

Analytical solutions for sketch-based convolution surface modeling on the GPU

Xiaoqiang Zhu · Xiaogang Jin · Shengjun Liu · Hanli Zhao

Published online: 8 December 2011
© Springer-Verlag 2011

Abstract Convolution surfaces are attractive for modeling objects of complex evolving topology. This paper presents some novel analytical convolution solutions for planar polygon skeletons with both finite-support and infinite-support kernel functions. We convert the double integral over a planar polygon into a simple integral along the contour of the polygon based on Green's theorem, which reduces the computational cost and allows for efficient parallel computation on the GPU. For finite support kernel functions, a skeleton clipping algorithm is presented to compute the valid skeletons. The analytical solutions are integrated into a prototype modeling system on the GPU (Graphics Processing Unit). Our modeling system supports point, polyline and planar polygon skeletons. Complex objects with arbitrary genus can be modeled easily in an interactive way. Resulting convolution surfaces with high quality are rendered with interactive ray casting.

Electronic supplementary material The online version of this article (doi:10.1007/s00371-011-0662-z) contains supplementary material, which is available to authorized users.

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

X. Zhu
e-mail: zhuxiaoqiang@zjucadcg.cn

S. Liu
School of Mathematical Science and Computing Technology,
Central South University, Changsha, 410083, P.R. China
e-mail: shjliu.cg@gmail.com

H. Zhao
College of Physics & Electronic Information Engineering,
Wenzhou University, Wenzhou, 325035, P.R. China
e-mail: hanlizhao@gmail.com

Keywords Convolution surface · Closed-form solution · Planar polygon skeleton · Sketch-based modeling · CUDA

1 Introduction

Traditional modeling systems can generate complex objects, however, they take too much time for novice users to learn. As one of the most intuitive and simplest ways, drawing is usually one of the best choices for such users. Moreover, it is desirable to express the ideas and visions of designers at the earliest possible stages. Therefore, a lot of sketch-based modeling systems have been developed recently [11, 17, 24, 27, 31].

Rotund objects are usually expected for sketch-based systems. Therefore, convolution surfaces are a good choice for sketch-based modeling because of its pleasing superposition property. A convolution surface is defined by convolving a skeleton with a three-dimensional low-pass filter kernel. The skeletal elements can be points, polylines, polygons, and even volumes. As skeletons are natural abstractions for shapes, convolution surfaces offer a new means for sketch-based surface modeling [1, 2, 27]. In the sketch-based modeling system, constructing a 3D shape based on the 2D silhouette is the most important tool to create new objects [11]. In the convolution surface-based *ConvMo* system [27], Tai et al. approximate the medial axis of a given silhouette curve as a set of line segments, and create a convolution surface by convolving a linearly weighted kernel along each segment. Later, Alexe et al. perform extraction of a graph of branching polylines and polygons from silhouette contours [2]. The polygons in the graph have to be further triangulated to create the convolution surface of the graph. Unlike these systems, we use the sketched silhouette as a new type of skeleton and compute its convolution surface on the GPU. By

converting the double integral over a planar polygon into a simple integration along the contour of the polygon based on Green's theorem, the triangulation process for the planar polygon, and hence the redundant computations involved are avoided. As a result, our approach allows for parallel computation on the GPU.

The majority of previously available sketch-based modeling systems relying on implicit surfaces used the iso-surface mesh extraction algorithms. However, details may be lost because of the sampling resolution of the polygonization process. This motivates us to render the resulting surface using ray-casting because it allows us to produce high-quality images. As ray-casting convolution surfaces is usually computational intensive, we try to use GPU and the natural bounding volumes of the skeletons to accelerate the ray/surface intersection so that ray-casting can be performed interactively.

The main contributions of the paper are the derivation of closed-form solutions for convolving planar polygon skeletons and its practical use in a prototype sketch-based modeling system on the GPU. We calculate the fields of all the sketched skeletons on the GPU. For a sketched silhouette curve, we convert it into a closed planar polygon and then calculate its field. By transforming the double integral of the polygon into a line integral using Green's theorem, our solution is both analytical and efficient. A fast ray-casting method is also presented to render high-quality sketched convolution surfaces by making use of parallel computation on the GPU.

The remainder of this paper is organized as follows. After introducing the related work in Sect. 2, we derive the closed-form convolution surface solutions for planar polygon skeletons with both the quartic polynomial kernel (with finite support) and the Cauchy kernel (with infinite support) in Sect. 3. In Sect. 4, a prototype modeling system using the closed-form solutions is presented. The implementation details and some modeling results with our system are discussed in Sect. 5, and our paper ends with the conclusion section.

2 Related work

Different from traditional 3D modeling software packages, sketch-based modeling can be used to create 3D shapes conveniently by novice users without professional skills. Convolution surfaces are often used in such systems because of their smooth blending behavior and automatic changes of topology defined by the underlying skeletons. In this section, we will introduce the related work on convolution surfaces and sketch-based modeling using implicit surfaces.

Convolution surfaces. Implicit surfaces are conveniently adopted to model complex objects, such as tree branches,

liquids, and organic shapes. The reason is that the field-based implicit surfaces can easily deal with smooth deformable shapes with varying topologies. Skeleton-based implicit surfaces, including metaballs, distance surfaces, and convolution surfaces, are quite suitable for animation because the skeletons can be directly used for skeleton-based animation. A convolution surface is defined as convolving geometric skeletons (e.g., lines, curves and polygons) with Gaussian kernel functions [5]. Later, many different kernels are introduced. These kernels can be classified into infinite support kernels (e.g., Cauchy kernel [22]) and finite support kernels (e.g., quartic polynomial kernel [15]). For finite support kernels, the field falls to zero when the distance from the point in question to the skeleton is greater than the effective radius of the kernel. For infinite support kernels, the field never falls to zero even the point in question is quite far from the skeleton. Theoretically, any geometric primitive can be used as a skeleton to create convolution surfaces. However, closed-form solutions for convolution surfaces depend on both the kernel functions and the skeleton primitives. Researches show that analytical convolution solutions exist for skeletons such as line segments, planes, triangles, arcs, and quadratic spline curves [12–14, 22, 26]. Before the closed-form solutions for convolving triangle skeletons with quartic polynomial kernels is introduced [15], the analytical convolution solutions for triangle skeletons with Cauchy kernels were presented by McCormack and Shertshyk [22], which allows the authors of the sketch-based system to create models with planar surfaces [2]. Angelidis et al. [3] model convolution surfaces based on subdivision curves and subdivision surfaces. Another application of closed-form solutions for line skeletons is introduced by Kravtsov et al. [20]. They simulate the interaction of viscous objects with meshes in which a convolution surface is embedded. Recently, Hubert [10] independently derived similar analytical solutions for convolving planar polygon skeletons by applying Green's theorem.

