Robust Creation of Implicit Surfaces from Polygonal Meshes

Gary Yngve and Greg Turk, *Member*, *IEEE*

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. 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. Iteratively, additional constraints are added; the resulting surfaces are evaluated, and the errors guide the placement of subsequent constraints. We have applied our method successfully to a variety of polygonal meshes and consider it to be robust.

Index Terms—Geometric modeling, surface representations, implicit surfaces.

1 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 (as well as to texture and render). 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 that are used as primitives for implicit surfaces can often have nonobvious 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. Fig. 1 shows 23 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

- *G. Yngve is with the Department of Computer Science and Engineering, University of Washington, 114 Sieg Hall, Box 352350, Seattle, WA 98195. E-mail: gyngve@cs.washington.edu.*
- *G. Turk is with the GVU Center, College of Computing, Georgie Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280. E-mail: turk@cc.gatech.edu.*

Manuscript received 17 Nov. 2000; revised 6 July 2001; accepted 30 July 2001. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 113171.

1077-2626/02/$17.00 © 2002 IEEE

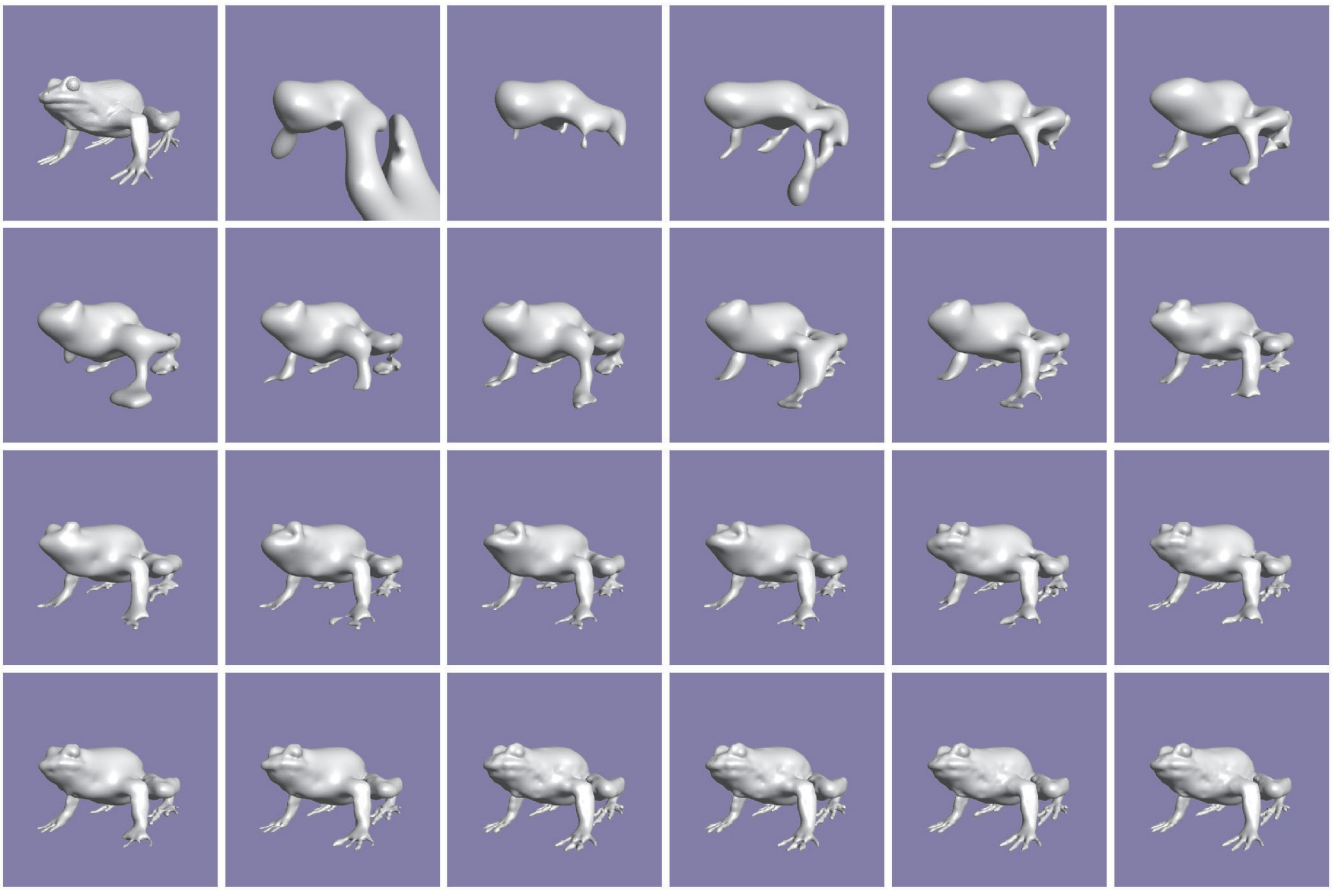

Fig. 1. A polygonal frog (top left) is converted to an implicit surface via our incremental improvement algorithm. The algorithm took 23 iterations to reach the final result. The results from each iteration are shown in successive images. The frog is a near fit after the first three rows; in the last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

would mean that we can avoid having to create special purpose modeling programs for implicit surfaces.

The rest of the paper will proceed as follows: We briefly discuss previous work in implicit surface modeling in Section 2. In Section 3, we explain the variational implicit surface representation. Then, in Section 4, we introduce a set of tools that will be used by the algorithm. We present the algorithm in Section 5, analyze the algorithm's parameters in Section 6, and show results in Section 7. Finally we conclude and discuss future work in Section 8.

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 [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:

$$f_i(x) = A_i e^{b_i \, ||x - c_i||^2}. \quad (1)$$

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 such 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. Recent work by Sherstyuk [11] shows promising results for creating convolution surfaces interactively. Offset surfaces are used to approximate the convolution surfaces during interactive editing.

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. Ferley et al. [12] maintain a cube list representing the isosurface of an implicit function. By updating the cube list, they can achieve interactive editing rates. Their technique is limited to the fixed resolution of the volume,

but wavelet techniques may grant them more flexibility. Still, however, for creating complicated models, more automatic methods are needed.

