Creating Smooth Implicit Surfaces from Polygonal Meshes

Gary Yngve
gary@cc.gatech.edu

Greg Turk
turk@cc.gatech.edu

Abstract

Implicit surfaces have long been used for a myriad of tasks in computer graphics, including modeling soft or organic objects, morphing, and constructive solid geometry. Although operating on implicit surfaces is usually straight-forward, 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 models. Whereas much previous work has been done with primitives such as "blobbies," we use surfaces based on a variational interpolation technique (the 3D generalization of thin-plate interpolation). Given a polygonal mesh, we convert the data to a volumetric representation and use this as a guide to create the implicit surface iteratively. Carefully chosen metrics evaluate each intermediate surface and control further refinement. We have applied this method successfully to a variety of polygonal meshes.

1 Introduction

The task of constructing smooth surfaces is ubiquitous throughout computer graphics. The main representations that are used are parametric surfaces and implicit surfaces. Parametric surfaces specialize in the ability to identify specific locations on a surface. Although implicit surfaces are at a disadvantage here, they provide more information than parametric surfaces about their interior and exterior. A parametric surface is much like a coastline with mileage markings, whereas an implicit surface is a coastline plus the height of the land both above and below the water. Determining interior and exterior for an implicit surface is trivial; also, implicit surfaces are innately manifold.

Because points can be evaluated easily as being inside or outside an implicit surface, many applications considered challenging for parametric surfaces (including polygonal meshes) become simple when applied to implicit surfaces. Constructive solid geometry reduces to just looking at the signs of the implicit functions — there is no need to consider geometry such as convexity or topology such as genus. Operations on implicit surfaces that may cause a change in genus of the isosurface have simple implementations because the operations affect every point in space — on the isosurface, inside, and outside. Interpolating between implicit surfaces effectively morphs between the two shapes of arbitrary manifold topology. Implicit surfaces can be collided and deformed, with fusions and separations done automatically. Often in graphics, implicit functions are created by summing several continuous and infinitely differentiable functions, yielding surfaces that are smooth and seamless. The forms that they can represent are useful for modeling organic shapes and for some classes of machine parts that require blends and fillets.

Although implicit surfaces have these benefits, they can be difficult to model. Parametric surfaces allow for local finite control, while infinitely differentiable radial basis functions, which are often used as primitives for implicit surfaces, can have non-obvious influences on surface position. Modeling with "blobbies"[2] suffers from this problem since each blobby primitive only indirectly influences the position of the isosurface.

Rather than using the more traditional blobby primitives approach to implicit surfaces, 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 provides the guarantee that the surface will interpolate a given set of constraint points. In addition to this interpolating property, variational implicit functions are smooth when the basis functions are appropriately chosen to satisfy an energy equation that is 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 four stages of this iterative process for a triceratops model.

We have focused on the creation of implicit models from polygonal meshes because of the large numbers of existing high-quality polygonal meshes. Having an automatic conversion procedure from meshes to implicits should provide a pathway towards creating a large library of implicit surfaces. Moreover, all of the interactive modeling tools for creating polygonal meshes could then used

Figure 1: Implicitization of a Triceratops: initial attempt, iteration 4, iteration 10, iteration 18 of refining.

to create implicit surfaces. This ability means 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 discuss previous work in implicit-surface modeling in Section 2. In Section 3, we explain the variational implicit surface model. Then in Section 4, we introduce a set of tools that will be used by the algorithm. We present the algorithm in Section 5 and then show the results in Section 6. Finally we conclude and discuss future work in Section 7.

2 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. Blinn generalized these implicit surfaces for the purpose of modeling molecules [2]. Basing his model from electron densities, he developed the blobby molecule model, which consists of Gaussian-like primitives blended together.

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

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 [10] are widely used in the computer graphics community.

Another genre of implicit surfaces is the convolution surface [3]. A skeleton is convoluted with a kernel, such as a Gaussian, to produce a smooth solid. The convolution causes joints to be smoothed. However, the resulting surface can have undesirable bulges, especially where joints meet. The skeletons for the convolutions can also be 2D and 3D shapes, which tends to eliminate the problems with bulging. In either of these two classes of surfaces, the control points do not necessarily lie on the resulting surfaces.

