Combining CSG modeling with soft blending using Lipschitz-based implicit surfaces

Daniel Dekkers, Kees van Overveld, Rob Golsteijn

October 29, 1996

Abstract

In this paper a general method is given for combining CSG modeling with soft blending using implicit surfaces. A class of various blending functions sharing some desirable properties like differentiability and intuitive blend control are given. The functions defining the CSG objects satisfy the *Lipschitz* condition which gives the possibility of fast root-finding, but can also prove useful in the field of collision detection and adaptive triangulation.

Introduction

Methods for defining smooth surfaces can be divided into two categories:

- **Parametric functions**
  Parametric functions are functions of the form $f(u,v) = (f_x(u,v), f_y(u,v), f_z(u,v))$. Typical examples are Bezier or B-spline patches (See [Böhm84] for an overview). The surface is defined by control points and the surface can be adjusted by moving the control points. The surface can be rendered by evaluating the function for different well-chosen values of u and v.

- **Implicit functions**
  Implicit functions have the form $f(p) = k$ and split space into two half spaces. One for which $f(p) \geq k$ and one for which $f(p) < k$. The points for which $f(p) = k$ are called iso-surfaces (for value k). We introduce the term *zero-surface* for the collection of points for which $f(p) = 0$.

An example showing a parametric function and an implicit function defining the same object (a sphere with center m and radius r) is given by Bloomenthal [Bloo88]:

Parametric:
$f(\theta,\phi) = (m_x + rsin(\theta)cos(\phi),\ m_y + rsin(\theta)sin(\phi),\ m_z + rcos(\theta))$,
$\theta \in [0,\pi],\ \phi \in [0,2\pi)$
Implicit:
$f(p) = r - ||p - m|| = r - \sqrt{(p_x - m_x)^2 + (p_y - m_y)^2 + (p_z - m_z)^2}$
where $f(p) = 0$ defines the iso-surface (i.e. $k = 0$)

Parametric functions have been studied in great detail, until recently implicit functions have received less attention.

The blending functions proposed in this paper are inspired by the blobby objects (or soft objects) introduced by Wyvill and McPheeters (see [Wyvill90] for an overview). Blobby objects originated from molecule models: implicit functions were used to visualize electron clouds [Blin82]. Wyvill and McPheeters use the analogy of spatial temperature distributions: if one moves away from a

heat source, the temperature drops. In [Wyvill90], this decline in temperature is represented by a function:

$$f(p) = 1 - \frac{||p - m||^2}{maxdist^2}$$

It has a maximum value of 1 at the heat source m and a minimum value 0 at a threshold distance $maxdist$ (points for which $||p - m|| > maxdist$ have value 0 as well). Different heat sources can be combined into one temperature distribution by adding the individual functions $f_1(p)$, $f_2(p)$, etc. By introducing an iso-surface value k between 0 and 1, a surface is obtained representing points with the same temperature value. When the heat sources are far apart, this surface will consist of separate spheres. When two heat sources move closer together, the two spheres will gradually blend together resulting in one single sphere when $m_1 = m_2$ (figure 1).

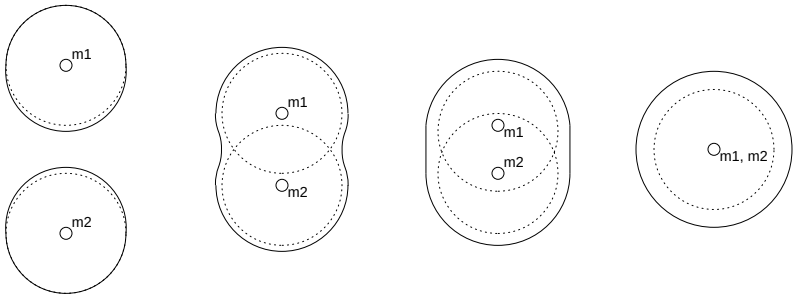


Figure 1: *Blobby spheres.*

The functions introduced in this paper do not follow the heat source analogy. Instead of adding and subtracting potential values to form the compound object, we focus on the CSG combination methods resulting in a combination method which is based on maximum and minimum operators. A second important difference with blobby objects is the interpretation of the function value. For blobby objects this interpretation concerns 'heat', in our method we interpret the function value as 'distance'. This notion of distance gives valuable information about the position of the surface while evaluating different points. As a consequence, the term 'potential' is misplaced, but will still be used due to historical reasons.

By introducing the blending functions presented in this paper we aim to fill a gap between work done by A. Pasco and J. Hart. Hart has convincingly shown that the Lipschitz property is very desirable for efficient sampling of iso-surfaces [Hart93]. Pasco introduced R-functions as a framework for creating objects from subobjects using CSG based operations [Pasko95]. Although R-functions show great 'expressive power', the exact form of the functions is not argumented and seems to be based on aestheticly pleasing results. Furthermore, the R-functions do not possess the Lipschitz property.

