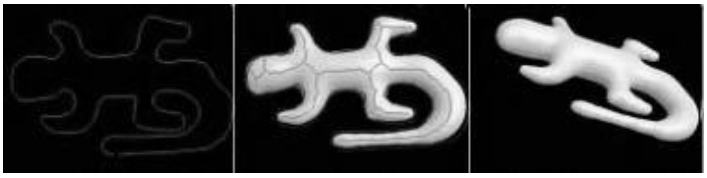
Interactive Modeling of Implicit Shapes From Sketches

Ileana Anca ALEXE, Véronique GAILDRAT

University of Toulouse III, Department of Computer Science,
Computer Graphics and Virtual Reality Group
Route de Narbonne 118, 31062 Toulouse Cedex 4, France
Email: alexe@irit.fr, gaildrat@irit.fr



Abstract

We present here a new system that interactively creates freeform smooth shapes from user strokes. The shapes have a natural rounded looking appearance. Sketching is a very intuitive modeling interface for shape creation. Smoothness is guaranteed by using implicit convolutions surfaces representation. We propose an efficient algorithm to compute the implicit convolution surface that best fits the stroke, and we analyze several implicit convolution kernels, in order to chose the appropriate one for our solid representation.

1 Introduction

The typical role of CAD systems is to provide user with assistance to efficient and rapid shape creation. However this is not an easy task especially with free form shapes.
CAD users still begin their work with paper and pencil. During the conception step they sketch the shape that they intent to design. Then they use CAD systems to create the complex shape by composing standard simple primitives. This is obviously a very tedious task. Another problem is strictly related to classic interfaces, which are hard to use and too complex to be efficient. Menu-based interfaces (WIMP) add an important amount of cognitive overhead to the user's task [10][13]. This means that user spends too much time to search through commands and menus, and generally he must have goods skills with the modeling software.
Zeleznik introduced the concept of sketching for intuitive scene modeling [16]. Igarashi [7] developed this concept by implementing the Teddy modeler, in order to experiment freeform shape design with strokes. With Teddy, the surface is reconstructed by elevating the medial axis of the closed stroke polygon. However the generated mesh is sometimes unsuitable and it has bulges and creases. Implicit surfaces can solve this problem, since they are smooth and continuous. Also, manipulation of object parts is not possible with Teddy. For instance, if we want a character to raise its arm, we have to entirely redraw the arm, instead of just manipulating the arm's skeleton by dragging its points. With implicit shapes, manipulating the skeleton is very easy, and the changes in the resulting surface can be automatically recomputed. Recently Karpenko [17] proposes sketching with implicit variational surfaces. But in the presented system, user can only draw implicit ellipses, and the thickness of the produced surface is constant. This limits the shapes to be produced. Our algorithm automatically computes local surface thickness, achieving a natural appearance for the shape.
We were also inspired by previous studies on reconstruction of point clouds using implicit blobs [2] [8] [14]. Although these methods could be adapted to approximate strokes, they are very slow, since their original purpose is a very precise

approximation. As a consequence they cannot reach interactive rates. Also, the blob is a very expensive implicit function, since it contains an exponential factor. We have tested several spherical implicit functions to determine which one suits best for rapid computation and satisfying stroke approximation. Another disadvantage of previous approaches is that they focus on progressive optimization of surface parameters, and initial values are randomly chosen. Based on simple and intuitive observations, we propose a technique that directly computes very satisfying initial values for the surface parameters. The surface generated using these initial values is very close to the stroke's shape, and only a few adjustments are necessary to produce a good fit, so computing times are interactive.

In section 2 we give a general overview of our system, while section 3 presents some theoretical concepts about implicit surfaces and most particularly implicit convolution surfaces. In section 4 the surface computation algorithm is discussed, and section 5 describes in detail the formulas used to compute the implicit function parameters. Section 6 discusses the choice of a convolution kernel, and section 7 draws conclusions and perspectives.

2 General overview

Our goal was to conceive a system easy to use, allowing intuitive rapid creation of freeform smooth shapes. A good application of this system would be fast shape prototyping. After being designed, the produced shape can be exported to other modeling software that can add details, for instance.

The user draws strokes, which are interpreted to interactively produce implicit shapes.

The implicit surface is composed of implicit convolution spheres. We also analyzed and compared the 7 convolution kernels (as reported in [12]), in order to determine which kernel would be the most appropriate for surface representation.

