finally we discuss how the algorithm terminates. Empirically we have found that the models tend to refine themselves quickly at first and then slow as they converge to the goal models (see Figure 1). We terminate the algorithm under four conditions. The algorithm terminates if the model has reached a maximum error of one voxel, if a model has not improved in the previous four iterations, if too many iterations have passed (we use 30), or if too many constraints have been placed (we use a maximum of 5000). If successive models have the same Hausdorff error, we pick the best model based from a similarly derived RMS error.

VI. Program Parameters

Our algorithm has several parameters that govern its behavior. We will discuss each of these parameters and show that the quality of the results are largely insensitive to their values.

A. Volume Resolution

Because our algorithm attempts to fit an implicit surface to a signed distance function over voxels, the performance is dependent on the voxel resolution. A low-resolution volume might not capture much of the detail from the original polygonal model, and a high-resolution volume can be computationally expensive. For the models shown in Section VII, we use high-resolution volumes, making sure that the volumetric models lose little detail from the polygonal originals. However, our method can also make implicit surfaces out of coarse volumetric data. In this case, much detail cannot be captured in the implicit surface because it is not present in the volumetric model, but the implicit surfaces can be computed in mere minutes.

Table I shows the results from varying the voxel resolution. For the lowest-resolution models, our algorithm rapidly generates a nearly exact fit, in part because most of the high-frequency details from the original models are smoothed away in the volumetric models. Figure 4 shows the volumetric models and resulting implicit surfaces for models of a horse and Spock's head at two low voxel resolutions.

TABLE I

VOLUME RESOLUTION

Model	Size of Volume	Iterations to Finish	Max Error (in voxels)	RMS Error (in voxels)	Total Constraints	Total Time (h:m)
Bunny	$60 \times 70 \times 69$	12	1	0.1055	742	7
Bunny	$99 \times 120 \times 119$	13	1	0.1399	3166	5:00
Bunny	$138 \times 170 \times 168$	16	2	0.2798	1501	2:27
Horse	$104 \times 70 \times 119$	11	1	0.1416	1358	1:19
Horse	$195 \times 120 \times 228$	18	2	0.1773	3952	4:50
Horse	$286 \times 170 \times 337$	17	2	0.3003	2641	5:21
Spock	$69 \times 69 \times 76$	12	1	0.1591	990	15
Spock	$118 \times 120 \times 135$	13	2	0.1815	3742	3:45
Spock	$168 \times 170 \times 193$	18	2	0.2949	2543	5:57

B. Number of Initial Constraints

The final implicit surface is dependent on the number of constraints used to construct the initial surface before the refinement passes. (For all the examples shown in Section VII, 50 boundary con-