1 Primitive and compound objects

In this paper, we propose the construction of compound objects such that both the Lipschitz property and a large expressive power are obtained. For a primitive object, the potential in a point p denotes the signed minimal distance to the primitive object. For compound objects, this exact distance interpretation cannot be held (unfortunately) due to the influence of soft blending. But even for compound objects, the potential will still give distance information: if the potential in a point p with respect to a compound object is equal to pot then the distance from p to the compound object will be at least $|pot|$, and not very much larger than $|pot|$.

In this section, we introduce the primitive functions that we will be using: these will be skeletal objects. The techniques used to apply the standard CSG operations to the primitive objects will be discussed, followed by a generalization of the standard CSG operations, allowing soft blending.

1.1 Primitive objects: skeletals

The primitive objects are the 'building blocks' used for creating the compound objects. A large number of commonly used geometrical shapes can be defined by so called 'skeletal objects' [Wyvill86a, Bloo88]. These objects can be interpreted as a sweep of a sphere along a predefined 'skeletal shape'. For the variant used in this paper, the function value $f(p)$ denotes the minimal distance from p to the objects surface. From now on, we will always assume the zero-surface to be the objects boundary. Following the convention in recent implicit surface literature, the sign of $f(p)$ is negative when p is outside the object and positive when p is inside the object. A sphere is defined by a point and a radius, a rounded cylinder by two points and a radius, a torus by a circle and a radius, etc. More complex basic shapes like splines or Bezier-patches can be defined similarly.

- The implicit object sphere from the introduction is an example of a skeletal object. It represents a sphere with center point m and radius r (see figure 2):

$$f(p) \;=\; r - ||p - m||$$

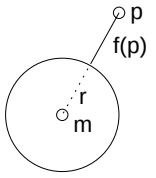


Figure 2: *Skeletal object: sphere.*

- A second example is the rounded cylinder. Calculation of the potential value $f(p)$ is done in two steps. First, point p' is calculated by projecting p on the line segment (m_1, m_2) (see figure 3). After this step, the function describing the sphere from the first example can be used (with center point p' and radius r).

1.2 Compound objects: combining CSG with soft blending

Following Ricci [Ricc73], minimum and maximum operations can be used to construct functions that perform the standard CSG set operations *union*, *intersection* and *difference*. A new class of objects is formed whose members are called *compound* objects.

If f_1 en f_2 are implicit functions representing two objects we can define the operators *union*, *intersection* and *difference* by:

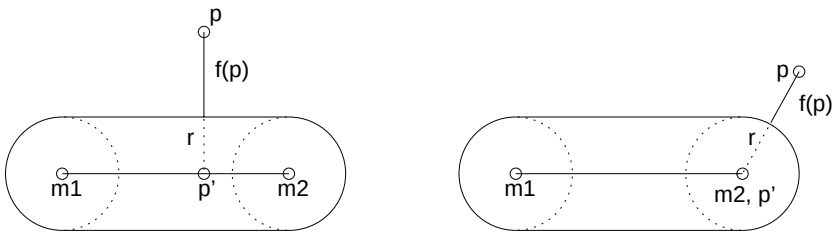


Figure 3: *Skeletal object: rounded cylinder.*

$$\begin{array}{lll} \textit{union:} & f_\cup(f_1, f_2) & := \; max(f_1, f_2) \\ \textit{intersection:} & f_\cap(f_1, f_2) & := \; min(f_1, f_2) \end{array}$$

Note that $f_\cap(f_1, f_2) = -f_\cup(-f_1, -f_2)$. The difference operator can be expressed in terms of a negation and an intersection:

$$\begin{array}{lll} \textit{difference:} & f_-(f_1, f_2) & := \; f_\cap(f_1, -f_2) \\ & & = \; min(f_1, -f_2) \end{array}$$

Figure 4 shows an example of the union, intersection and difference operator aplied to two spheres. The figures on the left show the potential as a function of position along a ray through the centers of two spheres. The figures on the right show a top view of the resulting compound objects.

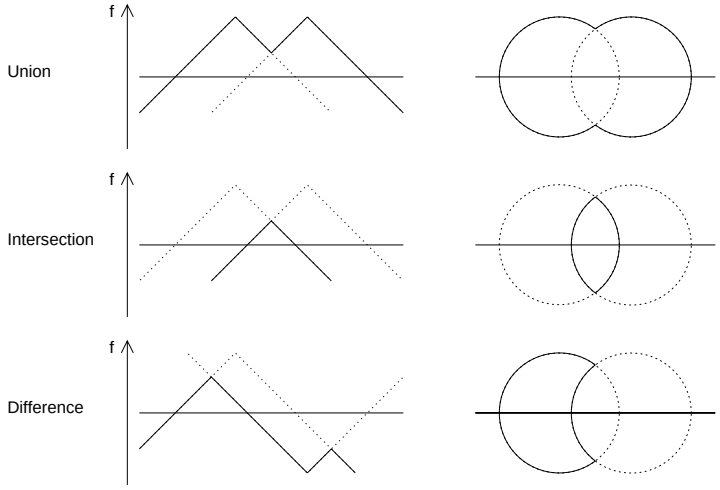


Figure 4: *Set operations (standard).*

The standard set operations give sharp edges at the intersection curves of primitive objects. One goal is to soften these intersection curves by adding and substracting 'material' at these points. In geometric modeling, this proces is known as *blending*. We do so by introducing a function f_b which represents the amount of blend added or substracted. For the union and the intersection operation, f_b depends on the difference in potential between the two subobjects. The second parameter, n, is a user-adjustable non-negative real and directly represents the amount of blending; $n = 0$ will correspond to the absence of blending.

