

Kernel functions in convolution surfaces: a comparative analysis

Andrei Sherstyuk

School of Computer Science and Software Engineering
Monash University, Victoria 3168, Australia
email: ash@csse.monash.edu.au,

Abstract

A comprehensive analysis of various convolution kernels is presented. Computational complexity and compatibility between the kernels and a number of modeling primitives are examined. A number of practical suggestions are given how to choose the proper kernel function, with a special attention to polynomial kernels. Mathematical formulations for convolved line segments are given.

Key words: Geometric modeling – Isosurfaces – Polynomial line segments – Implicit modeling primitives

1 Introduction

A convolution surface is the set of points (x, y, z) that satisfy

$$f(x, y, z) = T \quad (1)$$

where T is some scalar value and the field function $f(x, y, z)$ is obtained via a 3D convolution of a *kernel* function $h(\mathbf{p})$ and a *skeleton* function $g(\mathbf{p})$:

$$f(\mathbf{p}) = \int_S g(\mathbf{r})h(\mathbf{p} - \mathbf{r}) d\mathbf{r}, \quad (2)$$

integrating for all points \mathbf{r} that belong to the skeleton S . Skeleton elements may be points, line segments, curves, polygons, and other geometrical modeling primitives. Kernels may be represented by a number of functions, such as a Gaussian function

$$h(x, y, z) = b e^{-a(x^2+y^2+z^2)} \quad (3)$$

Figure 1 gives an example of a skeleton and a corresponding convolution surface.

Convolution surfaces were introduced into computer graphics by Bloomenthal and Shoemake (1991) as a logical extension of point-based implicit surface models: Blinn (1982), Nishimura et al. (1985), Wyvill et al. (1986). Due to their valuable properties, such as a superposition and compactness, convolution surfaces provide a powerful and flexible environment for geometric modeling. Convolution surfaces are particularly well suited for designing objects of

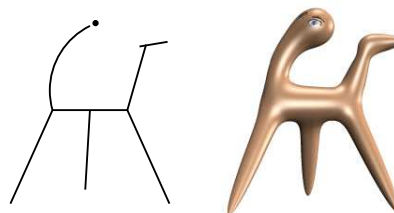


Figure 1. An example of modeling with convolution surfaces: a skeleton (left) and a convolution surface (right). The figure is modeled after *Angel*, a sculpture by Deborah Halpern.

organic origin. The modeling capabilities of convolution surfaces are examined in Bloomenthal (1985, 1995, 1997). The special topics of creating bulge-free surfaces are discussed in Bloomenthal (1997a).

While modeling potentials of convolution surfaces are very attractive, the actual mathematical formulation of such surfaces still has many open questions. The major problem is that the convolution integral (2) seldom yields closed-form solutions that can be evaluated directly. The choice of kernel functions $h(\mathbf{p})$ and geometrical functions $g(\mathbf{p})$ that can be convolved together analytically, is very limited.

Several researches suggested point-sampling methods in order to circumvent this problem. Bloomenthal and Shoemake (1991) described how to evaluate the convolution integral (2) for polygonal primitives. They used a precomputed mosaic of images of polygons, scan-converted in the (xy) -plane and filtered with a Gaussian kernel (3) in 2 dimensions. To evaluate the field function $f(\mathbf{p})$, a point of interest \mathbf{p} is first projected onto an appropriate image cell and the convolution values are obtained from the image. Then, this value is scaled by a perpendicular z -component of the Gaussian function. The demands on memory to store the filtered images are $O(n^2)$ for an $n \times n$ image.

Sealy and Wyvill (1996) suggested evaluation of the convolution integral (2) using volume sampling techniques and tri-linear interpolation between the nodes. They replaced the integration (2) by a discrete summation over the

model space, aiming for a more general procedural solution. The trade-off for this generality is the usual ‘accuracy-vs-storage’ problem. In a brute-force uniform voxel representation memory demands are $O(n^3)$, therefore, Sealy and Wyvill used an adaptive octree data representation.

We would like to emphasize, that at present, the lack of closed-form methods of evaluating the convolution integral (2) is the major drawback of the convolution surface model. In general, the model requires a raster representation for two-dimensional primitives and volume representation for three-dimensional primitives. Both representations use point-sampling evaluation techniques that may cause undersampling artifacts. Also, they require large volumes of memory for storage, which may become prohibitive for complex scenes.