Based on our first results, which are rather encouraging, we are now implementing other operations, such as directly manipulating the skeleton points, in order to rearrange the object's parts, extruding, selecting, and cutting. Extruding parts of the objects should be done using strokes, but all the other operations can be directly done on the skeleton. Then the implicit surface can be recalculated from its skeleton, still remaining smooth and continuous. In fact we intent to make use of the *skeleton - shape* duality. Skeleton is computed from sketch, and shape can be computed from skeleton. When we need to create a shape we use the sketch, but when we need to operate on it we use its skeleton points. Both of these modeling approaches are intuitive. This procedure allows us to get the best of both of them, as each approach is used where it is most suitable. Also the transformation between those two representations can be easily done.

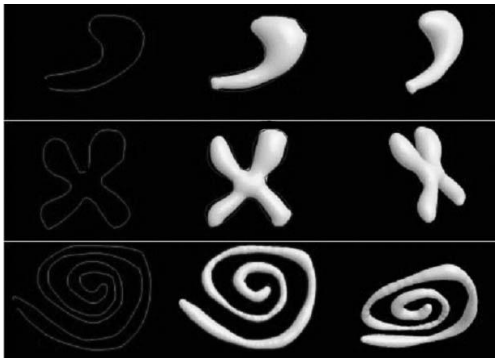


Figure 1: A few sketched implicit objects.

In order to render the shape we use a marching cube style algorithm [5]. It operates by dividing this space into equally sized cells. The potential value in cell corners is used to form polygons. The polygonization's speed depends on two factors: complexity of the function to be evaluated and cubic size cell. Here the user can fix the cubic size cell, which allows controlling the mesh's resolution. We will discuss the other factor in Section 6.

3 Theoretical concepts

Implicit surfaces [18] are a powerful modeling tool due to their blending properties, which allow them to create smooth shapes, and easily be manipulated by the mean of their skeleton.

An implicit function is defined by the following formula:

$$F(x,y,z)-T=0, \qquad (1)$$

where F is called a scalar field function, or scalar potential. F associates a field value to every point in space. All the points in space in which $F(x,y,z)=T$ form the implicit surface. T is called the iso surface constant. Blinn was the first to use implicit surfaces in computer graphics [3]. He used an exponential function. As this function was too slow to evaluate, several approximations were proposed shortly after [9] [15]. A very important step in the implicit surfaces modeling history was the introduction of implicit distance surfaces, which are surfaces defined by their skeleton. The skeletons can be any elementary geometric primitives such as points, line segments, curves or polygons. However, blending two skeleton shapes together sometimes produces unwanted bulges and creases in the resulting surface. To solve this problem, Bloomenthal and Shoemake proposed convolution surfaces [5]. Convolution surfaces are defined by the formula:

$$f(p)=g(p)*h(p)\,, \qquad (2)$$

g being the skeleton's characteristic function and h being the *convolution kernel*. g characterizes the object's geometry and it is also called *geometry function*, while h characterizes its shape.
g can be expressed as

$$g(p)=\sum_{i=1}^{N} g_i(p) \qquad (3)$$

g_i being the characteristic function of the i^{th} skeleton element. The convolution product describes the implicit surface. Bloomenthal and Shoemake observed that the additive property of convolution integrals makes implicit convolution surfaces suitable for modeling complex surfaces from simple primitives, and, the most important thing, each primitive can be evaluated separately. Skeleton variation produces fluid variations in the surface, and no creases or bulges are present when joining skeletons together. In order to obtain the implicit surface we just need to sum the contribution of every primitive. This contribution is limited by the kernel width.

4 Overview of surface computation algorithm

We will first describe the general reconstruction process, and then the next section will add more details on how we compute the surface, and how we tested our method on various convolution kernels.
First the user draws a stroke, which must be not self-intersecting. Then the stroke is closed (if the user hasn't already closed it). The stroke is approximated by a polygon, which preserves the sharp angles, and has equally sized edges. The next step is to compute the skeleton of the polygon. We start by performing a Delaunay triangulation [1] on polygon points. Then we eliminate all triangles that have edges outside the polygon. Then we compute the chordal axis [11] by an algorithm similar to [7]. The algorithm just joins together the intern edges middle points. If there is an intern triangle, the skeleton branches join into its center, as shown in Figure 2 d). Triangles with two extern edges are the extremities of the shape. These triangles are split in two, and only the intern point is considered to be a skeleton point, as in Figure 2 d). Skeleton does not touch the polygon boundary.