Muraki developed a method to approximate range data by a blobby implicit surface [13]. 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 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 et al. addressed the modeling of an implicit surface from volume data [14]. 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 positive primitives along the medial axis in substantially less time than Muraki's approach. They also developed a similar method in which the user dissects an object into intuitive reconstruction windows [15]. The optimization problems run inside the individual reconstruction windows for a significant gain in speed. However, the implicit surfaces that they generated with their method were small (the largest had only about 50 primitives). These three methods only calculate a one-way error, namely for the points on the goal surface, seeing how far the isosurface deviates away. To prevent stray components of the implicit surface that the error metric would not be able to catch, they use an E_{shrink} term that minimizes the field of influence for each primitive. Our method, on the other hand, uses a two-way error metric. Another difference between our method and the above methods is they calculate the deviation of the isosurface by its value at the goal surface, rather than the Euclidean distance of the zero-set from the goal surface. It should also be noted that these methods cannot be applied directly to variational implicit surfaces.

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. [16].

3 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.

3.1 Basic Formulation

Variational implicit surfaces are created by solving a scattered data interpolation problem [17]. 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 $\phi(\mathbf{x}) = |\mathbf{x}|^3$, which minimizes the curvature functional

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

The reason why $\phi(\mathbf{x}) = |\mathbf{x}|^3$ minimizes this functional is nonobvious, and we refer the interested reader to [18] for details. 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

$$f(\mathbf{x}) = \sum_{j=1}^{n} d_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}). \tag{3}$$

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

$$h_i = \sum_{j=1}^{k} d_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}). \tag{4}$$

Because (4) 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 $c_i = (c_i^x, c_i^y, c_i^z)$ and let $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$. Then, the linear system can be written as the following matrix equation:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \ldots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \ldots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \ldots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \ldots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \ldots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \ldots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \ldots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{5}$$

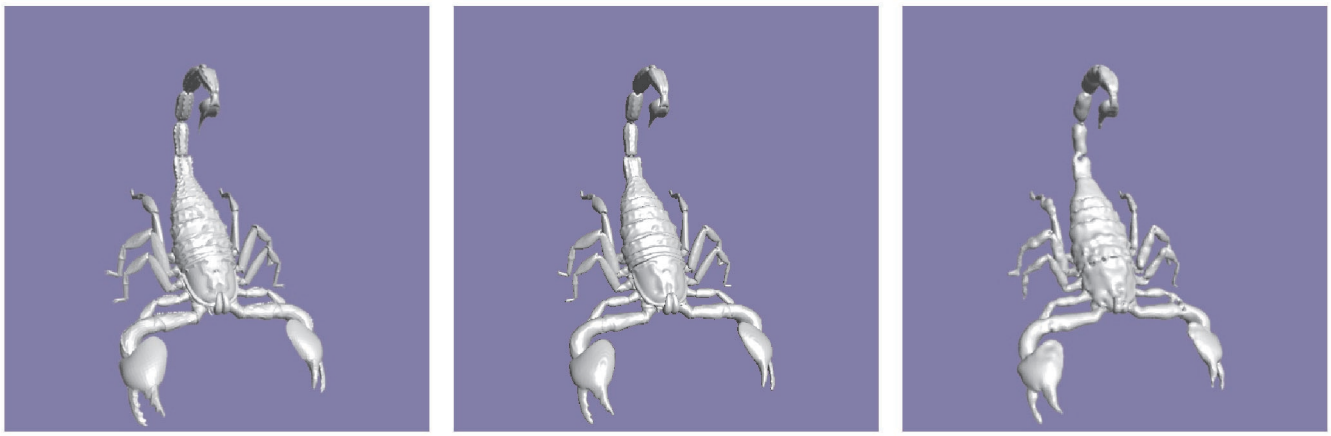

Fig. 2. Original polygonal model of a scorpion (a), intermediate volumetric representation (b), and final implicit surface (c) generated by our algorithm. The volumetric representation captures all but the finest detail, such as the hairs on the tail. Our algorithm refined the implicit surface using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

The above matrix system is symmetric and is the standard one used to solve this type of interpolation problem [18]. 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 ds and ps), evaluating the implicit function from (3) becomes simple. We refer the interested reader to [3], [17], [19] for more mathematical details about this form of implicit function.

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 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 [17]. 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 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. Another possibility would have been to use just the mesh vertices to guide the implicit surface, but this would undersample the surface where the original mesh polygons are large. 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. In Section 4, we 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 approximating 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. Using wavelets or subdivision could reduce the complexity; we delegate investigation of such techniques to future work.

4.1 Voxelization of a Polygonal Mesh

To convert a polygonal model into a volumetric representation, we cast a grid of parallel rays through the mesh and regularly sample the points along these rays. Each sample becomes one voxel. A parity count of surface crossings determines interior from exterior. 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 [20]. For example, Fig. 2 shows the original polygonal scorpion model (Fig. 2a), the intermediate volumetric model (Fig. 2b), and the final implicit representation generated by our algorithm (Fig. 2c). 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. A number of other researchers have also produced methods of voxelizing polygons [21], [22], [23], and any of these methods could have been used.

4.2 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. We use a three-dimensional version of Danielsson's method for computing Euclidean distances [24] to compute the signed distance in $O(n^3)$. 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. We could also have used one of the published methods for converting polygons directly to a distance field, such as [1], [25]. We chose our two-step approach for speed and to allow subvoxel constraint placement.