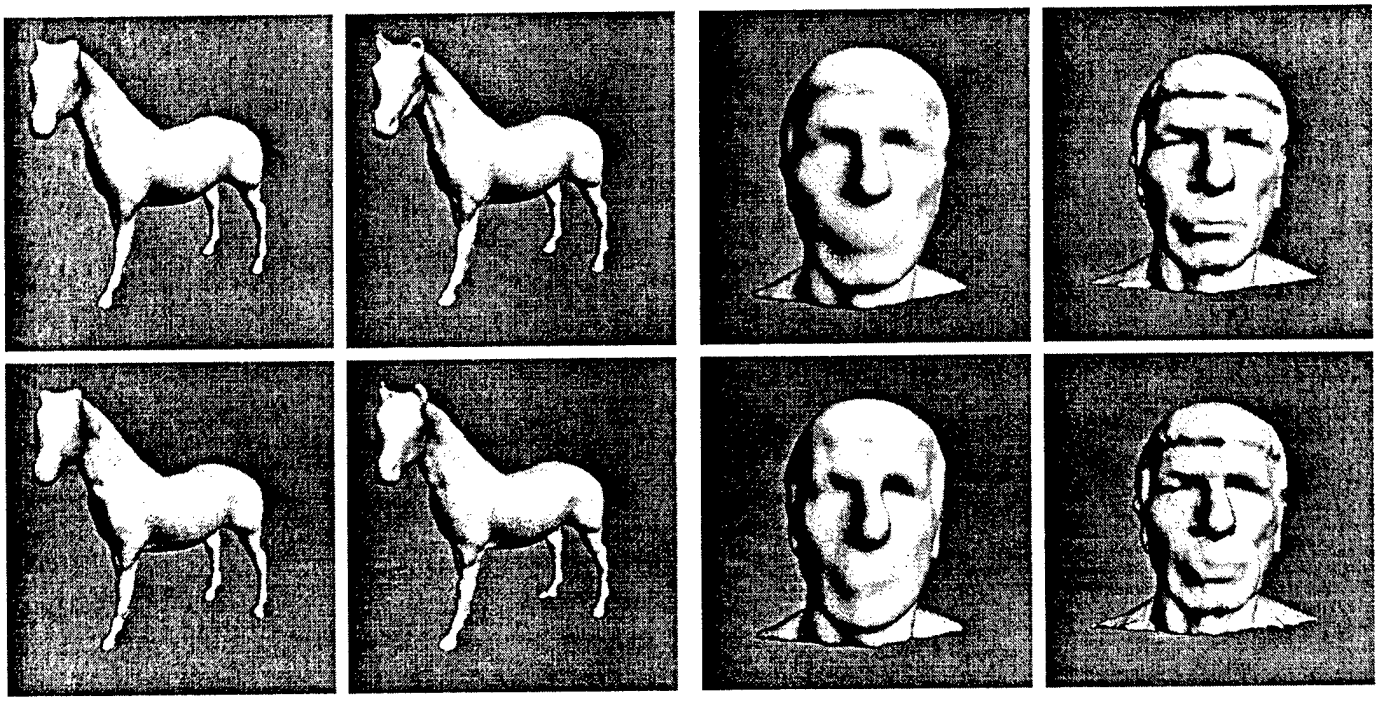


Fig. 4. Top row: low-resolution volumetric representations of the horse ($104 \times 70 \times 119$, $195 \times 120 \times 228$) and Spock ($69 \times 69 \times 76$, $118 \times 120 \times 135$). Bottom row: respective implicit surfaces generated by our algorithm.

TABLE II

NUMBER OF INITIAL CONSTRAINTS

Model	Initial Constraints	Max Error	RMS Error	Final Constraints	Max Error	RMS Error	Iterations to finish	Total time (h:m)
Bunny	100	53	5.895	2556	2	0.0638	14	7:02
Bunny	200	36	5.335	2515	2	0.2715	16	6:15
Bunny	400	18	2.869	2145	2	0.2655	11	6:08
Bunny	800	31	0.961	2774	2	0.2701	7	5:42
Bunny	1600	5	0.389	2671	2	0.2768	5	7:35
Bunny	3200	6	0.187	4057	2	0.0715	4	11:13
Triceratops	100	25	5.039	1803	2	0.2732	17	3:24
Triceratops	200	27	3.061	1953	2	0.0655	15	4:16
Triceratops	400	17	2.073	1910	2	0.0722	11	3:24
Triceratops	800	24	1.311	1906	3	0.2665	12	6:35
Triceratops	1600	16	0.709	2230	3	0.2658	10	7:13
Triceratops	3200	13	0.4712	3630	2	0.2549	7	12:25

straints and a total of 50 interior and exterior constraints were used for initialization.) Too few initial constraints could be inefficient because the algorithm has to spend time up front just trying to get a rough fit of the goal, such as trying to make the surface bounded. Too many constraints could result in too much time being spent on solving for unnecessary constraints. For example, in Table II, both the bunny and triceratops models took the longest to calculate when given the most initial constraints.

We experimented with seeding the implicit surface with values ranging from 50 boundary constraints to 1600 boundary constraints (100 to 3200 total constraints). For all the different configurations, the algorithm returned satisfactory results. Furthermore, simply adding more initial constraints does not alone produce a good surface. The results in Table II indicate that although the first iterations of the surfaces get better with more constraints, they are still nowhere near the desired fit. Figure 5 illustrates using varying numbers of initial constraints on the triceratops model. The implicit surfaces created from the initial constraints alone get consistently better with more constraints, but even with 3200 total constraints, key features such as the horns are still lacking. However, for all the triceratops tests, the final implicit surfaces captured these key features. Not only do these results indicate that the

amount of initial constraints is relatively inconsequential, but they also demonstrate why the iterative refinement process is necessary.

For all of the results shown in Section VII, we initialize with 50 zero constraints and 50 interior and exterior constraints. Using fewer initial constraints tends to give results more quickly, and with fewer constraints the algorithm still produces comparable results to those that started with more constraints.