The definitions of the generalized set operators are:

$$
\begin{array}{lll}
\textit{gen. union:} & f^*_\cup(f_1,f_2) & := \; max(f_1,f_2) + f_b(|f_1 - f_2|, n) \\
\textit{gen. intersection:} & f^*_\cap(f_1,f_2) & := \; -f^*_\cup(-f_1,-f_2) \\
 & & = \; -max(-f_1,-f_2) - f_b(|-f_1+f_2|, n) \\
 & & = \; min(f_1,f_2) - f_b(|f_1 - f_2|, n)
\end{array}
$$

Again, the difference is expressed in a negation and an intersection:

$$
\begin{array}{lll}
\textit{gen. difference:} & f^*_/(f_1,f_2) & := \; f^*_\cap(f_1,-f_2) \\
 & & = \; min(f_1,-f_2) - f_b(|-f_2-f_1|, n)
\end{array}
$$

To simplify discussion and to guide intuition about the desired properties of the generalized set operations, we will first give a qualitive sketch of a possible blending function (figure 5) and the results of the generalized set operators (figure 6).

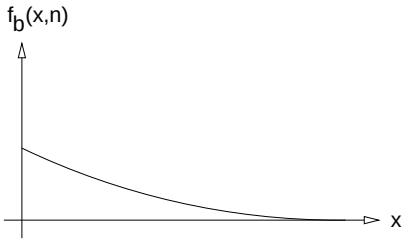


Figure 5: *The function f_b.*

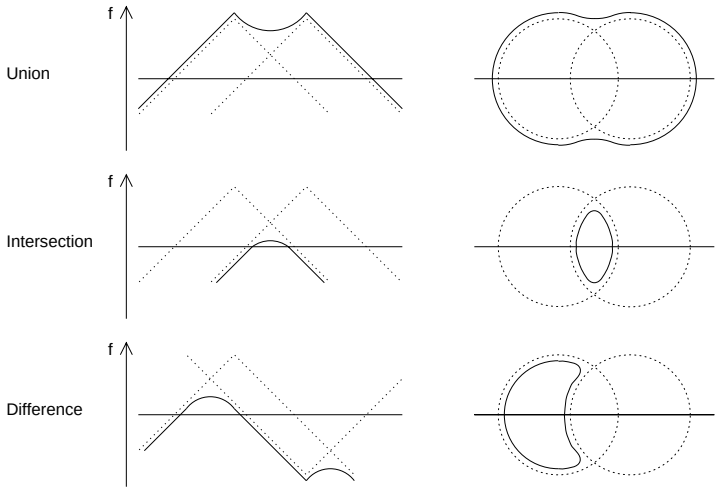


Figure 6: *Set operations (generalized).*

In the sequel, we use f_i (e.g. f_1, f_2) to denote subobjects that are combined with one of the generalized set operators into a compound function f.

2 Properties of the compound objects

In this section, four important properties of the blending functions will be summarized. Namely the Lipschitz condition, differentiability, locality and intuitive blend control.

2.1 The *Lipschitz* condition

One of the main disadvantages of implicit surface rendering is the fact that points on the iso-surface can not be generated directly (as with parametric functions). Finding the exact position of the iso-surface often works via sampling, involving numerous function evaluations. A single function evaluation will only give local information, that is: a classification of the point as being an interior, exterior or boundary point.

Lipschitz-based implicit functions are functions where the rate of growth or decline of the function value is bounded. For three dimensional functions, the formal definition is given by:

$$\forall p, q \in R^3 : |f(p) - f(q)| \leq \lambda ||p - q||, \quad \text{where } \lambda \text{ is a parameter characterizing } f$$

The fact that the Lipschitz condition is usefull can be seen from the following observation: An evaluation of a function in point p will result in a potential value $f(p)$, giving local information about the position of p with respect to the object defined by f. But due to Lipschitz, statements about the environment of p can be made as well. If $f(p)$ is negative, all points q that are closer to p than $|f(p)|/\lambda$ will be negative as well. This implies that the sphere with radius $|f(p)|/\lambda$ around p is guarenteed to be 'empty' (that is: the zero-surface does not intersect with this sphere).
For the functions presented in this paper, the aim will be Lipschitz for $\lambda = 1$ [1]. As proven in appendix A, a sufficient condition for f_b is:

$$-1 \leq f_b'(x, n) \leq 0, \quad \text{for all } x \geq 0,\ n \geq 0$$

When a Lipschitz property holds, we can apply *Sphere Tracing* [Hart93]. Sphere Tracing is a technique for finding the first intersection point between a ray and an implicit surface. Lipschitz gives a reliable method for finding the first intersection point. When moving allong a ray, we can take a step of magnitude $|f(p)|$ without risking a 'boundary cross' (see figure 7). If an intersection point exists, it will be better approximated with every jump. If it does not exist the points will move to infinity without ever reaching a potential close to zero. Special care must be taken to prevent this from happening.

