

Quasi-Convolutional Smoothing of Polyhedra

Richard J. Lobb

Department of Computer Science
University of Auckland, New Zealand.
[richard@cs.auckland.ac.nz]

ABSTRACT

This paper describes a system for the representation and rendering of polyhedral scenes in which individual components can have different user-specified amounts of rounding applied to edges and corners. Objects are represented by structures similar to CSG trees but with arithmetic operators at internal nodes rather than set membership operators. An object's smoothing attribute specifies the radius of a spherical smoothing filter, and the smoothed object's surface is defined by an iso-density surface after low-pass filtering. The filtering is an approximation to true convolutional filtering, but allows rapid determination of iso-density surfaces. The rounded surfaces are similar to those achieved by use of filleting and surface blending techniques during modelling, but are much easier to specify, far more economical in storage, and simpler to compute. By varying the smoothing radii, a wide range of effects can be obtained, from near-perfect polyhedra through to "blobby models".

1 INTRODUCTION

A large proportion of computer graphics scenes are modelled with polyhedra, even where the natural objects being modelled are themselves curved. Smooth shading techniques readily eliminate the defects of polyhedral approximations to curved surfaces, but synthetic images usually have at least one significant flaw: edges and corners of the models remain sharp, whereas natural polyhedral objects usually have some smooth curvature connecting adjacent planes. Apart from having a generally softer appearance, curved edges and corners often exhibit bright highlights or, where they are in contact with other objects or the floor, associated shadows. Such highlights and shadows help the eye delineate the object and distinguish it from a similarly coloured background.

The computer aided geometric design field has a rich literature on the subject of blending surfaces; see for example Hoffman and Hopcroft [8] or the review by Woodward [21]. The usual goal in this field is to determine a mathematical description for a surface that smoothly blends with two or more other surfaces. Many modern CAD packages offer surface blending facilities. For example, a user may be able to select two planar surfaces, enter a blending radius, and have the system automatically construct a blending surface where the planes intersect. Such surfaces themselves need to be blended at corner points, and the problem can become mathematically intractable with some topologies. Usually, though, a solution is possible, and the CAD

system outputs a polygonized representation of the resultant surface. However, such surface blending is in general mathematically difficult, hard to implement, awkward for the user to control, and expensive in storage space for the polygonized scene description.

Colburn [4][5] takes a different approach to rounding, with what he calls a "global blending" method that works on arbitrary solid models. He regards a solid object as a classification function $f(x, y, z)$, which has value 1 at points inside the object and -1 outside it. That function can be low-pass filtered by convolving it with a smoothing filter, which has the effect of removing the high frequencies associated with edges and corners. The surface of the smoothed model is then defined to be all points at which the filtered function value is 0. Since the method used in this paper starts from essentially the same idea, it will be outlined in more detail in Section 2.1.

Colburn's application area is computer-aided manufacturing, for which he requires a polygon mesh representation of the smoothed surface. He locates surface points by tracing rays through an octree approximation to the solid, using a root finding method along each ray. Surface points are combined into a mesh structure, which is subdivided in regions of high curvature. The mesh can then be displayed using normal polygonal rendering methods, or used as input to numerically controlled machinery.

Bloomenthal and Shoemake [3] also used convolution to construct surfaces. Their goal was very different from Colburn's: they were concerned with "fleshing out" skeletons to represent complex smooth surfaces, such as of a human limb. They describe an ingenious fast rendering scheme that exploits the separability property of a gaussian filter to allow them to render images by compositing convolution images of individual components of the skeleton. A major restriction, however, is that the skeleton components have to be planar, though the resulting surfaces are decidedly three dimensional. Their method does not create convolution surfaces around solid skeletons, can not easily represent planar surfaces, and is not suitable for general-purpose modelling and display of synthetic scenes.

Our initial goal was to find a way of rendering standard polygonal scenes with object edges and corners rounded. However, it is difficult to recreate the original topology from just the polygons, and furthermore the goal is too restrictive, because:

- (a) The rounding radius of edges in typical scenes varies by many orders of magnitude from negligible to

several millimetres: it is a property of a particular object or even of a particular edge.

