

Chapter 2

Formulation of Convolution Surfaces

2.1. INTRODUCTION

An *implicit surface* S is the isosurface in some scalar field F , defined in 3D space as

$$S = \{\mathbf{p} \in R^3 \mid F(\mathbf{p}) - T = 0\}, \quad (2.1)$$

where parameter T is the isopotential value. Although the scalar field function $F(\mathbf{p})$ may take any form, one of the most commonly used functions is a summation of point potentials:

$$F(\mathbf{p}) = \sum_{i=1}^N f_i(\mathbf{p}), \quad (2.2)$$

where $f_i(\mathbf{p})$ are usually Gaussian-like bumps, decreasing with the distance from their centers. More formally, any offset T may be contained within the definition of scalar field F . An example of a typical implicit surface and its constituent parts is shown in Figure 2.1. In this form, implicit surfaces were introduced into computer graphics by Blinn [7] as *blobby molecules*, and by Nishimura et al. [49] as *metaballs*. Later, Wyvill et. al. described their *soft objects* model [73].

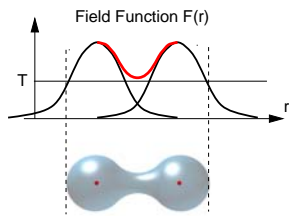


Figure 2.1: An implicit surface generated by two point sources at isopotential value T .

The constituent field functions f_i of equation (2.2) 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 isosurfaces provide smooth blending between sources as they are brought together. In animation, surfaces undergo fluid variations, following the changes in position of their constituents.

These features, combined with the ability to construct complex shapes that are difficult for other modeling primitives to describe, have made implicit surfaces a popular tool for many modeling tasks, particularly where the shapes to be modeled are from the natural world [7, 12, 13], or exhibit ‘soft’ properties [73].

The capabilities of any modeling system based on equations (2.1) and (2.2) depend on the choices of modeling primitives, denoted as field functions f_i , and this choice is the essence of the model. As with many modeling 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 isosurface equation (2.1), many different approaches have been described. The earliest methods used only simple primitives, such as spherical or ellipsoidal bumps, either Gaussian [7] or polynomial [50, 73]. More complex primitives used in the context of implicit modeling have included superquadrics [38, 78], generalized implicit cylinders [20] and convolution surfaces [12].

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 locate the isosurfaces efficiently. Naturally, these constraints contradict each other.

Models using simple primitives are well adapted for direct rendering methods such as ray-tracing or ray-casting, because they yield a relatively simple implicit equation (2.1) that can be solved at run-time. However, modeling with point field sources has a number of serious disadvantages. Firstly, using point sources makes the design of complex shapes tedious. In addition, the use of point sources during an interactive modeling session provides little indication of how the final isosurface will appear when rendered. Finally, point-based field functions have difficulty describing circular structures well and can only approximate flat regions.

The use of more complex primitives can solve many of the short-comings of point sources. Convolution surfaces, introduced by Bloomenthal and Shoemake [12] operate with modeling primitives of higher dimensions. These primitives, represented by their field functions f_i , are obtained by spatial convolution between skeletal primitives (such as polygons or curves) and a Gaussian distribution function. Nice properties of convolution, such as superposition and compactness, make modeling with such functions very intuitive.

Convolution surfaces incorporate the smooth blending power and easy manipulability of potential surfaces while expanding the skeletons from points to lines, polygons, planar curves and regions, and in principle, any geometric primitive.

Bloomenthal and Shoemake [12].

While the convolution surface technique is an appealing method for modeling, it is difficult to determine an analytical solution of the field functions f_i , for all but the simplest of primitives. In the original work[12] this problem was circumvented by using precomputed tables. Potentially, this may result in undersampling artifacts and may cause storage problems when the number of primitives used in the model is large.

In this chapter a new method of creating convolution surfaces is presented, based upon analytical solutions for an extended set of potential source primitives. A new potential function is described and a set of field functions is derived for the following widely used primitives: points, line segments, polygons, arcs and planes. Analytical solutions offer the advantage that they provide an exact representation of the surface. Such accuracy can be important, for example, when dealing with primitives of widely varying scales. The resulting functions may be used both for polygonizing or direct rendering of the implicit surfaces formed by the combination of primitives.

2.2. DEFINITIONS

Definition 1 *A convolution surface is the implicit surface based upon a field function $f(\mathbf{p})$, obtained as a spatial convolution of two scalar tri-variate functions $g(\mathbf{p})$ and $h(\mathbf{p})$:*

$$f(\mathbf{p}) = g(\mathbf{p}) \star h(\mathbf{p}) = \int_{R^3} g(\mathbf{r})h(\mathbf{p} - \mathbf{r}) d\mathbf{r} \quad (2.3)$$

The geometry function $g(\mathbf{p})$ gives the spatial distribution of potential sources. The potential function $h(\mathbf{p})$ describes the decay of a potential produced by a single point source. Convolved together, they produce the potential distribution generated by all sources in 3D space. Although convolution is a commutative operation, for our purposes we say that the field function $f(\mathbf{r})$ is obtained by convolving a geometry function $g(\mathbf{r})$ with a potential function $h(\mathbf{r})$, also called a *convolution kernel*.

Definition 2 *A convolution kernel $h(\mathbf{p})$ is a tri-variate function*

$$h(\mathbf{p}) : R^3 \rightarrow R \quad (2.4)$$

that describes the scalar field generated by a single point source.

Figure 2.2 gives an example of one-dimensional convolution with a typical bell-shaped kernel function.

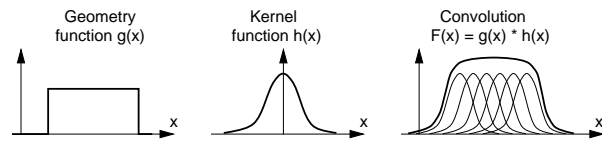


Figure 2.2: One-dimensional convolution.

The general convolution equation (2.3) makes no assumptions about the properties of the geometry function $g(\mathbf{p})$. To evaluate the integral in practice, we restrict our attention to continuous bounded functions $g(\mathbf{p})$ that can be associated with some solid primitive P :

$$g(\mathbf{p}) = \begin{cases} 1 & \mathbf{p} \in P; \\ 0 & \text{everywhere else;} \end{cases} \quad (2.5)$$

For a point modeling primitive, $g(\mathbf{p})$ is a delta-function and the integration (2.3) yields the convolution kernel itself, that is, $f(\mathbf{p}) = h(\mathbf{p})$. For non-point primitives, the convolution integral (2.3) must be evaluated over the volume \mathbf{V} of the primitive:

$$f(\mathbf{p}) = \int_{\mathbf{V}} h(\mathbf{p} - \mathbf{r}) d\mathbf{r} \quad (2.6)$$

Formula (2.6) describes a generic field function which is a component of a modeling equation for a convolution surface. For practical reasons, we will also refer to it as an *implicit primitive*.