Sketch-based modeling using implicit surfaces. As implicit surfaces can be merged together naturally and easily, lots of researches adopt implicit surfaces as modeling primitives. These systems include BlobTrees [29, 30], RBF-based [18], and level set-based [7]. Compared to the explicit surface-based sketching modeling system, implicit surface-based methods can design smooth models with complex topology easily. Karpenko et al. [18] propose a sketch-based system using RBF, in which the input sketches are used to generate the 3D constraint points. However, with the increase of the number of constraint points, the system slows down rapidly because of the linear system involved. Moreover, the linear system must be recalculated if a point of the model is modified. BlobTree [30] extends the CSG operations by allowing intuitive global and local operations; it is a tree-based shape representation and has been successfully

used in the sketch-based ShapeShop [23, 24] system. As a skeleton-based modeling tool [5], convolution surfaces were naturally introduced into sketch-based modeling [27]. The sketch-based system in [1] extracts the point skeletons of the sketched outlines, and then utilizes spherical implicit functions to construct the blobby 3D models. In order to model complex objects such as cylindrical and planar surfaces, too many point-skeletons are needed to avoid bumps. Cylindrical surfaces can be easily modeled using line skeletons [2, 4, 27], as convolution solutions for line skeletons can be obtained analytically [12–14, 26]. The triangle skeletons are also adopted by the system in [2] to construct the palm component of a hand model where an extracted polygon has to be triangulated in order to allow for the evaluation of its field values.

3 Field value calculation for planar polygon skeletons

In this section, after giving a brief introduction to convolution surfaces, we deduce analytical solutions for planar polygon skeletons with different kernels using a curve integral method. Our approach allows us to efficiently evaluate the field value at any point analytically.

3.1 Convolution surfaces

A convolution surface is an iso-surface defined in an implicit scalar field whose values are evaluated by accumulating contributions from 3D points of a skeleton. The contribution usually decreases rapidly when the distance between a point in space and a skeleton is increased. Here, we adopt the convolution surface definition by McCormack and Sherstyuk [22]. Let $\mathbf{p}(x, y, z)$ be a space point in \mathbb{R}^3 , and $g : \mathbb{R}^3 \rightarrow \mathbb{R}$ be a geometric function representing a modeling skeleton V :

$$g(\mathbf{p}) = \begin{cases} 1, & \mathbf{p} \in \text{skeleton } V \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Let $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ be a potential function which defines the field with a single point in the skeleton V , and let \mathbf{q} be

a point in V . Then the total field value contributed by the skeleton at point \mathbf{p} is the convolution of functions f and g :

$$F(\mathbf{p}) = \int_V g(\mathbf{q})f(\mathbf{p} - \mathbf{q}) dV = (f \otimes g)(\mathbf{p}). \quad (2)$$

f is also called the convolution kernel function. Theoretically, any geometric primitive can be used as a skeleton. However, closed-form solutions depend on both the kernel and the skeleton.

3.2 Curve integral for planar polygon skeletons

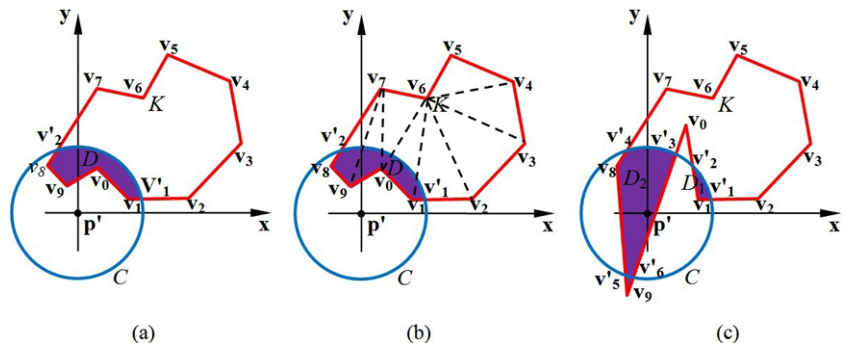
Let $K(\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2 \cdots \mathbf{v}_n\mathbf{v}_0)$ be a planar polygon skeleton, the field value at a point \mathbf{p} for this skeleton is computed by a double integral according to (2). For a kernel function with infinite support, the integration is evaluated on the entire polygon. However, for a finite support kernel function, the valid skeletons are areas in the clipping sphere centered at \mathbf{p} , as shown in Fig. 1(a). Generally, the intersecting area is a polygon or multiple polygons whose boundaries consist of arcs and line segments. It is difficult to perform the integration on the intersecting area directly. One solution is to decompose the polygon into triangle segments and chord segments, and then evaluate the convolution integral for each type [15], as shown in Fig. 1(b). However, such a solution is not suitable for GPU implementation because GPU branching is usually slow. Here, we present a closed-form solution by converting the double integral into a curve integral using Green’s theorem. As the integral computation can be performed for each edge one by one, we can make better use of the GPU’s enormous floating point computing capacity.