The necessity of point sampling and storage makes the convolution surface model conceptually discrete, even though its mathematical foundation requires and implies continuity. Thus, we advocate direct solutions for the convolution surface integral as a non-compromising approach for evaluating convolution surfaces. Consequently, we restrict our attention to those convolution kernels and modeling primitives that yield closed-form solutions for the integral (2).

Ideally, one would want to have a kernel that can be convolved directly with as many primitives as possible. Such kernels are few, and the more primitives are required, the smaller is the number of possible kernels. However, many modeling tasks do not require a large variety of primitives. Sometimes, spheres and cylinders are sufficient, as described in Beier (1990). Under such circumstances, the software developers may choose among a wide variety of kernels.

In this paper, we examine a number of kernels that may be used with the convolution surfaces directly. We present a kernels/primitives compatibility chart and timing analysis that provide good criteria for choosing the best kernel for each particular modeling task. In addition, we present mathematical expressions for a line segment modeling primitive $f_{line}(\mathbf{p})$, convolved with the various kernels discussed in the paper. These field functions may be used for the purposes of geometric modeling.

2 The seven kernels

To support smoothing and blending properties that are characteristic for implicit surfaces in general and convolution surfaces in particular, a usable convolution kernel has to be a low-pass filter. In other words, the kernel should be represented by a bell-shaped function, similar to a Gaussian distribution (3) which is plotted in Figure 2. 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 $\mathbf{p} = (x, y, z)$, i.e., $r^2 = x^2 + y^2 + z^2$.

This notation will be used further, unless specified otherwise.

Kernels that resemble the Gaussian distribution are in abundance, as shown by Blanc and Schlick (1995) and Wyvill and Wyvill (1989). However, few of them may be convolved analytically with anything more complex than a point modeling primitive. For the purposes of our analysis, we have chosen those kernels that yield closed-form solutions of the integral (2) for at least a line segment. These kernels are introduced next.

2.1 Gaussian function

$$h(r) = \exp(-a^2 r^2), \quad r > 0 \quad (4)$$

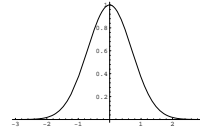
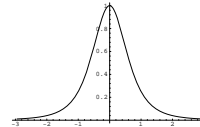


Figure 2. A Gaussian function.

This function was first used for the purposes of implicit modeling by Blinn (1982) in his blobby model. It was also used by Bloomenthal and Shoemake (1991) as a convolution kernel in their original paper on convolution surfaces.

2.2 ‘Cauchy’ function

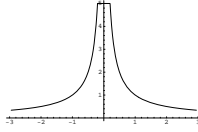
$$h(r) = 1/(1 + s^2 r^2)^2, \quad r > 0 \quad (5)$$



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, for example, in Buchanan and Turner (1992). Alternatively, this kernel 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 refer to this kernel as the Cauchy function, keeping in mind that it is in fact a squared Cauchy function. This function was first employed as a convolution kernel by McCormack and Sherstyuk (1998).

2.3 Inverse function

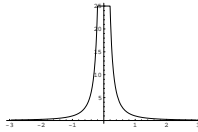
$$h(r) = 1/r, \quad r > 0 \quad (6)$$



Although this function has a singular point at $r = 0$, it still can be used as a convolution kernel. 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 (1). Such clipping ensures 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 (1996).

2.4 Inverse squared function

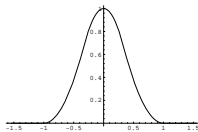
$$h(r) = 1/r^2, \quad r > 0 \quad (7)$$



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.

2.5 Metaballs

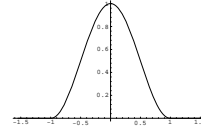
$$h(r) = \begin{cases} 1 - 3r^2 & 0 \leq r \leq \frac{1}{3}; \\ \frac{3}{2}(1 - r)^2 & \frac{1}{3} < r \leq 1; \\ 0 & r > 1; \end{cases} \quad (8)$$



Metaballs are composed of three pieces of quadratic polynomials. They were described by Nishimura et al. (1985).