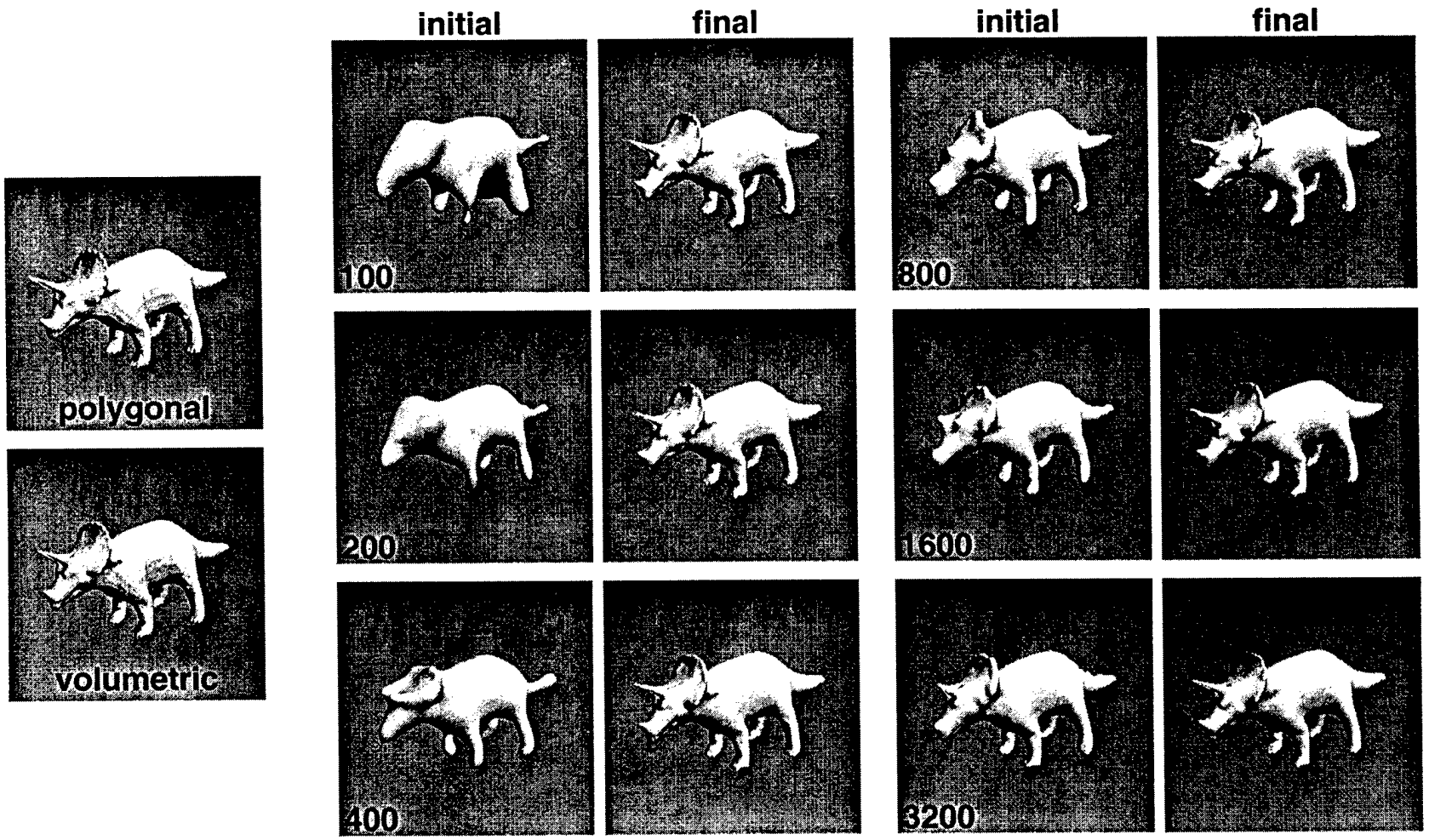


Fig. 5. Triceratops model, showing the effect of the number of initial constraints. At far left is original polygon model and the volumetric model. Pairs of models show results after just the initial constraints (left image of pairs, with number of constraints in corner), and after subsequent refinement until the algorithm terminates (right image of pairs).

C. Regularization Parameter

If we wish to approximate rather than exactly interpolate some of the constraints, we can use a slight adaptation of matrix equation 6 for creating the implicit surfaces. We modify the matrix's diagonal entries of the form ϕ_{ii} to $\phi_{ii} + \lambda_i$. Since $\phi_{ii} = 0$, a constraint with a non-zero λ is not interpolated exactly. Instead, the constraint's position becomes a weighted average of the desire for interpolation and the desire for regularization (smoothness). The experiments we describe next will explore the tradeoffs between approximation and interpolation.

First we conducted a simple test of changing the λ values of the interior and exterior constraints. For each of these constraints c_i, we evaluated whether $f(c_i)$ had the same sign as the constraint weight (positive for interior and negative for exterior). For $\lambda < 10^3$, less than one percent of the constraints had a different sign than when the implicit function was evaluated there. The boundary constraints for the implicit surfaces were still interpolated exactly and produced just about no noticeable change in appearance. However for $\lambda > 10^7$, even though only about twenty percent of the constraints had a different sign, the resulting models were inferior, especially around sharp features. When only affecting the interior and exterior constraints, making λ larger causes those constraints to have less of an effect on the implicit surface.