4.3 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

$$H = \max\left[\max_{x\in\partial f_{goal}}\left[\min_{y\in\partial f_{curr}}||x-y||\right], \max_{x\in\partial f_{curr}}\left[\min_{y\in\partial f_{goal}}||x-y||\right]\right]. \tag{6}$$

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 $sd_{goal}(x)$ and $sd_{curr}(x)$ for the goal and current surfaces as lookup tables, the new constraint location is defined concisely and calculated efficiently as

$$C_{new} = \arg\max\left[\max_{x\in\partial f_{goal}}|sd_{curr}(x)|, \max_{x\in\partial f_{curr}}|sd_{goal}(x)|\right]. \tag{7}$$

The error metric has a two-fold purpose: to evaluate the attempted fit and to suggest where to refine the implicit representation further. Fig. 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). Fig. 3a, Fig. 3b, Fig. 3c, Fig. 3d, and Fig. 3e show the refinement at the third 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, Fig. 3b 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 Fig. 3a and Fig. 3b and choose points with the brightest background color (furthest distance from the other curve). The newly chosen constraints are shown as magenta dots in Fig. 3c (boundary constraints) and Fig. 3d (interior and exterior constraints). The implicit curve with these refinements is in Fig. 3e. Fig. 3f, Fig. 3g, Fig. 3h, Fig. 3i, and Fig. 3j show the respective information for the refinement at the seventh iteration. After later iterations, the implicit curve becomes nearly indistinguishable from the original data.

5 Iterative Improvement of Model

We now describe how the above tools let us model implicit surfaces. Using (7) to find candidate locations for new constraints, we can design an iterative method to refine the surfaces. Below is pseudocode for our algorithm.

```
Algorithm MakeImplicitSurface(Volume f_goal,
          SDFunc sd_goal)
Begin
  Constraints = InitialConstraints(NumInit)
  Repeat
    f_curr = MakeImplicit(Constraints)
    sd_curr = SignedDist(f_curr)
    GenerateCandidateConstraints()
    Repeat
      PruneCandidateList()
      NewCandidate = SelectHighestError()
      Constraints.add(NewCandidate)
    Until NoMoreCandidates
  Until DoneRefining
End
```

First, `NumInit` initial constraints are chosen, discussed in Section 5.1. Then, for each iteration, the following steps are taken: The implicit surface for the current set of constraints is generated by solving (5). The result is evaluated over the volume to obtain f_{curr} (see Section 5.2), and its signed distance sd_{curr} is calculated. New candidate constraints are selected by (7). They may be pruned due to proximity to other constraints or other reasons; see Section 5.3. The remaining candidates become new constraints, and the evaluation-refinement process repeats. The algorithm terminates when the surface fits f_{goal} well enough; termination criteria is discussed in Section 5.4.

5.1 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 do not want to overwhelm the system with too many constraints. To get a good balance between these two extremes, we sample 50 boundary constraints from the points on the surface ∂f_{goal}. Three-dimensional Poisson-disk sampling bandlimits the spatial frequency of constraints; when constraints are close to each other tend to have more influence on the rest of the system and can cause (5) to be ill-conditioned. We

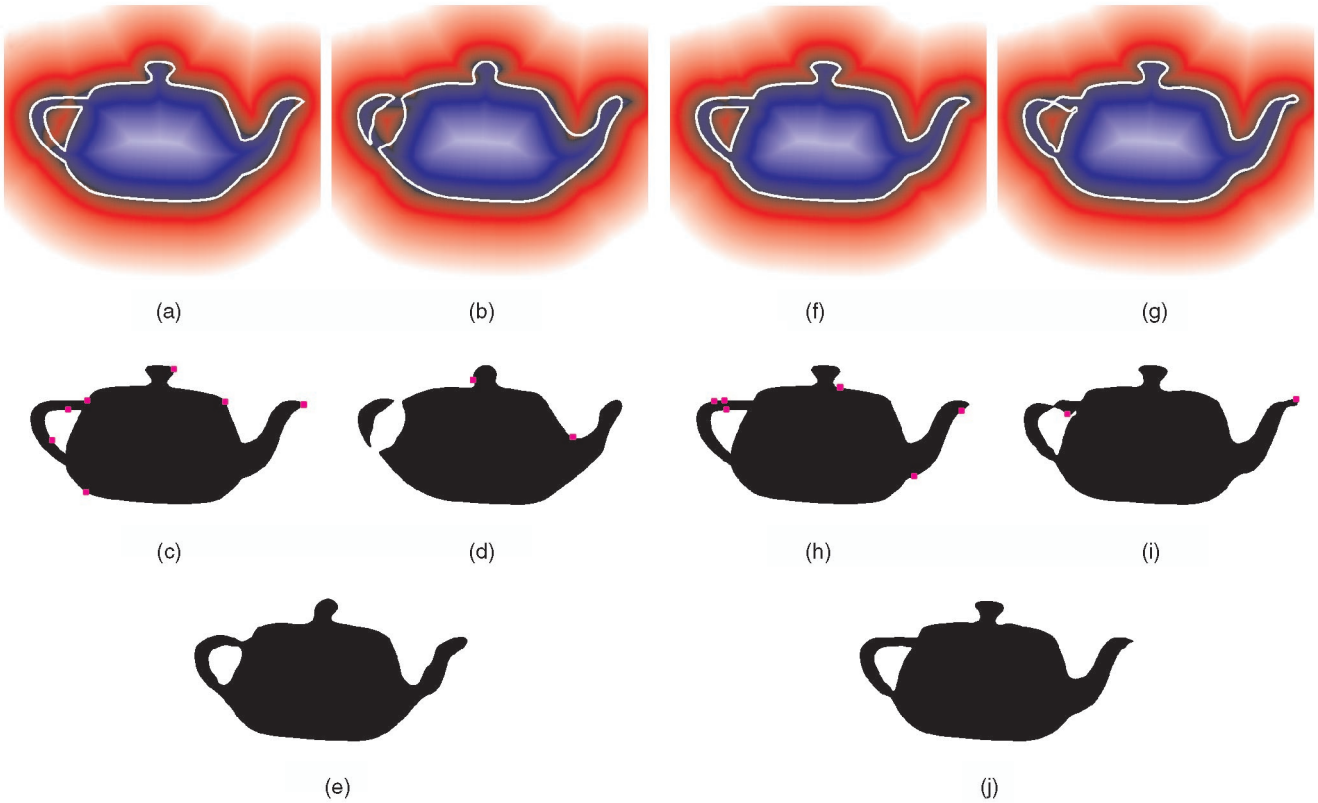

Fig. 3. Creating a two-dimensional implicit curve: To find the locations with greatest errors, we walk around the boundary of the goal curve and see how far it is from the boundary of the current curve. This query is done via a signed distance lookup, seen in (a). (b) Shows the symmetric counterpart,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

recommend the article by Mitchell for information on Poisson disk sampling [26]. Other methods such as a point-repulsion technique may yield a more uniform sampling, but we feel our simple initialization technique is quite satisfactory.