- (b) Given two cuboids as in Figure 1(a), we wanted to be able to produce at least the two different roundings of Figures 1(b) and 1(c). 1(b) is appropriate for a box sitting on a table, while 1(c), with the same initial polygon topology, is appropriate for a welded machine part.

To support these two requirements more information is needed at scene construction time. Hence, the system we describe is both a scene description method and a rendering algorithm.

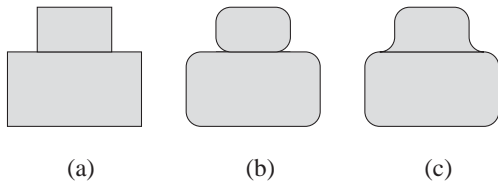


Figure 1. Two different rounding options (b and c) for the same initial polygon configuration (a).

2 SCENE DESCRIPTION

2.1 Convolutional Smoothing

The underlying principle of our smoothing method is very similar to that of Colburn's. We restrict ourselves to a polyhedral scene description and we associate with each polyhedral component a spherical smoothing filter with a given radius. The surface of the smoothed polyhedron is then defined as all points such that when a sphere of the specified radius is positioned at the point, exactly half of the volume of the sphere is inside the polyhedron and half is outside.

Figure 2 illustrates this in two dimensions. The circles are shown centred at various points on the surface of the smoothed polygon, shown shaded: half of each circle's area is inside the unsmoothed polygon.

More formally, given a smoothing filter of radius r and a polyhedral region of space P defined by

$$P: (x,y,z) \rightarrow \text{density}$$

where *density* has a value of 1 inside the polyhedron and 0 outside it¹, the surface of the smoothed polyhedron is all points (x,y,z) for which

$$\int P(u,v,w)h(x-u, y-v, z-w) du dv dw = 0.5 \quad (1)$$

where the integral is over all space, and

$$h(x,y,z) = \begin{cases} \frac{1}{4\pi r^3/3} & x^2 + y^2 + z^2 < r^2 \\ 0 & \text{otherwise} \end{cases}$$

The term *density* should not be confused with the physical density of an actual scene object; it is simply a

convenient term for the real-valued quantity of “insidedness” that can be manipulated as in equation (1).

From a signal-processing standpoint, our method involves convolving the density function by a spherical filter h , and displaying the isosurface of value 0.5. Since the filter h is a low-pass filter, the rounding of edges and corners can be seen as the result of filtering out high-frequency spatial components from the original density function.

We use a ray-tracer for rendering, so one can imagine a small sphere moving along a ray from the eye into the scene, and being “reflected” at the first point where half the volume of the sphere is inside a scene object.

In practice, two major problems occur.

- The volume of intersection of a sphere and a polyhedron is difficult to compute, and is certainly too expensive to allow equation (1) to be solved for every ray during a ray traced rendering.
- For non-convex polyhedra, the surface of the smoothed volume can lie outside the unsmoothed volume, so that it is possible for a ray that misses the unsmoothed object to hit the smoothed object. This potentially makes it difficult to know where to search for solutions to equation (1).

Before addressing these problems, it is necessary to be more explicit about our scene definition data types.

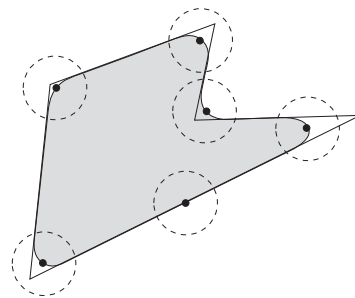


Figure 2. Convolutional smoothing of a polygon by a circular filter.

2.2 Scene Object Data Types

Figure 3 shows, in the language Clean [13][14], simplified versions of the data types we use for scene construction. The syntax will be obvious to readers who are familiar with modern functional languages such as Haskell [9] or Miranda [19], but others should be able to understand the notation with the help of some explanatory comments. The type definitions, which are indicated by a double colon, specify that a scene is just a scene object, which itself is one of

- A Union, Intersection or SetDifference of two other scene objects, or
- A Rounded (with a specified real radius value) DensityTree, to be explained shortly, or
- A Halfspace defined by a plane (and its directed normal).