Next we tried varying λ for the boundary constraints as well. Instead of running our algorithm again with the new λ, we simply took the constraints already produced and solved matrix equation 6 with the new λ. Although our algorithm refined the set of constraints given the old $\lambda = 0$, the implicit surface with the new λ is still a nice fit. Although it may seem like a good idea to run the algorithm again with the new λ, the errors will be in different places (partially because of the Poisson-disk sampling), and hence, the results will be harder to compare.

Figure 6 shows enlarged images of the implicit horse with $\lambda = 0$ and $\lambda = 100$. Using $\lambda = 100$ made the ears of the horse nicer but sacrificed some detail around the nostrils. The smoother version of the leg with $\lambda = 100$ are more pleasing than the $\lambda = 0$ version, especially the hoof and the dimple on the

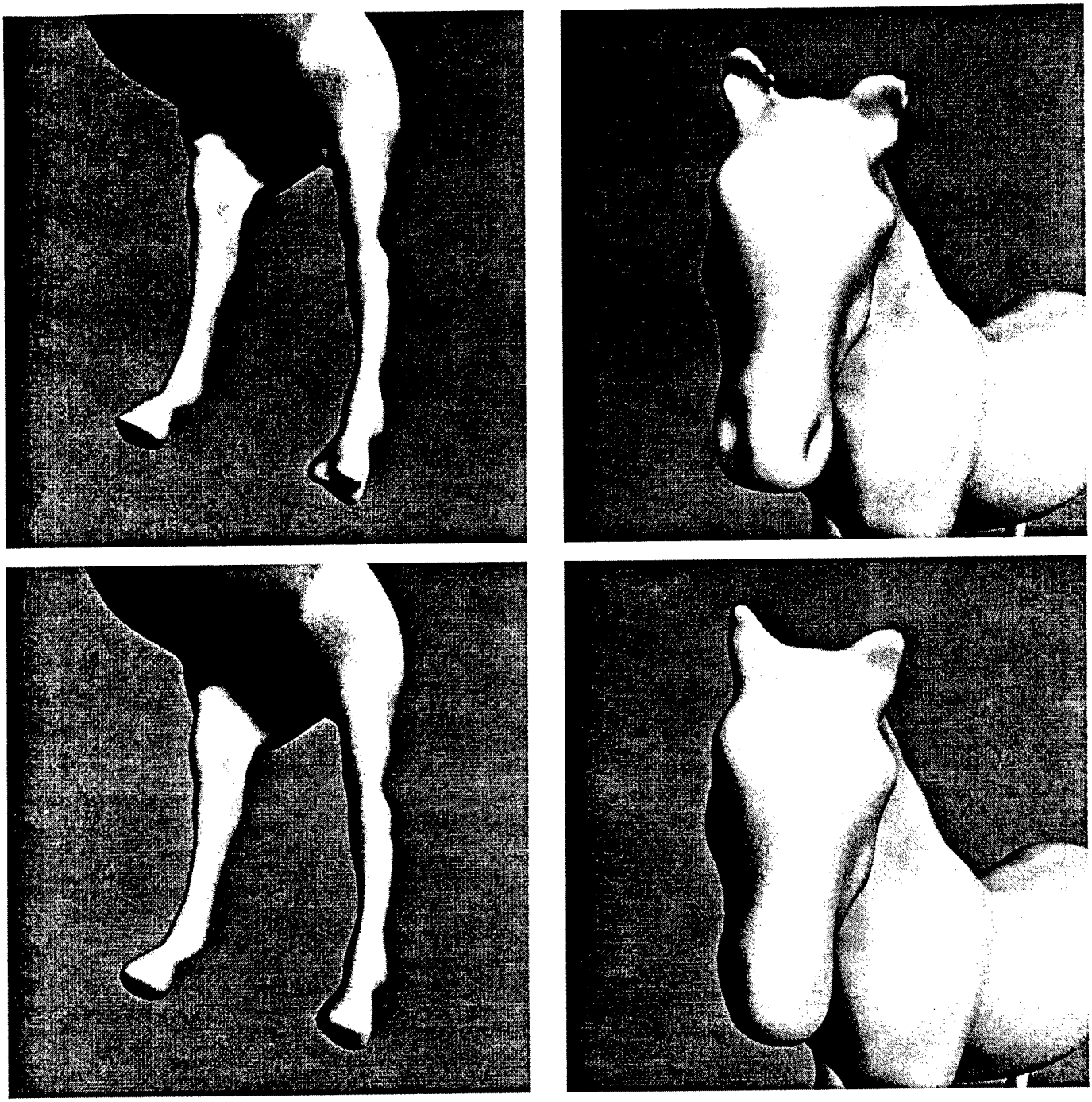


Fig. 6. Implicit horse: head and leg. Top shows $\lambda = 0$ (exact interpolation), and bottom shows $\lambda = 100$ (weighted average of exact interpolation and curvature minimization).

inner thigh. All of the results in Section VII use $\lambda = 0$; a user who wishes a smoother model can set λ accordingly.

