Visual Comput (2009) 25: 279–288
DOI 10.1007/s00371-008-0267-3

ORIGINAL ARTICLE

Xiaogang Jin
Chiew-Lan Tai
Hailin Zhang

Implicit modeling from polygon soup using convolution

Published online: 15 July 2008
© Springer-Verlag 2008

X. Jin (✉) · H. Zhang
State Key Lab of CAD & CG,
Zhejiang University, Hangzhou, 310027,
P.R. China
jin@cad.zju.edu.cn

C.-L. Tai
Hong Kong University of Science &
Technology, Hong Kong, P.R. China
taicl@cse.ust.hk

Abstract We present a novel method for creating implicit surfaces from polygonal models. The implicit function is defined by convolving a kernel with the triangles in the polygonal model. By adopting a piecewise quartic polynomial kernel function with a finite support, we derive a convolution model that has a closed-form solution, and thus can be efficiently evaluated. The user only needs to specify an effective radius of influence to generate an implicit surface of desired closeness to the polygonal model. The resulting implicit surface is fast to evaluate, not requiring accumulating evaluation results using any hierarchical data structure, and can be efficiently ray-traced to reveal the detailed features.

Keywords Convolution surfaces · Implicit surfaces · Ray tracing

1 Introduction

Recent progress in scanning technology has led to an abundance of polygonal models. Polygonal models are the preferred representation in many applications due to their simplicity and the hardware support for fast rendering. However, polygonal data reconstructed from scanned point clouds are often imperfect, possibly containing holes, non-manifold parts, or bad quality triangles (slivers). Such general polygonal data are referred to as polygonal soup. On the other hand, implicit representations define surfaces as isosurfaces of a scalar field in 3D. They are basically mesh-independent, with the desired mesh only generated when needed. Implicit representations allow for efficient constructive solid geometry (CSG) operations for several reasons: boolean operations can be easily defined through inside–outside tests; contours and blends can be efficiently generated; the resulting solid models are guaranteed to be manifolds and thus manufacturable. They have also been proven to be good representations for modeling and animation of smooth time-varying deformable objects of complex topology, such as liquid, snow, cloud and organic shapes [3, 6, 8, 16, 17].

In this paper, we present a method to transform a polygon data set to an implicit surface using convolution. The polygon set is triangulated into triangles before computing the implicit surface. We develop a closed-form convolution model for convolving a piecewise quartic kernel function with the triangle primitive. The analytical solution enables the resulting convolution surface to be efficiently evaluated. The user can intuitively control how closely the resulting implicit surface approximates the input polygon data by selecting a proper effective radius.

Our method is most closely related to that of Shen et al. [20]. They use a moving least-squares formulation

to compute an implicit surface that approximates or interpolates the given polygonal data. The tightness of the surface is controlled through an iterative procedure that adjusts the constraint values over each polygon. In addition, all input vertices are ensured to be enclosed by the implicit surface through an iterative procedure. In contrast, our method produces an implicit surface that is arbitrarily close to the input polygons by simply controlling an effective radius parameter. Our formulation leads to a field function that can be evaluated quickly, allowing the resulting surface to be efficiently ray-traced. Moreover, thanks to our skeleton-based modeling approach, our method is versatile for animation. A model can be easily separated into individual skeletal primitives or combined with other skeletal primitives, achieving complex shapes and special effects for animation purpose.

2 Related work

Skeleton-based implicit surfaces. A skeleton-based implicit surface is commonly defined as a level set satisfying

$$S = \left\{ (x, y, z) \;\middle|\; \sum F_i(x, y, z) - T = 0 \right\} \tag{1}$$

where F_i is the field function of the i-th contributing source and T is the threshold field. Metaballs (or blobs, soft objects) [1, 15, 23, 24] were the first skeletal-based implicit surface introduced, and they are now widely used in commercial software. However, metaballs use only point fields, making them inefficient for representing flat surfaces. The convolution surface was introduced by Bloomenthal and Shoemake [2] as a natural and powerful extension to point-based field surfaces: by convolving a three-dimensional low-pass Gaussian filter kernel along arbitrary skeletons. The resulting iso-surfaces possess the superposition property, thus avoiding bulges and creases [5].

Theoretically, convolution surfaces can be defined in terms of any geometric primitives. However, for rendering purpose, the function $F_i(x, y, z)$ needs to be easy to evaluate so as to efficiently locate the iso-surfaces. The best case is when the the convolution integral yield a closed-form solution that can be directly evaluated. The possibility of deriving a closed-form solution depends on both the kernel function and the skeletal primitive. Bloomenthal and Shoemake [2] calculate the field numerically using a point-sampling method, thus suffering from potential under-sampling artifacts. Later, researchers proposed the Cauchy function [13, 21] as the kernel, leading to analytical solutions for convolving with points, line segments, triangles, arcs and planes. However, it has an infinite support, and thus requires the use of a bounding box in practice to truncate the influence region, causing noticeable artifacts. In this paper, we propose to use a finite-support kernel function that allows efficient evaluation and produces smoother surfaces without approximation artifacts.

No kernel functions are known to produce closed-form solutions when integrating along a free-form curve or surface. Hornus et al. [9] focus on surfaces whose skeletons are a graph of interconnected subdivision curves and surfaces and adopt a new kernel function that provides a closed-from solution when convolving along a line skeleton with varying radius. Piecewise quartic kernel function has been proposed to provide a closed-form solution for convolving along a quadratic curve [10].

Polynomial kernel functions are popular due to their small computation cost. Nishimura and Kawai [15] use piecewise quadratic polynomials to define metaballs. Wyvill and Wyvill [23] introduce a six-degree polynomial to model soft objects. Quartic polynomials are used in ray-tracing software such as Rayshade and POV-Ray. Jin and Tai [10] use piecewise quartic polynomials for convolving with lines, arcs, quadratic curves. In this paper, we show that the piecewise quartic polynomials also provide a closed-form solution when convolving with the triangle primitive.

