Creating and Rendering Convolution Surfaces

Jon McCormack and Andrei Sherstyuk

Department of Computer Science
Monash University, Victoria, Australia

Abstract
Implicit surfaces obtained by convolution of multi-dimensional primitives with some potential function, are a generalisation of popular implicit surface models: blobs, metaballs and soft objects. These models differ in their choice of potential functions but agree upon the use of underlying modelling primitives, namely, points. In this paper a method is described for modelling and rendering implicit surfaces built upon an expanded set of skeletal primitives: points, line segments, polygons, arcs and planes. An algorithm for ray-tracing the surfaces formed through convolution of any combination of these primitives is also presented. The algorithm employs analytical methods only, which makes it computationally effective.

Keywords: *convolution surfaces, geometric modelling, implicit surfaces, ray-tracing.*

1. Introduction

Geometric modelling with superimposed field functions (also known as density distributions) was independently introduced by Blinn[1] and Nishimura et al.[11]. In general, fields have contributions from a number of sources, each of which defines a density distribution in three-dimensional space. The contribution from each source is summed, and a surface is formed at some threshold value. The modelling equation may be specified in an implicit form:

$$\sum_{i=1}^{N} F_i(x, y, z) - T = 0, \qquad (1)$$

where F_i are the source potentials and T is the iso-potential value. A set of points (x, y, z) satisfying equation (1) forms an iso-surface. Such iso-surfaces are commonly known in computer graphics as implicit surfaces. The constituent field functions F_i of equation (1) are usually defined to be monotonically decreasing, with a negligible contribution beyond a certain distance from the source. With appropriate field function selection, the resulting iso-surfaces provide smooth blending between sources as they are brought together. In animation, the structural and topological changes are continuous. These features, combined with the ability to construct complex shapes that are difficult for other modelling primitives to describe, have made implicit surfaces a popular tool for many modelling tasks, particularly where the shapes to be modelled are from the natural world[1, 3, 4], or exhibit a 'soft' structure[15].

The capabilities of any modelling system based on equation (1), will be dependent on the possible choices of modelling primitives, denoted as field functions F_i, and this choice is the essence of the model. As with many modelling techniques in computer graphics, the following dilemma exists: whether an object should be represented by a large number of simple primitives, or by a smaller number of complex ones. With respect to the iso-surface equation (1), many different approaches have been described. The earliest methods used only simple primitives, such as spherical or ellipsoidal bumps, either Gaussian[1] or polynomial[11, 15]. More complex primitives have included super-quadrics[8, 16], generalised implicit cylinders[5] and convolution surfaces[3].

These approaches have their advantages and disadvantages with respect to designing and rendering. For designing purposes, it is desirable to have a wide range of field functions F_i representing commonly used shapes. For rendering purposes, these functions should be easy to evaluate in order to efficently locate the iso-surfaces. Naturally, these constraints contradict each other.

Direct rendering of models (that is, without conversion to some intermediate form such as polygons), defined using simple point field primitives, has been described using a scan-line based approach utilising numerical methods[1], or via ray-tracing and ray-casting[8]. Simple point fields yield relatively simple implicit equations that can be solved during the rendering process, without the need for conversion into other geometric primitives, such as polygons. However, modelling with point field sources, makes the design of complex shapes cumbersome. In addition, the use of point sources during an interactive modelling session provides little indication of how the final iso-surface will appear when rendered. Attempts to circumvent this problem by visualising each source with a sphere or ellipsoid (that represents each source as though it were an isolated surface), can make the interactive modelling process a little more intuitive. Finally, datasets produced with point-based field functions have difficulty describing circular structures and can only approximate flat regions.

The use of more complex potential-source primitives can solve many of the short-comings of point sources. In most cases however, the iso-surface equations (1) become too complex to afford practical direct rendering times. Polygonisation methods are normally employed, which involves evaluation of field functions (1) over some $3D$-grid[2, 10]. This process is a piece-wise linear approximation to the true surface, and may result in under-sampling artefacts.

In a skeletal-based approach to modelling with implicit surfaces, elements such as lines, planar curves and polygons are employed as potential sources[4]. These primitives are structured to represent the underlying 'skeleton' of the object being modelled, the implicit surface created by combining the field contribution of each source using a blending function. Blending functions allow many of the common CSG set operations, as each implicit function may be considered a solid volume in the non-negative region.