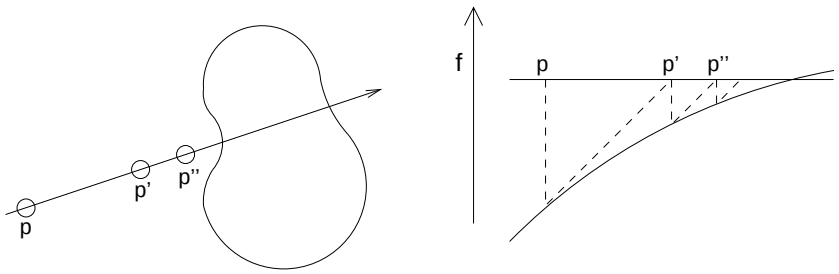


Figure 7: *Sphere Tracing.*

[1] Other values for λ are atainable straightforwardly; however, the value 1 allows the interpretation of potential-values as bieing distances.

These observations give rise to the following algorithm:

```
function Intersect(e: Vector, V: Vector): Vector;
{ pre: ||V||=1, V is the direction of the ray and e is the starting point }
|[  var  p: Vector;
         pot, λ: Real;
    λ:=0;
    pot:=f(e+λV);
    do (pot< −ε ∧ λ <outofscene) →
        λ:=λ+|pot|;
        pot:=f(e+λV)
    od ;
    if  λ <outofscene → return e+λV
    []  λ ≥outofscene → return 'non-existent'
    fi
]|
```

2.2 Differentiability

For lighting calculations, among other things, it is necessary to compute the surface normal in arbitrary points on the zero-surface. The surface normal equals the normalized negated gradient vector in a specific point. It follows that the gradient vector should be computable for points on the objects boundary. We might be tempted to think, that for primitive objects, it is sufficient to be able to calculate the gradient vectors solely on the zero-surface. There are two reasons why this is not the case. When combining primitive objects, the zero-surface of the compound object will deviate from the zero-surfaces of the subobjects. When p is a point on the compound object's zero-surface, the gradient $\nabla f(p)$ will depend on the gradient vectors $\nabla f_1(p)$ and $\nabla f_2(p)$. In general, p will not lie on the zero-surfaces of f_1 or f_2. A second reason is given by the fact that in general reliable information about the position of the zero-surface does not exist, and function evaluations in different points are used to approximate it. The gradient vectors in interior and exterior regions can (in combination with the potential value) be a valuable aid in finding the zero-surface.

The defining functions of the primitive objects must be chosen in such a way that they are in all points (except unavoidable critical points) differentiable. As an example, we look at the sphere again defined by the function:

$$f(p) = r - ||p - m||$$

The gradient vector is given by

$$\nabla f \;=\; \langle \tfrac{\partial f}{\partial x}, \tfrac{\partial f}{\partial y}, \tfrac{\partial f}{\partial z} \rangle \;=\; \frac{1}{||p-m||} \langle m_x - p_x, m_y - p_y, m_z - p_z \rangle$$

As can be seen from the formula mentioned above, the gradient vector $\nabla f(m)$ is undefined. In general this is the case for all points on the underlying skeleton of a skeletal object.

We propose to set $\nabla f = \langle 0, 0, 0 \rangle$ in these critical points, as the direction of the nul-vector is undefined as well. Moreover, this choice gives correct results in configurations like the one in figure 8. Here, m_2 is a critical point of f_2, while the nearby point p is regular. When we compute the gradient of the intersection of the two primitives (using the formulas given at the end of this section) we find that in both m_2 and p, the correct gradients are found.

The normal vector $-\nabla f / ||\nabla f||$ on any compound object can be found analytically during evaluation of the function. The gradient can be written as a function of the partial derivatives of the composing subobjects.

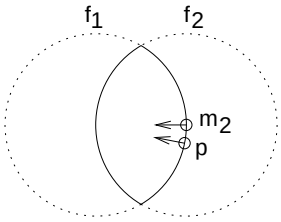


Figure 8: *critical points on the zero-surface.*

- Union, $f = f^*_{\cup}(f_1, f_2)$

$$\begin{array}{ll} \nabla f = \nabla f_1 + f'_b(f_1 - f_2, n)(\nabla f_1 - \nabla f_2) & \text{if } f_1 \geq f_2 \\ \nabla f = \nabla f_2 + f'_b(f_2 - f_1, n)(\nabla f_2 - \nabla f_1) & \text{if } f_1 \leq f_2 \end{array}$$

- Intersection, $f = f^*_{\cap}(f_1, f_2)$

$$\begin{array}{ll} \nabla f = \nabla f_2 - f'_b(f_1 - f_2, n)(\nabla f_1 - \nabla f_2) & \text{if } f_1 \geq f_2 \\ \nabla f = \nabla f_1 - f'_b(f_2 - f_1, n)(\nabla f_2 - \nabla f_1) & \text{if } f_1 \leq f_2 \end{array}$$