Definition 3 *An implicit primitive $f(\mathbf{p})$ is completely defined by its geometry function (2.5), kernel function (2.4) and the convolution integral (2.3)*

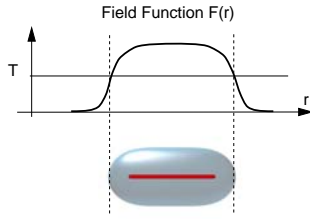


Figure 2.3: A convolution surface, produced by an implicit line segment, rendered in isolation.

When used with the modeling equation (2.1), an implicit primitive $f(\mathbf{p})$ generates an isosurface, as shown in Figure 2.3.

One can easily envisage an implicit primitive as if its potential distribution function $h(\mathbf{p})$ has been ‘spread’ along the volume of the primitive (see Figures 2.2 and 2.3). Each type of modeling primitive has its own unique geometry (point, line, triangle etc.), and, therefore, yields its own unique field function f_i that can be used in isosurface equations (2.1) and (2.2).

In what follows, the terms ‘implicit primitives’, ‘field functions’ and ‘modeling functions’ are used interchangeably. They all refer to an object, given by definition (3). Similarly, the words ‘potential function’, ‘kernel function’ or just ‘kernel’ are used to denote the convolution kernel as given by definition (2).

2.3. KERNELS AND PRIMITIVES

The formulation of a convolution surface, as given in definition (1), depends on the primitive’s geometry $g(\mathbf{r})$ and the convolution kernel $h(\mathbf{r})$. The selection of both components is very important and may influence the effectiveness of the resulting formulation dramatically.

A usable kernel should have the properties that it is continuous, monotonic, diminishes to a negligible contribution beyond a certain distance from the center, and exhibits zero or near zero gradient at this distance. In other words, the kernel should be represented by a bell-shaped function, similar to a Gaussian distribution

$$h(r) = b \exp(-a r^2)$$

which is shown in Figure 2.4. Parameters b and a control the height and the width of the distribution and are set to 1 in this example; r denotes a Euclidean distance to a point of interest (x, y, z) , i.e., $r^2 = x^2 + y^2 + z^2$. This notation will be used further in this section, unless specified otherwise.

Kernels that resemble the Gaussian distribution are in abundance (see references [5, 78] and Appendix A).

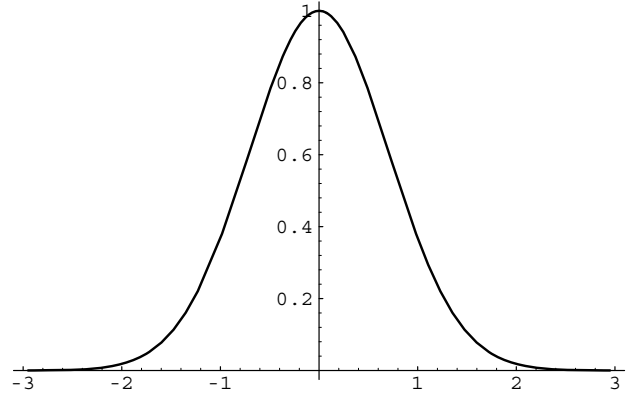


Figure 2.4: A Gaussian function.

However, few of them may be convolved analytically with anything more complex than a point modeling primitive. In fact, at the time this work was undertaken, no convolution kernel has been described that yields a closed-form solution of the integral (2.6) for any modeling primitive but a line segment.

2.3.1. A new kernel function

We propose a new convolution kernel that allows direct analytical solutions of convolution integral (2.6) for a wide family of modeling primitives. The kernel exhibits all the desirable properties for creating smooth surfaces. With this kernel we are able to calculate the *exact* amount of field generated by primitives at an arbitrary point without sacrificing accuracy nor speed. The kernel is:

$$h(r) = \frac{1}{(1 + s^2 r^2)^2}, \quad (2.7)$$

where r is the distance from the point and coefficient s controls the width of the kernel. The kernel is plotted in Figure 2.5.

This function has not been named properly yet. It is reminiscent of a function used in an example by Runge, which is of the form $1/(1 + r^2)$. Runge used this function to demonstrate an oscillatory behavior of the Lagrangian polynomial interpolants, as described in many texts on numerical methods (see, for example [17]). Alternatively, the new kernel (2.7) may also be called a quasi-Cauchy function, because it resembles a Cauchy or Lorentzian distribution, which is $\frac{1}{\pi} \frac{1}{1+r^2}$. For better readability, we will address kernel (2.7) as Cauchy function, keeping in mind that it

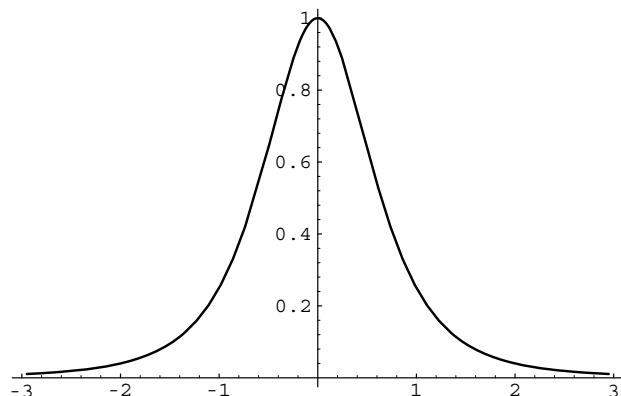


Figure 2.5: A new kernel function.

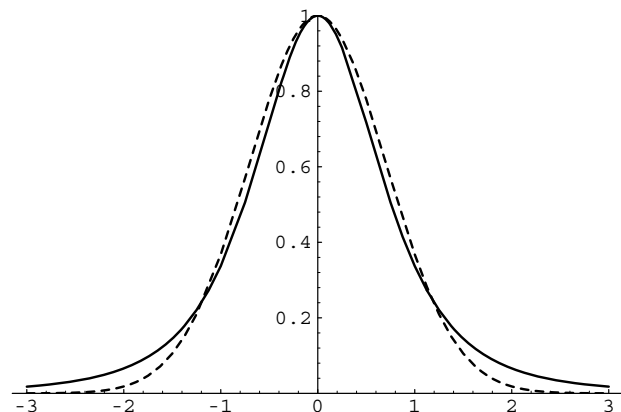


Figure 2.6: A Gaussian (dotted) and the new kernel (solid). Here $s = 0.85$.

is in fact a squared Cauchy function ¹.