2.6 Soft objects

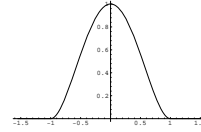
$$h(r) = \begin{cases} 1 - (\frac{4}{9})r^6 + (\frac{17}{9})r^4 - (\frac{22}{9})r^2, & r \leq 1; \\ 0 & r > 1; \end{cases} \quad (9)$$



This function is derived from a piecewise Hermite cubic interpolation over the region of influence of a point source. It was introduced by Wyvill et al. (1986).

2.7 Quartic polynomial

$$h(r) = \begin{cases} (1 - r^2)^2, & r \leq 1; \\ 0 & r > 1; \end{cases} \quad (10)$$



This function is used for modeling blobby objects in many public domain ray-tracing programs, e.g. *Rayshade* and *POV-Ray*. The infinite wings of this quartic polynomial are clipped at the unit distance from the center.

As the plots indicate, all polynomial kernels (metaballs, soft objects, quartics) have very similar shapes. Since the quartic function (10) 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.

3 Primitives/kernels compatibility chart

Applied to the main convolution integral (2) 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.

In most cases, the integration was carried out with the aid of the symbolic-computation package *Mathematica 3.0* by Wolfram (1996), which appeared to be very useful for that purpose. It should be mentioned that the previous versions of *Mathematica* (2.0 and 2.2) are not capable of performing the integration for all cases — version 3.0 or higher must be used. The results are summarized in Table 1.

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

Table 1. Primitives/kernels compatibility chart: (•) closed-form solution exists, (•) no closed-form solution, (∞) integral yields infinite value, (elliptic) 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. The actual formulas, resulting from the integration (2) with the Cauchy kernel for all 5 primitives are given by McCormack and Sherstyuk (1998).

4 Computational efficiency

Besides the issues of compatibility between kernels and primitives, there is another important characteristic that may in some cases help choosing 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 have chosen to use only two implicit primitives: a sphere and a tapered cylinder, corresponding to a point and a line segment. More on practices of modeling with implicit surfaces at PDI may be found in Beier (1990). In order to produce implicit points and line segments, as Table 1 demonstrates, all five kernels may be employed. Computational cost analysis is needed to choose the most effective implementation.

Table 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 kernels' definitions and include the additional expenses of computing the distance r from an arbitrary point. The lower table describes five implicit line segment primitives, obtained via convolution integral (2). The resulting expressions are given in Appendix A.

Kernel	Point primitives				Special functions	
	Floating point operations					
	*	/	+	-	Total	
Gaussian	4		2	3	9	1 exp
Cauchy	4	1	3	3	11	
Inverse	3	1	2	3	9	1 sqrt
Squared	3	1	2	3	9	
Polynomial	4		2	4	10	

Kernel	Line segment primitives				Special functions	
	Floating point operations					
	*	/	+	-	Total	
Gaussian	11		5	11	27	1 exp 2 erf
Cauchy	23	6	11	13	53	2 atan 1 sqrt
Inverse	9		9	13	31	2 log 2 sqrt
Squared	7	3	9	13	32	2 atan 1 sqrt
Polynomial	33	3	14	22	72	1 sqrt

Table 2. Computational costs for points and line segments, convolved with various kernels. A polynomial line segment includes additional operations for finding integration boundaries, which involves clipping the segment again a sphere (7 multiplications, 6 additions, 5 subtractions and 1 square root).

As Table 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 A for the layout of the bounding boxes and other relevant parameters for line segments). 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 3 and 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 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 4. Summary of the timing tests and speed ratings for all kernel functions.

As Tables 3 and 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.

5 A note on polynomial kernels

As Table 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 results in effective bounding volumes for the modeling primitive. Finally, the polynomial kernel demonstrated a reasonably good compatibility with a variety of modeling primitives. Unfortunately, there is one serious problem that makes polynomial kernels difficult to implement for higher-dimensional primitives.

Table 1 shows the existence of closed form expressions for the definite integrals that are computed over the whole volume of each primitive. The limited range of influence of the polynomial kernels make the integration domain position dependent. While theoretically it is still possible to find closed form expressions for polynomial-based convolution integrals, practical computations become prohibitively difficult. The following two examples illustrate this problem.