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 [9]. Particles diffuse across the surface and are created and destroyed as necessary. They implemented their technique on blobbies; their technique is adaptable to variational implicit surfaces as well. However, for creating complicated models, more automated methods are needed.

Muraki developed a method to approximate range data by a blobby implicit surface [6]. Muraki's method incrementally added 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 function. 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 [1]. 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).

This brief summary barely scratches the surface of work on implicits in computer graphics. For an excellent overview of the area, see the book by Bloomenthal et al. [3].

3 Variational Implicit Surfaces

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

3.1 Basic Formulation

Variational implicit surfaces are created by solving a scattered data interpolation problem [8]. The particular so-

lution 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. Variational implicit 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 resulting implicit surface will exactly pass through these points. In addition, we can specify that certain points will be interior or exterior or 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). The radial basis function that has given us the best results is $\phi(\mathbf{x}) = |\mathbf{x}|^2 \log(|\mathbf{x}|)$. Using this radial basis function, the function that we wish to create has the form

*-0.10in

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

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.

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 1 for $f(\mathbf{c}_i)$, which gives:

$$h_i \quad = \quad \sum_{j=1}^{k} d_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}) \qquad (2)$$

Since this equation 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 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 the linear system can be written as follows:

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

We used LU decomposition to solve this system for all of the examples shown in this paper. With the coefficients from the matrix solution, evaluating the implicit function from Equation 1 is simple.

3.2 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 polygonal models, 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 [8]. 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 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 that they help to compute. Following this, 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.

4 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. However, our largest volumes are about $200 \times 200 \times 200$ and our running times are still quite reasonable.

4.1 Voxelization of a Polygonal Mesh

The process of converting a polygonal model into a volumetric representation is much like scan-converting a 2D polygon into a set of pixels. In 2D polygon scan conversion the important operation is to find where a scanline intersects the edges of the polygon, and to use a parity

count of the number of such intersections to determine whether a pixel is inside or outside 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 final densities vary continuously between zero and one. Further details of the voxelization process, including variations to handle troublesome meshes, can be found in[5].

4.2 Signed Distance Function

We use a signed distance transform to measure the error between our original 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. If one views the signed distance transform of a 2D domain as a height field, it resembles a mountain range with straight-walled ridges and valleys.

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 3D version of Danielsson's method for computing Euclidean distances [4] to achieve a running time of $O(n^3)$. This method requires making a few 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.

4.3 Error Metrics

To guide our surface creation method, we make use of a metric to evaluate how closely our current variational implicit surface matches with the original data. We are concerned most about the surface points, i.e. the voxels with signed-distance values of zero. We will use ∂ symbol to denote these boundary voxels. Using the signed distance transform, we can quickly calculate how far a given boundary voxel is from the other surface's boundary. We calculate this distance for all the boundary points in the original model and similarly for all of the boundary points on the current the implicit surface. The incremental algorithm uses these measurements to find the next places at which to improve the surface. We define our error metric as follows, basing it on the Hausdorff metric. Note that the evaluation is fast because it consists of lookups of the signed distance functions. We use f_{goal} to represent the original model that we wish to approximate, f_{curr} to represent the current implicit surface, and sd_{goal} and sd_{curr} as their respective signed distances.

$$E = \max \left[\max_{x \in \partial f_{goal}} |sd_{curr}(x)|, \max_{x \in \partial f_{curr}} |\mathrm{sd}_{goal}(x)| \right] \tag{3}$$

We use E to find candidate locations for new constraints. In Figure 2 we illustrate the use of the metric E in two dimensions. 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). Part (a) of the figure shows the goal shape, and part (b) shows an implicit contour that we wish to improve. Part (d) shows the boundary (as thin lines) of (b) overlaid atop the signed distance of (a), and part (e) is similar but reverses the roles of (a) and (b). Our algorithm traverses the boundary overlaid in (e) and finds places where the signed distance takes on large values (the darkest colors). These large signed distance values are then used as additional constraints for the next iteration of the algorithm, and the magenta dots in (a) are exactly such constraints that were found from (e). Similarly, we traverse the boundary of (d) and gather additional constraints (not shown). Adding those magenta constraints shown in (a), we obtain a new implicit contour, shown in (c). After a few more iterations, we obtain (f), a near facsimile to the goal shape (a).