Convolution surfaces, introduced by Bloomenthal and Shoemake[3], convolve polygonally-based skeletal primitives with a three-dimensional Gaussian filter kernel. This process overcomes some of the problems of related distance surfaces, which include bulges and curvature discontinuity where there is a field contribution from more than one non-convex primitive. Finding a direct analytical representation for even a single convolved polygon is difficult, so Bloomenthal and Shoemake take advantage of the planer nature of their convolution primitives and perform a series of 1 and 2 dimensional convolutions, which are superimposed to obtain the final convolution. The resulting iso-surface in then found using polygonisation techniques.

In this paper we describe a method of creating and rendering convolution surfaces using an extended set of potential source primitives. A new potential function is described and we derive a set of field functions for the following widely used primitives: points, line segments, polygons, arcs and planes. These functions may be used both for polygonising or direct rendering of the implicit surfaces formed by the combination of primitives. We also describe an algorithm for ray-tracing the surfaces formed and discuss some implementation issues.

2. Creating convolution surfaces

2.1. Definition

Let f be a tri-variate function $f : \Re^3 \rightarrow \Re$, representing the geometry of a modelling primitive P:

$$f(\mathbf{r}) = \left[\begin{array}{ll} 1 & \mathbf{r} \in P; \\ 0 & otherwise; \end{array} \right.$$

Let h be a potential function that describes the field generated by a single point of that primitive:

$$h : \Re^3 \rightarrow \Re$$

The total amount of field at point $\mathbf{r}$, generated by the whole primitive is

$$F(\mathbf{r}) = \int_{\Re^3} f(\mathbf{p}) h(\mathbf{r} - \mathbf{p})\, d\mathbf{p}, \qquad (2)$$

which is a convolution of two functions f and h. Although convolution is a commutive operation, for our purposes we say that the field function $F(\mathbf{r})$ is obtained by convolving a geometry function $f(\mathbf{r})$ with a potential function $h(\mathbf{r})$, also called a *convolution kernel*. Thus, equation (2) may be conveniently re-written as an integral of the potential function $h(\mathbf{r})$ over the volume of the primitive:

$$F(\mathbf{r}) = \int_{\mathbf{V}} h(\mathbf{r} - \mathbf{v})\, d\mathbf{v} \qquad (3)$$

Each type of primitive has its own unique geometry (point, line, triangle etc.), and, therefore, yields its own field function $F_i(\mathbf{r})$ that can be used in iso-surface equation (1).

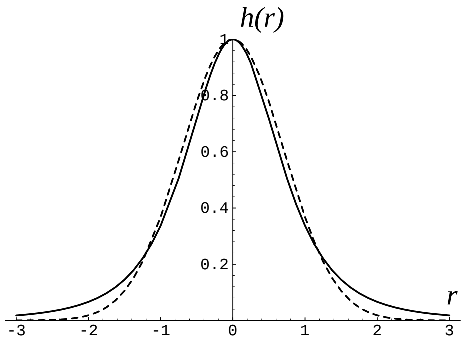


Figure 1: Exponential kernel (dotted line) and the kernel of equation (4) (solid line).

2.2. Convolution kernels

The ability to perform the actual integration depends on both the primitive and potential function. Most previous choices for potential functions h prove themselves to be illsuited for practical use with equation (2) when the geometry function f is more complex than just a point, particularly when an analytical solution to the integration is required. For example, kernels with a finite support, such as piece-wise polynomials, can not be convolved analytically with primitives whose dimensions are larger than the kernel's width. Kernels with infinite support (e.g. Gaussian, rational functions) do not yield closed-form expressions for all desired types of primitives[13].

Bloomenthal and Shoemake[3] used a Gaussian kernel, approximated by a cubic spline. For surface evaluation, they developed a technique of evaluating the field function (3) for planar primitives, using pre-computed mosaic of scan-converted polygons, filtered in 2 dimensions with a Gaussian filter.

A useable kernel should have the properties that it is continuous, monotonic, diminishes to a negligible contribution beyond a certain distance from the centre, and exhibits zero or near zero gradient at this distance.