5.1 Computing the field function for a line segment

First, consider a simple task of finding the field function of a line segment using the main convolution integral (2) and some generic kernel $h(\mathbf{p})$:

$$f_{line}(\mathbf{p}) = \int_{V_{line}} h(\mathbf{p} - \mathbf{r}) d\mathbf{r} \quad (11)$$

Here V_{line} is the volume of the primitive, in this case, the length L of the line segment. Introducing x as a distance along the line segment and keeping in mind that most of the kernels (4 – 10) are defined via the squared distance to the point \mathbf{p} , re-write (11) as

$$f_{line}(\mathbf{p}) = \int_0^L h(r^2(\mathbf{p}, x)) dx. \quad (12)$$

As usual, $r(x)$ is the distance from a point of interest \mathbf{p} to a point on the line segment x . For illustrative purposes, 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 ¹:

¹This simplified integration is used for illustrative purposes only. The

$$f_{line}(x) = \int_0^L h((x-t)^2) dt, \quad 0 < x < L, \quad (13)$$

where x denotes a point on a line segment (see Figure 3). Now, we can try to plug various kernels $h(x)$ into integral (13).

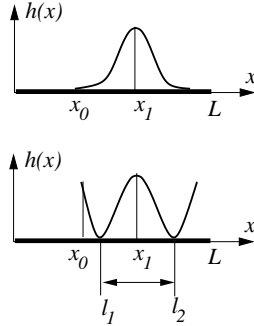


Figure 3. Convoluting a line segment with an infinite-support kernel (top) and a polynomial kernel (bottom).

It is easy to see that for convolution with kernels of infinite support (4 – 7), we can integrate along the whole length of the segment L . Consider Figure 3, top. In this case, all points x_0 , lying on the segment, will contribute the correct amount of their field at the point of interest x_1 , where the function $f_{line}(x_1)$ is being evaluated. Thus, integral (13) may be calculated from 0 to L for all points x_1 .

For polynomial kernels (8 – 10), the situation is very different, as shown in Figure 3, 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, such kernels must be windowed by an appropriate interval of integration $I = [l_1, l_2]$, as pictured in Figure 4. The size of this interval depends on the relative position of the line segment and point of interest x_1 . Once the values for l_1 and l_2 are obtained, integral (13) can be evaluated:

$$f_{line}(x) = \int_{l_1}^{l_2} h((x-t)^2) dt, \quad 0 < x < L, \quad (14)$$

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

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, i.e., a region where the kernel is defined. This operation requires solving one quadratic equation, which is easy to do.

real field functions for line segments are defined over 3D space in the world coordinates.

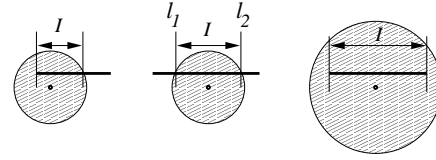


Figure 4. Finding integration domain for line segments: three cases of line/sphere intersections.

5.2 Computing a field function for a triangle

For triangles, finding an integration domain is much more difficult. Clipping a triangle against a sphere yields a variety of possible configurations. Some of them are pictured in Figure 5.

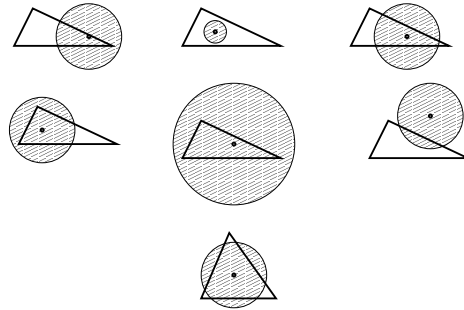


Figure 5. Finding integration domain for triangles.

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

$$f_{triangle}(\mathbf{p}) = \int_{S(\mathbf{p},x,y)} h(r^2(\mathbf{p},x,y)) dx dy \quad (15)$$

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 5, 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.

6 A short summary

In this paper, 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 solution to a convolution integral (Table 1);
- computational complexity (Table 4).

Auxiliary kernel characteristics can be also of some interest, and deserve a short note. They include, for example, the tightness of a bounding volume that the modeling primitive may be enclosed by, or complexity of the normal vector computations, which are required for surface shading. However, these characteristics are less important, since all kernel functions described above may be clipped and scaled within almost arbitrary bounding values. Expressions for normal vectors may be easily obtained by computing gradients of the field functions, then negating and normalizing the resulting vectors.