The question to ask is: what makes the kernel (2.7) better than the multitude of other similar functions, which may look simpler and seemingly more suitable for modeling purposes? In the following sections, we will provide answers to these questions. We will introduce a pool of alternative kernels that have gained wide recognition in the field of implicit modeling and compare them with the newly introduced kernel (2.7).

2.3.2. The seven kernels

The following seven potential functions meet all the requirements for implicit modeling: they are smooth, monotonic and bounded. Their plots are given in Appendix A.

- **Cauchy function**

$$h(r) = 1/(1 + s^2 r^2)^2, \quad r > 0 \quad (2.8)$$

The newly introduced Cauchy function heads the list of possible convolution kernels for alphabetical reasons.

- **Gaussian function**

$$h(r) = \exp(-a^2 r^2), \quad r > 0 \quad (2.9)$$

This function was first used for the purposes of implicit modeling by Blinn in his blobby model [7]. Also, it was used by Bloomenthal and Shoemake as a convolution kernel in the original paper on convolution surfaces [12] and later in [13].

- **Inverse function**

$$h(r) = 1/r, \quad r > 0 \quad (2.10)$$

Although this function has a singular point at $r = 0$, it still can be used as a potential function. To avoid the division error, the function must be clipped by $1/\epsilon$ for all arguments $0 < r < \epsilon$. The value of ϵ must be chosen small enough so that $1/\epsilon \gg T$ (see equation (2.1)). This will ensure that creases will not occur on the isosurface formed at threshold T . This modeling function was reported to be successfully used in implicit modeling by Wyvill and van Overveld in [80].

- **Inverse squared function**

$$h(r) = 1/r^2, \quad r > 0 \quad (2.11)$$

The same as above, squared. This function also needs clipping as prescribed for the inverse kernel function $1/r$ to ensure that a division overflow does not occur.

¹A reader might be wondering why a squared Cauchy function is better than Cauchy function. The reason is that although a Cauchy distribution has a suitable shape for a low-pass filter, it does not yield closed form solutions of the convolution integral. The squared Cauchy function does.

- **Metaballs**

$$h(r) = \begin{cases} 1 - 3r^2 & 0 \leq r \leq \frac{1}{3}; \\ \frac{3}{2}(1 - r)^2 & \frac{1}{3} < r \leq 1; \\ 0 & r > 1; \end{cases} \quad (2.12)$$

Metaballs are composed of three pieces of quadratic polynomials. They were introduced by Nishimura et. al. in [50].

- **Soft objects**

$$h(r) = \begin{cases} 1 - (\frac{4}{9})r^6 + (\frac{17}{9})r^4 - (\frac{22}{9})r^2, & r < 1; \\ 0 & r > 1; \end{cases} \quad (2.13)$$

This functions is derived from a piecewise Hermite cubic interpolation over the region of influence of a point potential. Introduced by Wyvill et. al. in [73].

- **W-shaped quartic polynomial**

$$h(r) = (1 - r^2)^2, \quad r < 1 \quad (2.14)$$

This function is used for modeling blobby objects in many public domain ray-tracing programs, e.g. [56, 58]. The infinite wings of this quartic polynomial are clipped at the unit distance from the center.

As Appendix A indicates, all polynomial kernels (metaballs, soft objects, quartics) have very similar shapes. Since the quartic function is the simplest among them, we have chosen it for further analysis, assuming that the conclusions will be valid for other polynomial kernels as well.

2.3.3. Primitives/kernels compatibility chart

Applied to the main convolution integral (2.6) the kernels h listed above, performed with different degrees of success for different types of geometry functions g . Closed-form solutions were obtained for five types of primitives: point (trivial), line segment, plane, arc and triangle. There have been several attempts made to perform analytical convolution with a cubic curve, which is a very attractive primitive for purposes of implicit modeling. Unfortunately, none of the kernels produced closed-form solutions for this primitive. The difficulties with cubic curves arise from the necessity of using variable arc lengths under the convolution integral. This additional factor ensures that each point on a curve contributes a correct amount of field into the resulting integral. Line segments and arcs may be naturally parameterized by their length, thus reducing this scaling factor to 1. For cubic curves, variable arc length can not be eliminated easily which makes the integration very

difficult, if at all possible ². The results of the integration are summarized in Table 2.1.

Kernel	Modeling primitives				
	point	line	plane	arc	triangle
Cauchy	•	•	•	•	•
Gaussian	•	•	•	.	.
Inverse	•	•	∞	ellip	•
Squared	•	•	∞	•	.
Polynomial	•	•	•	•	.

Table 2.1: Primitives/kernels compatibility chart: (•) closed-form solution found; (.) no closed-form solution found; (∞) integral yields infinite value; (ellip) a solution is expressed via elliptic integrals;

At a glance, the Gaussian kernel produced three implicit primitives and stopped after planes. This is the smallest number of closed-form solutions.

Inverse and inverse squared kernels yielded solutions for various primitives. Solutions for planes, however, did not converge and the solution for arcs is expressed via elliptical integrals of the first kind. For all practical purposes, this result may be dismissed as prohibitively expensive to compute, as it is not expressed via elementary functions.

The polynomial kernel produced solutions for 4 modeling primitives; unfortunately, a strategically important triangular primitive is not one of them.

Finally, the remaining Cauchy kernel covered the whole set of geometric primitives, demonstrating the best modeling flexibility among all the kernels compared.

2.3.4. Computational costs

Besides the issues of compatibility between kernels and primitives, there is another important characteristic that may in some cases help choose the right kernel. This characteristic is computational cost.

Often, a modeling system that employs implicit surfaces, is not designed with the intention of going beyond linear primitives. (For example, the computer graphics production company Pacific Data Images has chosen to use only two implicit primitives, corresponding to a sphere and a tapered cylinder. More on practices of modeling with implicit surfaces

²In most cases, the actual integration was carried out with the aid of the symbolic-computation package *Mathematica 3.0* [72], which appeared to be very useful for that purpose. It should be mentioned that the previous versions of *Mathematica* (2.2 and 2.0) are not capable of performing the integration for all cases — version 3.0 or higher must be used.

at PDI may be found in [3].) In order to make implicit points and line segments, as Table 2.1 demonstrates, all five kernels may be employed. Computational cost analysis is needed to choose the most effective implementation.