In addition to choosing boundary constraints, we place nonzero constraints to indicate what portions of space are interior or exterior. It is essential to have these further specifications; otherwise, the iso-surface could fit the goal exactly but the implicit function might be inside-out. These nonzero 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 along the gradient of $sd_{goal}(x)$ until we reach a local extremum (or the boundary of the volume) and place a constraint there. The local extrema are the tops of ridges and the bottoms of valleys of the signed distance function—that is, positions on the medial axis of the shape. To decide whether to traverse the gradient uphill or downhill, we try both directions and pick the longer path (hitting the boundary of the volume is by default the longer path). Shorter paths are usually in the direction that is locally concave. 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 $sd_{goal}(x)$ not only tells the surface what direction is outside but also suggests the surface normal.

5.2 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 5.

Evaluating the implicit surface throughout the volume can be costly. Although surface-following isosurface-extraction techniques can reduce the complexity by an order of magnitude, they make assumptions about topology, e.g., they may miss a detached portion of the surface. We wish ∂f_{curr} to capture all connected components since they may indicate error in the current implicit surface. Because the radial basis functions we use have infinite support, a refinement could create errors elsewhere. We do not want to overlook any surface components, but we do not want to evaluate over the entire volume. 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 of the surface, we sample that cube voxel by voxel; otherwise it is filled uniformly. Likewise, if a cube is within eight voxels ∂f_{goal}, we sample it finely. Although this coarse sampling could possibly miss very thin components, we have not found this to happen for any of the models that we have tried.

5.3 Refinement

Now that the boundary voxels can be evaluated by the metric, we can add constraints to refine the implicit surface. We want to avoid adding constraints one by one because performing an iteration per constraint would be costly. However, we also want to avoid refinements interfering with each other. Likewise, making fine changes could be useless if coarse changes are made elsewhere on the surface because such changes may have a nonlocal effect.

We scan through ∂f_{curr} and ∂f_{goal} to find the voxel with the maximum error. In a tie, we favor ∂f_{goal}. 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 use the notation $|sd(x)|$ to represent both these cases. Searching for the maximum error is equivalent to walking along the overlaid boundaries in Fig. 3a and Fig. 3b, 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 subvoxel precision by trilinearly interpolating the densities of the voxels. Another way to place the constraints more accurately would be to return to the actual polygonal data, but we have not done so. 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 will improve the surface. To avoid having (5) become ill-conditioned, we discard candidates less than one voxel from any preexisting constraint. 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 spheres of radius $sd(x_i)$ centered at x_i will be disjoint. Often, adding a constraint in one location will greatly improve the surface nearby. Boundary voxels with errors less than half the maximum error at the current iteration are considered too fine an adjustment and are discarded.

5.4 Termination

Finally we discuss how the algorithm terminates. Empirically, we found that the models tend to refine quickly at first and then slow as they converge to the goal (see Fig. 1). We terminate the algorithm under four conditions: 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 5,000). When successive models have the same Hausdorff error, we pick the best model based from a similarly derived root-mean-square (RMS) error. For most of our results, termination was from the model not improving over the last few iterations.

6 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. We note that the parameter settings can have some effect on the running times to generate the results, especially when using extreme numbers of initial constraints. All of the timing data in the tables were from using one 195 MHz R10000 MIPS processor of an SGI Origin.

6.1 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 coarse volume might not capture much detail from a polygonal model, and a fine volume can be computationally expensive. We use one byte per voxel, and by far, the largest factor for memory usage in our algorithm is the size of the voxel grids. For the models shown in Section 7, 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 surfaces because of the absence of detail in the volumetric models, but they can be computed in mere minutes. We also tried running a few models at higher voxel resolutions. There were no noticeable improvements, which is not surprising because the results in Section 7 had errors larger than a voxel. To determine the resolution, a user could view the model with the radius of regularity or the radius of curvature color-coded and textured.

Table 1 shows the results from varying the voxel resolution. For the coarsest models, our algorithm rapidly generates a nearly exact fit because the high-frequency details are smoothed in the volumetric models. Fig. 4 shows the volumetric models and resulting implicit surfaces for models of a horse and the head at two resolutions.

6.2 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 7, 50 boundary constraints 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. While using many initial constraints can cut down the number of iterations needed, using too many initial constraints results in too much time being spent on solving for unnecessary constraints. For example, in Table 2, both models took the longest to calculate when given the most initial constraints.

We experimented with seeding the implicit surface with values ranging from 50 to 1,600 boundary constraints (100 to 3,200 total constraints). For all configurations, the algorithm returned satisfactory results.

TABLE 1
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
Head	$69 \times 69 \times 76$	12	1	0.1591	990	15
Head	$118 \times 120 \times 135$	13	2	0.1815	3742	3:45
Head	$168 \times 170 \times 193$	18	2	0.2949	2543	5:57

Furthermore, simply adding more initial constraints did not alone produce a good surface. The results in Table 2 indicate that, although the first iterations of the surfaces get better with more constraints, they are still nowhere near the desired fit. Fig. 5 illustrates using varying amounts 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 3,200, key features such as the horns are still lacking, yet the final implicit surfaces captured these key features. Not only do these results indicate the insignificance of the amount of initial constraints, but they also demonstrate the need for iterative refinement.

For all of the results shown in Section 7, we initialize with 50 zero constraints and 50 interior and exterior constraints. Using fewer initial constraints produces results faster, and the results are comparable to those started with more constraints.

6.3 Regularization Parameter

If we wish to approximate rather than exactly interpolate some of the constraints, we can use a slight adaptation of (5) 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 nonzero 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).