- Difference, $f = f^*_{/}(f_1, f_2)$

$$\begin{array}{ll} \nabla f = -\nabla f_2 - f'_b(f_1 + f_2, n)(\nabla f_1 + \nabla f_2) & \text{if } f_1 \geq -f_2 \\ \nabla f = \nabla f_1 - f'_b(-f_1 - f_2, n)(-\nabla f_1 - \nabla f_2) & \text{if } f_1 \leq -f_2 \end{array}$$

When adding the blendfunction f_b to the standard CSG operations we must first note that compound objects combined with the standard CSG operations will, in general, produce discontinuities in the first derivatives on all the intersection curves of the subobjects (i.e. in all points p where $f_1(p) = f_2(p)$). The main goal of blending is to soften these intersection lines, that is: to remove these discontinuities. When the generalized set operations are used, differentiability is guaranteed in points p where $f_1(p) = f_2(p)$ if:

$$f'_b(0, n) = -\frac{1}{2}, \quad \text{for all } n \geq 0$$

(The proof is given in appendix B)

Note the fact that since $-1 \leq f'_b \leq 0$, the gradient vector on the composite object forms a convex combination with respect to the gradient vectors of the two subobjects. For the union and intersection operator, it is a convex combination of ∇f_1 and ∇f_2, while for the difference operator, it is a convex combination of ∇f_1 and $-\nabla f_2$. As a result, the gradient vector on any compound object is bounded by the convex hull, spanned by the (sometimes negated) gradient vectors on all the primitive objects.

2.3 Locality

It is desirable to have a limited domain of influence for each primitive object. Objects that are far apart should not have any influence on each other. As a consequence, optimalisations in the evaluation of compound functions are possible, decreasing potential and gradient evaluation time. Locality is obtained when f_b equals 0 after some threshold potential difference k. The function f_b should be C1 continuous in k, to obtain a smooth connection of blended and non-blended surface regions.

2.4 Intuitive blend control

The parameter n gives control to the user concerning the amount of blend used in a combination step. A larger value of n should result in more added material when the union operator is used and in more subtracted material when the intersection or difference operator is used. Formally, the following should hold:

$$n > m \Rightarrow f_b(x, n) > f_b(x, m)$$

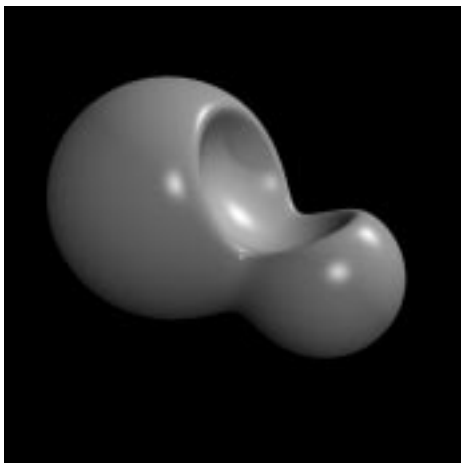
For $n = 0$, no blend should be added or subtracted, resulting in functions describing the standard CSG operations where $f_b(x, 0) = 0$ for all x. This function conflicts with the differentiability constraint stating $f_b(0, n) = -1/2$ as it should, since standard CSG is not differentiable. The standard CSG operations can be approximated by the differentiable generalized CSG operations by using an arbitrarily small value for n.

2.5 Some examples of blending functions

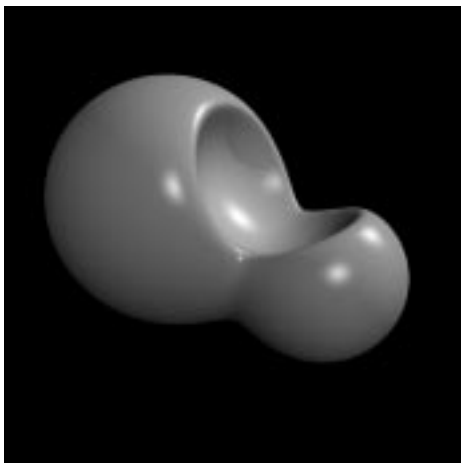
We will give six examples of different blending functions. Only three of them satisfy all of the above mentioned constraints. The others are given to provide some understanding on the effects of different types of blending functions. The first two functions give aesthetically pleasing results, but are non-local. The second is prefered to the first, since the square root can be calculated in advance. The third function shows the effect of 'over-blending'. The fourth and fifth function produce nice, clean blends. The last function demonstrates the effect of a non-differentiable blending function.