VII. Results

The main goal of our research is to create an entirely automatic algorithm for creating implicit models from polygons, that is, to have a method that does not require human intervention and is robust. We tested our algorithm on a variety of polygonal models, and we are convinced from its performance that it will behave robustly for all but the most pathological of polygonal models.

Figure 7 and Figure 8 show the resulting implicit surfaces from our algorithm. The sub-images in these figures are arranged in pairs, with the intermediate volumetric models (the left sub-images in a pair) shown side-by-side with the resulting implicit surface create by our method (right sub-image of pair). We choose to compare with the intermediate volumetric model because the implicit surface can only capture features represented by it. For example, some surfaces that are paper-thin are problematic for our algorithm because the voxelization will alias those regions or not even capture them. However, we are not concerned by these pathological examples because they are difficult to represent well by any type of implicit surface.

Figure 7 shows our results on six high-resolution models obtained from laser-scanning. In all cases, our algorithm produced implicit surfaces that capture all of the main features of each model. The bunny and teeth implicit surfaces look nearly identical to their volumetric counterparts. The two head models are also close matches, although small but sharp creases such as wrinkles or eyelids are not always captured. The algorithm performs the least well on the Buddha model, apparent from both the image and the data in Table III. It is hard to fit a surface through the Buddha model because of all the detail, especially high curvature areas such as the folds on the robe. The implicit surface for the dragon model captures all of the macroscopic features but does not capture the fine scales. Perhaps more constraints could capture these very fine details, but the computational expense would be extreme. A more reasonable approach would be to encode the scales as a bump map or displacement map (see Future Work). We do not know of any other method for creating implicit surfaces that would give results comparable to those shown in this figure.