The field value at point \mathbf{p} for a planar polygon skeleton can be computed directly without the polygon triangulation. With the integration domain D and its boundary l , the field value at point \mathbf{p} is:

$$F(\mathbf{p}) = \iint_D f(r) dx dy, \quad (3)$$

where $r = \|\mathbf{p} - \mathbf{q}\|$ denotes the Euclidean distance between points \mathbf{p} and \mathbf{q} , $\mathbf{q} \in D$. This double integral can be computed

Fig. 1 The regions for the convolution integral. The valid polygon skeleton for finite support kernels is shadowed in (a). Illustration in (b) triangulates the polygon before using closed-form convolution surface solutions for triangle skeletons. (c) shows our clipping between a general polygon and the clipping sphere



by converting it into a curve integration using Green’s theorem [28]:

$$\iint_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy = \oint_l P dx + Q dy, \tag{4}$$

where P and Q are functions with continuous first-order derivatives. In order to satisfy (4), P and Q must be designed elaborately. Here, we set $P = 0$ and rewrite (4) as follows:

$$\iint_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy = \iint_D \frac{\partial Q}{\partial x} dx dy = \oint_l Q dy. \tag{5}$$

In our case, the boundary l consists of arcs and line segments. Taking Fig. 1(a) as an example, the boundary of the integration domain D includes an arc and five line segments. The field contribution at \mathbf{p} is the sum of the integrals along each boundary primitive:

$$\begin{aligned} F(\mathbf{p}) &= \iint_D f(r) dx dy = \oint_l Q dy \\ &= \int_{arcs} Q dy + \int_{linesegments} Q dy \\ &= \sum_i \int_{c_i} Q dy + \sum_j \int_{l_j} Q dy. \end{aligned} \tag{6}$$

Once the kernel function is chosen, the continuous function Q can be determined. In the next three subsections, we will deduce the analytical field functions for two commonly used field functions which are adopted in our system.

3.3 Evaluation of a valid skeleton for planar polygons

In this section, we will discuss how to compute the valid skeleton for a planar polygon skeleton. As we convert the double integral into a line integral, the valid skeleton for infinite support kernels is just the union of the edges of the polygon. However, for finite support kernels, a key step is to determine the valid skeletons which have field contribution.

For a 3D point \mathbf{p} , let R be the effective radius of the finite support kernel, we define the sphere centered at \mathbf{p} with radius R as the *clipping sphere*. We compute the valid skeleton by adopting an approach similar to the Weiler–Atherton polygon clipping algorithm [6]. In our case, the clipping window is a clipping circle. The circle is the intersection of the clipping sphere and the plane the polygon lies. Given a polygon $K(\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2 \cdots \mathbf{v}_n\mathbf{v}_0)$ (here, we take $n = 9$ as an example in Fig. 1(c)) and a circle C , the boundary of the intersection consists of some boundaries from K and C . Connecting the intersection points forms the boundary. The intersection points are classified into *income point* at which the boundary of K comes into the circle and *outcome point*.

We use Algorithm 1 to describe the clipping algorithm. After applying the clipping algorithm to the given polygon, a

Algorithm 1 Skeleton clipping

- 1: Create a vertex list L_K for the given polygon $K(\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2 \cdots \mathbf{v}_n\mathbf{v}_0)$ and an empty list L_C for the clipping circle C ;
- 2: Find the intersection points between polygon K and circle C , flag them with income point or outcome point, insert them into list L_K along the polygon boundary, and list L_C according to the angles formed by the lines from the circle center to the intersection points and a line from the circle center to the first intersection point. Create a double link between the same intersection points in two lists L_K and L_C ;
- 3: Clipping. If there are intersection points which have not been traced, do the following steps:
- 4: Create an empty vertex list L_r for resulting polygons;
- 5: Choose an intersection point randomly as a start point, and insert it into L_r ;
- 6: If the intersection point is an income point, we trace the polygon boundary along list L_K ; otherwise, we trace the clipping circle boundary along list L_C ;
- 7: Trace the polygon boundary L_K or L_C . Insert polygon vertices into L_r until a new intersection point is found;
- 8: Insert the intersection point into L_r , and change the tracing direction according the double link at the intersection point;
- 9: Repeat steps 7 and 8 until we are back to the start point;

set of vertices are obtained. They are intersection points and some polygon vertices. Two adjacent vertices form a path of our convolution integral. The path is either a line segment or an arc. For two adjacent vertices, if one of them is a vertex of the polygon, then the path is a line segment. Otherwise, both of them are intersection points. In this case, if the first vertex is an income point, then the path is a line segment; otherwise, the path is an arc. Taking Fig. 1(c) as an example, after performing the intersection calculation, we create a vertex list L_K of the given polygon $K(\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2 \cdots \mathbf{v}_9\mathbf{v}_0)$ and a list L_C for the clipping circle. The intersection points $\mathbf{v}'_1, \mathbf{v}'_2, \mathbf{v}'_3, \mathbf{v}'_4, \mathbf{v}'_5, \mathbf{v}'_6$ will be inserted into two lists L_K and L_C , and a double pointer will be created at each intersection point between two lists, as shown in Fig. 2(a). In the clipping procedure, we choose an intersection point with an income point flag as the start point for tracing. As shown in Fig. 2(b), \mathbf{v}'_2 is the start point which is flagged with a small red triangle. By tracing two lists, a set of resulting points, $L_{r,1}$ is obtained. The second tracing is shown in Fig. 2(c). After clipping, there are two subpolygons stored in the resulting lists $L_{r,1}$ and $L_{r,2}$.

However, it will consume a lot of GPU memory if we store list L_K for each thread on the GPU. We solve this problem by performing the convolution integral as soon as a valid line skeleton is available. In this way, we need not

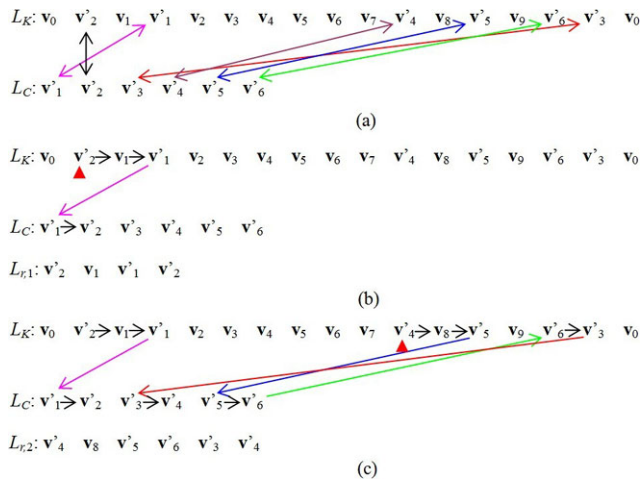


Fig. 2 The polygon clipping procedure. The intersection points were inserted into Lists L_K and L_C with a double link in (a). Two new sub-polygons were created in (b) and (c)

store L_K at all. The list L_C for valid arc skeletons is created by inserting the intersection point with flag *income point* or *outcome point*. As the list L_C is usually not too long, we can store it in the local memory of each thread. A list of sixteen intersection points is empirically enough in our implementation. Taking Fig. 1(c) as an example, as soon as a clipped line segment ($v'_2v_1, v_1v'_1, v'_4v_8, v_8v'_5$, or $v'_6v'_3$) between the clipping sphere and the planar polygon skeleton is obtained, we calculate its integral. In the meantime, we store the intersection points ($v'_2, v'_1, v'_4, v'_5, v'_6$, and v'_3) in list L_C . Then we calculate the algebraic sum of integrals along the clipped arcs.