Implicit surfaces from polygonal models. Construction of implicit surfaces from polygonal models can borrow ideas from the extensive research work on construction from point clouds. We only mention a few representative works here. In [7] and [22], the function is represented using a globally supported radial spline, while locally supported functions were used in [18]. In addition to representing function as sums of continuous basis functions, level-set methods have also been used to fit a surface to a point cloud [14].

Yngve et al. [25] use a globally supported radial spline to match the polygon data, however, the resulting surface deviates substantially from the polygons. Shen et al. [20] extend the moving least-squares framework from point sets to polygon data by replacing the evaluation of coefficients with an integration over the polygons. Since there is no closed-form solution for the integration, they propose an approximation method, which is quite slow if accurate results are needed. When the weights in the moving least-squares system are set to interpolate the polygon soup, the integration over points on the polygons leads to infinity. They solve this problem by finding a suitable weight parameter that is smaller than the smallest feature size in the model. A disadvantage of the algorithm is that an iterative procedure is needed to produce an implicit surface that not only approximates the polygon model, but also encloses all polygons. Finally, the cost of evaluating the implicit surface function formulated with moving least-squares is linear in the number of polygon constraints, which is undesirable for huge models. They address the high computation cost by maintaining a hierarchical tree to store the integration of every polygon in the leaves and define an approximate function with the evaluation aided by the tree.

Kanai et al. [12] address the problem of implicit surface reconstruction from polygon soup using an error-driven approach. By borrowing the idea of interpolation over points, they divide the points into smaller sets, interpolate each smaller set independently, then use the partition of unity method to evaluate the implicit surface function. Their algorithm approximates every polygon by a quadratic implicit function that minimizes the distance and gradient errors between the quadratic function and the polygon. A hierarchical tree is then built by storing the quadratic functions in the leaf nodes and constructing the internal nodes by merging two children that produce the smallest distance and gradient errors. A disadvantage of their method is the large memory requirement for constructing the tree structure.

Our convolution formulation leads to an analytical surface, which is fast to evaluate, without requiring special data structure.

2.1 Convolution surface

A convolution surface is an iso-surface in a scalar field implicitly defined by a skeleton consisting of three-dimensional points and a potential function representing the contribution of each skeletal point to the scalar field.

In this paper, we adopt the following definition of convolution surface proposed by McCormack and Sherstyuk [13]. Let $P(x, y, z)$ be a space point in R^3, and let $g : R^3 \to R$ be the geometry function that represents a modeling skeleton V:

$$g(\boldsymbol{P}) = \begin{cases} 1, & \boldsymbol{P} \in \text{skeleton } V \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Let $f : R^3 \to R$ be a potential function generated by a single point in the skeleton V, and let $\boldsymbol{Q}$ be a point in the skeleton. Then the total field contributed by the skeleton at a point $\boldsymbol{P}$ is the convolution of two functions f and g:

$$F(\boldsymbol{P}) = \int_V g(\boldsymbol{Q})\, f(\boldsymbol{P} - \boldsymbol{Q})\, \mathrm{d}V = (f \otimes g)(\boldsymbol{P}). \tag{3}$$

Thus, f is also called the *convolution kernel*. The existence of a close-form solution to this integration depends on both the primitive and potential function. We adopt a piecewise quartic polynomial since it leads to the simplest computation:

$$f(r^2) = \begin{cases} \left(1 - \frac{r^2}{R^2}\right)^2, & r \le R \\ 0, & r > R \end{cases} \tag{4}$$

where R is the effective radius of the kernel. Note that it has a finite support; specifically, skeleton primitives that have distance larger than R from point $\boldsymbol{P}$ will have zero field contribution to $\boldsymbol{P}$.

An important property of convolution surfaces is the superposition, or the evaluation-independent property, which means that summing the convolution surfaces generated by two separate skeletal primitives yields the same surface as that generated by the combined skeleton [21].

3 Field computation for triangular models

We assume that the input polygon model, which may be a mesh with manifold connectivity or a polygon soup, consists of only triangles (i.e., other types of polygons are assumed to have been triangulated). We use all the triangles as skeleton primitives to generate a convolution surface. To evaluate the field function F at a point $\boldsymbol{P}$ we must first identify the integration domain of each triangle in its influence region. We represent the finite support of the kernel function as a *clipping sphere*, centered

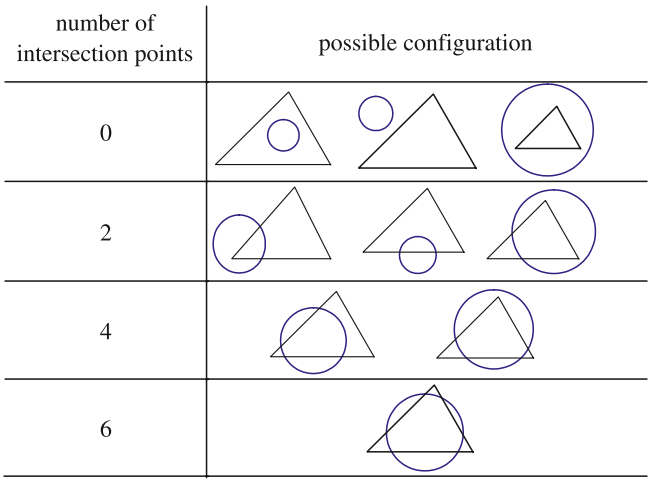

Fig. 1. All possible intersection configurations between a clipping sphere and a triangle

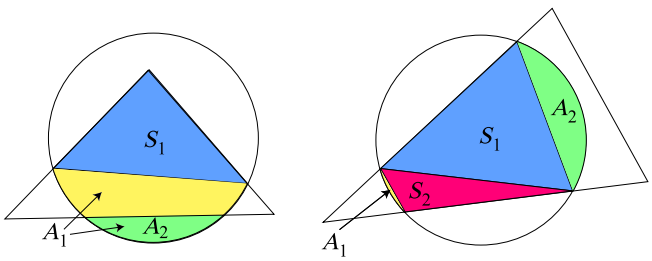

Fig. 2. The intersecting area of the clipping sphere and a triangle defines the integration domain. The integration domains for the two examples are: $S_1 + A_1 - A_2$ (*left*) and $S_1 + S_2 + A_1 + A_2$ (*right*), where S_i are triangle segments and A_i are chord segments

at $\boldsymbol{P}$ with radius R. The area of intersection between the clipping sphere and the triangle defines the effective area that contributes to that field, i.e., the integration domain.