Although we find the above distance metric to be very useful, it has a few shortcomings. The metric E only identifies absolute errors. For example, consider the picture of the triceratops in Figure 3. A slightly thicker or thinner belly would probably not be noticeable. However, slightly thicker or thinner horns would be quite noticeable. Using only E to refine the implicit surface would result in many unnecessary constraints being added around the belly, possibly overconstraining the system. In addition, because evaluating an implicit surface at a point is $O(n)$, where n is the number of primitives, and solving the variational problem as a matrix equation is even more expensive, unnecessary constraints can become costly. We use a modified version of the above metric to eliminate this problem.

We use a discrete analog of the *radius of regularity* to tackle these issues of curvature and thinness. A shape is *r-regular* [7] if every circle of radius r tangent to the boundary of the shape does not contain points in both the interior and exterior of the shape. Our discrete analog $rreg()$

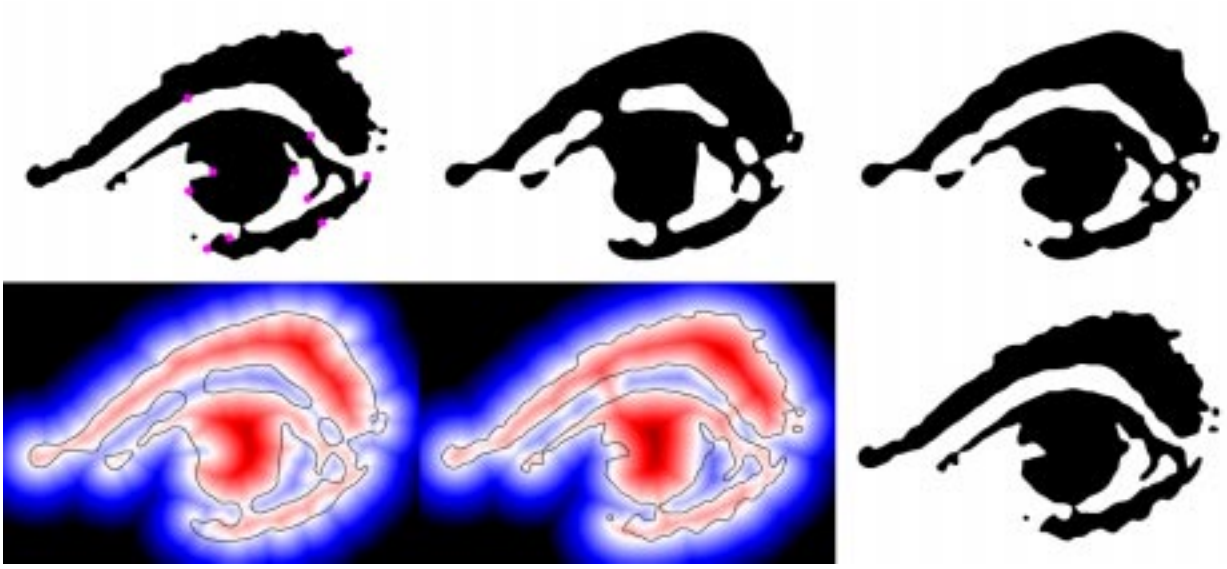

Figure 2: Creating a two-dimensional implicit function. First row: (a) the goal shape, obtained from blurring and thresholding the Graphics Interface logo, (b) implicit contour at iteration 3, (c) implicit contour at iteration 4. Second row: (d) signed distance of *a* with the boundary of *b* overlaid (red is positive, blue is negative, darker is further from 0), (e) signed distance of *b* with the boundary of *a* overlaid, (f) implicit contour upon completion after 14 iterations.