We examined the results of our algorithm using nonzero λ values at the boundary constraints. Instead of running our algorithm again with the new λ, we simply took the constraints already produced and solved (5) 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. Fig. 6 shows enlarged images of the implicit horse with $\lambda = 0$ and $\lambda = 100$. Using $\lambda = 100$ improved the ears but sacrificed detail around the nostrils. The smoother version of the leg with $\lambda = 100$ is more pleasing than the

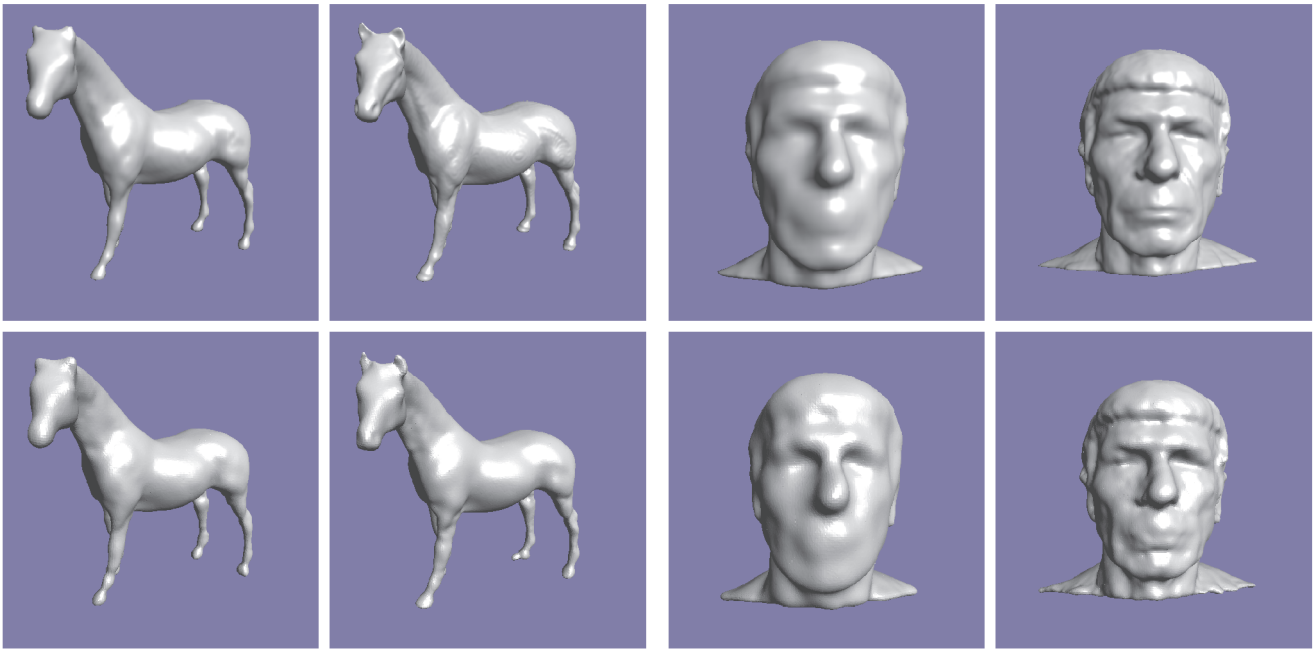

Fig. 4. Top row: low-resolution volumetric representations of the horse ($104 \times 70 \times 119$, $195 \times 120 \times 228$) and the head ($69 \times 69 \times 76$, $118 \times 120 \times 135$). Bottom row: respective implicit surfaces generated by our algorithm.

TABLE 2
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

$\lambda = 0$ version, especially the hoof and the dimple on the inner thigh. All of the results in Section 7 use $\lambda = 0$; a user who wishes a smoother model can set λ accordingly.

7 Results

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

Fig. 7 and Fig. 8 show the resulting implicit surfaces from our algorithm. The subimages in these figures are arranged in pairs, with the intermediate volumetric models (the left subimages in a pair), which do not capture all the details from the polygonal models, shown side-by-side with the resulting implicit surfaces created by our method (right subimage of pair). 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. While some forms of implicit surfaces can represent such thin features, we do not know of any automatic method for creating thin-featured models from polygons.

Fig. 7 shows our results on six high-resolution models obtained from laser range scanning. In all cases, our