We first identify all the possible configurations of the clipping sphere intersecting a triangle. We note that the number of intersection points between a circle and the edges of a triangle is always even, and is at most six, discounting the degenerate cases such as when an edge is tangential to the circle. Therefore, there are four possible cases: the number of intersection points is 0, 2, 4 or 6. Considering symmetry features, Fig. 1 shows all the possible scenarios. The cases of zero or six intersecting points are simple. For the case of two intersecting points, the circle may either intersect one edge at two points or intersect two edges each at one point. Finally, for the case of four intersecting points, the circle may either intersect two edges each at two points or intersect one edge at two points and two other edges each at one point. For each of the six cases, the integration domain can be decomposed in terms of two types of primitives: triangle segments and chord segments. Figure 2 shows two examples of decomposition.

3.1 Convolution integral over triangle segments

Let $\boldsymbol{P} \in R^3$ be an arbitrary point at which we want to evaluate the field contribution of a triangle segment $\boldsymbol{ABC}$ (Fig. 3). Let $\boldsymbol{Q}(x, y)$ be a point on the line segment $\boldsymbol{BC}$. Expressing the vector $\boldsymbol{AQ}$ as $u\boldsymbol{AB} + v\boldsymbol{AC}$, where $v = 1-u$, we can write the squared distance from $\boldsymbol{P}$ to $\boldsymbol{Q}$ as $((\boldsymbol{P}-\boldsymbol{A}) - (u\boldsymbol{AB} + v\boldsymbol{AC}))^2$. Let $(0, 0)$, $(r, 0)$, (s, t) be the coordinates of the vertices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, respectively, in a Euclidean system. For the coordinates (x, y) of $\boldsymbol{Q}$, we can write $(x, y) = u(r, 0) + v(s, t)$, thus $\mathrm{d}x\,\mathrm{d}y = rt\,\mathrm{d}u\,\mathrm{d}v$. Noting that rt is twice the triangle area, and using the sine rule for triangles, we get $\mathrm{d}x\,\mathrm{d}y = |\boldsymbol{AC}|\,|\boldsymbol{AB}|\sin(\alpha)\,\mathrm{d}u\,\mathrm{d}v$, where α is the angle between $\boldsymbol{AB}$ and $\boldsymbol{AC}$.

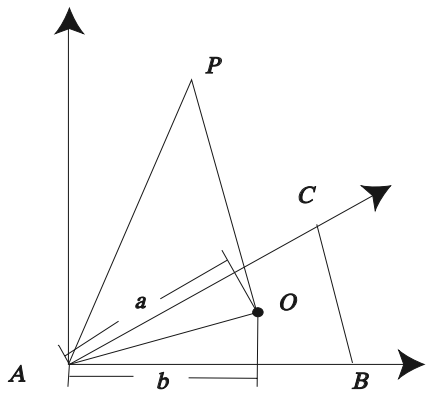

Fig. 3. Convolution integral over a triangle segment $\boldsymbol{ABC}$, where $\boldsymbol{O}$ is the projected point of $\boldsymbol{P}$

To simplify the expressions of the integration result, we first define

$$l_1 = |\boldsymbol{AC}|, \quad l_2 = |\boldsymbol{AB}|, \quad h^2 = |\boldsymbol{AP}|^2 - R^2,$$
$$a = \boldsymbol{AP} \cdot \frac{\boldsymbol{AC}}{|\boldsymbol{AC}|}, \quad b = \boldsymbol{AP} \cdot \frac{\boldsymbol{AB}}{|\boldsymbol{AB}|}, \quad c = \cos(\alpha).$$

Then, using the quartic kernel function in Eq. 4, the field function of the triangle segment $\boldsymbol{ABC}$ at point $\boldsymbol{P}$ is:

$$\begin{aligned}
&F(\boldsymbol{P}) \\
&= \int_0^1 \int_0^{1-u} \left(1 - \frac{((\boldsymbol{P}-\boldsymbol{A}) - (u\boldsymbol{AB} + v\boldsymbol{AC}))^2}{R^2}\right)^2 \\
&\quad \times l_1 l_2 \sin(\alpha)\,\mathrm{d}v\,\mathrm{d}u \\
&= l_1 l_2 \sin(\alpha) \int_0^1 \int_0^{1-u} \mathrm{d}v\,\mathrm{d}u \\
&\quad \times \left(\frac{-h^2 + 2l_2 bu + 2l_1 av - l_2^2 u^2 - l_1^2 v^2 - 2uvl_1 l_2 c}{R^2}\right)^2 \\
&= \frac{l_1 l_2 \sin(\alpha)}{6R^4}\Big[3h^4 + \big(-4(al_1 + bl_2) + \big(l_1^2 + l_2^2 + l_1 l_2 c\big)\big)h^2 \\
&\qquad + 2\big(a^2 l_1^2 + b^2 l_2^2 + abl_1 l_2\big) \\
&\qquad - \frac{2}{5}\big(al_1\big(3l_1^2 + 2cl_1 l_2 + l_2^2\big)\big) \\
&\qquad - \frac{2}{5}\big(bl_2\big(l_1^2 + 2cl_1 l_2 + 3l_2^2\big)\big) \\
&\qquad + \frac{1}{5}\big(\big(l_2^4 + l_2^4\big) + cl_1 l_2\big(l_1^2 + l_2^2\big)\big) \\
&\qquad + \frac{1}{15}(1 + 2c^2) l_1^2 l_2^2\Big]. \qquad (5)
\end{aligned}$$