works as follows: From a point on the surface, proceed up the gradient of the signed distance function. When the distance traveled is greater than the closest distance from that point to the boundary (plus a two voxel tolerance), we stop and return that distance[1]. Using $rreg()$, our new weighted error metric becomes

$$E_w = \max\left[\max_{x\in\partial f_{goal}} \frac{|sd_{curr}(x)|}{rreg_{goal}(x)}, \max_{x\in\partial f_{curr}} \frac{|sd_{goal}(x)|}{rreg_{curr}(x)}\right] \quad (4)$$

An error of a few voxels near the belly would be given a small weight because of the large $rreg()$ values in that region. Errors of a few voxels in the redder areas of Figure 3 would be given a higher weight. Strictly scaling the object would respectively scale the radii, so E_w is scale-independent. Empirically, we have found that the areas of smallest $rreg()$ are the hardest regions for the implicit surface to fit. We use the metric E_w to help decide whether or not a candidate constraint should be used, and this is described further in Section 5.3.

5 Iterative Improvement of Model

We will now describe how the above tools allow us to model the implicit surfaces. Here is an outline of the algorithm:

[1]Due to discretization errors, this small tolerance is necessary. This is enough tolerance so that a large voxelized sphere is assigned a uniform radius of regularity over its entire surface.

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

5.1 Initialization

The algorithm needs to start with an initial guess at the implicit surface before we can start refining. We need to decide how many constraints to place — we want to create a reasonable first surface, but we don't want to overwhelm the system with too many constraints. To get a good balance between these two extremes, we select 50 boundary constraints randomly from the points on the surface ∂f_{goal}. In the event that two constraints are too close to each other we do not add the newer of the constraints. Constraints that are close to each other tend to have more influence on the rest of the system and

Figure 3: This figure illustrates the radius of regularity. Thin regions such as the dinosaur's collar and high-curvature regions such as the tips of the horns have a smaller radius and are colored red, and lower curvature regions are more green.

can cause the matrix to be ill-conditioned. Likewise, we do not add constraints where the radius of regularity is too small. A point-repulsion technique could balance the constraints throughout the surface, but we have found that our simple initialization technique is quite satisfactory.

In addition to placing boundary constraints, we also place non-zero constraints to indicate what portions of space are interior or exterior, and to give a general height relation between the constraints. For each of the boundary constraints that we placed as described previously, we follow the gradient of the signed distance function until we reach a local extremum. We then place an interior or exterior constraint at this location. To decide whether to place the interior or exterior constraint up the gradient or down the gradient, we traverse both ways and pick the longer path. Shorter paths are in the direction that is generally locally concave. Interior and exterior constraints then tend to "fan out" instead of getting clustered in ridges or valleys of the signed distance function.

5.2 Implicit Function Evaluation

We solve the variational problem for the current set of constraints to obtain the basis-function weights for the corresponding implicit surface. We then evaluate the implicit surface throughout the volume to find the boundary voxels ∂f_{curr} and the signed-distance function sd_{curr}. Using the boundary voxels and the signed-distance function, we can apply the metric E described in Section 4.

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. However we do not want to evaluate 500 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 cube does not have its eight vertices entirely in the interior or exterior or if it is within eight voxels of a boundary of f_{goal}, we sample that cube voxel by voxel. Otherwise, the $4 \times 4 \times 4$ cube is filled uniformly. This speedup brings the running time of implicit function evaluation on par with the other steps of the algorithm.

5.3 Refinement

Now that we can evaluate the boundary voxels according to the metric E, 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. Likewise, making fine adjustments to regions of the surface could be ineffectual if more coarse adjustments are made elsewhere on the surface because each adjustment has a global effect.

We scan through ∂f_{curr} and ∂f_{goal} to find the voxel with the maximum error. 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 2 (d) and (e) and finding the largest magnitude (darkest 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 soft interior or exterior constraints, and take on the values given by the signed distance function of f_{goal}.