Table 2.2 presents the computational complexity of all five kernels. The upper table describes the kernels themselves: the numbers of floating point operations and special function calls are taken directly from the kernel's definitions, plus additional expenses to compute the squared distance r^2 from an arbitrary point. The lower table describes five implicit line primitives, obtained via convolution integral (2.6). The resulting expressions are rather bulky. An interested reader may find them in Appendix B. All of these functions define density distributions in 3D-space that are somewhat similar to the example shown in Figure 2.3.

Point primitives						
Kernel	Floating point operations					Special functions
	*	/	+	-	Total	
Cauchy	3	1	3	3	10	
Gaussian	3		2	3	8	1 exp
Inverse	3		2	3	8	1 sqrt
Squared	3		2	3	8	
Polynomial	4		2	4	10	

Line primitives						
Kernel	Floating point operations					Special functions
	*	/	+	-	Total	
Cauchy	29	6	11	13	53	2 atan 1 sqrt
Gaussian	11		5	11	27	1 exp 1 erf
Inverse	9		9	13	31	2 log 2 sqrt
Squared	7	3	9	13	32	2 atan 3 sqrt
Polynomial	33	3	14	22	72	1 sqrt

Table 2.2: Computational costs for point and line primitives.

As Table 2.2 demonstrates, it is not an easy task to choose the fastest kernel function, judging solely on the number of floating point operations. The use of special functions obscures the analysis and calls for a more practical method of comparison.

In most cases, the computational costs of evaluating non-algebraic functions are argument-dependent. The reason is the following: non-algebraic functions internally are computed via iterations. These iterations converge at different rates, depending on the value of the argument passed. To simulate the ‘real-life’ situation, a series of timing tests have been conducted for the point and line segment primitive functions. These functions have been evaluated over the bounding boxes of corresponding primitives, of dimensions (6,6,6) for points and (6,16,6) for line segments (see Appendix B for the layout of the bounding boxes and other relevant parameters for line seg-

ments). Each volume was organized into a uniform grid of 150^3 nodes, yielding over three million function evaluations. The tests have been performed on a Pentium processor running at 90 MHz and on a 150 MHz Silicon Graphics machine. The running times and relative speed ratings are given in Tables 2.3 and 2.4.

Point primitives			
Kernel	Pentium	SGI	Combined Rating
	Time:Rating	Time:Rating	
Gaussian	13.93 sec : 5	4.54 sec : 4	5
Cauchy	7.19 sec : 3	4.51 sec : 3	3
Inverse	10.32 sec : 4	5.40 sec : 5	4
Squared	6.54 sec : 2	4.24 sec : 2	2
Polynomial	5.32 sec : 1	3.45 sec : 1	1

Line segment primitives			
Kernel	Pentium	SGI	Combined Rating
	Time:Rating	Time:Rating	
Gaussian	48.22 sec : 5	20.29 sec : 4	5
Cauchy	40.79 sec : 4	23.35 sec : 5	4
Inverse	29.54 sec : 3	14.52 sec : 2	2
Squared	28.62 sec : 2	17.48 sec : 3	3
Polynomial	15.14 sec : 1	8.80 sec : 1	1

Table 2.3: Timing test results and speed ratings for points and line segments.

Points and line segments				
Kernel	Prim	Time	Time	Overall Rating
		Pentium	SGI	
Gaussian	point			5
	segment			
Cauchy	point			4
	segment			
Inverse	point			3
	segment			
Squared	point			2
	segment			
Polynomial	point			1
	segment			

Table 2.4: Summary of the timing tests and speed ratings for all kernel functions.

As Tables 2.3 and 2.4 show, the polynomial kernel consistently produces the implicit point and line modeling primitives that are the fastest to evaluate on both processors. This result confirms that polynomial formulations of point-based modeling primitives in implicit models, such as metaballs in Nishimura et al. (1985) and soft objects in Wyvill et al. (1986), are generally considered as an improvement over the

original blobby model, introduced by Blinn (1982), which employed the Gaussian function.

2.3.5. Problems with polynomial kernels

As Table 2.4 shows, the polynomial kernel has the best speed rating among other kernels for point and line primitives. In addition, it has a finite support (i.e., it is defined over a limited range of distances) which yields effective bounding volumes for the resulting modeling primitive. Finally, the polynomial kernel demonstrated a reasonably good compatibility with a variety of modeling primitives. Why not use polynomial kernels, then?

This is an important questions and we'll spend some time on it. First, consider a simple example of finding the field function of a line segment using the main convolution integral (2.6) ³ and some generic kernel $h(\mathbf{p})$:

$$f_{line}(\mathbf{p}) = \int_{V_{line}} h(\mathbf{p} - \mathbf{v}) d\mathbf{v} \quad (2.15)$$

Here V_{line} is the volume of the primitive, in this case, the length of the line segment L . Introducing x as a distance along the line segment and keeping in mind that most of the kernels (2.8 – 2.14) are defined via the squared distance to the point, re-write (2.15) as

$$f_{line}(\mathbf{p}) = \int_0^L h(r^2(\mathbf{p}, x)) dx. \quad (2.16)$$

As usual, $r(x)$ is the distance from a point of interest \mathbf{p} to a point on the line segment x . Without the loss of generality, let us restrict our attention to points that belong to the line segment, which allows us to re-write the convolution integral as a function of a one-dimensional argument in the local coordinate system of the line segment ⁴:

$$f_{line}(x) = \int_0^L h((x - t)^2) dt, \quad 0 < x < L, \quad (2.17)$$

where x denotes a point on a line segment (see Figure 2.7). Now, we can try to plug various kernels into integral (2.17).

It is easy to see that for convolution with kernels of infinite support (2.8 – 2.11), we can integrate along

³It should be clearly stated that integration here is analytical, not numerical. Numerical integration can easily be performed by scattering sample points along a primitive.

⁴This simplified integration is used for illustrative purposes only. The real field functions for line segments are defined over 3D space in the world coordinates. They are presented in Section “Development of Implicit Primitives” and also in Appendix B).