3.2 Convolution integral over chord segments

Integration over chord segments involve two possible cases: the angle subtending the arc is smaller than or greater than $\frac{\pi}{2}$ (Fig. 4). For the first case, the chord segment is the area of the pie minus the isosceles triangle. For the second case, the chord segment is the area of the big pie plus the isosceles triangle. Let $\boldsymbol{e_1}$ be a unit vector along one of the edges subtending the arc and $\boldsymbol{e_2}$ be a unit vector perpendicular to $\boldsymbol{e_1}$. Let point $\boldsymbol{O}$ be the projection of the point $\boldsymbol{P}$ onto the plane of the intersecting triangle. Note that $\boldsymbol{O}$ is always the center of the arc (clipping sphere). The value of the convolution integral over a pie slice at a given point $\boldsymbol{P}$ is:

$$F(\boldsymbol{P}) = \int_{\Omega} \left(1 - \frac{(\boldsymbol{OP} - u\boldsymbol{e_1} - v\boldsymbol{e_2})^2}{R^2}\right) \mathrm{d}\Omega \qquad (6)$$

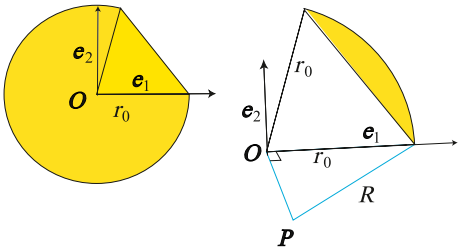

Fig. 4. Convolution integral over a chord segment. Two scenarios: the subtending angle is greater than (*left*) and smaller than (*right*) $\pi/2$

Letting $u = r\cos(\theta)$, $v = r\sin(\theta)$, and using the fact that $|\boldsymbol{OP}|^2 = R^2 - r_0^2$, where r_0 is the radius of the arc, we can obtain

$$\begin{aligned} F(\boldsymbol{P}) &= \int_0^{\alpha}\int_0^{r_0}\left(1-\frac{(\boldsymbol{OP}-(r\cos(\theta)e_1+r\sin(\theta)e_2))^2}{R^2}\right)^2 \\ &\quad \times r\,\mathrm{d}r\,\mathrm{d}\theta \\ &= \int_0^{\alpha}\int_0^{r_0}\left(1-\frac{\boldsymbol{OP}^2-r^2}{R^2}\right)^2 r\,\mathrm{d}r\,\mathrm{d}\theta \\ &= \int_0^{\alpha}\int_0^{r_0}\left(\frac{r_0^2-r^2}{R^2}\right)^2 r\,\mathrm{d}r\,\mathrm{d}\theta = \frac{\alpha\times r_0^6}{6R^4}. \end{aligned} \tag{7}$$

Equation 5 can be applied directly to compute the convolution integral over the isosceles triangle. Noting that the vertex $\boldsymbol{A}$ of the triangle $\boldsymbol{ABC}$ in the derivation of Eq. 5 is in fact the point $\boldsymbol{O}$ here, we obtain $l_1 = l_2 = r_0$, $h^2 = |\boldsymbol{AP}|^2 - R^2 = \boldsymbol{OP}^2 - R^2 = -r_0^2$, $a = 0$, and $b = 0$. Thus the integration over the isosceles triangle is

$$F(\boldsymbol{P}) = \frac{(22 - 9\times\cos(\alpha) + 2\times\cos^2(\alpha))\times r_0^6\sin(\alpha)}{90R^4}. \tag{8}$$

Finally, combining both the integrations over the pie area and over the isosceles triangle, the field function over the chord segment is

$$\begin{aligned} F(\boldsymbol{P}) &= \frac{\alpha\times r_0^6}{6R^4} \\ &\quad \pm\frac{(22-9\times\cos(\alpha)+2\times\cos^2(\alpha))\times r_0^6\sin(\alpha)}{90R^4}. \end{aligned} \tag{9}$$

4 Rendering convolution surfaces

With the field function defined, Eq. 1 can be used to locate the surface. Several methods exist for rendering implicit surfaces. Depending on the application, a different rendering method may be most suitable. Existing methods can be divided into two main categories. Indirect methods polygonize the implicit surface to within a given tolerance, allowing the use of polygon-rendering techniques and hardware for interactive inspection. Direct methods render the implicit surfaces directly without polygonizing them.

Although polygonal representation has become a standard in computer graphics industry, polygonizing implicit surfaces has two drawbacks. First, the polygonization may not accurately detect disconnected components or small features. For models with such features, it is necessary to extract at a very fine resolution, resulting in a large number of polygons. Second, the surface has to be repolygonized each time the modeling implicit surface equation $F(x, y, z) = T$ changes. Such circumstances arise in animated metamorphoses, with each component of an object modeled using an implicit equation and move relative to each other.

Sederberg and Zundel [19] developed a direct scan-line method for rendering more accurately algebraic implicit surfaces at an interactive speed. Ray-tracing is another important direct rendering method. Compared with polygonization, it may be slower, but it provides an accurate rendering for visualizing implicit surfaces.

We propose a ray tracing algorithm for rendering the proposed convolution surfaces. Let

$$\boldsymbol{r}(t) = \boldsymbol{a} + \boldsymbol{b}t \tag{10}$$

be a parametrically defined ray that is anchored at $\boldsymbol{a}$ in the direction of the unit vector $\boldsymbol{b}$. Substituting $\boldsymbol{r}(t)$ into the field function in Eq. 1 yields the following equation:

$$\sum F_i(\boldsymbol{r}(t)) = \sum F_i(\boldsymbol{a}+\boldsymbol{b}t) = \sum f_i(t) = f(t) = T. \tag{11}$$

The solutions to Eq. 11 are the intersections of the ray with the implicit surface. The challenge is how to solve the above equation quickly and reliably. Rewriting Eq. 3, we get

$$\sum_{i\in I\setminus S} F_i(\boldsymbol{a}+\boldsymbol{b}t) = T \tag{12}$$

where S denotes the set $\{i\,|\,F_i(\boldsymbol{a}+\boldsymbol{b}t)\equiv 0\}$.