Function	Differentiability	Lipschitz	Locality	Intuitive blend control
1	×	×		×
2	×	×		×
3	×	×	×	×
4	×	×	×	×
5	×	×	×	×
6		×	×	×

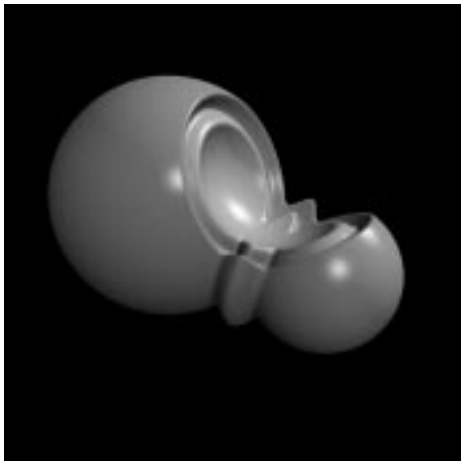
1. $f_b(x, n) = \frac{\sqrt{x^2+n}-x}{2}$
2. $f_b(x, n) = \frac{n}{x+\sqrt{2n}}$
3. $f_b(x, n) = \begin{cases} n\cos(\frac{\pi}{6} + \frac{x}{n}) + n & \text{for } x < \frac{5\pi n}{6} \\ 0 & \text{for } x \geq \frac{5\pi n}{6} \end{cases}$
4. $f_b(x, n) = \begin{cases} n\cos(\frac{5\pi}{6} + \frac{x}{n}) + n & \text{for } x < \frac{\pi n}{6} \\ 0 & \text{for } x \geq \frac{\pi n}{6} \end{cases}$
5. $f_b(x, n) = \begin{cases} n\left(\frac{x}{n} - \frac{1}{4}\right)^2 & \text{for } x < \frac{n}{4} \\ 0 & \text{for } x \geq \frac{n}{4} \end{cases}$
6. $f_b(x, n) = \begin{cases} -\frac{1}{2}x + n & \text{for } x < 2n \\ 0 & \text{for } x \geq 2n \end{cases}$



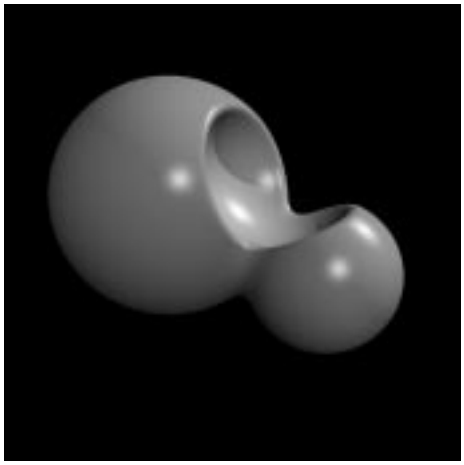
Function 1



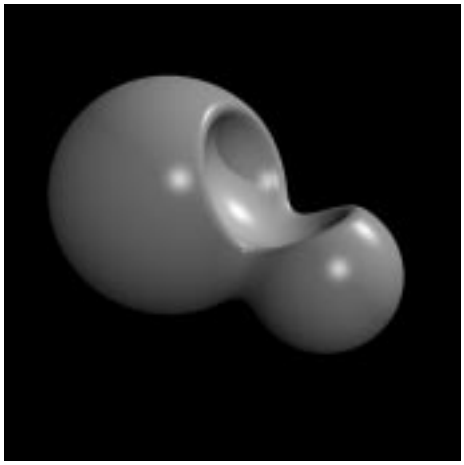
Function 2



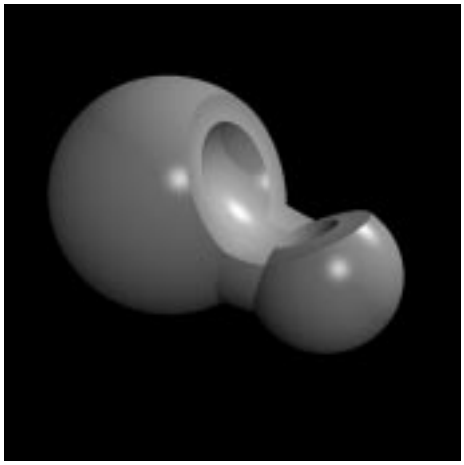
Function 3



Function 4



Function 5



Function 6

3 Conclusions

In this paper we have constructed a framework to describe a large class of complex objects using implicit functions. The combination method to form compound objects from primitive ones is based on CSG operations. The CSG operations have proven to offer a good modeling tool in practice. A generalization of the standard CSG operations was given, allowing variable amounts of blending. Four desirable properties were described and carefully translated into constraints on the generalized CSG operations. With these functions, CSG objects with controllable blending can be obtained where we can benefit from the Lipschitz condition to allow rapid sampling.

Acknowledgements

The authors wish to thank Dick Groot and Jan Bruijns for their important contributions to this work. The authors would further like to thank Jan Bruijns, Miriam Egas, Walter Lewin, and Patrick Meyers for their valuable comments after having read draft versions of this work in various states of completion.

References

[Blin82] Blinn, J., A generalization of algebraic surface drawing, ACM Trans. Graph., Vol. 1, pp. 234, 1982

[Bloo88] Bloomenthal, J., Techniques for implicit modeling, Xerox PARC technical report P89-00106 (November 1988)

[Böhm84] Böhm, W. et al., A survey of curve and surface methods in CAGD, Computer Aided Geometric Design, vol. 1, No. 1, 1984, pp. 1-60.