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). 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 (3, 4) may serve as a good criterion for choosing the best kernel. For a better variety of modeling primitives, a Cauchy function should be used instead. Also, it is always possible to combine primitives convolved with various kernels into one skeleton. The two-headed horse-like objects, pictured in Figure 1, contains such diverse primitives: a Gaussian point (the left head), a Cauchy arc (the left neck) and Gaussian line segments (the rest of the body).

7 Acknowledgements

Many thanks to Ken Shoemake for his help and discussion on variable arc length in convolution of cubic primitives, which still remains an unsolved problem. I am also thankful to David Albrecht and Alexander Kolesnikov for their help with finite-support kernels and to Peter Tischer for his scientific and literary criticism. Encouraging reports from the anonymous referees were also greatly appreciated.

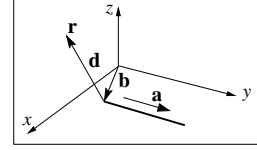
A Field functions for line primitives

The following functions describe the scalar fields produced by a line segment convolved with various kernels. A line segment of length L is defined as:

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

where

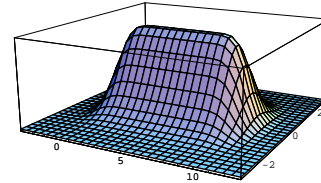
- b** segment base (bx, by, bz)
- a** segment axis (ax, ay, az) , normalized
- r** point of interest (x, y, z)
- d** vector from segment base to a point **r**
- d length of **d**
- x dot product of **d** and **a**



Three-dimensional plots show the field distributions in $z = 0$ plane. In all plots, the parameters are $\mathbf{b} = (0, 0, 0)$, $\mathbf{a} = (1, 0, 0)$, $L = 10$. All functions are defined in world coordinates.

A.1 Gaussian line segment

$$f_{line}(\mathbf{r}) = e^{-a^2(d^2-x^2)}(\text{erf}[a(L-x)] + \text{erf}[ax]), \quad a = 1$$



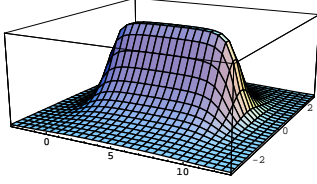
A.2 Cauchy line segment

$$f_{line}(\mathbf{r}) = \frac{x}{2p^2(p^2 + s^2x^2)} + \frac{L-x}{2p^2q^2} + \frac{1}{2sp^3}(\text{atan}[\frac{sx}{p}] + \text{atan}[\frac{s(L-x)}{p}]),$$

where kernel width $s^2 = 0.25$ and p and q are

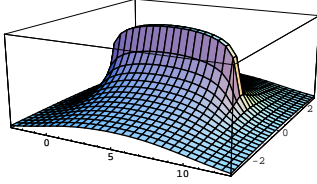
$$p^2 = 1 + s^2(d^2 - x^2),$$

$$q^2 = 1 + s^2(d^2 + L^2 - 2Lx),$$



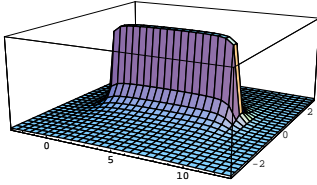
A.3 Inverse potential line segment

$$f_{line}(\mathbf{r}) = \ln(L - x + \sqrt{d^2 - 2xL + L^2}) - \ln(d - x)$$



A.4 Inverse squared line segment

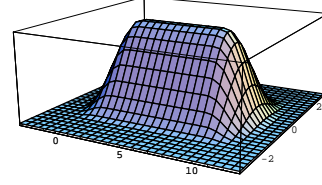
$$f_{line}(\mathbf{r}) = \frac{\operatorname{atan}\frac{x}{\sqrt{d^2-x^2}} + \operatorname{atan}\frac{L-x}{\sqrt{d^2-x^2}}}{\sqrt{d^2-x^2}}$$



A.5 Polynomial line segment

$$\begin{aligned} f_{line}(\mathbf{r}) = & ((l_2 - l_1)(R^2 - d^2)^2 \\ & + 2(l_2^2 - l_1^2)x(R^2 - d^2) \\ & + 2(l_2^3 - l_1^3)((2x^2 + d^2 - R^2))/3 \\ & - (l_2^4 - l_1^4)x \\ & + (l_2^5 - l_1^5)/5)/R^4 \end{aligned}$$

where R is the width of the kernel and $[l_1, l_2]$ is the integration interval I as shown in Figure 4. Here $R = 2$.