We use a six-degree piecewise polynomial $p_i(t)$ to approximate the function $F_i(\boldsymbol{a}+\boldsymbol{b}t)$, satisfying the following equations:

$$p_i(c_i) = F_i(\boldsymbol{a}+\boldsymbol{b}c_i) = 0 \tag{13}$$
$$p_i(d_i) = F_i(\boldsymbol{a}+\boldsymbol{b}d_i) = 0 \tag{14}$$
$$p_i'(c_i) = \boldsymbol{b}\bullet\nabla F_i(\boldsymbol{a}+\boldsymbol{b}c_i) = 0 \tag{15}$$
$$p'(d_i) = \boldsymbol{b}\bullet\nabla F_i(\boldsymbol{a}+\boldsymbol{b}d_i) = 0 \tag{16}$$
$$p_i(c_i+\Delta t) = F_i(\boldsymbol{a}+\boldsymbol{b}(c_i+\Delta t)) = f_1 \tag{17}$$
$$p_i(c_i+2\Delta t) = F_i(\boldsymbol{a}+\boldsymbol{b}(c_i+2\Delta t)) = f_2 \tag{18}$$
$$p_i(c_i+3\Delta t) = F_i(\boldsymbol{a}+\boldsymbol{b}(c_i+3\Delta t)) = f_3. \tag{19}$$

From Eqs. 13–19, we obtain

$$p_i(t) = \begin{cases} (t-c_i)^2(t-d_i)^2\left(\frac{f_1+f_3}{18\Delta t^6} - \frac{f_2}{16\Delta t^6}\right)(t-c_i-2\Delta t)^2 \\ \quad + \frac{f_3-f_1}{18\Delta t^5}(t-c_i-2\Delta t) + \frac{f_2}{16\Delta t^4}, & c_i \le t \le d_i \\ 0, & \text{otherwise} \end{cases}$$

where $[c_i, d_i]$ denotes the i-th interval with a nonzero field along the ray. The approximation error can be represented by the following equation:

$$\frac{f_i^{(7)}(\xi)}{7!}(t-c_i)^2(t-d_i)^2(t-c_i-\Delta t) \times (t-c_i-2\Delta t)(x-c_i-3\Delta t), \qquad c_i < \xi < d_i.$$

Now the equation $\sum_{i\in S} p_i(t) = T$ is a six-degree piecewise polynomial. We use the simple and effective Bezier clipping [16] algorithm to solve the above equation. The normal vector at the ray/surface intersection point is computed as

$$\boldsymbol{n} = \left(-\frac{F(x+\Delta x, y, z)}{\Delta x}, -\frac{F(x, y+\Delta y, z)}{\Delta y}, -\frac{F(x, y, z+\Delta z)}{\Delta z}\right).$$

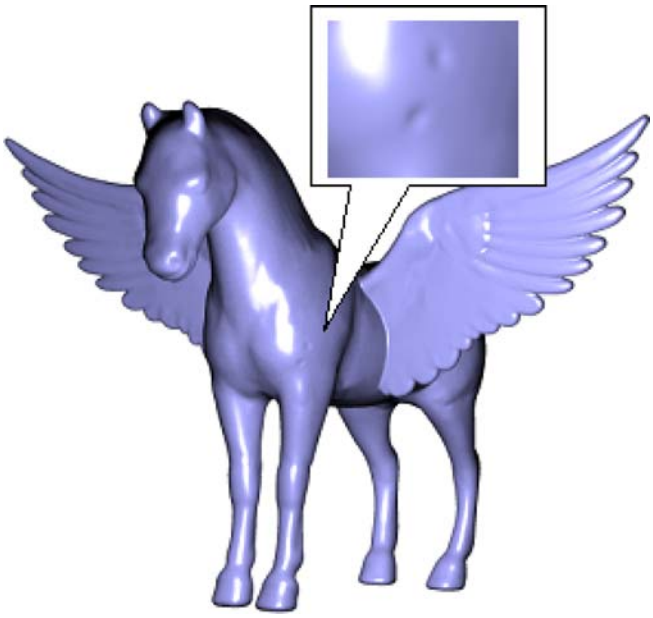

Fig. 5. An unsatisfactory repair result. The horse polygonal model has two holes on a front leg. Our implicit surface fills the holes but is not smooth. Using a larger effective radius can solve this problem

Fig. 6. Results of convolving a triangular segment using the Cauchy kernel (*left*) and using our method with $R = 0.25$ (*right*). Their computation time is 32 311 ms, with 165 millions evaluations per second. Our method has comparable performance: the number of evaluations (in millions) per second are 223 ($R = 0.25$), 163 ($R = 0.5$), 150 ($R = 0.75$), 166 ($R = 1.0$), 210 ($R = 1.25$)

Octree partitioning is a popular technique to accelerate polygonization [4] and ray tracing of implicit surfaces [11]. We also use spatial partitioning to accelerate the computation of the intervals $[c_i, d_i]$. Each node of the octree contains a list of triangles with nonzero contribution to the corresponding cell. One after another, the ray traverses through each cell and compute the intersecting intervals. Experiment results show that spatial partitioning can substantially reduce the computation cost.

5 Results and discussion

We have implemented the proposed algorithm on a Pentium 4 CPU 3.0 GHZ with 1 GB main memory. Figure 7 shows some examples of convolution surfaces constructed from a variety of polygon models using different effective radii ($R = 0.00001$, 1.0, 2.0, 5.0). Instead of normalizing the effective radius using the average triangle size, in these experiments, we simply scale all input models to a $(-200, -200, -200) \times (200, 200, 200)$ volume. For simplicity, we also use the same number

Model	Triangles	$R = 0.00001$	$R = 1.0$	$R = 2.0$	$R = 5.0$
Heptoroid	35 840	4.7	39.5	156.4	261.5
Jackolan	11 002	4.5	17.2	54.3	92.1
Hand	654 666	2.3	882.0	9751.0	25 734.9
Turtle	267 931	3.4	227.7	1326.4	3169.6
Armadillo	345 944	4.3	393.2	2218.3	4813.3
Dragon	118 793	4.3	108.1	501.9	1008.5
Buddha	160 182	2.8	165.6	1003.7	1905.1