We choose a new kernel that allows direct analytical solution of equation (3) for a wide family of primitives. In other words, with this kernel we are able to calculate the *exact* amount of field generated by a primitive at an arbitrary point without sacrificing accuracy nor speed. The kernel is:

$$h(r) = \frac{1}{(1+s^2r^2)^2}, \qquad (4)$$

where r is the distance from the point and coefficient s controls the width of the kernel, similar to the standard deviation parameter, a, in Blinn's blobby model[1]:

$$h(r) = e^{-a^2r^2}$$

A comparison of both kernels is shown in Figure 1, with $s = 0.85$ and $a = 1$. Notice, that kernel (4) is spherically symmetric and quadratic, which allows us to re-write the field equation (3) simply as:

$$F(\mathbf{r}) = \int_{\mathbf{V}} \frac{dv}{(1+s^2r^2(v))^2} \qquad (5)$$

2.3. Field functions

Using the kernel shown in (4), the field function for a number of primitives can be derived. These primitives are points, line segments, arcs, triangles and planes. Other primitives, e.g. circles and polygons, may be built by combining, respectively, arcs and triangles. We now describe the field function of each primitive in detail.

Points. A point is a 0-dimensional object, so its geometry is a simple delta-function at point $\mathbf{p}$. Integration of equation (5) yields the kernel function itself:

$$F_{point}(\mathbf{r}) = \frac{1}{(1+s^2|\mathbf{p}-\mathbf{r}|^2)^2} \qquad (6)$$

Figure 2 shows, clockwise from top-left: diagramatic representations of $\mathbf{p}$ and $\mathbf{r}$; an intensity plot of F_{point} over the $Z = 0$ plane; and a rendered version of an isolated point, $\mathbf{p}$ (in red) and it's corresponding iso-surface (transparent surface).

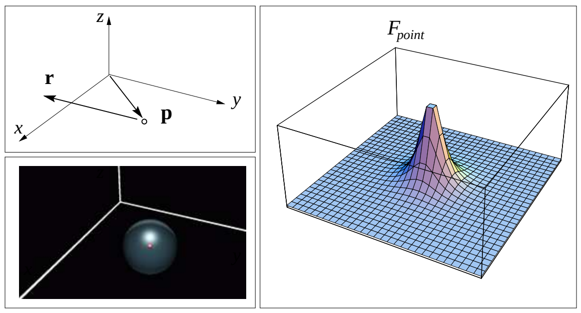


Figure 2: Point modelling primitive.

Line segments. A line segment of length l is defined as:

$$\mathbf{p}(k) = \mathbf{b} + k\mathbf{a}, \quad 0 \le k \le l,$$

where $\mathbf{b}$ is the base vector, $\mathbf{a}$ is the normalised axis. The squared distance between an arbitrary point $\mathbf{r}$ and a point on the line segment is

$$r^2(k) = d^2 + k^2 - 2k\mathbf{da},$$

where $\mathbf{d}$ is a vector from segment base to $\mathbf{r}$: $\mathbf{d} = \mathbf{r} - \mathbf{b}$. To obtain the field function, we substitute r^2 into the general formula (5) and integrate:

$$\begin{aligned}
&F_{line}(\mathbf{r}) = \\
&= \int_0^l \frac{dk}{(1+s^2r^2(k))^2} = \\
&= \frac{x}{2p^2(p^2+s^2x^2)} + \frac{l-x}{2p^2q^2} + \\
&+ \frac{1}{2sp^3}(\mathrm{atan}[\frac{sx}{p}] + \mathrm{atan}[\frac{s(l-x)}{p}]), \qquad (7)
\end{aligned}$$

where $x = \mathbf{da}$ and p and q are distance terms:

$$\begin{aligned}
p^2 &= 1 + s^2(d^2 - x^2), \\
q^2 &= 1 + s^2(d^2 + l^2 - 2lx)
\end{aligned}$$

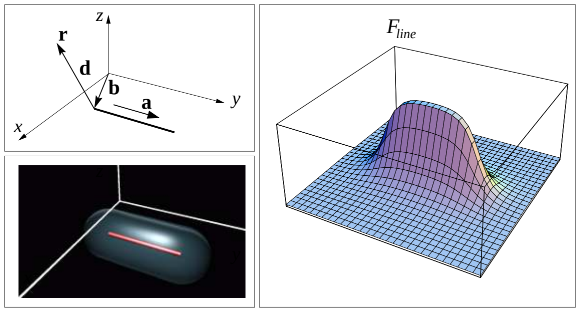


Figure 3: Line modelling primitive.