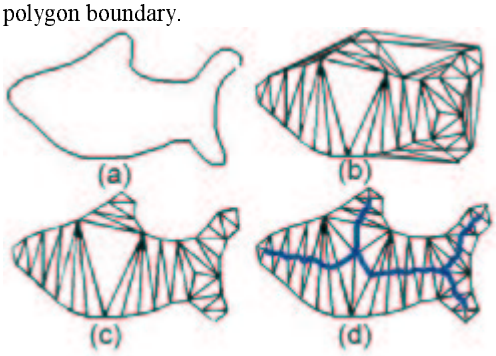


Figure 2: a) Original contour b) Delaunay triangulation c) Cutting extern triangles d) Computing the skeleton.

The next step in the reconstruction the implicit surface is to place an implicit sphere in every point of the skeleton. If we look back to the convolution definition (2) we just defined the geometry function g. g is the sum of the skeleton characteristic functions g_i, defined as follows (p_i are the skeleton points):

$g_i(p)=1$, if $p=p_i$ and (4)
$g_i(p)=0$ in all the other cases.

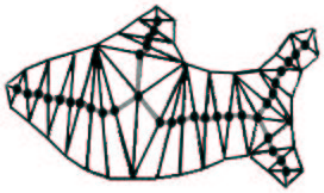


Figure 3: Spheres are placed in every point of the skeleton.

Now we need to determine the convolution kernel. The kernel must provide:

1. fast computation time;
2. the resulting surface must match the original contour (with as few adjustments as possible).

In order to respect these requirements we have tested the 7 convolution kernels (see [13]) with a set of random strokes.

Each of those kernels produces an implicit surface. For every sphere, there is at least one parameter to compute in order to adjust it to the stroke. This parameter is usually the radius. Depending on kernel, we can have a second parameter, which controls the blending.

The formulas used for computing these parameters will be discussed in the next section. This step produces an implicit surface very close to the initial sketch.

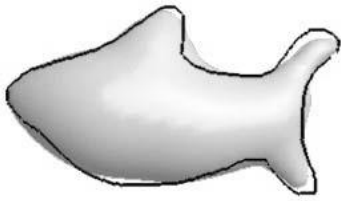


Figure 4: Originally computed implicit surface.

The next step performs an adjustment of the implicit sphere parameters. We progressively select the sphere whose surface is farthest from the original contour. We adjust the sphere to better fit the contour, and the process is reiterated until a minimum distance criterion is reached. This "greedy" technique evaluates a local criterion and has the advantage of being fast and easy to implement.

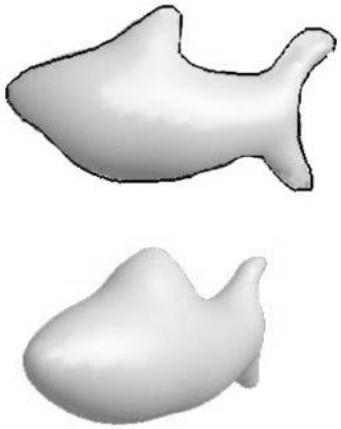


Figure 5: Adjusting the resulting surface to better fit the contour.

The surface is then polygonized for rendering [5] and a compression of the mesh is performed [6].

5 Computation of implicit surface parameters

Blinn has shown that any implicit primitive can be expressed as a function of two distinct parameters: *radius in isolation* R and *blobbiness* B. Radius in isolation is the characteristic distance between the primitive's skeleton and the implicit surface that it produces. Blobbiness is a parameter that controls blending between primitives. As the blobbiness grows, the surface becomes rounder.

These two parameters are very interesting because they are universal. We made some simple remarks that helped us to automatically compute satisfying values for those parameters. For instance, R represents the distance to the surface, so we can conclude that a good approximation for the radius in isolation would be the average of the point's distances to its' neighbors on the surface, as shown in Figure 6. Blobbiness controls blending, so it must depend on the distance to neighboring spheres. We decided to approximate it by the average distance to the centers of its neighboring spheres, as in Figure 6.

Figure 6: Computing the radius in isolation and the blobiness for the center point. The light gray points participate in radius in isolation computing. The dark gray points contribute to the blobbiness value.

When the skeleton function $g_i(p)$ is a point set, considering the equation (3), the convolution formula yields:

$$f(p)=\int_{R^3}\left(\sum_{i=1}^{n} g(x)\right)h(p-x)dx. \qquad (5)$$

By replacing g definition (4) we get:

$$f(p)=\sum_{i=1}^{n}\left(f(p-p_i)-T_i\right). \qquad (6)$$

where T_i is a constant. The contribution from each primitive is $h(p-p_i)-T$ or, if we put $p-p_i=r$, $h(r)-T$.