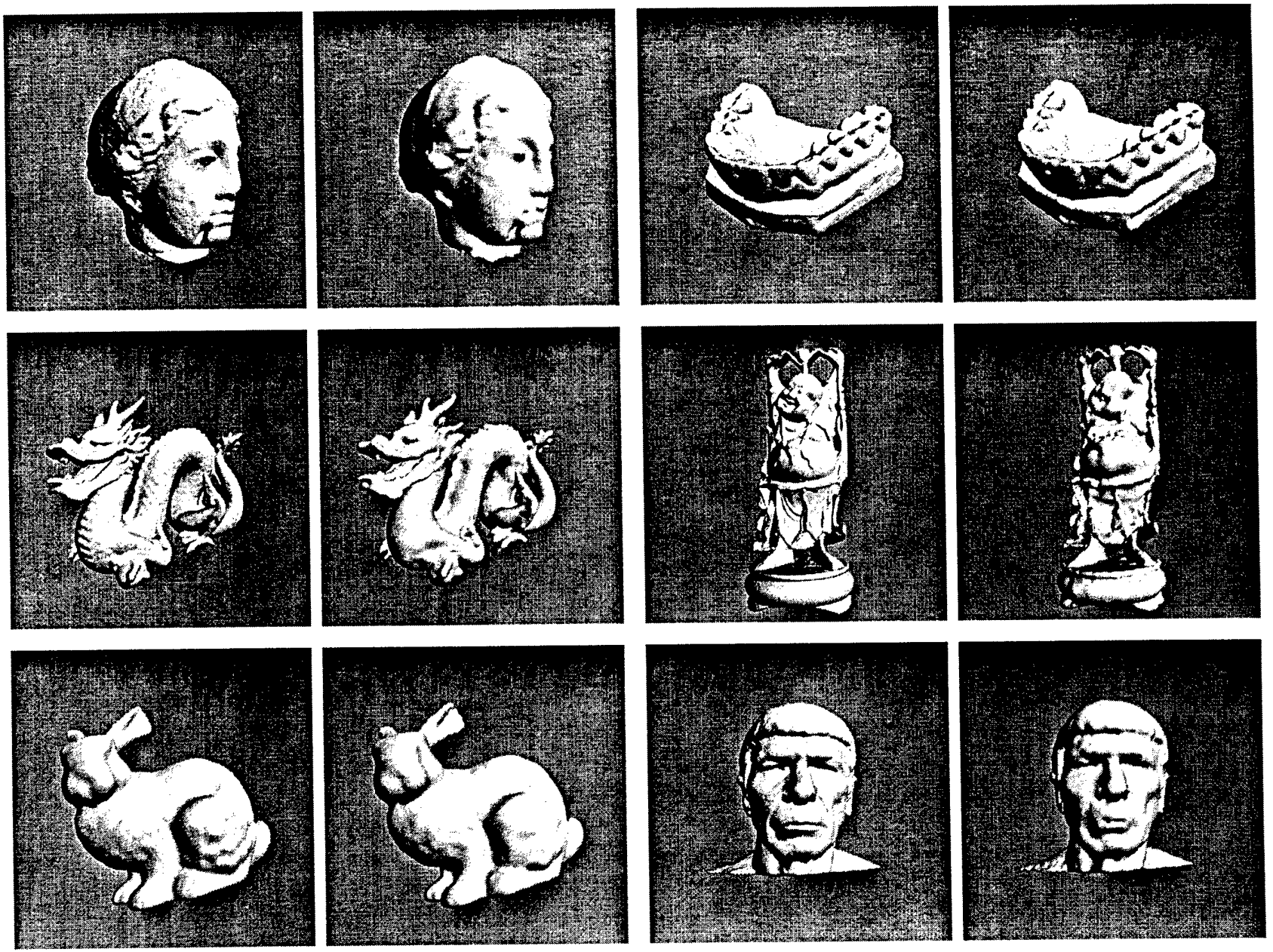


Fig. 7. Six large scanned polygonal models: intermediate volumetric representations (left images of pairs) and final implicit surfaces (right images of pairs) from our algorithm.

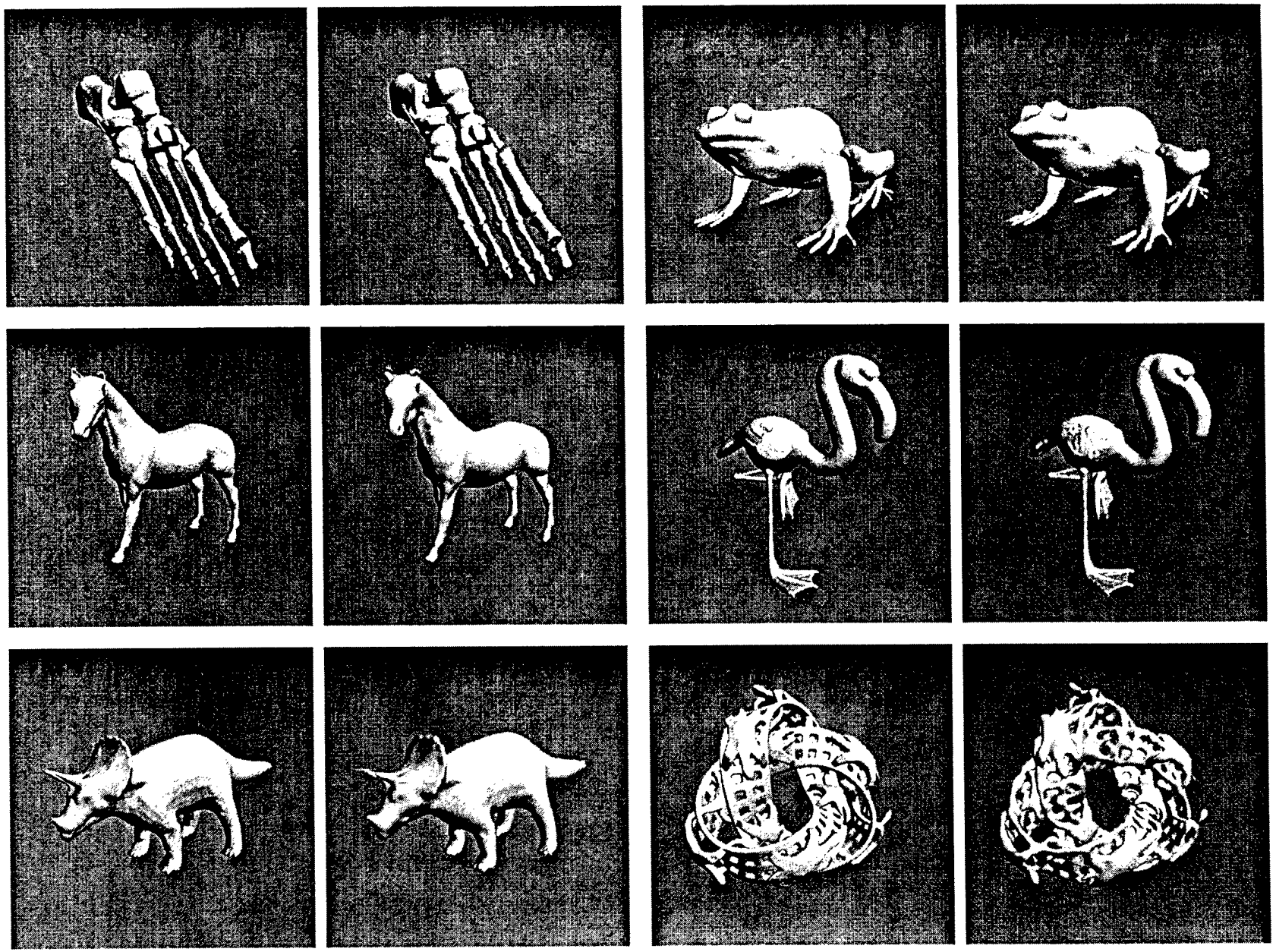


Fig. 8. Six models with thin or high-curvature regions: intermediate volumetric representations (left images of pairs)and final implicit surfaces (right images of pairs) from our algorithm.