We need to be careful about adding new constraints. Some constraints may be redundant; others are even counterproductive. We take our current set of candidate constraints and prune the list based on two criteria. We prune candidates based on distance from other constraints and based on the error metrics. First, we eliminate any candidate that is within $2 \times sd(x)$ of a voxel x where a

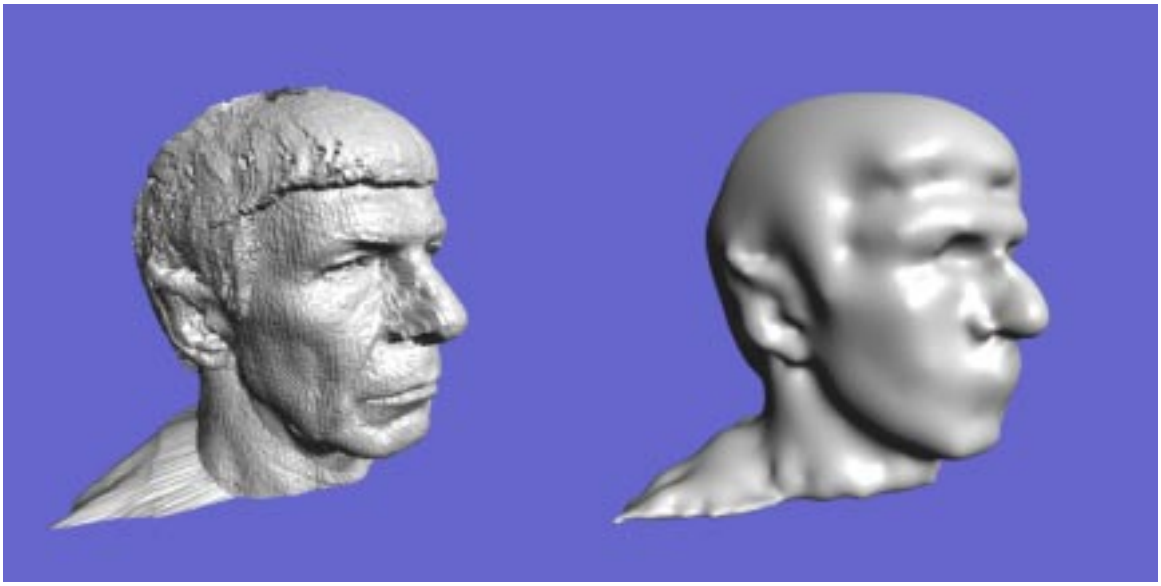

Figure 4: Spock mesh (left) and resulting implicit surface (right).

constraint was added on the current iteration. This distance restriction, along with 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. Second, if a candidate is less than two voxels from any pre-existing constraint, we remove it from the candidate list. Third, we consider boundary voxels with errors less than half the maximum error E (see equation 1) to require too fine an adjustment, so those voxels are eliminated from the candidate list. Finally, a candidate at a voxel that has a weighted error less than 0.125 is eliminated, as the difference is most likely unnoticeable. Filtering out based on the weighted error prevents many unnecessary points from being added.

5.4 Termination

Algorithm termination is an important issue. Empirically we have found that the models tend to refine themselves quickly at first and then reach a plateau. Adding too many constraints may even give worse results. We terminate the algorithm under three conditions. The algorithm terminates if the model has reached a certain level of accuracy by either of the error metrics, if a model has not improved from iteration i by iteration $i+4$, or if too many iterations have passed. If successive models have the same E, we pick the best based from a similarly-derived RMS error.

6 Results

We tested our method on a number of different models from a variety of sources. We used five models: Spock (Figure 4), the bunny, the foot bones, the horse (all in Figure 5) and the triceratops (Figure 1). Our method performed well on all of these models, in contrast to the simple method briefly described in Section 3.2, which produces unacceptable results for most of these models. Most of the noticeable errors fall in areas with a low radius of regularity, such as the sharp indentations by the bunny's leg or Spock's lip. Our method performs the least well on the foot model, which probably is not a good candidate for an implicit surface representation. The difficulty is the low radius of regularity where the joints meet — basically, two flat regions that are barely separated. The thin bones also are difficult, but thin features in other models, such as the horse's legs, are handled quite well.