[Hart93] Hart, J., Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. To appear: The Visual Computer. An earlier version appeared in the SIGGRAPH '93 course notes as WSU Technical Report EECS-93-015

[Pasko95] Pasko, A., Adzhiev V., Sourin A., Savchenko V., Function representation in geometric modeling: concepts, implementation and applications, The Visual Computer, vol. 11, No. 8, 1995, pp. 429-446.

[Ricc73] Ricci, A., Constructive geometry for computer graphics, Comput. J. (GB), Vol. 16, pp. 157-160, 1973.

[Wyvill86a] Wyvill, B., McPheeters, C., and Wyvill, B., Data Structure for Soft Objects, Visual Computer, 2, 4 (August 1986a), pp. 227-234

[Wyvill90] Wyvill, B., A Computer Animation Tutorial, Springer-Verlag, New York 1990

A Lipschitz

The proof obligation is:

$$\forall p, q \in R^3 : |f(p) - f(q)| \leq ||p - q||$$

We will prove this by induction on the structure of the tree defining the compound object.
To make the formulas manageable we apply the following abbreviations:

$$\begin{array}{ll} f_1(p) & = a \\ f_1(q) & = b \\ f_2(p) & = c \\ f_2(q) & = d \\ ||p-q|| & = l \end{array}$$

The induction hypothesis is:

$$\forall p, q \in R^3 : |f_1(p) - f_1(q)| \leq ||p - q||$$
$$\forall p, q \in R^3 : |f_2(p) - f_2(q)| \leq ||p - q||$$

Using the abbreviations:

$$|a - b| \leq l$$
$$|c - d| \leq l$$

This directly implies:

$$|max(a, c) - max(b, d)| \leq l \quad (1)$$
$$|min(a, c) - min(b, d)| \leq l \quad (2)$$

We will show that the following rules (which are implied by $-1 \leq f_b' \leq 0$) are sufficient for Lipschitz:

$$x \geq y \;\equiv\; f_b(x, n) \leq f_b(y, n) \quad (3)$$
$$x \geq y \;\equiv\; f_b(x, n) \geq f_b(y, n) \;-\; (x - y) \quad (4)$$

For the base step of the induction we note that the skeletal primitives are defined in terms of an exact (signed) distance property: $f(p)$ denotes the minimal distance from p to the zero-surface of f. This directly implies that Lipschitz holds for these skeletal elements.

For the induction step we differentiate between the three possible combination functions: union, intersection and difference.

Union

$$\begin{array}{ll} & |f(p) - f(q)| \leq l \\ \equiv & \{\ f = f^*_{\cup}(f_1, f_2)\ \} \\ & |\,[max(a, c) + f_b(|a - c|, n)] - [max(b, d) + f_b(|b - d|, n)]\,| \leq l \\ \equiv & \\ & -l \leq max(a, c) - max(b, d) + f_b(|a - c|, n) - f_b(|b - d|, n) \leq l \end{array}$$

- Consider the upper bound:

$max(a,c) - max(b,d) + f_b(|a-c|,n) - f_b(|b-d|,n) \leq l$
$\Leftarrow$ { • Case 1: Assume $|a-c| \geq |b-d|$, from (3) follows:
$f_b(|a-c|,n) \leq f_b(|b-d|,n)$
}
$max(a,c) - max(b,d) \leq l$
$\Leftarrow$ { From (1) }
Induction hypothesis
$\Leftarrow$ { • Case 2: Assume $|a-c| < |b-d|$, from (4) follows:
$f_b(|a-c|,n) - f_b(|b-d|,n) \leq |b-d| - |a-c|$
}
$max(a,c) - max(b,d) + |b-d| - |a-c| \leq l$
$\equiv$
$min(a,c) - min(b,d) \leq l$
$\Leftarrow$ { From (2) }
Induction hypothesis

- Consider the lower bound:

$-l \leq max(a,c) - max(b,d) + f_b(|a-c|,n) - f_b(|b-d|,n)$
$\Leftarrow$ { • Case 1: Assume $|a-c| \geq |b-d|$, from (4) follows:
$f_b(|a-c|,n) - f_b(|b-d|,n) \geq |b-d| - |a-c|$ }
$-l \leq max(a,c) - max(b,d) + |b-d| - |a-c|$
$\equiv$
$-l \leq min(a,c) - min(b,d)$
$\Leftarrow$ { From (2) }
Induction hypothesis
$\Leftarrow$ { • Case 2: Assume $|a-c| < |b-d|$, from (3) follows:
$f_b(|a-c|,n) > f_b(|b-d|,n)$ }
$-l \leq max(a,c) - max(b,d)$
$\Leftarrow$ { From (1) }
Induction hypothesis

Intersection

The intersection operator is defined in terms of negation and union operations (see figure 9): $f^*_{\cap}(f_1, f_2) = -f^*_{\cup}(-f_1, -f_2)$

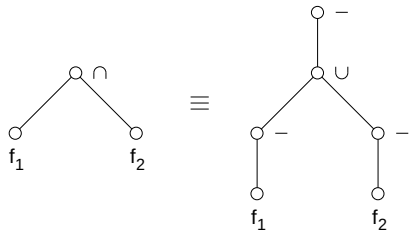


Figure 9: *Intersection expressed in union and negation operations.*

Since we have proven that the union operator does not violate the Lipschitz condition, it is sufficient to show that negation preserves Lipschitz as well.