Arcs. An arc and its distance function r^2 are conveniently defined in the arc's local z-aligned coordinate system as:

$$\begin{aligned}
\mathbf{p}(t) &= (r\cos(t), r\sin(t), 0), \quad 0 \le t \le \theta, \\
r^2(t) &= (x - r\cos(t))^2 + (y - r\sin(t))^2 + z^2,
\end{aligned}$$

where r is the radius of the circle the arc lies on, θ is the arc angle (Figure 4, top-left). Integrating along the angle, we obtain the field function in local coordinates:

$$\begin{aligned}
&F_{arc}(x,y,z) = \\
&= \int_0^\theta \frac{dt}{(1+s^2r^2(t))^2} = \\
&= \frac{by}{xp^2(kx-b)} + \frac{k(x^2+y^2)\sin(\theta) - by}{xp^2(k(x\cos(\theta)+y\sin(\theta))-b)} + \\
&+ \frac{2b}{p^3}(\mathrm{atanh}[\frac{ky}{p}] + \mathrm{atanh}[\frac{(kx+b)\tan(\frac{\theta}{2}) - ky}{p}]), (8)
\end{aligned}$$

where $k = 2rs^2$ and distance terms d, b and p are:

$$\begin{aligned}
d^2 &= x^2 + y^2 + z^2, \\
b &= 1 + r^2s^2 + s^2d^2, \\
p^2 &= -r^4s^4 + 2r^2s^2(s^2(d^2 - 2z^2) - 1) - (1 + s^2d^2)^2.
\end{aligned}$$

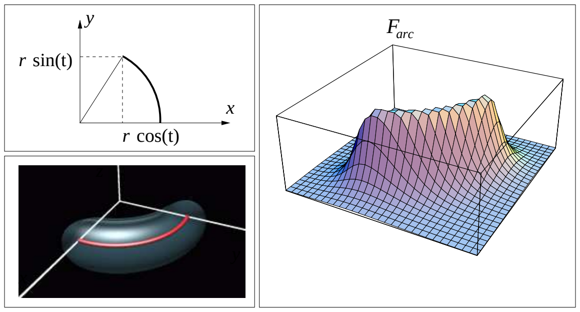


Figure 4: Arc modelling primitive.

Triangles. We split a general triangle into two right-angled triangles, do integration for both halves and sum the result. We define the following parameters (see Figure 5, top-left): a point $\mathbf{b}$, the projection onto the longest edge of the opposite vertex; vectors $\mathbf{u}$ and $\mathbf{v}$ that form the local surface coordinate system, with $\mathbf{b}$ as its origin and $\mathbf{u}$ aligned in the direction of the longest edge. We also define scalars a_1, a_2 that subdivide the longest edge at $\mathbf{b}$, and h, the distance from $\mathbf{b}$ to the apex of the triangle. Introducing the vector $\mathbf{d} = \mathbf{r} - \mathbf{b}$ and scalars $u = \mathbf{du}$ and $v = \mathbf{dv}$, the final field function of an arbitrary triangle is:

$$\begin{aligned}
&F_{triangle}(\mathbf{r}) = \\
&= \frac{1}{2qs}(\frac{n}{A}(\mathrm{atan}[\frac{s(vh + a_1(a_1+u))}{A}] + \mathrm{atan}[\frac{s(gh + a_1u)}{-A}]) + \\
&+ \frac{m}{B}(\mathrm{atan}[\frac{s(vh + a_2(a_2-u))}{-B}] + \mathrm{atan}[\frac{s(gh - a_2u)}{B}]) + \\
&+ \frac{v}{C}(\mathrm{atan}[\frac{s(a_1+u)}{C}] + \mathrm{atan}[\frac{s(a_2-u)}{C}])), \qquad (9)
\end{aligned}$$

where

$$\begin{aligned}
A^2 &= a_1^2w + h^2(q + s^2u^2) - 2hs^2a_1ug, \\
B^2 &= a_2^2w + h^2(q + s^2u^2) + 2hs^2a_2ug, \\
C^2 &= 1 + s^2(d^2 - u^2), \\
g &= v - h, \\
q &= 1 + s^2(d^2 - u^2 - v^2), \\
w &= c^2 - 2hs^2v + h^2s^2, \\
m &= a_2g + uh, \\
n &= uh - a_1g
\end{aligned}$$

Planes. Unbounded planes have the following field function:

$$F_{plane}(\mathbf{r}) = \frac{\pi}{s^2(1+s^2d^2)}, \qquad (10)$$

where d is distance between point $\mathbf{r}$ and the plane.

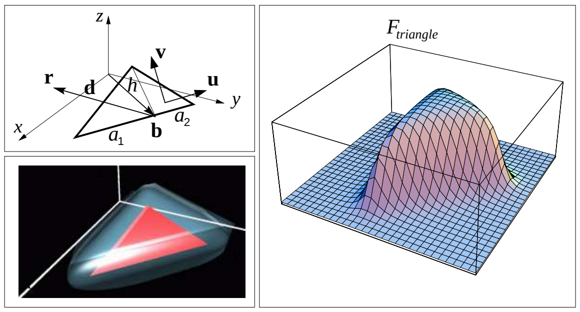


Figure 5: Triangle modelling primitive.