Table 1. Rendering times in seconds for models of different sizes and using different effective radii, all ray-traced at 800×600 resolution

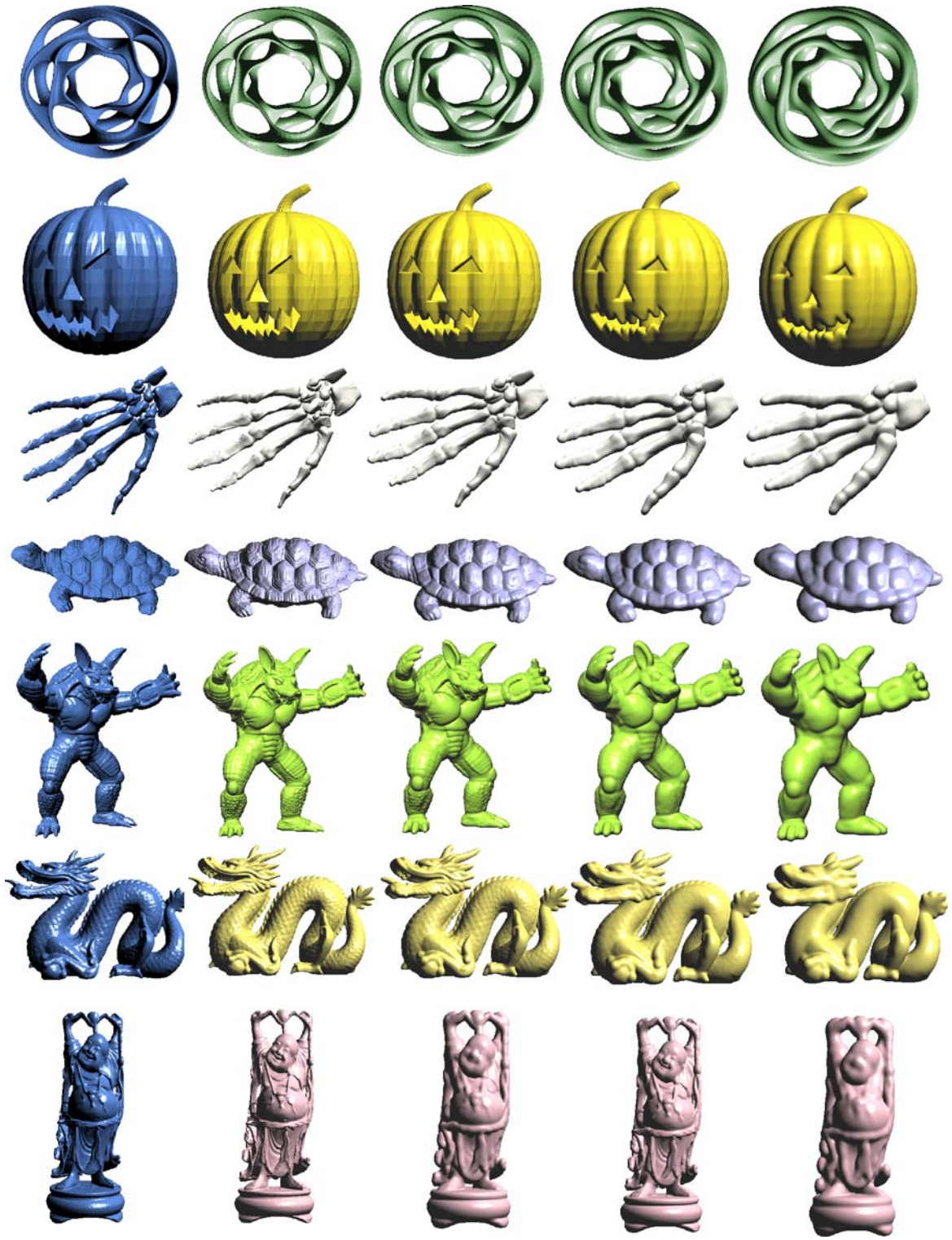

Fig. 7. *Leftmost column* shows Phong-shaded polygonal models, each containing 12 400 triangles. The *remaining four columns* are ray-traced implicit surfaces created with different effective radius (0.00001, 1.0, 2.0, 5.0)