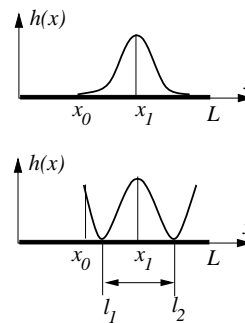


Figure 2.7: Convolving a line segment with Cauchy kernel (top) and polynomial kernel (bottom).

the whole length of the segment L , for any point x_0 (see Figure 2.7, top). In this case, all points x_1 , lying on the segment, will contribute the correct amount of their potential at the point of interest x_0 . Thus, integral (2.17) may be calculated from 0 to L for all points x_0 .

For polynomial kernels (2.12 – 2.14), the situation is very different. Consider the same point x_0 (Figure 2.7, bottom). The contribution from point x_0 , positioned farther than the half of the kernel's width $|l_2 - l_1|$, is grossly incorrect. To cut off the infinite wings of polynomial kernels, these kernels must be windowed, i.e., the appropriate interval of integration must be obtained, as pictured in Figure 2.8. The size of this interval $I = [l_1, l_2]$ depends on the mutual position of the line segment and point of interest x_0 . Once the values for l_1 and l_2 are obtained, integral (2.17) can be evaluated:

$$f_{line}(x) = \int_{l_1}^{l_2} h((x - t)^2) dt, \quad 0 < x < L, \quad (2.18)$$

The result of this integration may be found in Appendix B.

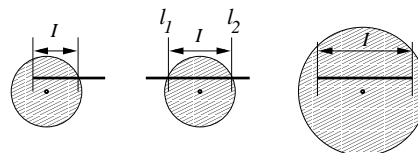


Figure 2.8: Finding integration domain for line segments: three cases of line/sphere intersections.

Thus, finding the proper integration boundaries introduces additional costs for using polynomial kernels. For line segments, these costs involve intersecting a line segment with a sphere — a region where

the kernel is defined. This operation requires solving one quadratic equation, which is easy to do.

For triangles, this task is much more difficult. Clipping a triangle against a sphere yields a variety of possible configurations. They are pictured in Figure 2.9.

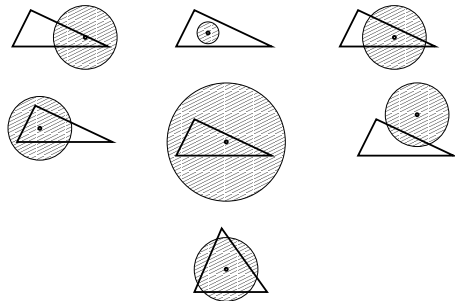


Figure 2.9: Finding integration domain for triangles.

In each particular case, the area of intersection between the sphere and the triangle defines the planar integration domain S :

$$f_{triangle}(\mathbf{p}) = \int_{S(p,x,y)} h(r^2(\mathbf{p}, x, y)) dx dy. \quad (2.19)$$

integrating in the triangle's plane inside region S . In the general case, this task is not suitable for analytical solution. The only easy way to compute the convolution between a triangle and a polynomial kernel (or any other kernel with a finite support) is to ensure that the kernel is always defined over the area of the whole primitive (Figure 2.9, center). This may be achieved by using small triangles or widening the kernel. In both cases, however, triangles would appear more like point primitives and will lose their geometric identity and modeling value, as their size decreases with respect to the kernel's width.

To conclude: because of the finite domain size, polynomial kernels are inherently ill-suited for finding analytical solutions of the convolution integral (2.6), unless the characteristic size of the primitive is much smaller than the width of the kernel.

2.3.6. A short summary

In this section, we have compared a number of kernel functions. We examined their suitability for using with the convolution surface model, paying special attention to

- existence of an analytical solutions to a convolution integral (Table 2.1);

- computational complexity (Table 2.4).

As we have demonstrated, choosing the right convolution kernel is a delicate task that should be approached carefully. Before making that choice, one must decide which modeling primitives have to be 'implicitized' with the convolution integral (2.6).

Sometimes, only the simplest modeling primitives are required, e.g. points and line segments. In such cases, a wide family of kernels may be considered for practical implementation. The speed ratings, given in Tables (2.3, 2.4) may serve as a good criterion for choosing the best kernel.

Another possible selection criterion is the esthetic appeal of the resulting shapes. Each kernel has its unique 'signature' which shows in the way the surface components blend together. For instance, the image of a coral tree shown later in this chapter, is produced with a Cauchy kernel. The same coral tree appears in Chapter 4, this time convolved with a Gaussian kernel. Although the skeletons of both coral trees are identical, different convolution kernels result in slightly different looking shapes. The difference may be especially noticeable between kernels with finite and infinite support. To illustrate, consider images of blobs, produced with a Gaussian kernel (Figure 1.1) and a polynomial kernel (Figure 2.1). Some people may argue that exponential blends look more 'organic' and polynomial blends are more 'mechanical'. This is a matter of personal choice.

For a wider variety of modeling primitives, the first priority should be given to kernels which can be convolved with these primitives analytically. Thus, we have chosen the Cauchy kernel that demonstrated the best computability among other kernels and primitives.

2.4. DEVELOPMENT OF IMPLICIT PRIMITIVES

Using the new convolution kernel (2.7) shown in Figure (2.5), the field functions for a number of primitives can be derived. These primitives are

- point sources
- line segments
- arcs
- triangles
- planes

Other primitives, e.g. circles and polygons, may be built by combining, respectively, arcs and triangles. In the following section, development of field functions for each primitive is given in detail.

2.4.1. Point sources

A point is a 0-dimensional object, so its geometry function is a simple delta-function at point \mathbf{p} (see Figure 2.10). Integration (2.6) yields the kernel function, projected into 3D-space as:

$$F_{point}(\mathbf{r}) = \frac{1}{(1 + s^2|\mathbf{p} - \mathbf{r}|^2)^2} \quad (2.20)$$

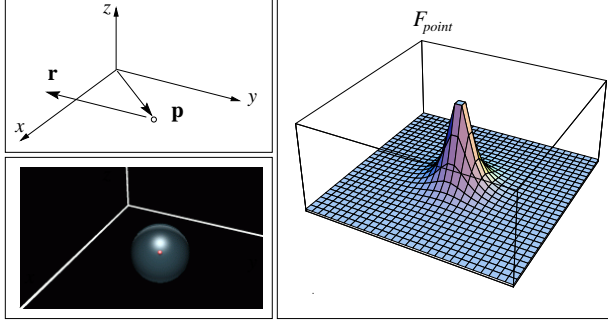


Figure 2.10: Point modeling primitive: geometry (top-left); an intensity plot of F_{point} over the $Z = 0$ plane (right); a rendered version of an isolated primitive (in red) and it's corresponding isosurface (transparent surface).