3.4 Finite support kernel functions

If we take the following quartic polynomial as the kernel function:

$$f(r) = \begin{cases} (1 - \frac{r^2}{R^2})^2, & r \leq R \\ 0, & r > R \end{cases} \tag{7}$$

where R is the effective radius of the kernel, the analytic solution of (5) can be obtained by using the curve integral method:

$$\iint_D \frac{\partial Q}{\partial x} dx dy = \oint_l \left(\int \frac{\partial Q}{\partial x} dx \right) dy. \tag{8}$$

To deduce the solution conveniently, in polygon K , we set up a local coordinate system whose origin is the projection point of \mathbf{p} on the polygon. Two coordinate axes \mathbf{x} and \mathbf{y} are any two orthogonal vectors. Here, we set the \mathbf{x} -axis parallel with one edge of the polygon and the \mathbf{y} -axis perpendicular with \mathbf{x} -axis, then the \mathbf{z} -axis is the cross product of \mathbf{x} and \mathbf{y} . In Fig. 1, the \mathbf{x} -axis and edge v_1v_2 are parallel. In the new coordinate system, $p = (0, 0, z_0)$, where z_0 is the distance

from \mathbf{p} to the plane where polygon K lies. From (7) and (8), we can get the following equation:

$$\begin{aligned} F(\mathbf{p}) &= \frac{1}{R^4} \oint_l \left[\int (r_0^2 - x^2 - y^2)^2 dx \right] dy \\ &= \frac{1}{R^4} \oint_l M dy \\ &= \frac{1}{R^4} \left(\sum_i \int_{c_i} M dy + \sum_j \int_{l_j} M dy \right), \end{aligned} \tag{9}$$

where $r_0^2 = R^2 - z_0^2$, and $M = \frac{x^5}{5} - \frac{2}{3}x^3(r_0^2 - y^2) + x(r_0^2 - y^2)^2$. By calculating the integrals along arcs and line segments respectively, we can obtain the field for \mathbf{p} .

(1) The integration along an arc c_i

Let $(x, y) = (r_0 \cos \theta, r_0 \sin \theta)$, and $(dx, dy) = (-r_0 \sin \theta, r_0 \cos \theta)d\theta$, from (9) we can obtain

$$\begin{aligned} &\int_{c_i} M dy \\ &= \int_{y_1}^{y_2} M dy \\ &= r_0^6 [60(\theta_2 - \theta_1) + 45(\sin 2\theta_2 - \sin 2\theta_1) \\ &\quad + 9(\sin 4\theta_2 - \sin 4\theta_1) + (\sin 6\theta_2 - \sin 6\theta_1)]/360, \end{aligned} \tag{10}$$

where θ_1, θ_2 are corresponding angles of the start point and the end point of arc c_i .

(2) The integration along a line segment l_j

A line segment from $v_j(x_j, y_j)$ to $v_{j+1}(x_{j+1}, y_{j+1})$ can be represented in a parametric form as following:

$$\begin{cases} x = x_j + (x_{j+1} - x_j)t = x_j + at \\ y = y_j + (y_{j+1} - y_j)t = y_j + bt \end{cases}, \quad 0 \leq t \leq 1. \tag{11}$$

It is easy to know that $dx = (x_{j+1} - x_j)dt = a dt$, $dy = (y_{j+1} - y_j)dt = b dt$, where $a = x_{j+1} - x_j$, $b = y_{j+1} - y_j$. Then we can obtain:

$$\begin{aligned} &\int_{l_j} M dy \\ &= \int_{y_j}^{y_{j+1}} M dy = \int_0^1 Mb dt \\ &= b(3a^5 + 18a^4x_j + 6a^2x_jE + 3aF \\ &\quad + a^3G + 2x_jH)/90, \end{aligned} \tag{12}$$

where

$$\begin{aligned} p &= x_j^2 + y_j^2 - r^2, & q &= x_j^2 + 3y_j^2, & A &= r^2 - q, \\ B &= q - 3r^2, & C &= 3x_j^2 + y_j^2 - r^2, \end{aligned}$$

$$\begin{aligned}
 D &= x_j^2 + 9y_j^2 - 3r^2, \\
 E &= 6b^2 + 15by_j + 10p, \\
 F &= 5b^4 + 24b^3y_j - 15b^2A + 40by_jp + 15p^2, \\
 G &= 10b^2 + 24by_j + 15C, \\
 H &= 9b^4 + 45r^4 + 9x_j^4 + 45b^3y_j + 30x_j^2y_j^2 + 45y_j^4 \\
 &\quad - 30r^2q + 30by_jB + 10b^2D.
 \end{aligned}$$

3.5 Infinite support kernel functions

Infinite support kernel functions, such as Gaussian kernel, Cauchy kernel and power inverse kernel, are also well adopted in convolution surface modeling. For such kernels, skeletons impose contributions to any point in question even if its distance to the skeleton is large.

Here, we take the Cauchy kernel function as an example to deduce the analytical field function:

$$f(r) = \frac{1}{(1 + s^2r^2)^2}, \tag{13}$$

where s is a parameter to control the width of the kernel.