6 Choosing a convolution kernel

The 7 kernels are given by the following formulas:

- *Cauchy function* [12]:

$$f_i(r)=\frac{1}{\left(1+s_i^2 r^2\right)^2}, \qquad (7)$$

s being the stiffness parameter.

- *Blinn's blob* [3]:

$$f_i(r)=b_i\, exp\left(-a_i^2 r^2\right). \qquad (8)$$

- *Inverse function* [19]:

$$f_i(r)=\frac{1}{r}. \qquad (9)$$

- *Inverse squared function*:

$$f_i(r)=\frac{1}{r^2}. \qquad (10)$$

- *Meta ball function* [9]:

$$f_i(r)=\begin{cases} 1-3r^2 & 0<r<\frac{1}{3} \\ \frac{3}{2}(1-r)^2 & \frac{1}{3}<r<1 \\ 0 & r>1 \end{cases}. \qquad (11)$$

- *Soft Objects* [15]:

$$f_i(r)=\begin{cases} 1-\frac{4}{9}r^6+\frac{17}{9}r^4-\frac{22}{9}r^2 & r<1 \\ 0 & r>1 \end{cases}. \qquad (12)$$

- *W-shaped polynomial* [20] [22]:

$$f_i(r)=\begin{cases} \left(1-r^2\right)^2 & r\le 1 \\ 0 & r>1 \end{cases}. \qquad (13)$$

In this section we analyze each of the above kernels, by considering the contribution of a sphere using the respective kernel. Then we express this contribution as a function of radius in isolation and blobbiness (if possible), as described in section 5. Some functions only have one parameter, so they will only depend on radius in isolation.
Tests are run with a set of random strokes. Then the surface is computed, polygonized and rendered. This allows us to compare the performances of the kernels, in terms of speed and approximation accuracy.
The condition that we use in every case is that the contribution of the potential field f_i is 0 in a point at distance r from the sphere's center, if the point lies on the surface of the sphere, which means:

$$f_i(r)-T_i=0 \text{ when } r=R_i; \qquad (14)$$

where R_i is the radius in isolation of the sphere. Then we solve (14) for T_i and other kernel parameters, if they exist.
For efficiency reasons, all the following functions are windowed, that is they are considered to be 0 outside the radius in isolation. This will increase the computation speed.

- *Cauchy function*

The condition (14) yields $\frac{1}{\left(1+s_i^2 R_i^2\right)^2}=T_i$ or

$s_i^2=\frac{1}{R_i^2}\left(\frac{1}{\sqrt{T_i}}-1\right)_i$, and by putting $\frac{1}{B_i}=\frac{1}{\sqrt{T_i}}-1$,

we get:

$$f_i(r)=\frac{\frac{1}{\left(1+\frac{r^2}{B_i R_i}\right)^2}-\frac{1}{\left(1+\frac{1}{B_i}\right)^2}}{1-\frac{1}{\left(1+\frac{1}{B_i}\right)^2}}. \quad (15)$$

- *Blinn's blob*

Blinn has already done the calculus for this function. He obtained: $f_i(r)=F_i\exp\left(-\frac{B_i}{R_i^2}r^2-B_i\right)$. As Blinn noticed, there is a T_i factor in every sphere function, so by putting $T_i=1$ we get a very convenient formula:

$$f_i(r)=\exp\left(-\frac{B_i}{R_i^2}r^2-B_i\right)-1. \quad (16)$$

- *Inverse function*

With condition (14) we get $T_i=\frac{1}{R_i}$, so

$$F_i(r)=\frac{1}{r}-\frac{1}{R_i}. \quad (17)$$

- *Inverse squared function*

The condition (14) produces $T_i=\frac{1}{R_i^2}$, and

$$F_i(r)=\frac{1}{r^2}-\frac{1}{R_i^2}. \quad (18)$$

- *Meta ball function*

Meta ball is usually an implicit sphere with unit radius, but a R_i radius sphere will be much more useful for us. There are 3 main situations:

1. $0<r<\frac{R_i}{3}$, which produces $T_i=1-3R^2$,
2. $\frac{R_i}{3}<r<R_i$, which gives $T_i=\frac{3}{2}(1-R_i)^2$,
3. r>1, $f_i(r)=0$.