2.4.2. Line segments

A line segment of length l is defined as:

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

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

$$r^2(t) = d^2 + t^2 - 2t\mathbf{d}\mathbf{a},$$

where $d = \|\mathbf{d}\|$ is the magnitude of a vector from the segment base to \mathbf{r} : $\mathbf{d} = \mathbf{r} - \mathbf{b}$. To obtain the field function, r^2 is substituted into the general formula (2.6) and integrate:

$$\begin{aligned} F_{line}(\mathbf{r}) &= \int_0^l \frac{dt}{(1 + s^2 r^2(t))^2} = \\ &= \frac{x}{2p^2(p^2 + s^2 x^2)} + \frac{l-x}{2p^2 q^2} + \\ &+ \frac{1}{2sp^3} \left(\text{atan}\left[\frac{sx}{p}\right] + \text{atan}\left[\frac{s(l-x)}{p}\right] \right), \end{aligned} \quad (2.21)$$

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

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

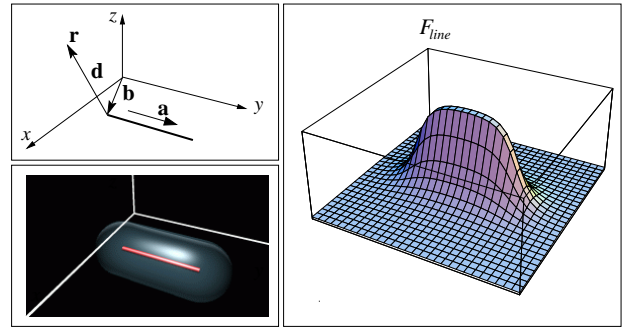


Figure 2.11: Line modeling primitive.

2.4.3. Arcs

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

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

where r is the radius of the circle the arc lies on, θ is the arc angle (Figure 2.12, top-left). Integrating along the angle, the field function in local coordinates is obtained:

$$\begin{aligned} F_{arc}(x, y, z) &= \int_0^\theta \frac{dt}{(1 + s^2 r^2(t))^2} = \\ &= \frac{by}{xp^2(kx - b)} + \frac{k(x^2 + y^2)\sin(\theta) - by}{xp^2(k(x\cos(\theta) + y\sin(\theta)) - b)} + \\ &+ \frac{2b}{p^3} \left(\text{atanh}\left[\frac{ky}{p}\right] + \text{atanh}\left[\frac{(kx + b)\tan(\frac{\theta}{2}) - ky}{p}\right] \right), \end{aligned} \quad (2.22)$$

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

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

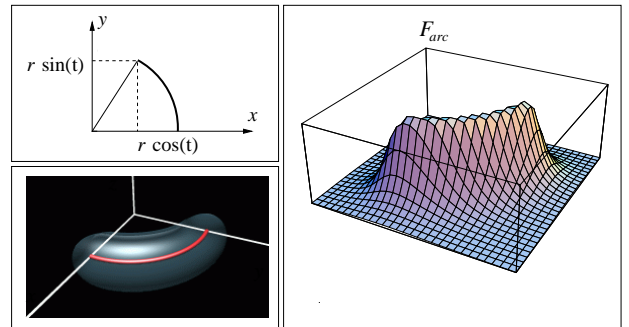


Figure 2.12: Arc modeling primitive.

2.4.4. Triangles

A triangular primitive requires integration in two dimensions, which is slightly more complicated than for one-dimensional cases. First, a triangle is split along the longest edge into two right-angled triangles (Figure 2.13).

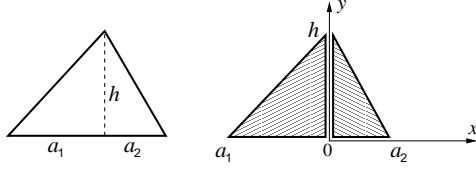


Figure 2.13: Integration parameters.

Next, the field function for the right half is obtained:

$$F_{right}(x, y) = \int_{y=0}^{h - \frac{hx}{a_2}} \int_{x=0}^{a_2} \frac{dx dy}{(1 + s^2 r^2(x, y))^2}$$

and F_{left} is derived from F_{right} by replacing x by $-x$ and a_2 by a_1 . Finally, F_{left} and F_{right} are added together.

For an arbitrarily positioned triangle (Figure 2.14, top-left), the following parameters are defined: 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. h is the distance from \mathbf{b} to the apex of the triangle. Introducing the vector $\mathbf{d} = \mathbf{r} - \mathbf{b}$ and scalars $u = \mathbf{d} \cdot \mathbf{u}$ and $v = \mathbf{d} \cdot \mathbf{v}$, the final field function is:

$$\begin{aligned} F_{triangle}(\mathbf{r}) = & \frac{1}{2qs} \left(\frac{n}{A} \left(\text{atan}\left[\frac{vh + a_1(a_1 + u)}{A}\right] + \text{atan}\left[\frac{gh + a_1u}{-A}\right] \right) + \right. \\ & + \frac{m}{B} \left(\text{atan}\left[\frac{vh + a_2(a_2 - u)}{-B}\right] + \text{atan}\left[\frac{gh - a_2u}{B}\right] \right) + \\ & \left. + \frac{v}{C} \left(\text{atan}\left[\frac{a_1 + u}{C}\right] + \text{atan}\left[\frac{a_2 - u}{C}\right] \right) \right), \end{aligned} \quad (2.23)$$

$$A^2 = a_1^2 w + h^2(q + u^2) - 2a_1 h u g,$$

$$B^2 = a_2^2 w + h^2(q + u^2) + 2a_2 h u g,$$

$$C^2 = 1/s^2 + d^2 - u^2,$$

$$g = v - h,$$

$$q = C^2 - v^2,$$

$$w = C^2 - 2vh + h^2,$$

$$m = a_2 g + u h,$$

$$n = u h - a_1 g$$

Geometrically, the field function (2.23) is a sum of three 3D step-functions, one for each edge of the underlying triangle. They are pictured in Figure 2.15. These step-functions cancel each other at infinity and form a positive triangular bump between the edges.

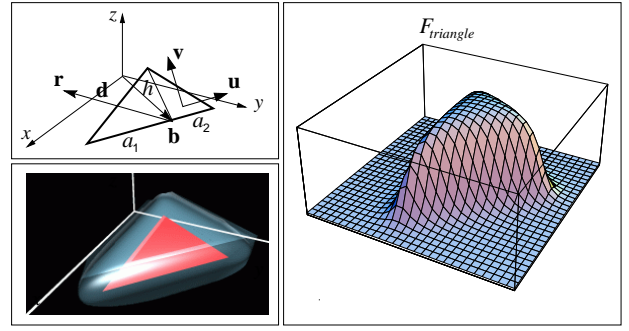


Figure 2.14: Triangle modeling primitive.