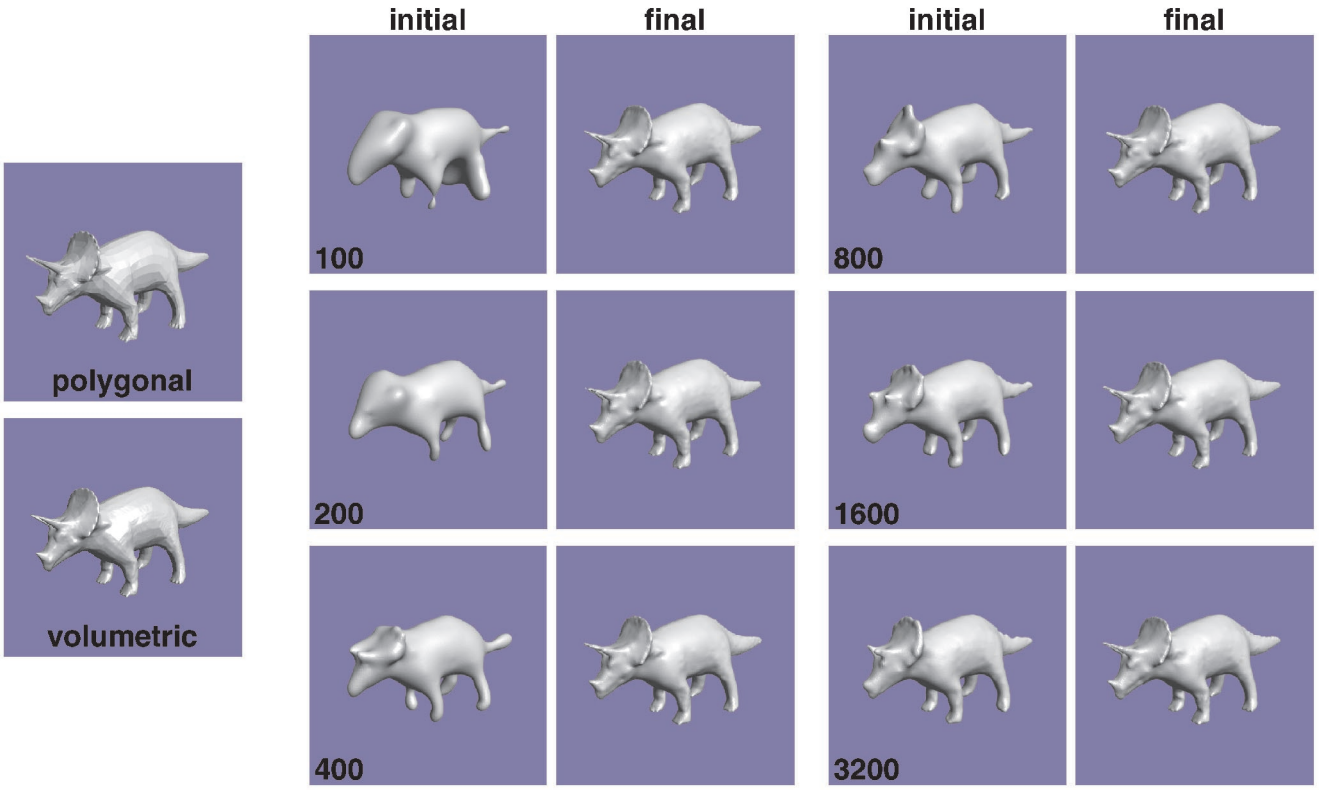

Fig. 5. Triceratops model, showing the effect of the number of initial constraints. At far left is original polygon model and the volumetric mode. Pairs of models show results after just the initial constraints (left image of pairs, with number of constraints in corner), and after subsequent refinement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

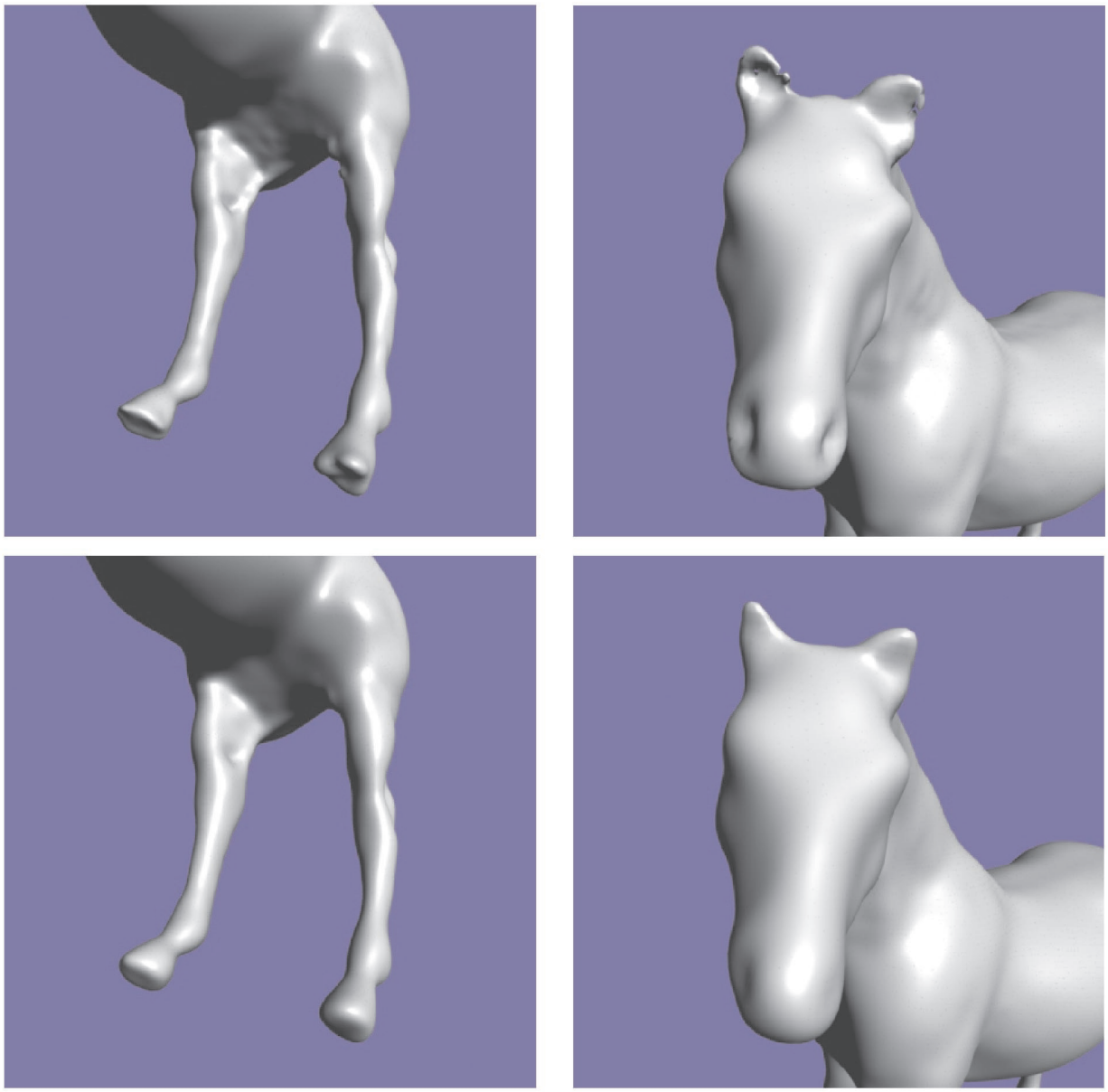

Fig. 6. Implicit horse: head and leg. Top shows $\lambda = 0$ (exact interpolation) and bottom shows $\lambda = 100$ (weighted average of exact interpolation and curvature minimization).

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 Table 3. The Buddha model is difficult because of the fine 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 at too great a cost. 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 using radial basis functions that would give results comparable to those shown in this figure.

Fig. 8 shows our results on six polygonal models that contain high-curvature regions or thin surfaces. The triceratops has sharp horns, and the horse has skinny legs and pointy ears. All of these features are captured by the surface created by our method. The foot model contains many slender bones, but the most difficult feature is not the bones, but the 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 many holes. The implicit representation we use has the effect of rounding off the sharp edges of this model. For the other five models, despite their high-curvature features, our results fit the original data well.