We prefer the easy to read version:

$$F_i(r)=\begin{cases}1-3r^2-1+3R_i^2 & 0<r<\frac{R_i}{3}\\ \frac{3}{2}(1-r)^2-\frac{3}{2}(1-R_i)^2 & \frac{R_i}{3}<r<1\\ 0 & r>R_i\end{cases}. \quad (19)$$

- *Soft Objects*

The same process as above, and replacing the unit radius by R_i radius gives us:

$$F_i(r)=\begin{cases}\frac{4}{9}r^6+\frac{17}{9}r^4-\frac{22}{9}r^2-\frac{4}{9}R_i^6+\frac{17}{9}R_i^4-\frac{22}{9}R_i^2, & r\le R_i\\ 0 & r>R_i\end{cases}. \quad (20)$$

- *W-shaped polynomial*

Condition (14) and changing the radius to R_i give:

$$F_i(r)=\begin{cases}\left(1-r^2\right)^2-\left(1-R_i^2\right)^2 & r\le R_i\\ 0 & r>R_i\end{cases}. \quad (21)$$

In order to determine the most efficient kernel for reconstruction, we considered 50 different random strokes. The criteria were:

1. *Computation time* for the implicit surface: computation time for spheres position, radius and blobbines.
2. *Polygonization time*: computation time for the surface mesh. Since we need interactive rates, this is a crucial factor.
3. *Approximation error*: the distance from the computed surface to the original stroke. One simple way to compute this error would be to make the sum of the potential values in every point of the original polygon (in absolute value):

$$e=\sum_{i=1}^{n}\left|F(p_i)\right|. \quad (22)$$

If we look back at the 7 functions we see that all except the inverse function and the inverse square function have the same characteristic: they are bell shaped functions, with unit value in 0 point and slowly decreasing to 0 in the same way (see their plots in [21]). So comparing their potential values in the same point has a meaning.

However this method is not efficient for evaluating the inverse function and the square inverse function, as they decrease more rapidly than the others. They start from an "infinite" value near 0 point (where they are not defined) and fall to 0 on the same interval as the others, but the slope is quite larger. So using (22) as an error measure has a signification only for the 5 remaining functions.

Here are the average results obtained on a 1400MHz Intel Pentium 4 computer:

Table 1: Computation times.

Kernel	Time (s)
Cauchy	0,02605
Blob	0,02555
Inverse	0,026
Inverse squared	0,02455
Meta ball	0,0255
Soft Object	0,02455
W shape polynomial	0,026

Table 2: Polygonization times.

Kernel	Time (s)
Cauchy	2,6408
Blob	3,59705
Inverse	2,7746
Inverse squared	2,62075
Meta ball	2,8644
Soft Object	2,8427
W shape polynomial	2,4741

Table 3: Approximation errors.

Kernel	Error
Cauchy	1,73349475
Blob	3,30165095
Inverse	31,928574
Inverse squared	267,0739279
Meta ball	6,21988875
Soft Object	5,16660585
W shape polynomial	4,86360465

One can notice that the computation time (Table 1) does not vary much. This is normal, since in our case the number of spheres is about a few dozens, which is a small number, and it does not produce significant differences. It is not a crucial factor.

For polygonization (Table 2), we chose a small cell size, in order to make differences more obvious. We can notice that the fastest function is the inverse square function. This was quite predictable, since it only contains a dividing operation, and it avoids the expensive square root operation (necessary for the inverse function, for instance).

Concerning approximation error (Table 3), the values for inverse function and inverse squared function are not relevant, as we explained before, so we will only refer to the others. We can see smaller errors for the two functions for which blobbiness can be fixed (Blob and Cauchy functions). We deduce that having the function depending on the two parameters is very important for accuracy. This gives us more control on the generated surface. This observation is also valid for the two functions that we were not able to compare, and it can be confirmed by directly visualizing the shapes, although the differences between shapes obtained with different kernels are rather small.

Cauchy kernel function offers the best approximation, and it is among the functions that are fast to polygonize. So we think it would be a good choice for our representation.

For these reasons we based our implementation on this kernel.

7 Conclusion and perspectives

We have presented a technique that allows rapid creation of complex smooth shapes. The objects are represented by implicit convolution surfaces. The generated shapes are natural looking, continuous, and there are no bulges and creases on the surface. Figure 7 compares a sketch reconstructed with our algorithm with the same sketch reconstructed using the algorithm described in [7]. We can see that the second surface (c) is much smoother than the first one (b).