The field for point \mathbf{p} can be computed according to (3), (5):

$$\begin{aligned}
 F(\mathbf{p}) &= \iint_D \frac{1}{(1 + s^2(x^2 + y^2 + z_0^2))^2} dx dy \\
 &= \frac{1}{s^4} \oint_l \left[\int \frac{1}{(m + x^2 + y^2)^2} dx \right] dy = \frac{1}{s^4} \oint_l M dy,
 \end{aligned} \tag{14}$$

where $m = z_0^2 + \frac{1}{s^2}$, and

$$M = \frac{x}{2(m + y^2)(m + x^2 + y^2)} + \frac{\arctan(\frac{x}{\sqrt{m+y^2}})}{2(m + y^2)^{3/2}}.$$

As the Cauchy kernel function is infinite support, the boundary of the integration domain must be the line segments of planar polygon skeletons. Then the curve integral can be rewritten as:

$$\oint_l M dy = \sum_i \int_{l_i} M dy. \tag{15}$$

For each line segment, we also use its parametric representation as shown in (11). The integral over a line segment $l_i(\mathbf{v}_i \mathbf{v}_{i+1})$ can be computed as:

$$\begin{aligned}
 \int_{l_i} M dy &= \int_{y_i}^{y_{i+1}} M dy = \int_0^1 bM dt
 \end{aligned}$$

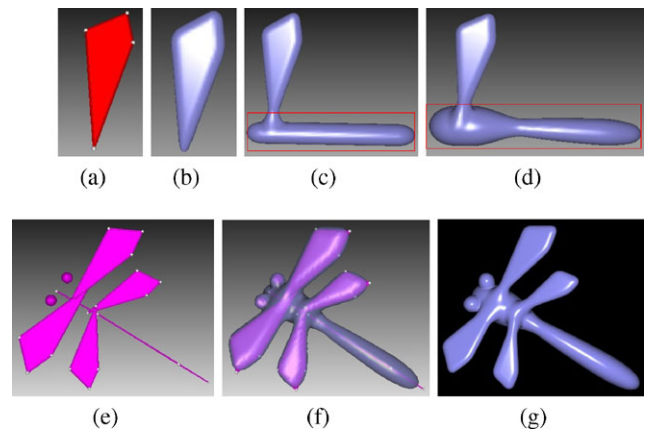


Fig. 3 The modeling framework. The model in (a) is a sketched silhouette which is used to create the convolution surface in (b). The figures in (c) and (d) illustrate the added components and edition of the component respectively. The overall skeletons can be seen in (e), and the final shape is rendered using marching cubes (f) and ray-casting (g)

$$\begin{aligned}
 &= \frac{1}{4m} \left[\frac{2B \arctan \frac{A}{w}}{w} + \left[2(bx_i - ay_i)p \arctan \frac{aA + bB}{q} \right. \right. \\
 &\quad \left. \left. - 2abmq \log |b| \right] / (pq) \right],
 \end{aligned} \tag{16}$$

where

$$\begin{aligned}
 A &= at + x_i, \quad B = bt + y_i, \quad w = \sqrt{m + B^2}, \\
 p &= b^2x_i^2 - 2abx_iy_i + a^2(m + y_i^2), \\
 q &= \sqrt{b^2(m + x_i^2) - 2abx_iy_i + a^2(m + y_i^2)}.
 \end{aligned}$$

4 Prototype modeling system

In this section, we introduce a prototype modeling system which employs the analytical solutions. To accelerate the ray-casting of the resulting surface, we make full use of the bounding volumes of the skeletons. As a result, the ray-casting of the created shapes can be performed in an interactive way.

4.1 Modeling framework

The input to our prototype modeling system is a set of sketched skeletons, including points, polylines and polygons. Their geometry can be modified interactively. Our system outputs a polygonal mesh generated by the marching cubes algorithm implemented on the GPU. High-quality images of the created surfaces are rendered interactively using a ray-casting algorithm on the GPU. The framework for our modeling system is illustrated in Fig. 3, which we describe further in the text.

Step 1. The user draws a 2D shape (or points, polylines) as the skeleton on a projection plane. Here, the sketched silhouette polygon is used for surface modeling in Step 2 directly. This step is performed on the CPU.

Step 2. The system generates a rotund generic convolution surface model using analytical field computation solutions (Sect. 3). Polynomial weighted distributions are employed for polyline skeletons [12]. For finite support kernels, the system calculates the valid part of the skeleton. A skeleton clipping algorithm is presented for finite support kernels (Sect. 3.3). The valid skeleton calculation and the field value computation are both performed on the GPU. A GPU version of marching cubes algorithm [9, 21] is provided to visualize the modeling surface.

Step 3. The user performs further editing operations on existing skeletons, such as editing the skeleton geometry, the weighting of parameters, and the kernel functions.

Step 4. In order to design complex shapes, the user rotates the partially designed shape to obtain a new projection plane, and repeats steps 1–3 to design another surface component, which can be merged smoothly with existing components.

Step 5. After the user finishes the design, an accurate visualizing result can be rendered interactively with a ray-casting algorithm for convolution surfaces (Sect. 4.2). This step is also performed on the GPU.


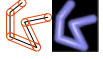

In this framework, step 1, step 3, and step 4 are common steps for a sketch-based modeling system using implicit surfaces. Our paper focuses on step 2 and step 5.

4.2 Ray-casting convolution surfaces on the GPU

Generally, there are mainly two methods to render implicit surfaces: ray-casting and polygonization. Polygonization is an indirect method which tessellates implicit surfaces into polygons within a given tolerance. On the contrary, ray-casting is an important direct rendering method. Compared to polygonization, it provides an accurate rendering method for visualizing implicit surfaces although it is slow. In our sketch-based modeling system, both interactive polygonization and ray-casting are supported. The sketched models with high resolution can be interactively generated with marching cubes on the GPU in our system, so the updated shape can be rendered interactively with new sketches.

As the intersection between a ray and the convolution surface is difficult to obtain analytically, iterative algorithms, such as the interval algorithm, Bezier clipping, the secant method, and the bisection method, are frequently adopted. Here, we use the bisection method. To provide interactive feedback, we use GPU and the bounding volumes of the skeletons to accelerate the rendering process. Our rendering can be made even faster if we employ advanced ray-casting techniques [8, 16, 19].