For more quantitative comparisons, Table 3 gives the numbers of iterations, the numbers of constraints in the final implicit surfaces, and the errors of the final implicit

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.

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 Fig. 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 4 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 time linear in the number of constraints. Our evaluation method tests a shell of voxels finely and a volume of voxels coarsely. However, the shell is rather thick, and in the later iterations, evaluating at a point becomes costly from so many constraints. For many of our models, the last few iterations amounted to half the total running time or more. Because we wanted to err conservatively regarding evaluating the implicit function as exactly as possible, there may be room for speeding up the evaluation, such as reducing the thickness of the shell. Solving (5) is also expensive for a large number of constraints. LU decomposition is cubic in the number of constraints, but because the evaluation was still more expensive, we did not focus on improving the matrix solution. Jacobi iteration with the last set of weights as the initial value could accelerate the matrix solution. The table also shows timing information to achieve root-mean-squared errors less than two voxels and less than one voxel; for most of the models, the algorithm reached these accuracies in substantially less time.

8 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 radial-basis-function 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.

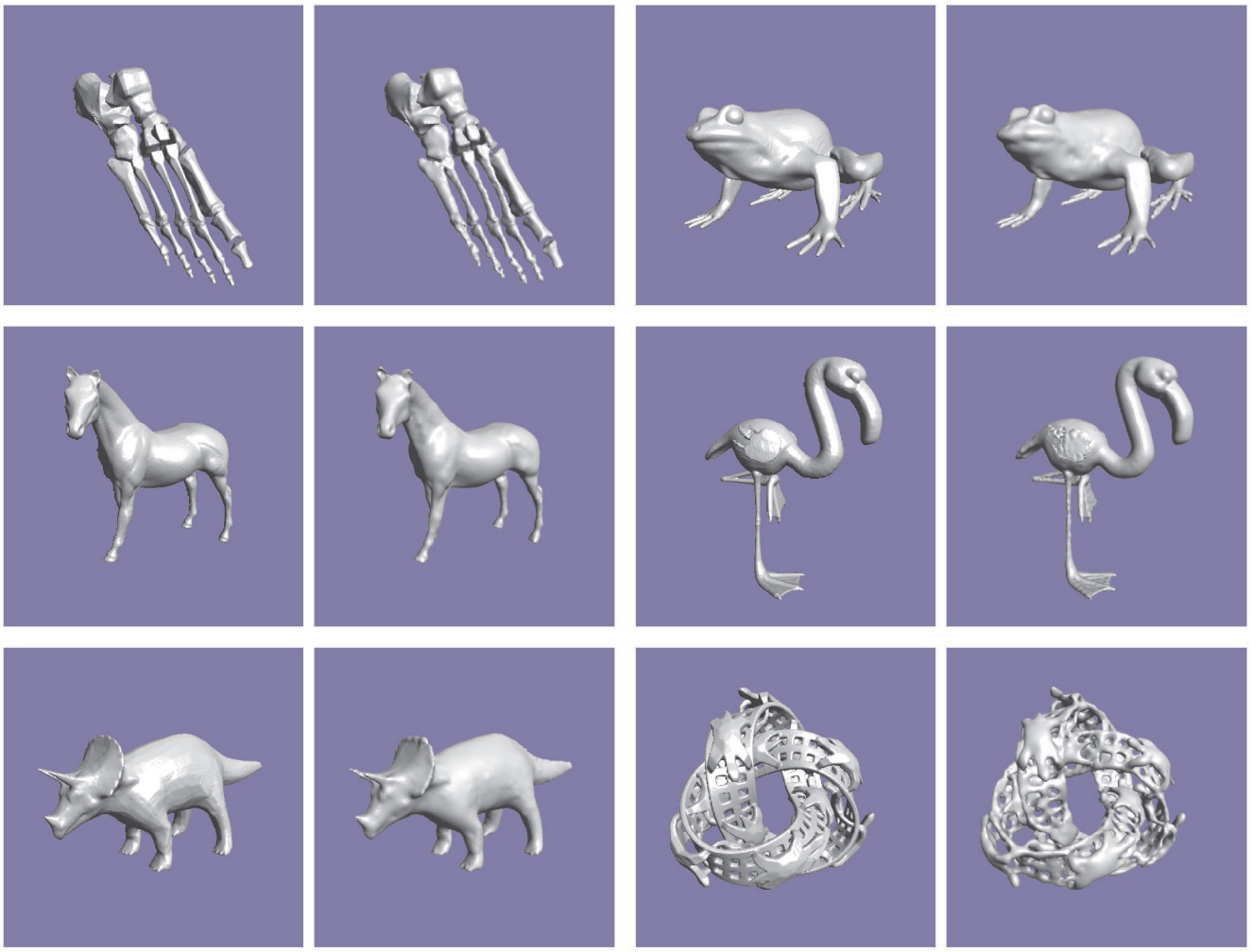

Fig. 8. Six models with thin or high-curvature regions: intermediate volumetric representations (left images of pairs) and final implicit surfaces (right images of pairs) from our algorithm.

- Provides a general framework for other basis functions or other implicit surface representations.

One avenue for future work 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 3. However, using such basis functions should speed up the

TABLE 3
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
Head	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 4
Running Time of Implicitization (In Hours: Minutes on a 195 MHz R10k)

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
Head	$168 \times 170 \times 193$	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

algorithm because the matrix will be sparse and implicit function evaluations will only have to use nearby basis functions. Recent work by Morse et al. has demonstrated the promise of using compactly supported basis functions [19].

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, such as the work in [27], [28], [29].

A different representation of the volumes could be used to reduce memory usage and computation times. Recently, researchers at MERL used adaptively sampled distance fields to represent surfaces [30] efficiently. Using wavelets to represent the volume data might have similar benefits [22]. Such an adaptive approach might be useful to speed up several of the algorithm tasks, especially in evaluating the metrics and searching for locations where the surface needs to be refined.