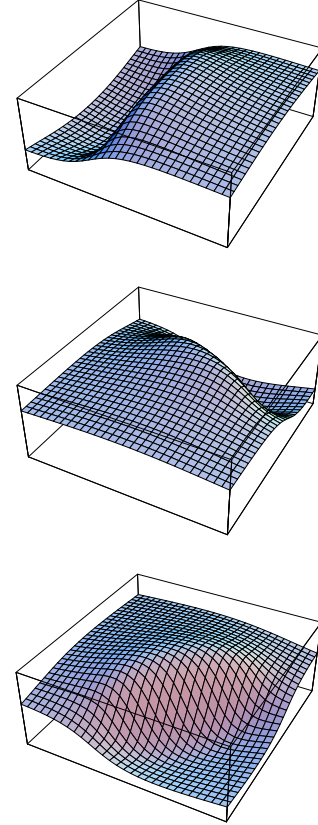


Figure 2.15: Field of a triangle decomposed into three step-functions.

2.4.5. Planes

The field function of an unbounded plane primitive is computed as follows:

$$\begin{aligned} F_{plane}(\mathbf{r}) &= \\ &= \int_S \frac{dS}{(1 + s^2 r^2(t))^2} = \\ &= \int_0^\infty \frac{2\pi t dt}{(1 + s^2 r^2(t))^2} = \frac{\pi}{s^2(1 + s^2 d^2)} \end{aligned} \quad (2.24)$$

where d^2 is the distance between point \mathbf{r} and the plane and dS is the area of an integration ring.

In this section, field functions for five modeling primitives are developed: points (2.20), lines (2.21), arcs (2.22), triangles (2.23) and planes (2.24). Some practical examples of using these functions are given next.

2.5. EXAMPLES

Several images have been computed to demonstrate the field functions developed in the previous section. All implicit primitives pictured in Figures 2.16 – 2.20 form blends with each other; all conventional primitives form unions. These images are rendered with a ray-tracing algorithm that will be described in Chapter 4.

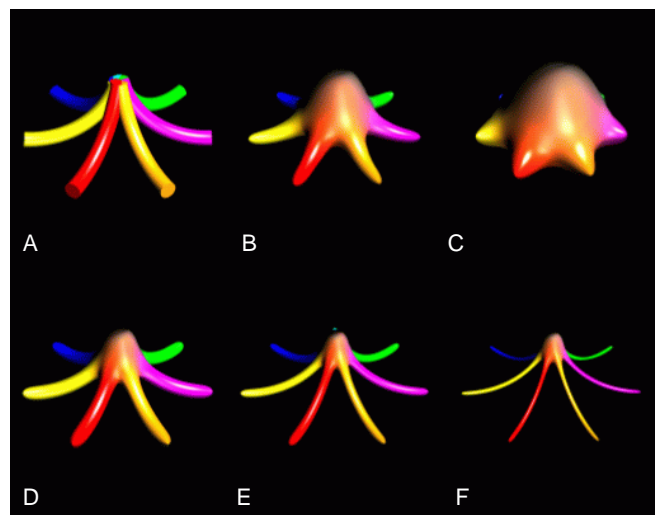


Figure 2.16: Family of convolved starfish. Notice how the colors of the tentacles are blended along the surface.

The starfish-like object in Figure 2.16 is modeled with seven arcs. The top-left image (A) shows the underlying skeletal representation (union of seven arcs). Images (B – F) show the arcs convolved with kernels of various widths and heights. Notice how the color of the tentacle is blended along the surface. Rendering

times are given in Table 2.5 (image size 426 x 512, antialiasing).

Image	Rendering time	Ratio to A
A	5 min 59 sec	1.00
B	18 min 47 sec	3.14
C	24 min 01 sec	4.01
D	12 min 02 sec	2.00
E	9 min 18 sec	1.55
F	7 min 51 sec	1.31

Table 2.5: Rendering convolved starfish.

As Table 2.5 indicates, image C in Figure 2.16 (a very “fat” starfish) takes the longest time to render. The reason is as follows: this particular starfish is modeled with implicit arcs that have very large regions of influence (3D areas where their field is significant). Consider Figure 2.17, which presents time-profiling diagrams for all images A – F. Regions of influence are clearly visible as light silhouettes. The rendering algorithm spent most time in these areas evaluating the field functions. The size of the regions of influence is directly proportional to the width of the kernel and the maximal height of the kernel. For example, “skinny” starfishes E and F have relatively small regions, so they were rendered faster.

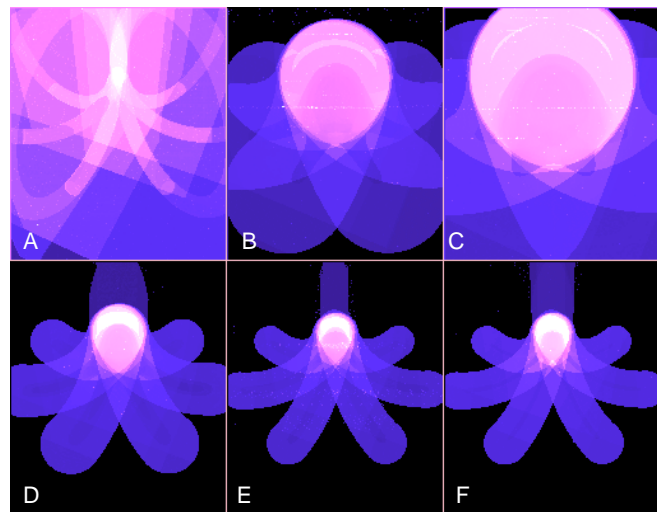


Figure 2.17: Rendering the family of convolved starfish. Lighter areas correspond to longer rendering time, revealing regions of influence of all arcs (pieces of tori capped with spheres). Each time-map image is self-normalized from white to black and gamma-enhanced.

Another example of using implicit arcs is shown in Figure 2.18. The well-known sphere-flake model of Eric Haines [31], has been enhanced by adding stems

to each sphere⁵. The stems are modeled with implicit arcs, the spheres – with implicit Gaussian points. Despite the fact that the convolution kernels are different, the resulting surface is perfectly smooth, both geometrically and photometrically.