Figure 8 shows our results on six polygonal models that contain high-curvature regions or thin surfaces. The triceratops model has long thin horns, and the horse model has long thin legs and small pointy ears. All of these features were captured by the implicit surface created by our algorithm. The foot model contains many long thin bones, but the most difficult part of the model is not the bones but the very thin gaps between the bones. The flamingo not only has wiry legs but also the thin foot webbing and wings. The wings are particularly hard to fit because they are separated from the rest of the body by a very small gap. The model with the ants crawling around the trefoil knot is also a very difficult model because of the high curvature around the holes. The isosurface has a high genus (many holes) – the algorithm has no knowledge of this fact but performs well nevertheless. For all of these models, despite their high-curvature features, the resulting implicit surfaces fit the original data quite well.

For more quantitative comparisons, Table III gives the numbers of iterations, the numbers of constraints in the final implicit surfaces, and the errors of the final implicit surfaces. Each final implicit surface was specified using roughly two to three thousand constraints and achieved a root-mean-squared error of less than one voxel. The Igea artifact model (upper left in Figure 7) took the fewest iterations; most likely the algorithm did not have to refine much because the model did not have many high-curvature regions. The scorpion and flamingo took the most iterations; we conjecture that the variational implicit surfaces had a difficult time fitting these goal models due to their high curvature.

Table IV shows the running times for the algorithm on all of the models. The running times ranged from a few hours to just over a day for the Buddha model. Most of the compute time for the algorithm was in the implicit function evaluation stage. Each evaluation of the implicit surface at a point requires $O(n)$ floating-point operations, where n is the number of constraints. Our evaluation method tests a shell of voxels, so the running time of the evaluation is $O(nx^2)$, where x represents one of the dimensions of the intermediate voxel volume. Unfortunately, the shell is rather thick, and in the later iterations, the number of constraints can get quite high. For many of our models, the

TABLE III

IMPLICITIZATION OF POLYGONAL MODELS

Model	Iterations to Finish	Zero Constraints	Interior Constraints	Exterior Constraints	Max. Error (in voxels)	RMS Error (in voxels)
Bunny	16	1718	63	734	2	0.2715
Dragon	22	1627	163	791	2	0.3329
Flamingo	26	1858	143	472	2	0.2726
Foot Bones	22	2307	138	681	2	0.3018
Frog	24	1900	137	791	2	0.2840
Happy Buddha	24	1788	196	648	4	0.6732
Horse	17	1829	88	724	2	0.3003
Igea Artifact	8	1591	54	842	2	0.3149
Scorpion	27	1992	179	483	3	0.7784
Spock	18	1444	81	1018	2	0.2949
Teeth Cast	17	2489	81	915	2	0.2655
Trefoil	23	2463	501	89	5	0.3844
Triceratops	17	1333	84	386	2	0.2732

TABLE IV

RUNNING TIME OF IMPLICITIZATION (IN HOURS:MINUTES)

Model	Size of Volume	Time until RMS Error < 2	Time until RMS Error < 1	Total Time
Bunny	$176 \times 220 \times 218$	1:15	1:45	7:02
Dragon	$113 \times 220 \times 163$	49	1:50	5:41
Flamingo	$237 \times 120 \times 288$	39	1:21	7:40
Foot Bones	$138 \times 320 \times 124$	47	1:30	6:46
Frog	$247 \times 220 \times 151$	1:44	3:46	16:07
Happy Buddha	$170 \times 170 \times 373$	6:15	15:42	24:47
Horse	$286 \times 170 \times 337$	1:58	3:01	5:21
Igea Artifact	$220 \times 220 \times 161$	14	52	2:16
Scorpion	$335 \times 220 \times 180$	1:40	2:57	3:56
Spock	$168 \times 170 \times 193$	27	1:18	5:57
Teeth Cast	$223 \times 170 \times 269$	1:09	2:44	11:35
Trefoil	$119 \times 220 \times 208$	2:11	7:30	9:23
Triceratops	$124 \times 320 \times 155$	41	1:03	3:24

last few iterations amounted to half the total running time or more. Because we wanted to err on the conservative side regarding evaluating the implicit function as exactly as possible, there may be a fair amount of room for speeding up the evaluation, such as reducing the thickness of the shell. Solving matrix equation 6 also is expensive for a large number of constraints. Using LU decomposition on a matrix with n constraints requires $O(n^3)$ operations. Because the evaluation was still more expensive, we did not focus on improving the matrix solution; using a method such as Jacobi iteration with the last set of weights as the initial value could accelerate the matrix solution. The timing information to achieve root-mean-squared errors less than two voxels and less than one voxel are also shown in the table; for most of the models, the times to reach these accuracies is substantially less than the total running times.