Each model required around twenty iterations, using a few hundred constraints. Most of the constraints are boundary constraints, while just a few interior or exterior constraints are necessary to tame the surface elsewhere. Thumbnail images from up to 40 iterations indicate that the termination conditions in Section 5.4 are appropriate. After this many iterations there may be no improvement or the system might have become overconstrained and ill-conditioned. Table 1 shows further information on the implicit surfaces produced.

All of the implicit surfaces were created within a few hours on a single 195 MHz R10000 processor. Table 2 shows the computation times per model. The computation times are dependent on the size of the volume, the number of constraints, and the number of iterations necessary.

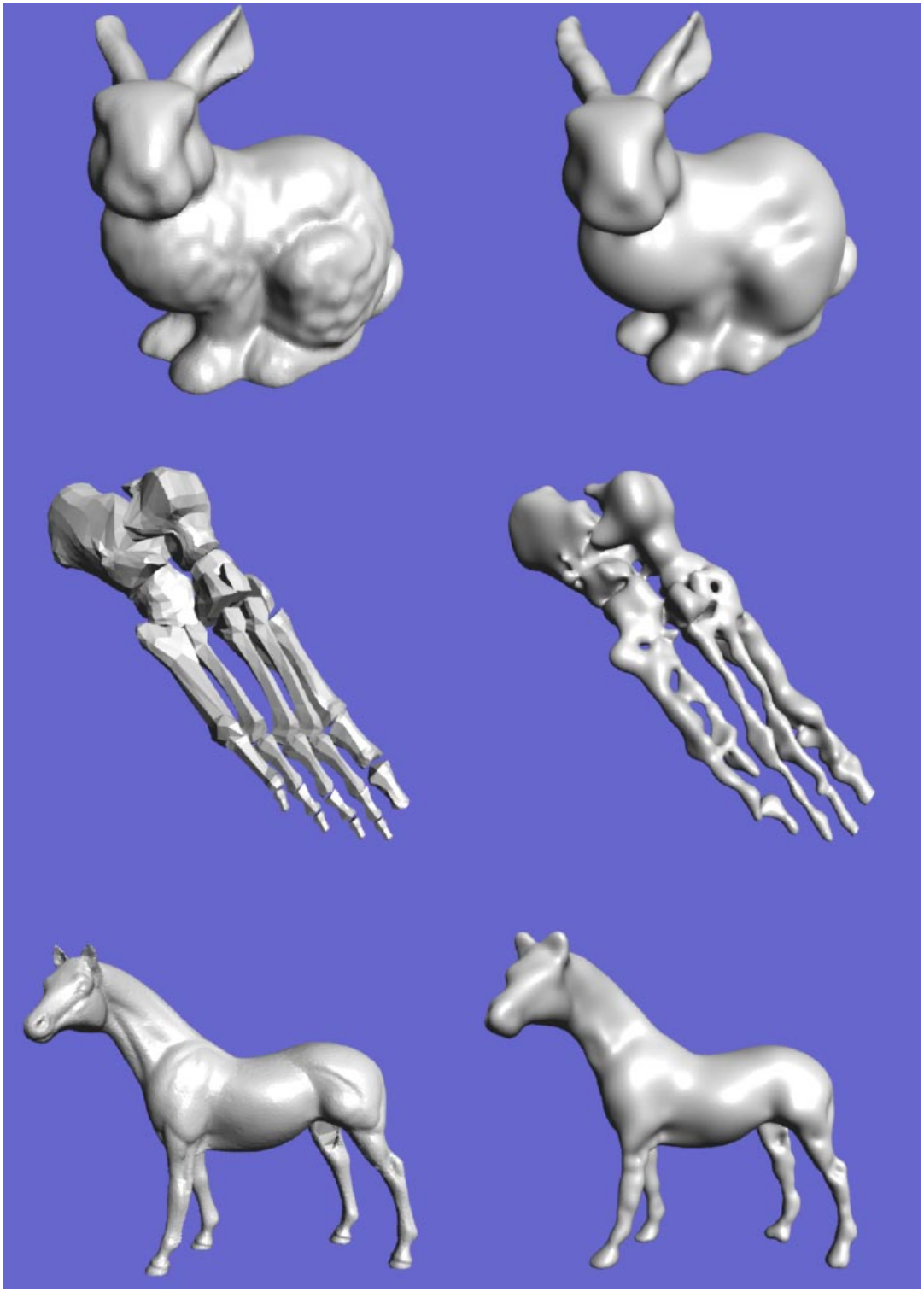

Figure 5: Original polygonal meshes (on left) and resulting implicit surfaces (on right).

Table 1: Implicitization of several polygonal models

Model	Number of Polygons	Size of Volume	Iterations to Finish	Zero Constraints	Interior Constraints	Exterior Constraints	Max. Error (in voxels)
Bunny	69451	$176 \times 220 \times 218$	20	337	109	69	4
Foot Bones	2339	$138 \times 320 \times 124$	24	469	70	133	5
Horse	39698	$286 \times 170 \times 337$	22	480	108	91	5
Spock	345436	$168 \times 170 \times 193$	15	307	92	72	5
Triceratops	2834	$124 \times 320 \times 155$	22	327	93	83	3

Table 2: Running time of implicitization (in minutes)

Model	Size of Volume	Time for Signed Distance	Time for Implicit Evaluation	Time for Misc Computations	Total Time
Bunny	$176 \times 220 \times 218$	22.25	53.60	37.45	113.30
Foot Bones	$138 \times 320 \times 124$	17.27	51.10	18.78	87.15
Horse	$286 \times 170 \times 337$	47.42	71.72	51.35	170.49
Spock	$168 \times 170 \times 193$	10.87	22.47	19.87	53.21
Triceratops	$124 \times 320 \times 155$	17.78	40.25	24.75	82.78

7 Conclusions

We have presented an effective and viable technique to convert polygonal meshes into implicit surfaces. Using a volumetric model as an intermediate representation, we fit an implicit function to match the signed distance of the volumetric model. We refine the implicit function iteratively, identifying areas of improvement and then constraining them. We improve on previous polygon-to-implicit conversion methods in both speed and quality of results.