Table 1 Bounding volumes of primitives

Primitive	Bounding volume
Point	1 sphere: 
Polyline ($N \geq 2$)	$(N - 1)$ cylinders \cup N spheres: 
Polygon ($N \geq 3$)	N cylinders \cup N spheres \cup 1 prism: 

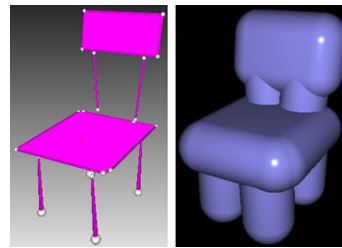


Fig. 4 Sketched skeletons (*left*) and bounding volume (*right*) of a chair

4.2.1 Bounding volume construction

The intersection point can be efficiently calculated if we can obtain the interval containing only the first intersection point. Similar to [25], we employ the bounding volume of the skeleton to isolate the domain which contains the ray/surface intersections. This will accelerate the intersection calculation significantly. The construction of bounding volumes for different primitives are illustrated in Table 1. Figure 4 shows a group of sketched skeletons and their corresponding bounding volumes.

4.2.2 Offset distance for computing the bounding volume

For a finite support kernel function, its corresponding offset distance d used for computing the bounding volume is just its effective radius. For the Cauchy kernel, as an unbounded plane has the following field function [22]:

$$\begin{aligned}
 F_{\text{plane}}(p) &= \int_0^{2\pi} \int_d^{+\infty} \frac{1}{(1 + s^2 r^2)^2} r \, dr \, d\theta \\
 &= \frac{\pi}{s^2(1 + s^2 d^2)},
 \end{aligned}
 \tag{17}$$

we calculate d so that it satisfies the following equation:

$$\frac{\pi}{s^2(1 + s^2 d^2)} = \frac{T}{N},
 \tag{18}$$

where T is the threshold of the resulting implicit surface and N is the maximum number of skeletons within a clipping

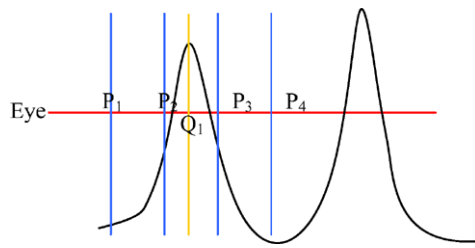


Fig. 5 Adaptive bisection marching

sphere. It is easy to follow that:

$$d = \frac{1}{s} \sqrt{\frac{N\pi}{Ts^2} - 1}. \quad (19)$$

4.2.3 Finding the intersection point

In general, it is hard to determine the suitable step-size for searching an intersection point. In our case, as the intersection point is isolated within the bounding volume of the skeleton, we set the initial step-size for marching forward as half the bounding radius. In this way, it is not easy to miss the first intersection even if the step-size is longer than needed. That is, if the first intersection is lost, we can detect it with the negative derivative D_f along the ray, and then step backwards with half the step-size and so forth. As illustrated in Fig. 5, points P_1 and P_2 are outside of the convolution surface and their derivatives along the ray are positive, so we step forwards with the current step-size to point P_3 . P_3 is outside of the surface, and the derivative at P_3 is negative, so it is necessary to step backward with half the step-size to Q_1 which is within the surface. Then we can perform the bisection/secant/regula falsi algorithm between point P_2 and Q_1 until a point is found which is close enough to the convolution surface within the prescribed tolerance.

Our ray-casting is performed on GPU using a thread for each pixel. The vertices of the sketched planar polygons are stored as textures on GPU (Fig. 6).

5 Results

Our prototype system is tested on a PC with a 2.67 GHz Intel Core 2 Quad 9400 CPU (only 1 core is used) with 3 GB main memory, and an NVIDIA GeForce GTX 260 GPU (with 27 stream multiprocessors) with 896 MB graphics memory. For such a sketched modeling system, surface creation and visualization are the bottlenecks as they involve massive field computation. In our approach, after transferring the skeletal data into the graphics card, the evaluation of the field is performed on the GPU, which allows us to create and visualize convolution surfaces interactively. The results (Fig. 3, Fig. 6, Fig. 7) in this paper are all interactively sketched with

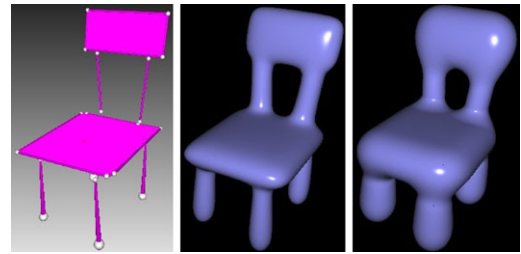


Fig. 6 Skeletons of a chair (left), ray-casting images with Cauchy kernel (middle) and quartic polynomial kernel (right)

our system. The resolution for marching cubes algorithm is $128 \times 128 \times 128$, and the screen resolution for ray-casting convolution surfaces is 512×512 .

A user interface similar to ShapeShop [23] is developed in our system. The user can add, delete, and modify skeletons. With the analytical solutions and the adoption of GPU, the reextraction of the updated iso-surface can be performed interactively without any special data structure. In addition, various skeletons are supported in our system, such as points, lines, and planar polygon skeletons. Our system is convenient for modeling complex objects with arbitrary topology. Multiple components can be fused together smoothly and trivially. Figure 7 shows some sketched models with their skeletons, rendered with marching cubes and ray-casting.

To test the efficiency of our novel analytical solution for planar polygon skeletons, we have compared three different strategies to calculate the potential field values. (1) DIT: Convolution with *double integral* based on the *triangulated* polygon; (2) CIT: Convolution with *curve integral* based on the *triangulated* polygon; (3) CI: Convolution with *curve integral* based on the polygon (ours). The statistical results for two examples shown in Fig. 8 are listed in Table 2. The time listed in Table 2 is the total time to calculate the potential field values for all the vertices of cubes (with the resolution $128 \times 128 \times 128$). We label the calculation time in the form of “ $\alpha_ \beta_ \gamma$ ”, where α represents the kernel type (quartic polynomial kernel or Cauchy kernel), β represents the processor (CPU or GPU), and γ represents the integration type (DIT, CIT or CI). The results show that our method has the best performance both on CPU and GPU. For the triangulation strategies, we have not counted in the consumed time for triangulation and the additional time and memory to transfer the triangles to GPU.