$|f(p) - f(q)| \leq l$
$\equiv\ -l \leq f(p) - f(q) \leq l$
$\equiv\ l \geq -(f(p) - f(q)) \geq -l$
$\equiv\ |-f(p) + f(q)| \leq l$

Difference

The difference operator does not violate the Lipschitz condition since it is defined as the composition of a negation and an intersection operator (see figure 10): $f^*_{/}(f_1, f_2) = f^*_{\cap}(f_1, -f_2)$.

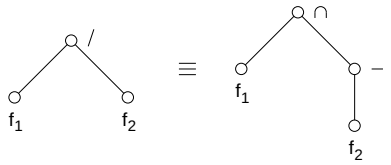


Figure 10: *Difference expressed in intersection and negation operations.*

B Differentiability

From the definition of the gradient vectors for union and intersection objects (see section 2.2), it is clear that the gradient is continuous if and only if the gradient for $f_1 \geq f_2$ is equal to the gradient for $f_1 \leq f_2$ in points p where $f_1(p) = f_2(p)$.

Union

$$
\begin{aligned}
& \nabla f_1(p) + f'_b(f_1(p) - f_2(p), n)[\nabla f_1(p) - \nabla f_2(p)] = \\
& \nabla f_2(p) + f'_b(f_2(p) - f_1(p), n)[\nabla f_2(p) - \nabla f_1(p)] \\
\equiv\ & \{ \text{ Calculus, } f_1(p) = f_2(p) \ \} \\
& 2f'_b(0,n)\nabla f_1(p) - 2f'_b(0,n)\nabla f_2(p) = \nabla f_2(p) - \nabla f_1(p) \\
\equiv\ & \{ \text{ Calculus } \} \\
& 2f'_b(0,n)(\nabla f_1(p) - \nabla f_2(p)) = \nabla f_2(p) - \nabla f_1(p) \\
\equiv\ & \{ \text{ Calculus } \} \\
& 2f'_b(0,n) = -1 \\
\equiv\ & \{ \text{ Calculus } \} \\
& f'_b(0,n) = -\frac{1}{2}
\end{aligned}
$$

Similar proofs can be given for the intersection and difference operator. Note that for the difference operators, points p where $f_1(p) = -f_2(p)$ must be considered.

C The resulting potential evaluation function

The compound object is stored in a binary tree. The leaves contain the primitive objects, the nodes contain a set operator and a blending factor. A non-zero blending factor defines a soft set operator, a zero blending factor defines a hard set operator. The function $EvalField$ returns the potential with respect to a primitive or compound object at a given point p. As a side-effect, the gradient vector in p is calculated and returned in ∇.

```
function EvalField(node: TreeType; p: Vector; var ∇: Vector): Real
|[ var pot_l, pot_r: Real;
       ∇_l, ∇_r: Vector;
   if node.objtype=primitive →
      pot := 'the signed minimal distance from p to the primitive object';
      ∇ := 'the gradient vector in p (⟨0,0,0⟩ for a stationary point).'
   [] node.objtype=compound →
```

```
        pot_l := EvalField(node.lefttree, p, ∇_l);
        pot_r := EvalField(node.righttree, p, ∇_r);
        if node.op=union →
            if pot_l > pot_r →
                pot := pot_l + f_b(pot_l − pot_r, node.blend);
                ∇ := ∇_l + f'_b(pot_l − pot_r, node.blend)(∇_l − ∇_r);
            [] pot_l ≤ pot_r →
                pot := pot_r + f_b(pot_r − pot_l, node.blend);
                ∇ := ∇_r + f'_b(pot_r − pot_l, node.blend)(∇_r − ∇_l)
            fi
        [] node.op=intersection →
            if pot_l > pot_r →
                pot := pot_r − f_b(pot_l − pot_r, node.blend);
                ∇ := ∇_r − f'_b(pot_l − pot_r, node.blend)(∇_l − ∇_r);
            [] pot_l ≤ pot_r →
                pot := pot_l − f_b(pot_r − pot_l, node.blend);
                ∇ := ∇_l − f'_b(pot_r − pot_l, node.blend)(∇_r − ∇_l)
            fi
        [] node.op=difference →
            if pot_l > −pot_r →
                pot := −pot_r − f_b(pot_l + pot_r, node.blend);
                ∇ := −∇_r − f'_b(pot_l +pot_r, node.blend)(∇_l + ∇_r);
            [] pot_l ≤ −pot_r →
                pot := pot_l − f_b(−pot_l − pot_r, node.blend);
                ∇ := ∇_l − f'_b(−pot_l − pot_r, node.blend)(−∇_l − ∇_r)
            fi
        fi
    fi
]|
```