We have already applied the implicit surfaces we generated to morphing, as is demonstrated in the accompanying video. One area for future research is to add back fine detail with a bump or a displacement map on top of the implicit surface. Another avenue we want to explore is animation of the implicit surfaces. A user may wish to animate a surface by changing joint angles or by performing a deformation. We envision such changes being applied to the locations of the constraint positions, and then invoking the iterative improvement algorithm if the surface requires adjustments.

References

[1] 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)*, volume 14, pages 457–468, 1995.

[2] James F. Blinn. "A Generalization of Algebraic Surface Drawing." *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[3] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*, chapter Convolution of Skeletons, pages 222–241. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[4] Per-Erik Danielsson. "Euclidean Distance Mapping." *Computer Graphics and Image Processing*, 14:227–248, 1980.

[5] F.S. Nooruddin and Greg Turk "Simplification and Repair of Polygonal Models Using Volumetric Techniques." Technical Report GIT-GVU-99-37, Graphics, Visualization and Usability Center, Georgia Institute of Technology, 20 pages, 1999.

[6] Shigeru Muraki. "Volumetric Shape Description of Range Data Using 'Blobby Model'." In *Computer Graphics (SIGGRAPH 91)*, volume 25, pages 227–235, July 1991.

[7] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

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

[9] Andrew P. Witkin and Paul S. Heckbert. "Using Particles to Sample and Control Implicit Surfaces." In *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 94)*, pages 269–278, July 1994.

[10] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. "Data Structures for Soft Objects." *The Visual Computer*, 2(4):227–234, 1986.