6 Conclusions and discussions

We have derived analytical convolution solutions for the planar polygon skeletons which are commonly used to model 3D objects. Our solution proves to be more efficient than the brute-force based triangulation method. Furthermore, we

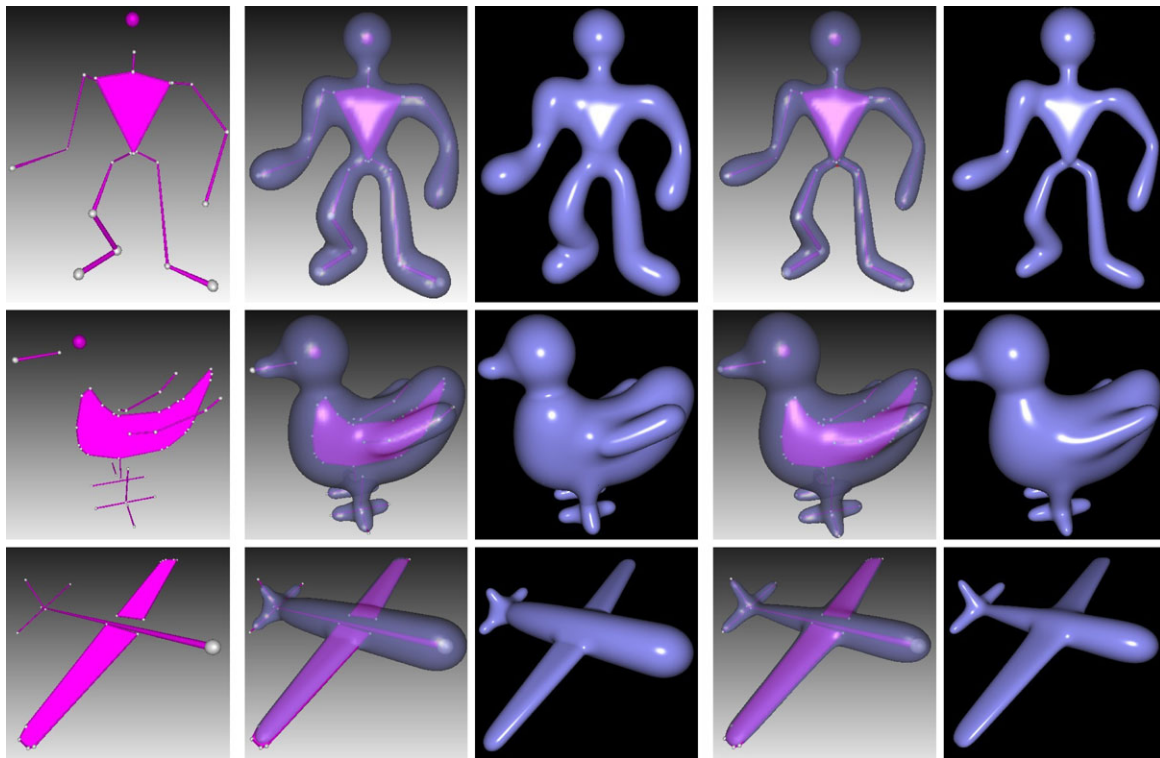


Fig. 7 The *leftmost column* shows the sketched skeletons. The *second* and the *third columns* show results rendered with marching cubes and ray-casting respectively using the quartic polynomial kernel. The

last two columns show results rendered with marching cubes and ray-casting respectively using the Cauchy kernel

Table 2 Comparison of calculation time (s)

	Ex1	Ex2
Number of Vertices	69	223
Number of Triangles	67	221
Quartic_CPU_DIT	5.902	22.292
Quartic_CPU_CIT	13.670	50.769
Quartic_CPU_CI	2.772	7.779
Quartic_GPU_DIT	0.064	0.186
Quartic_GPU_CIT	0.115	0.341
Quartic_GPU_CI	0.037	0.110
Cauchy_CPU_DIT	96.932	327.216
Cauchy_CPU_CIT	268.491	895.512
Cauchy_CPU_CI	84.015	286.960
Cauchy_GPU_DIT	0.132	0.431
Cauchy_GPU_CIT	0.241	0.795
Cauchy_GPU_CI	0.090	0.291

have presented a sketch-based prototype modeling system using closed-form solutions for convolution surfaces on the GPU. Since skeletons are good abstractions for 3D objects and convolution surface is a skeleton driven modeling tool,

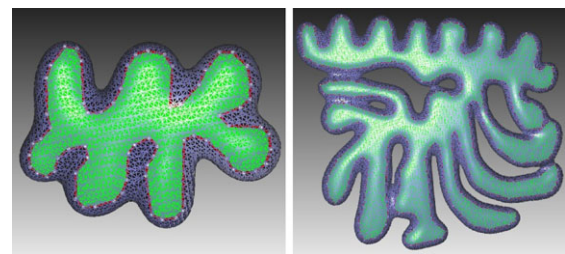


Fig. 8 The figure on the *left* and the figure on the *right* show Ex1 and Ex2, respectively, for comparing calculation time

our approach can create many aesthetically pleasing models with complex topology easily.

Compared to existing sketched-based modeling systems using convolution surfaces, our approach is easy to create flat surfaces (as shown in Fig. 3, Fig. 6, and Fig. 7), which are difficult to create using *ConvMo* [27]. On the other hand, this can be considered as a limitation of our approach because the result is different from the inflation operation in Teddy [11]. A set of functions provided by our prototype system is not as complete as in other sketch-based modeling systems. Currently, our system can ray-cast convolution surfaces with about twenty skeletons (each of which has about twenty vertices) interactively. For more complex skeletons,

we plan to employ advanced ray-casting acceleration solutions [8, 16, 19].

Acknowledgements Xiaogang Jin was supported by the National Key Basic Research Foundation of China (Grant No. 2009CB320801), the NSFC-MSRA Joint Funding (Grant No. 60970159), the National Natural Science Foundation of China (Grant No. 60933007), and the Zhejiang Provincial Natural Science Foundation of China (Grant No. Z1110154). Shengjun Liu was supported by the National Natural Science Foundation of China (Grant No. 61173119).