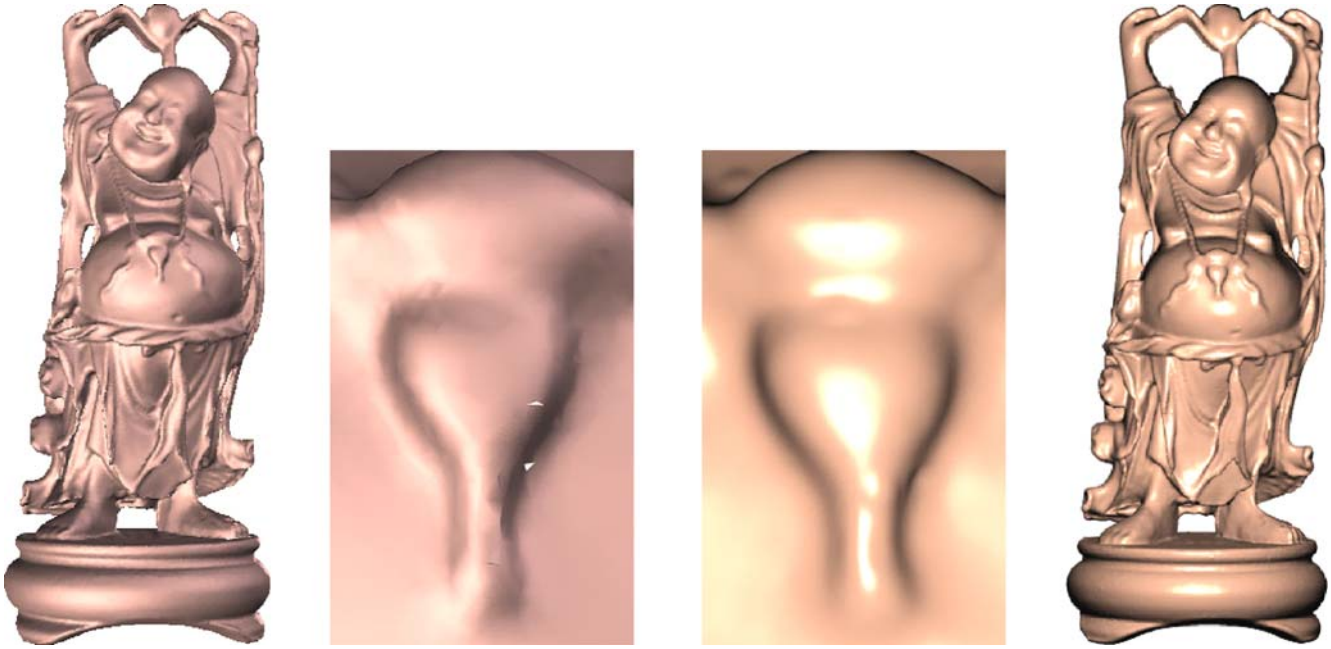

Fig. 8. The happy Buddha polygon model (*left*), which has two holes in a local area (*middle left*), rendered using Phong shading. The ray-traced convolution surface (*right*) with the holes repaired (*middle right*)

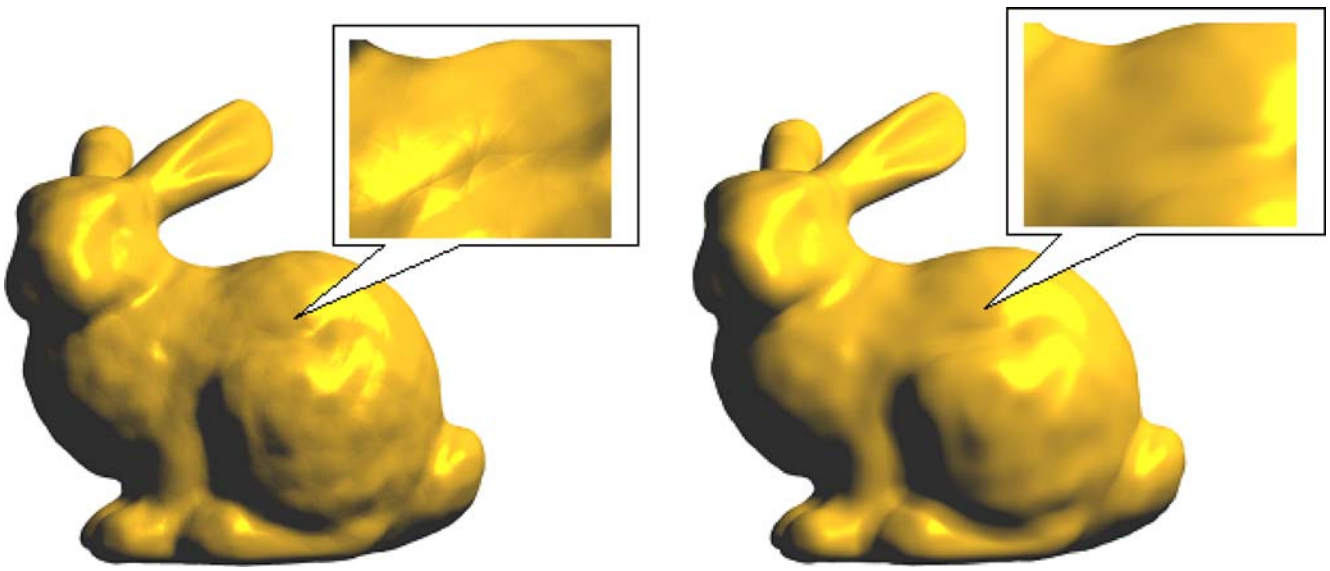

Fig. 9. Result using Sherstyuk method showing noticeable discontinuities due to truncation error (*left*) and smooth result using our method (*right*)

of triangles (i.e., 12 400) for all the models. Small effective radii (relative to the size of triangles) produce implicit surfaces that are closer to the polygon models. Larger radii produce blobby implicit surfaces. Table 1 lists the rendering times for different sizes of input models and effective radius values, with all the resulting surfaces ray-traced at 800×600 resolution. Note that the rendering time increases as the effective radius increases since evaluating at each point requires integrating over more triangles.

Like other implicit-surface construction algorithms, our algorithm can repair defects in polygonal models. Figure 8 shows a happy-Buddha model containing holes, self-intersections, and non-manifold structures. The resulting convolution surface is smooth, without these defects. To successfully repair a hole or a defective region, the effect-

ive radius needs to be sufficiently big, taking into account the size of the hole. Figure 5 shows an unsatisfactory repair result which can be overcome by increasing the effective radius.

McCormack and Sherstyuk [13] introduced the Cauchy kernel function $h(r) = 1/(1 + s^2r^2)^2$ that also leads to a closed-form solution for convolving with triangles. We compare our method with their method and find that the two computation costs are similar when evaluating at a specific point, as shown in Fig. 6. Their kernel has infinite support, to make the method computationally feasible, they introduce a bounding box for each skeletal primitive and consider its field value as zero when an evaluated point is outside the bounding box. However, such truncation errors may cause noticeable discontinuities in the implicit surfaces. This effect is evident in Fig. 9 as well as in [21]. Comparing with their method, our method has two merits. First, our method has no truncated errors, producing smoother results. Second, although both methods use a polynomial function to approximate the field function $F_i(\boldsymbol{a} + \boldsymbol{b}t)$ during rendering, we use a six-degree piecewise polynomial instead of a Hermite cubic polynomial, leading to more accurate results.

6 Conclusion

We present an efficient method to construct implicit surfaces from polygon data through deriving a closed-form solution for convolving a quartic polynomial kernel function with triangles. The finite support of the kernel function leads to a smaller computational cost as well as better visual results than previous formulation for convolving with triangles. Compared to previous implicit modeling approaches from polygon data, our method is computationally efficient, does not require building special data structure for the evaluation, and can be ray-traced efficiently. The method has potential in applications such as sketch-based modeling and cut-and-paste editing.