We have described field functions for five modelling primitives: points (6), lines (7), arcs (8), triangles (9) and planes (10). Notice that when the iso-surfaces generated by solitary skeletal primitives are rendered, they correspond to the following three-dimensional objects: spheres, cylinders, arc tubes, prisms and slabs. These objects may be conveniently used as visual modelling aids during the design stage, because they show the approximate structure of the composite implicit surface (1). The actual blending, of course, can only be obtained after the model is rendered. This is discussed in the following section.

3. Rendering convolution surfaces

Once the field functions (6, 7, 8, 9, 10) are defined, the main equation (1) may be used to locate the surface, and the object can be rendered. This may be done by approximating the surface with a mesh of polygons[2], which can then be rendered using standard polygon rendering techniques[6]. Alternatively, iso-surfaces may be rendered directly using ray-tracing, provided we can perform ray / iso-surface intersection tests efficiently. Here we describe a direct rendering algorithm using ray-tracing to locate the iso-surfaces generated from any combination of primitives described in the previous section. This algorithm is based on the original *'metaball'* ray-tracing algorithm developed by Nishimura et al.[12]. We briefly outline the ray-tracing algorithm, developed for *metaballs*, and show how it can be extended to render the set of new modelling primitives.

3.1. Ray-tracing algorithm for metaballs

In the *metaball* model, the constituents F_i of the iso-surface equation (1) are point potentials approximated by piece-wise quadratic polynomials:

$$F(\mathbf{r}) = \left[\begin{array}{ll} 1 - 3(r/R)^2 & 0 \le r \le R/3; \\ (3/2)(1 - r/R)^2 & R/3 < r \le R; \\ 0 & r > R; \end{array} \right. \qquad (11)$$

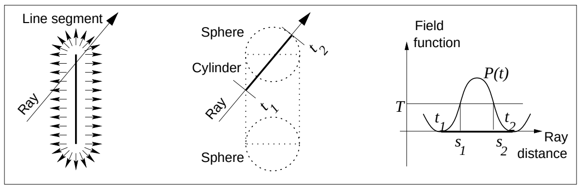


Figure 6: Computing intersection with a line segment.

where R is radius of influence of the metaball, r is the distance from its centre to point $\mathbf{r}$. The key idea of the algorithm is to keep the iso-surface equation (1) in the polynomial form, so it may be solved quickly during the ray / surface intersection test. To do that, the ray equation

$$\mathbf{r}(t) = \mathbf{a} + t\mathbf{b} \qquad (12)$$

($\mathbf{a}$ is the ray's origin and $\mathbf{b}$ is its normalised direction) is substituted into potential function (11). This yields at most three piece-wise quadratic polynomials per metaball that describe its field along the ray. When all metaballs have been processed in this manner, the whole extent of the ray inside their collective field becomes diced into a set of intervals with corresponding polynomials derived from equation (11). To find the intersections, the algorithm walks through these intervals, building and solving the iso-surface equation (1). Since all components are represented by quadratic polynomials, the collective iso-surface equation is also a quadratic, and all roots (hence intersections) can be found analytically.

3.2. Modified algorithm for new primitive types

The nature of the algorithm described above is that it may be applied to solve iso-surface equations generated by primitives of *any* type, provided the field function of each primitive can be represented in polynomial form, with ray distance t as an argument. For metaballs, this representation is straightforward, because the point source field function is already a polynomial. Field functions of more complex primitives, such as those discussed in the previous section, are not expressed in polynomial form, so additional processing is required in order to provide this representation. This can be achieved using polynomial interpolation. We evaluate the field function at specific points along the ray and apply either Lagrangian or Hermite interpolation to these evaluated points. The resulting polynomials are ready for use with the root-solving part of the algorithm.