VIII. Conclusions and Future Work

To the best of our knowledge, our method is the first approach that automatically converts an arbitrary polygonal mesh to a smooth implicit surface. We have tested our method on a variety of complex polygonal meshes, and we are convinced empirically that it behaves robustly. Specifically, we feel that our method makes the following new contributions to computer graphics:

- Automatically converts any manifold polygonal mesh to a smooth implicit surface.
- Generates implicit surfaces from low-resolution data very fast.
- Provides a general framework for other basis functions or other implicit surface representations.

There are several potential directions for future work. One avenue is looking at classes of basis functions that either have finite support or approach zero asymptotically. We will still be able to specify constraints to be interpolated exactly, but the basis functions will not minimize the aggregate curvature discussed in Section III. However, using such basis functions should speed up the algorithm because the matrix will be sparse and implicit function evaluations will only have to use nearby basis functions.

More work still needs to be done on representing high-frequency features such as thin surfaces and fine detail. Thin surfaces might be better represented by other radial basis functions or by basis functions that could be weighted along principal directions (much like the covariance matrix for a Gaussian). We do not feel that adding more constraints is the right way to capture fine detail. Rather, fine detail could be added by local-influence implicit surfaces. Another logical path to explore is adding fine features (such as scales on the dragon) using normal or displacement maps.

IX. Acknowledgements

We would like to thank Hughes Hoppe for providing the Spock dataset. We would also like to thank James O'Brien and the Geometry Group at Georgia Tech for helpful advice and discussions. This research was funded in part by ONR grant N00014-97-1-0223.

References

[1] Bradley Payne and Arthur Toga, "Distance field manipulation of surface models," *IEEE Computer Graphics and Applications*, vol. 12, pp. 65–71, 1992.

[2] John Hughes, "Scheduled fourier volume morphing," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 92)*, 1992, pp. 43–46.

[3] Greg Turk and James F. O'Brien, "Shape transformation using variational implicit functions," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 99)*, 1999, pp. 335–342.

[4] Mathieu Desbrun and Marie-Paule Gascuel, "Animating soft substances with implicit surfaces," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 95)*, 1995, pp. 287–290.

[5] Marie-Paule Gascuel, "An implicit formulation for precise contact modeling between flexible solids," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 93)*, 1993, pp. 313–320.

[6] James F. Blinn, "A generalization of algebraic surface drawing," *ACM Transactions on Graphics*, vol. 1, no. 3, pp. 235–256, 1982.

[7] Andrew P. Witkin and Paul S. Heckbert, "Using particles to sample and control implicit surfaces," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 94)*, 1994, pp. 269–278.

[8] Joshua Levin, "A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces," *Communications of the ACM*, vol. 19, pp. 555–563, 1976.

[9] Geoff Wyvill, Craig McPheeters, and Brian Wyvill, "Data structures for soft objects," *The Visual Computer*, vol. 2, no. 4, pp. 227–234, 1986.

[10] Jules Bloomenthal, "Convolution surfaces," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 91)*, 1991, pp. 251–256.

[11] Shigeru Muraki, "Volumetric shape description of range data using 'blobby model'," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 91)*, 1991, pp. 227–235.

[12] Eric Bittar, Nicolas Tsingos, and Marie-Paule Gascuel, "Automatic reconstruction of unstructured 3d data: Combining a medial axis and implicit surfaces," in *Computer Graphics Forum (Proceedings of Eurographics 95)*, 1995, vol. 14, pp. 457–468.

[13] Jules Bloomenthal, Ed., *Introduction to Implicit Surfaces*, chapter Convolution of skeletons, pp. 222–241, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[14] Greg Turk and James F. O'Brien, "Variational implicit surfaces," Tech. Rep. 15, Georgia Institute of Technology, 1999.

[15] F.S. Nooruddin and Greg Turk, "Simplification and repair of polygonal models using volumetric techniques," Tech. Rep. Technical Report GIT-GVU-99-37, Graphics, Visualization and Usability Center, Georgia Institute of Technology, 1999.

[16] Per-Erik Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227–248, 1980.