References

- Alexe, A., Gaildrat, V., Barthe, L.: Interactive modelling from sketches using spherical implicit functions. In: Proceedings of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, AFRIGRAPH '04, pp. 25–34. ACM Press, New York (2004)
- Alexe, A., Barthe, L., Cani, M., Gaildrat, V.: Shape modeling by sketching using convolution surfaces. In: Pacific Graphics, Short paper (2005)
- Angelidis, A., Cani, M.P.: Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In: Proceedings of the 7th ACM symposium on Solid Modeling and Applications, SMA'02, pp. 45–52. ACM Press, New York (2002)
- Bernhardt, A., Pihuit, A., Cani, M.P., Barthe, L.: Matisse: painting 2d regions for modeling free-form shapes. In: EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling, SBIM'08, pp. 57–64. Eurographics Association, Annecy (2008)
- Bloomenthal, J., Shoemake, K.: Convolution surfaces. *Comput. Graph.* **25**(4), 251–256 (1991)
- David, F.: Procedural Elements for Computer Graphics, 2nd edn. McGraw-Hill, New York (1997)
- Eyiyurekli, M., Grimm, C., Breen, D.: Editing level-set models with sketched curves. In: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling, SBIM'09, p. 45–52. Eurographics Association, New Orleans (2009)
- Gourmel, O., Pajot, A., Paulin, M., Barthe, L., Poulin, P.: Fitted bvh for fast raytracing of metaballs. *Comput. Graph. Forum* **29**(2), 281–288 (2010)
- <http://developer.nvidia.com/cuda-toolkit-32-downloads> (2011)
- Hubert, E.: Convolution surfaces based on polygons for infinite and compact support kernels. *Graph. Models* **74**(1) (2012). doi:10.1016/j.gmod.2011.07.001
- Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: a sketching interface for 3d freeform design. In: SIGGRAPH, pp. 409–416. ACM Press, New York (1999)
- Jin, X., Tai, C.: Analytical methods for polynomial weighted convolution surfaces with various kernels. *Comput. Graph.* **26**(3), 437–447 (2002)
- Jin, X., Tai, C.: Convolution surfaces for arcs and quadratic curves with a varying kernel. *Vis. Comput.* **18**(8), 530–546 (2002)
- Jin, X., Tai, C., Feng, J., Peng, Q.: An analytical convolution surface model for line skeletons with polynomial weighted distributions. *J. Graph. Tools* **6**(3), 1–12 (2001)
- Jin, X., Tai, C., Zhang, H.: Implicit modeling from polygon soup using convolution. *Vis. Comput.* **25**(3), 279–288 (2009)
- Kanamori, Y., Szego, Z., Nishita, T.: GPU-based fast ray casting for a large number of metaballs. *Comput. Graph. Forum* **27**(3), 351–360 (2008)
- Karpenko, O., Hughes, J.: Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph.* **25**(3), 589–598 (2006)
- Karpenko, O., Hughes, J., Raskar, R.: Free-from sketching with variational implicit surfaces. *Comput. Graph. Forum* **21**(3), 585–594 (2002)
- Knoll, A., Hijazi, Y., Kensler, A., Scjptt, M.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Comput. Graph. Forum* **28**(1), 26–40 (2007)
- Kravtsov, D., Fryazinov, O., Adzhiev, V., Pasko, A., Comninos, P.: Embedded implicit stand-ins for animated meshes: a case of hybrid modelling. *Comput. Graph. Forum* **29**(1), 128–140 (2010)
- Lorensen, W., Cline, H.: Marching cubes: a high resolution 3d surface construction algorithm. *Comput. Graph.* **21**(4), 163–169 (1987)
- McCormack, J., Sherstyuk, A.: Creating and rendering convolution surfaces. *Comput. Graph. Forum* **17**(2), 113–120 (1998)
- Schmidt, R., Wyvill, B., Sousa, M., Jorge, J.: Shapeshop: sketch-based solid modeling with blobtrees. In: EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling, SBIM'05, pp. 53–62. Eurographics Association, Dublin (2005)
- Schmidt, R., Wyvill, B., Sousa, M., Jorge, J.: Sketch-based modeling with the blobtree. In: SIGGRAPH, Technical Sketch. ACM Press, New York (2005)
- Sherstyuk, A.: Fast ray tracing of implicit surfaces. *Comput. Graph. Forum* **18**(2), 139–147 (1999)
- Sherstyuk, A.: Kernel functions in convolution surfaces: a comparative analysis. *Vis. Comput.* **15**(4), 171–182 (1999)
- Tai, C., Zhang, H., Fong, C.: Prototype modeling from sketched silhouettes based on convolution surfaces. *Comput. Graph. Forum* **23**(4), 71–83 (2004)
- Wilfred, K.: Advanced Calculus, 5th edn. Addison-Wesley Longman, Boston (2002)
- Wyvill, B., Overveld, K.: Tiling techniques for implicit skeletal models. In: SIGGRAPH courses. ACM Press, New York (1996)
- Wyvill, B., Guy, A., Galin, E.: Extending the csg tree: warping, blending and boolean operations in an implicit surface modeling system. *Comput. Graph. Forum* **18**(2), 149–158 (1999)
- Wyvill, B., Foster, K., Jepp, P., Schmidt, R., Sousa, M., Jorge, J.: Sketch based construction and rendering of implicit models. In: EUROGRAPHICS, Workshop on Computational Aesthetics in Graphics, Visualization and Image, pp. 67–74. Eurographics Association, Girona (2005)



Xiaoqiang Zhu is a Ph.D. candidate of the State Key Lab of CAD&CG, Zhejiang University. He received his B.Sc. degree in mathematics in 2006 and M.Sc. degree in computer science in 2009 from Hefei University of Technology. His research interests include implicit surface modeling and general purpose GPU computing.



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.



Hanli Zhao is a faculty member of Wenzhou University, Wenzhou, China. He received his B.Sc. degree in software engineering from Sichuan University in 2004 and his Ph.D. degree in computer science from Zhejiang University in 2009. His research interests include non-photorealistic rendering and general purpose GPU computing.



Shengjun Liu is currently a lecturer at the school of mathematical sciences and computing technology, Central South University, China. He received his B.Sc. degree in Computational Mathematics and Applied Software in 2000 and M.Sc. degree in Applied Mathematics in 2002 from Central South University, Ph.D. degree in Computer Science and Technology in 2007 from Zhejiang University. His research interests include geometric modeling in CAD/CAM, reverse engineering, and computer graphics.