To demonstrate, consider the simple example of finding all intersections between a ray and a surface modelled by a single line segment (Figure 6, left). First, we find the total extent of the ray inside the field, i.e., we intersect the ray with the bounding volume of the primitive. For line segments, the bounding volume is a union of a cylinder and two spheres at the endpoints (Figure 6, centre). The extent $[t_1, t_2]$ is the geometric location along the ray where the field is considered non-zero. After the extent is found, the actual geometry of the underlying primitive becomes irrelevant: we know that its field function F_{line} is given by (7) and it influences the ray on interval $[t_1, t_2]$. This interval is then split into five equal sub-intervals x_i, so that $x_1 = t_1$ and $x_5 = t_2$. We define $f_i = F_{line}(\mathbf{r}(x_i))$ to be the value of the field at a given position along the ray. Using the symmetry of F_{line} and the boundary conditions at x_1 and x_5, we obtain five samples of F_{line}, covering the whole extent of the field along the ray, within the bounding volume of the primitive:

$$\begin{aligned} f_1 &= 0, \\ f_2 &= F_{line}(\mathbf{r}(x_2)), \\ f_3 &= F_{line}(\mathbf{r}(x_3)), \\ f_4 &= f_2, \\ f_5 &= 0, \end{aligned} \tag{13}$$

where $\mathbf{r}(t)$ is the ray equation (12). Lagrangian interpolation of the constraints (13) produces a 4-degree polynomial $P(t)$ (Figure 6, right). Since there are no other primitives, the iso-surface equation (1) for this ray and this object becomes

$$P(t) - T = 0,$$

which has in this case two solutions within the interval $[t_1, t_2]$, s_1 and s_2, corresponding to intersection points $\mathbf{x}_1 = \mathbf{r}(s_1)$ and $\mathbf{x}_2 = \mathbf{r}(s_2)$. It's important to note that if we confine the interpolating polynomials to be of degree 4 or less, analytical root-finding methods can be used to obtain a solution to the ray / surface intersection test[14].

To conclude the discussion of the algorithm, we show the bounding volumes for all primitives (Figure 7).

3.3. Normal vector computation

The normal vector $\mathbf{n}$ at the ray/surface intersection point $\mathbf{x}$ is computed as a weighted sum of gradient vectors from all contributing primitives:

$$\mathbf{n} = -\sum_{i=1}^{N} w_i \nabla F_i(\mathbf{x}).$$

For the sake of brevity, we omit the exact expressions for $\nabla F_i(\mathbf{x})$: they may be obtained by taking partial derivatives of corresponding field functions. Scalar weights $w_i = F_i(\mathbf{x})$ are also used to interpolate between photometric characteristics of materials associated with modelling primitives. These usually include ambient and diffuse body colour, transparency, reflectivity and other information about the material of which the surface is composed.

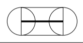
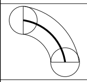
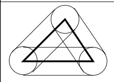
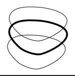
Primitive	**Bounding volume**	
Point		1 sphere
Line segment		1 cylinder + 2 spheres
Arc		1 piece of torus + 2 spheres
Triangle		3 cylinders + 3 spheres + 1 prism
Plane		1 infinite slab

Figure 7: Bounding volumes for modelling primitives.

4. Results

Several images have been computed to demonstrate the field functions described in Section 2.3. They were rendered using a ray-tracing program that computes the iso-surface intersections using the algorithm described in Section 3. All images were rendered on a 90 MHz Pentium, shooting at most 16 eye rays per pixel.

The starfish-like object in Figure 8 is modelled with seven arcs, each image shows the arcs convolved with kernels of various widths and heights. The top-left image shows the underlying skeletal representation (union of seven arcs). Rendering time, with respect to the top-left image (clockwise): 1, 4.5, 5.7, 2.9, 2.2, 1.05. Notice how the tentacle colour is blended along the surface.

Figure 9 shows some well known objects, that use arcs, line segments, triangles and planes as their skeletal elements. The unified polynomial representation of each modelling constituent allows primitives of different types to blend smoothly.

The images of coral trees in Figure 10 illustrate the modelling power of convolution surfaces built with field functions described in this paper. Both upper images are rendered using the same dataset, based upon Eric Haines's tree model[7]. Each tree consists of 31 branches, modelled with cylinders of various radii, and 512 randomly positioned spikes, also cylindrical. The only difference, from designer's point of view, is in material descriptions: the left coral is described as

```
surface Trunk: diffuse Pink
surface Spikes: diffuse White
```

while the right coral tree has its surface specified as

```
surface Trunk: diffuse Pink, blobbiness 100
surface Spikes: diffuse White, blobbiness 500
```

To switch between 'hard' and 'soft' representations of the surface, it suffices to change its blobbiness, which controls the width of the kernel (4). The default value is infinity, which makes the convolution kernel a delta function, yielding conventional, 'hard' surfaces, with no blending between primitives. In our implementation, objects with 'soft' or implicit surfaces have the same properties as their 'hard' siblings: they may be transformed, grouped into hierarchies, instantiated etc.