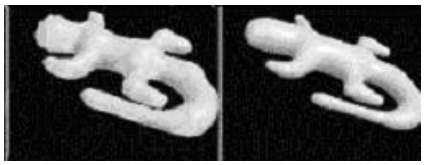


Figure 7: a) Stroke interpreted using the algorithm described in [7]. b) The same stroke interpreted with our system.

We also analyzed the 7 convolution kernels in order to choose the appropriate one for the representation.

Objects can be further manipulated easily, by directly operating on the skeleton points. Editing operations such as extruding, cutting, moving and rearranging object parts by the mean of their skeleton are now implemented.

The advantage of this system is to provide rapid prototyping, which is useful in the conception phase, and to avoid the tedious task of modeling with base primitives only. Perspectives include guiding the user's stroke (with a background image). We also think that stylized rendering and implementation on a PHANToM device would help achieving a better feeling of paper prototyping.

One disadvantage of present rendering method is the necessity to repolygonize the entire surface once it has been modified. This can affects the interactive times, so we are also working on using a local repolygonization algorithm.

References

[1] F Aurenhammer, "Voronoi diagrams - A survey of a fundamental geometric data structure", *ACM Computing Surveys*, 1991, 23:345-405.

[2] E.Bittar et al., "Automatic Reconstruction of Unstructured 3D Data: Combining Medial Axis and Implicit Surfaces", *Eurographics*, September 1995.

[3] J. F. Blinn "A Generalisation of Algebraic Surface Drawing", *ACM Trans Graphics*, Vol 1, No. 3 , July 1982, p 135-256.

[4] J.Bloomenthal, K.Shoemake, "Convolution Surfaces", *Computer Graphics*, Vol. 25, No.4, July 1991, p 251-256.

[5] J.Bloomenthal, "An Implicit Surface Polygonizer", *Graphics Gems IV* (P. Heckbert, ed.), Academic Press, New York, 1994.

[6] J. Cohen, et al. "Simplification Envelopes", *Computer Graphics Proceedings*, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 119-128.

[7] T.Igarashi et al. "Teddy: A Sketching Interface for 3D Freeform Design", *ACM SIGGRAPH 1999*.

[8] S.Muraki, "Volumetric shape description of range data using blobby model", *Computer Graphics*, 25(4): 227-235, July 1991.

[9] H. Nishimura et al., "Object Modelling by Distribution Function and a Method For Image Generation", *The Transaction for the Institute of Electronics and Communication for Engineers of Japan*, 1985, Vol J68-D, Part 4, p 718-725.

[10] D. A. Norman, "User Centred System Design", *Lawrence Erlbaum Associates, Inc.*, 1986.

[11] L. Prasad, "Morphological analysis of shapes", *CNLS Newsletter*, 139: 1-18, July 1997.

[12] A. Shersyuk, "Kernel functions in convolution surfaces: a comparative analysis", *The Visual Computer*, 15(4) 1999.

[13] S.A. Suchman, "Plans and Situated Actions: The Problem of Human-Machine Communication", *Cambridge Univ. Press. 1987.*

[14] N. Tsingos et al., "Semi-automatic reconstruction of implicit surfaces for medical applications", *Computer Graphics International*, June 1995.

[15] G. Wyvill et al., "Data Structure for Soft Objects", *The Visual Computer*, Vol.2, No. 4 August 1986a, p 227-234.

[16] R. Zeleznik, K. Herndon and J. Hughes "Sketch: An Interface for Sketching 3D Scenes", *Proceedings of SIGGRAPH'96,* 1996, p. 163-170.

[17] O. Karpenko, J. F. Hughes, R. Raskar, "Free-form Sketching With Varatioal Implicit Surfaces", *Eurographics 2002*, TR2002-27, June 2002.

[18] J. Bloomenthal "Implicit Surfaces", *Encyclopedia of Computer Science and Technology*, Marcel Dekker Inc., NY, 2000.

[19] B. Wyvill and K. van Overveld, "Tiling Techniques for Implicit Skeletal Models", *Implicit Surfaces for Geometric Modeling and Computer Graphics*, SIGGRAPH 1996, Course Notes, pp. C1.1-C1.26.

[20] POV-Ray , http://www.povray.org/.

[21] A.Shersyuk, "Convolution surfaces in Computer Graphics" *Ph.D. dissertation, School of Computer Sciences and Software Engineering Monosh University*, Australia 2000.

[22] Rayshade , http://www-grapics.stanford.edu/~cek/rayshade/rayshade.html.