Acknowledgement We would like to thank Hui Zhao for his help in making the examples in Fig. 5 and reporting the rendering times in Table 1. This work is supported in part by grants from the Natural Science Foundation of Zhejiang Province (R105431), the National Natural Science Foundation of China (60573153), the China 863 Program (2006AA01Z314) and the Research Grant Council of the Hong Kong Special Administrative Region, China (HKUST6295/04E).

References

1. Blinn, J.F.: A Generalization of algebraic surface drawing. ACM Trans. Graph. **1**(3), 235–256 (1982)
2. Bloomenthal, J., Shoemake, K.: Convolution Surfaces. In: Proceedings of SIGGRAPH '91, pp. 251–256. ACM, New York (1991)
3. Bloomenthal, J., Bajaj, C., Blinn, J., Cani, M., Rockwood, A., Wyvilland, B.G.: An Introduction to Implicit Surfaces. Morgan Kaufmann Publishers, Los Altos, CA (1997)
4. Bloomenthal, J.: Polygonization of implicit surfaces. Comput. Aided Geom. Des. **5**(4), 341–355 (1988)
5. Bloomenthal, J.: Bulge elimination in convolution surfaces. Comput. Graph. Forum **16**(1), 31–41 (1997)
6. Cani, M.P., Desbrun, M.: Animation of deformable models using implicit surfaces. IEEE Trans. Vis. Comput. Graph. **3**(1), 39–50 (1997)
7. Carr, J., Beatson, R., Cherrie, J., Mitchell, T., Fright, W., McCallum, B.: Reconstruction and representation of 3D objects with radial basis functions. In: Proceedings of SIGGRAPH '01, pp. 67–76. ACM, New York (2001)
8. Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T., Nishita, T.: A simple, efficient method for realistic animation of clouds. In: Proceedings of SIGGRAPH '00, pp. 19–28. ACM, New York (2000)
9. Hornus, S., Angelidis, A., Cani, M.P.: Implicit modeling using subdivision curves. Visual Comput. **19**(2–3), 94–104 (2003)
10. Jin, X., Tai, C.L.: Convolution surfaces for arcs and quadratic curves with a varying kernel. Visual Comput. **18**(8), 530–546 (2002)
11. Kalra, D., Barr, A.H.: Guaranteed ray intersections with implicit surfaces. In: Proceedings of SIGGRAPH '89, pp. 297–306. ACM, New York, NY (1989)
12. Kanai, T., Ohtake, Y., Kase, K.: Hierarchical error-driven approximation of implicit surfaces from polygonal meshes. In: Proceedings of Eurographics Symposium on Geometry Processing, pp. 21-30. Eurographics Association, Aire-la-Ville (2006)
13. McCormack, J., Sherstyuk, A.: Creating and rendering convolution surfaces. Comput. Graph. Forum **17**(2), 113–120 (1998)
14. Museth, K., Breen, D., Whitaker, R., Barr, A.: Level set surface editing operators. ACM Trans. Graph. **21**(3), 330–338 (2002)
15. Nishimura, H., Hirai, M., Kawai, T.: Object modeling by distribution function and a method of image generation. Image Generation Trans. IECE **68**(4), 718–725 (1985)
16. Nishita, T., Nakamae, E.: A method for displaying metaballs by using Bézier clipping. Comput. Graph. Forum **13**(3), 271–280 (1994)
17. Oeltze, S., Preim, B.: Visualization of vasculature with convolution surfaces: method, validation and evaluation. IEEE Trans. Med. Imaging **24**(4), 540–548 (2005)
18. Ohtake, Y., Belyaev, A., Seidel, H.P.: A multiscale approach to 3D scattered data interpolation with compactly supported basis functions. In: Proceedings of Shape Modeling International, pp. 292–300. IEEE Computer Society, Washington, DC (2003)
19. Sederberg, T.W., Zundel, A.K.: Scan line display of algebraic surfaces. In: Proceedings of SIGGRAPH '89, pp. 145–156. ACM, New York, NY (1989)
20. Shen, C., O'Brien, J.F., Shewchuk, J.R.: Interpolating and approximating implicit surfaces from polygon soup. ACM Trans. Graph. **23**(3), 896–904 (2004)
21. Sherstyuk, A.: Kernel functions in convolution surfaces: a comparative analysis. Visual Comput. **15**(4), 171–182 (1999)
22. Turk, G., O'Brien, J.F.: Modeling with implicit surfaces that interpolate. ACM Trans. Graph. **21**(4), 855–873 (2002)
23. Wyvill, B., Wyvill, G.: Field functions for implicit surfaces. Visual Comput. **5**(1–2), 75–82 (1989)
24. Wyvill, G., McPheeters, C., Wyvill, B.: Data structure for oft objects. Visual Comput. **2**(4), 227–234 (1986)
25. Yngve, G., Turk, G.: Robust creation of implicit surfaces from polygonal meshes. IEEE Trans. Vis. Comput. Graph. **21**(4), 346–355 (2002)

XIAOGANG JIN is a professor of the State Key Lab of CAD&CG, Zhejiang University. He received his B.Sc. degree in computer science in 1989, M.Sc. and Ph.D. degrees in applied mathematics in 1992 and 1995, all from Zhejiang University. His current research interests include implicit surface computing, special effects simulation, mesh fusion, texture synthesis, crowd animation, cloth animation and facial animation.

CHIEW-LAN TAI is currently an associate professor at the Department of Computer Science & Engineering, Hong Kong University of Science & Technology. She received her B.Sc. and M.Sc. in mathematics from the University of Malaya, M.Sc. in computer and information sciences from the National University of Singapore, and D.Sc. in information science from the University of Tokyo. Her research interests include digital geometry processing and computer graphics.

HAILIN ZHANG is a software engineer at ATI Technologies Inc., Shanghai. He has an M.Sc. in applied mathematics from Zhejiang University. His research interests include implicit surface modeling and animation.