Unexpected but pleasant results were obtained while experimenting with triangular field function as a skeletal modelling primitive. Triangular meshes are normally employed to approximate surfaces. Adding a potential function produces a texturing effect: the surface becomes much more interesting and realistic, especially when used to model skin of not quite human characters (see Figure 11). The images in the top row are rendered with traditional polygons and look very much like CG objects or plastic dolls as they lack the complex detail that characterises faces. The images in the bottom row were rendered using convolution surfaces, the original polygonal database providing the skeleton. The results give Yoda a much more complex and 'mature' look.

5. Conclusions

We have presented a method of creating convolution surfaces based upon exact evaluation of the field function for a wide set of modelling primitives. These primitives, used as components of the main iso-surface equation (1), provide far greater flexibility in modelling implicit surfaces than traditional point sources. The techniques described augment the possibilities for skeletal-based modelling, by extending the list of possible primitives from which field functions may be directly evaluated.

We have shown closed-form expressions for all field functions and their gradients. Most of the calculations can be performed in world coordinates (with the exception of arcs). All of the above makes direct rendering of convolution surfaces feasible. To illustrate this, we presented a ray-tracing algorithm, that builds and solves the iso-surface equations using analytical methods only. Therefore, one can chose if the convolution surface should be polygonised prior to rendering or rendered directly. In other words, polygonising becomes an option, not a requirement.

We are currently investigating the possibility of extending the set of primitives to include other geometric data types. These include non-planar parametric curves and surfaces, which would prove useful for the modelling of many natural tentacle and branching structures. It should be pointed out that even with the current list of primitive types, it is possible to decompose a parametric curve into line segments and surfaces into polygons.

The skeletal-based approach allows the construction and animation of a wide variety of natural forms, many of which would be difficult to construct, and importantly, animate, using other geometric modelling techniques.

References

1. Blinn J.F., "A Generalization of Algebraic Surface Drawing", *ACM TOG*, Vol. 1, No. 3, July 1982, pp. 235-256.
2. Bloomenthal J., "Polygonization of Implicit Surfaces", *Computer Aided Geometric Design*, 5(1988), pp. 341-355.
3. Bloomenthal J. and Shoemake K., "Convolution Surfaces", *SIGGRAPH Proceedings*, Vol. 25, No. 4, July 1991, pp. 251-256.
4. Bloomenthal J., "Skeletal Design of Natural Forms", *Ph.D. thesis, The University of Calgary, Department of Computer Science*, January 1995.
5. Crespin B., Blanc C. and Schlick C., "Implicit Sweep Objects", *Computer Graphics Forum*, Vol.15, No.3, pp. C165-74, 1996.
6. Foley J. D., van Dam A., Feiner S. K. and Hughes J. F., *Computer Graphics, Principles and Practice*, second edition. Reading, Massachusetts: Addison-Wesley, 1990.
7. Haines E., "A Proposal for Standard Graphics Environments", *IEEE Computer Graphics and Applications*, Vol. 7, No. 11, November 1987, pp. 3-5. The SPD (Standard Procedural Databases) package is available at ftp.princeton.edu:/pub/Graphics/SPD
8. Kalra D. and Barr A., "Guaranteed Ray Intersection with Implicit Surfaces", *SIGGRAPH Proceedings*, Vol. 23, No. 3, July 1989, pp. 297-306.
9. Kincaid D. and Cheney W., *Numerical Analysis: Mathematics of Scientific Computing*, Brooks/Cole Publishing Company, 1991.

10. Ning P. and Bloomenthal J., "An Evaluation of Implicit Surface Tilers", *IEEE Computer Graphics and Applications*, November 1993.

11. Nishimura H., Ohno H., Kawata T., Shirakawa I. and Omura K., "LINKS-1: A Parallel Pipelined Multimicrocomputer System for Image Creation", in *Proceedings of the Tenth International Symposium on Computer Architecture, ACM SIGARCH Newsletter*, Vol. 11, No. 3, 1983, pp. 387-394.

12. Nishimura H., Hirai M., Kawai T., Kawata T., Shirakawa I. and Omura K., "Object Modelling by Distribution Function and a Method of Image Generation", *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, 1985, Vol. J68-D, Part 4, pp. 718-725, in Japanese (English translation by Takao Fujuwara available).

13. Sherstyuk A., "Ray tracing implicit surfaces: a generalized approach", *Technical Report No 96/290, Monash University, Department of Computer Science*, December 1996.