Acknowledgments

The authors would like to thank Hughes Hoppe for providing the head dataset. They 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. The first author was funded in part by an US National Science Foundation Graduate Fellowship.

References

[1] B. Payne and A. Toga, "Distance Field Manipulation of Surface Models," *IEEE Computer Graphics and Applications,* vol. 12, pp. 65-71, 1992.

[2] J. Hughes, "Scheduled Fourier Volume Morphing," *Proc. SIGGRAPH '92,* pp. 43-46, 1992.

[3] G. Turk and J.F. O'Brien, "Shape Transformation Using Variational Implicit Functions," *Proc. SIGGRAPH '99,* pp. 335-342, 1999.

[4] M. Desbrun and M.-P. Gascuel, "Animating Soft Substances with Implicit Surfaces," *Proc. SIGGRAPH '95,* pp. 287-290, 1995.

[5] M.-P. Gascuel, "An Implicit Formulation for Precise Contact Modeling Between Flexible Solids," *Proc. SIGGRAPH '93,* pp. 313-320, 1993.

[6] J.F. Blinn, "A Generalization of Algebraic Surface Drawing," *ACM Trans. Graphics,* vol. 1, no. 3, pp. 235-256, 1982.

[7] A.P. Witkin and P.S. Heckbert, "Using Particles to Sample and Control Implicit Surfaces," *Proc. SIGGRAPH '94,* pp. 269-278, 1994.

[8] J. Levin, "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces," *Comm. ACM,* vol. 19, pp. 555-563, 1976.

[9] G. Wyvill, C. McPheeters, and B. Wyvill, "Data Structures for Soft Objects," *The Visual Computer,* vol. 2, no. 4, pp. 227-234, 1986.

[10] J. Bloomenthal and K. Shoemake, "Convolution Surfaces," *Proc. SIGGRAPH '91* pp. 251-256, 1991.

[11] A. Sherstyuk, "Interactive Shape Design with Convolution Surfaces," *Shape Modeling International,* 1999.

[12] E. Ferley, M.-P. Cani, and J.-D. Gascuel, "Practical Volumetric Sculpting," *Implicit Surfaces,* 1999.

[13] S. Muraki, "Volumetric Shape Description of Range Data Using 'Blobby Model'," *Proc. SIGGRAPH '91,* pp. 227-235, 1991.

[14] E. Bittar, N. Tsingos, and M.-P. Gascuel, "Automatic Reconstruction of Unstructured 3D Data: Combining a Medial Axis and Implicit Surfaces," *Computer Graphics Forum (Proc. Eurographics '95),* vol. 14, pp. 457-468, 1995.

[15] N. Tsingos, E. Bittar, and M.-P. Gascuel, "Semi-Automatic Reconstruction of Implicit Surfaces for Medical Applications," *Computer Graphics Int'l,* vol. 14, 1995.

[16] *Introduction to Implicit Surfaces,* J. Bloomenthal, ed., Morgan Kaufmann, 1997.

[17] G. Turk and J.F. O'Brien, "Variational Implicit Surfaces," Technical Report 15, Georgia Inst. of Technology, 1999.

[18] J. Duchon, "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," *Lecture Notes in Math. 571,* pp. 85-100, 1976.

[19] B. Morse, T. Yoo, P. Rheingans, D. Chen, and K.R. Subramanian, "Interpolating Implicit Surfaces from Scattered Surface Data Using Compactly Supported Radial Basis Functions," *Proc. Int'l Conf. Shape Modeling and Applications,* pp. 89-98, May 2001.

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

[21] A.T Kaufman, "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *Computer Graphics (Proc. SIGGRAPH '87),* vol. 21, no. 4, pp. 171-179, July 1987.

[22] L. Velho and J. Gomez, "Approximate Conversion of Parametric to Implicit Surfaces," *Computer Graphics Forum,* vol. 15, no. 5, pp. 327-337, Dec. 1996.

[23] J. Huang, R. Yagel, V. Filippov, and Y. Kurzion, "An Accurate Method for Voxelizing Polygon Meshes," *Proc. 1998 IEEE Symp. Volume Visualization,* pp. 119-126, Oct. 1998.
[24] P.-E. Danielsson, "Euclidean Distance Mapping," *Computer Graphics and Image Processing,* vol. 14, pp. 227-248, 1980.
[25] W.J. Schroeder and W.E. Lorensen, "Implicit Modeling of Swept Surfaces and Volumes," *Proc. Visualization '94,* pp. 40-45, Oct. 1994.
[26] D.P. Mitchell, "Generating Antialiased Images at Low Sampling Densities," *Computer Graphics (Proc. SIGGRAPH '87),* vol. 21, no. 4, pp. 65-72, July 1987.
[27] V. Krishnamurthy and M. Levoy, "Fitting Smooth Surfaces to Dense Polygon Meshes," *Proc. SIGGRAPH '96 Conf. Proc.,* H. Rushmeier, ed., pp. 313-324, Aug. 1996.
[28] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini, "Preserving Attribute Values on Simplified Meshes by Re-sampling Detail Textures," *The Visual Computer,* vol. 15, no. 10, pp. 519-539, 1999.
[29] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped Textures," *Computer Graphics Proc., Ann. Conf. Series (Siggraph '00),* pp. 465-470, 2000.
[30] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," *Proc. SIGGRAPH '00,* pp. 249-254, 2000.

Gary Yngve received the BS (2000) from Georgia Institute of Technology, where he worked with Greg Turk on implicit surfaces. He is pursuing a PhD in computer science at the University of Washington on a US National Science Foundation Graduate Fellowship. His current work focuses on character animation, and his other research interests include modeling and simulation, natural phenomena, photo-realistic rendering, and mathematical techniques for computer graphics.

Greg Turk received the PhD in computer science in 1992 from the University of North Carolina at Chapel Hill. He was a postdoctoral researcher at Stanford University for two years, followed by two years as a research scientist at UNC Chapel Hill. He is currently an associate professor at the Georgia Institute of Technology, where he is a member of the College of Computing and the Graphics, Visualization and Usability Center. His research interests include computer graphics, computer vision, and scientific visualization. He is a member of the IEEE.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.