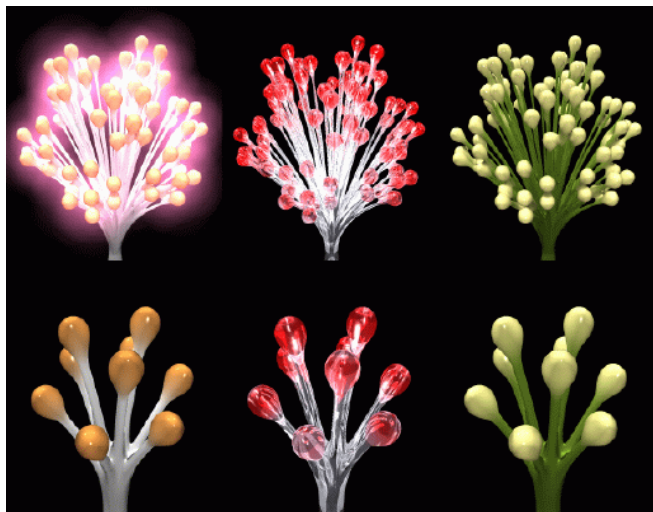


Figure 2.18: Modified implicit sphereflakes: Gaussian point sources (tops) blend with Cauchy arcs (stems).

Figure 2.19 shows some well-known objects, that use arcs, line segments, triangles and planes as their building elements. Primitives of different types blend together smoothly.



Figure 2.19: Blends between primitives of different types: arcs + lines (left); triangle + triangle (right); arcs + lines + planes (bottom).

The images of coral trees in Figure 2.20 illustrate the modeling power of convolution surfaces built with

⁵The sphere-flake model, grown to a full size of 7,381 components, will appear again in Chapter 4, where rendering issues will be discussed.

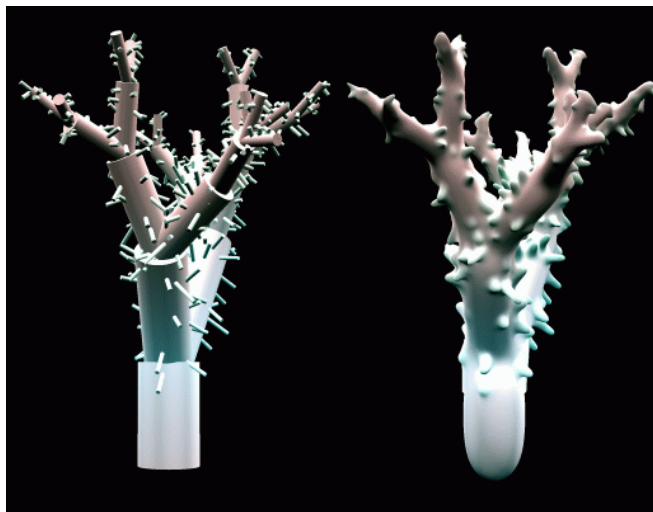


Figure 2.20: Coral tree made of line segments, ‘explicit’ and ‘implicit’ versions.

field functions developed in this chapter. The underlying model for both images is based upon a tree model generator by Eric Haines [31]. Both trees consist of 31 branches, modeled with cylinders of various radii, and 512 randomly positioned spikes, also cylindrical.

2.6. PERFORMANCE

The analytical solutions of the convolution integral (2.3), presented by field functions (2.20 – 2.24) are computationally more expensive than alternative procedural approaches [12, 20, 60]. To compare, evaluation of the analytical field function (2.23) of a triangular primitive requires 43 multiplications/divisions, 35 additions/subtractions, 3 square roots and 6 arctans. The procedural method discussed in the original convolution paper [12] uses a precomputed raster representation of a 2D convolution of a triangle, which allows to evaluate its field function using 14 multiplications and 9 additions only.

Some efficiency gains can be made by using tabulated versions of arctan and square root functions. Timing tests, similar to those performed for the comparative analysis of implicit points and lines, indicate the analytical method is approximately 5.4 times slower than the equivalent procedural solution. However, it is reasonable to expect that in a real rendering or polygonizing environment the timing results should be more favorable. The reason is for that is that analytical functions (2.20 – 2.24) are defined in the world coordinate system (except implicit arcs). Thus, there is no need to perform costly world-to-local coordinate system transformations that are typical for evaluation of field functions of complex implicit modeling primitives.

Between themselves, the newly developed implicit primitives demonstrated computational complexity of wide range, as shown in Table 2.6.






Prim	Time t_i	t_i/t_{point}	Bar chart
Point	3.18 sec	1.00	
Line	14.25 sec	4.48	
Arc	25.97 sec	8.17	
Triangle	45.66 sec	14.36	
Plane	3.26 sec	1.02	

Table 2.6: Timing tests for new modeling functions. The tests were conducted on a 90MHz Pentium, 125^3 evaluations for each primitive.

These data give a good indication when the convolution surface model becomes more effective computationally than a brute-force point-sampling technique. For instance, in a general situation an implicit line primitive that is “longer-than-five-points”, benefits from convolution representation, both geometrically (no wave pattern) and computationally (it can be evaluated faster).

2.7. CONCLUSIONS

We have presented a consistent and mathematically sound method of creating convolution surfaces based upon exact evaluation of the field functions for a wide set of geometric primitives. Functions (2.20–2.24) provide means to calculate values of their field and the gradient at an arbitrary point with an arbitrary precision.

Thus, we suggested a new formulation of the convolution surface model, that contains no granularity in its definition. The new formulation does not require volumetric computation and storage. We believe it to be a most important contribution to the convolution surface model.

We argue that the set of primitives developed in this chapter can be considered as a necessary minimal set of tools for modeling with convolution surfaces. Indeed, points, lines and triangles are the essential building elements of dimensionality 0, 1 and 2, respectively. In general case, they cannot be approximated by any other data types without introducing undersampling artifacts. On the other hand, due to the additive property of convolution, these primitives may be successfully used in creating more complex data types for implicit modeling. For example, arcs may be combined into circles, spirals and other 3D curves, triangles into polygons, segments into polylines etc.

Convolution surfaces allow modeling possibilities that would be difficult to achieve using other geometric modeling techniques. The analytical solutions

for the convolution of higher order primitives as presented above, extends the options for the modeling with implicit surfaces, which will be discussed in the next chapter.

We are all dreaming of a speech without words that utters the inexpressible and gives form to the formless.

–Hermann Hesse (1877-1962)
“*Steppenwolf*”