14. Schwarze J., "Cubic and Quartic Roots", *Graphics Gems* (editor, Andrew S. Glassner), Academic Press, Cambridge, MA, 1990, pp. 404-407.

15. Wyvill G., McPheeters C. and Wyvill B., "Data structure for soft objects", *The Visual Computer*, Vol. 2, pp. 227-234, 1986.

16. Wyvill B. and Wyvill G., "Field Functions for implicit surfaces", *The Visual Computer*, Vol. 5, pp. 75-82, 1989.

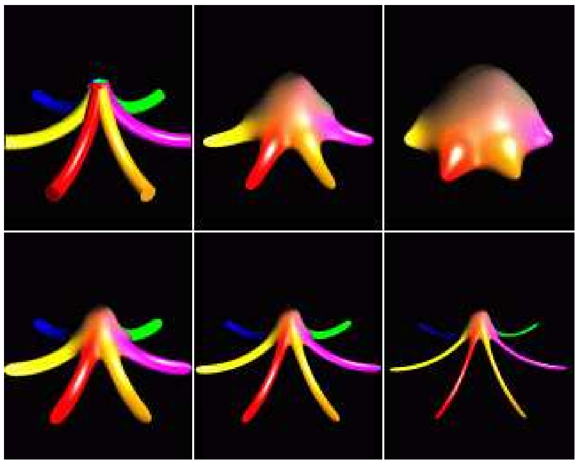


Figure 8: Family of convoluted starfish.

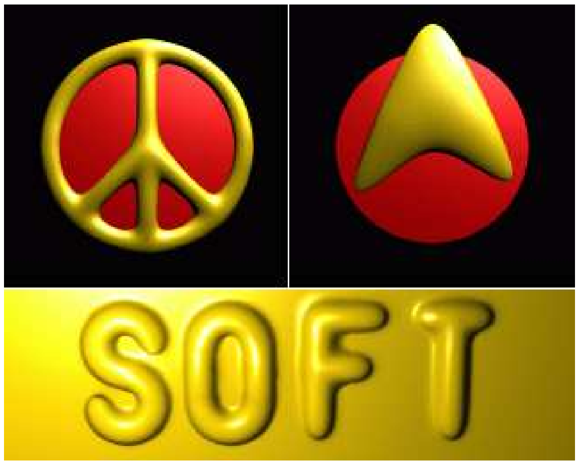


Figure 9: Blends: arcs, lines and planes (bottom); arcs and lines (left); triangle with triangle (right);

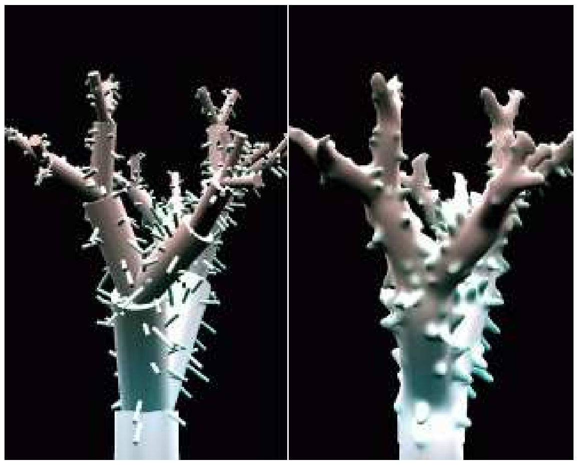


Figure 10: Coral tree made of line segments, 'explicit' and 'implcit' versions.

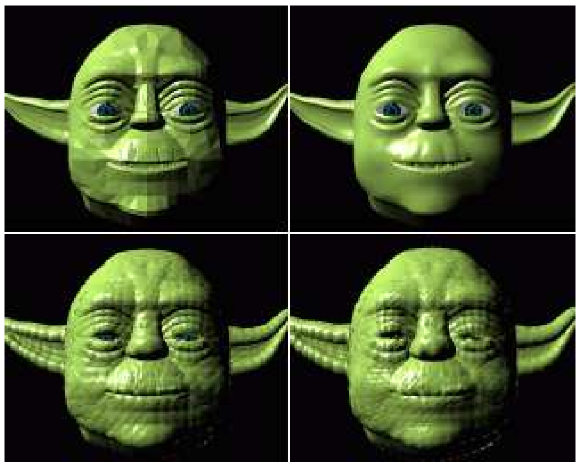


Figure 11: Aging Yoda: polygonal mesh (left top), polygonal mesh with smoothed normals (right top), polygonal mesh convolved with kernel function (bottom).