For ray-tracing of polynomial implicit line segments, a ray equation $\mathbf{r}(t) = \mathbf{A} + t\mathbf{B}$ may be substituted into $f_{line}(\mathbf{r})$. Collecting terms with respect to ray-distance t :

$$f_{line}(t) = (c_4 t^4 + c_3 t^3 + c_2 t^2 + c_1 t + c_0)/R^4$$

where

$$\begin{aligned} c_0 &= k_5 - 4k_4u + 2k_3(p^2 + 2u^2) + p^2(k_1p^2 - 4k_2u), \\ c_1 &= 4(q(k_3 - 2k_2u + k_1p^2) - v(k_4 - 2k_3u + k_2p^2)), \\ c_2 &= 2(q(2qk_1 - 4vk_2) + k_3 + 2v^2k_3 - 2k_2u + k_1p^2), \\ c_3 &= 4(qk_1 - vk_2), \\ c_4 &= k_1 \end{aligned}$$

$$\begin{aligned} k_1 &= (l_2 - l_1)/1, \\ k_2 &= (l_2^2 - l_1^2)/2, \\ k_3 &= (l_2^3 - l_1^3)/3, \\ k_4 &= (l_2^4 - l_1^4)/4, \\ k_5 &= (l_2^5 - l_1^5)/5 \end{aligned}$$

$$\begin{aligned} \mathbf{p} &= \mathbf{A} - \mathbf{b} = (p_x, p_y, p_z), \\ p^2 &= p_x^2 + p_y^2 + p_z^2 - R^2, \\ q &= \mathbf{B}\mathbf{p}, \\ u &= \mathbf{p}\mathbf{a}, \\ v &= \mathbf{B}\mathbf{a} \end{aligned}$$

References

- [1] Beier T (1990) Practical Uses for Implicit Surfaces in Animation. SIGGRAPH Course 23: 20.1–20.11
- [2] Blanc C, Schlick C (1995) Extended Field Functions for Soft Objects. Proc Implicit Surfaces: 21–32
- [3] Blinn J (1982) A Generalization of Algebraic Surface Drawing. ACM TOG 1(3): 235–256
- [4] Bloomenthal J (1985) Modeling the Mighty Maple. Proc ACM SIGGRAPH 19(3): 305–311
- [5] Bloomenthal J, Shoemake K (1991) Convolution Surfaces. Proc ACM SIGGRAPH 25(4): 251–257
- [6] Bloomenthal J (1995) Skeletal Design of Natural Forms. Doctoral dissertation, University of Calgary, Dept Computer Science
- [7] Bloomenthal J (1997) editor, Introduction to Implicit Surfaces. Morgan Kaufmann Inc
- [8] Bloomenthal J (1997a) Bulge Elimination in Convolution Surfaces. Computer Graphics Forum 16(1): 31–41
- [9] Buchanan J, Turner P (1992) Numerical Methods and Analysis. McGraw-Hill Inc
- [10] McCormack J, Sherstyuk A (1998) Creating and Rendering Convolution Surfaces. Computer Graphics Forum, to appear
- [11] Nishimura H, Hirai M, Kawai T, Kawata T, Shirakawa I, Omura K (1985) Object Modelling by Distribution Function and a Method of Image Generation. The Transactions of the Institute of Electronics and Communication Engineers of Japan J68-D(4): 718–725 (in Japanese; English translation by Takao Fujiwara, Advanced Studies in Computer Aided Art and Design, Middlesex Polytechnic, England, 1989)
- [12] Sealy G, Wyvill G (1996) Smoothing of three dimensional models by convolution. Proc Computer Graphics International: 184–190
- [13] Wolfram S (1996) The Mathematica Book. Third edition, Wolfram Media Inc and Cambridge University Press
- [14] Wyvill G, McPheeters C, Wyvill B (1986) Data Structure for Soft Objects. The Visual Computer 2(4): 227–234
- [15] Wyvill B, Wyvill G (1989) Field functions for implicit surfaces. The Visual Computer 5(1/2): 75–82
- [16] Wyvill B, van Overveld K (1996) Tiling Techniques for Implicit Skeletal Models. SIGGRAPH Course 11: C1.1–C1.26