A scene is thus a standard Constructive Solid Geometry tree (CSG tree) structure whose leaves are either half spaces or rounded density trees. We refer to the scene tree as the *primary CSG tree*, to distinguish it from the CSG-like structures that we will use for density trees.

¹We use values of 0 and 1 with an isosurface at 0.5, rather than the more conventional -1 and +1 with an isosurface at 0, to allow the direct use of arithmetic operators to simulate set operations, as will become clear later.

We focus on the geometry of scene objects and disregard the issues of specifying surface reflectance properties.

Rounded density trees are used to represent convolutionally smoothed polyhedra. They define a density value at any point in space, with the interpretation that a given point is inside the object if and only if the density value at that point is greater than 0.5.

```

:: Scene      =      SceneObject;
:: Radius     =      Real;
:: Plane      =      (Vector, Real);

:: SceneObject
  = Union      SceneObject SceneObject
  | Intersection SceneObject SceneObject
  | SetDifference SceneObject SceneObject
  | Rounded    Radius      DensityTree
  | Halfspace  Plane;

:: DensityTree
  = Sum        DensityTree DensityTree
  | Product    DensityTree DensityTree
  | Difference DensityTree DensityTree
  | DensityHalfspace Plane;

```

Figure 3: Scene definition data types.

A rounded density tree is a leaf of the primary CSG tree. It comprises the (unrounded) density tree together with the radius value of a convolutional smoothing filter. The density tree is like a CSG tree, but with the arithmetic operations of addition, multiplication and subtraction of other density trees used in lieu of union, intersection and set difference. The leaves of a density tree are DensityHalfspaces, defined to have a zero density on one side of a plane and unit density on the other.

Unless stated otherwise, it will be assumed that, prior to rounding, density trees define density values that are either 0 or 1 at all points. This makes density multiplication exactly equivalent to set intersection. Density addition is equivalent to set union provided the intersection of the two components is empty, and density subtraction is equivalent to set difference provided the intersection of the complement of the first component with the second component is empty.

Looking ahead somewhat, Figure 12 shows the different effects that can be produced by performing rounding at different stages during CSG-style modelling. The construction of this scene is discussed in Section 4.1.

2.3 The Density Tree

Figure 4 illustrates the meaning of the entire scene data structure by specifying, in the language Clean 1.0 [14], a point classification function PointInObject that tests whether a given point is inside a given scene object.

Function PointInObject, which implements standard CSG logic, is defined by a set of rules, exactly one of which should “pattern match” any particular invocation. For example, if the scene object is constructed from a union of a list of components, the value to return is given by:

(PointInObject p objA) Or (PointInObject p objB)

which in an imperative language would be written

PointInObject (p, objA) Or PointInObject (p, objB).

Of more interest is the case where the object being tested is a rounded density tree. In this case, the density due to the rounded object is calculated by function Density and the result compared to 0.5 to classify the point. The Density function, which takes as parameters a smoothing filter radius, a point in space and a density tree to be evaluated at that point, thus precisely defines the meaning of a rounded density tree. There are four separate cases to be handled by the Density function, the last of which if broken into three subcases by the use of “guards”: the syntax `Density a b c | d = e | f = g` should be read as “the value of the density, given parameters matching *a*, *b*, and *c*, is equal to *e* if condition *d* is satisfied, or else to *g* if condition *f* is satisfied”.

There are two ways of viewing the operation of the Density function: as an approximation to true convolutional filtering of a CSG model, or as an *exact* evaluation of a density field constructed from convolutionally-smoothed density halfspaces. The two views are complementary, and are covered in the following two sections.

2.3.1 Approximate Convolutional Filtering

In the first view, and with the assumptions of section

```

PointInObject :: Point SceneObject -> Bool;
PointInObject p (Union objA objB) = (PointInObject p objA) or (PointInObject p objB);
PointInObject p (Intersection objA objB) = (PointInObject p objA) and (PointInObject p objB);
PointInObject p (SetDifference objA objB) = (PointInObject p objA) and not (PointInObject p objB);
PointInObject p (Rounded r densityTree) = (Density r p densityTree) > 0.5;
PointInObject p (Halfspace plane) = InsidePlane p plane;

Density :: Radius Point DensityTree -> Real;
Density r p (Sum objA objB) = (Density r p objA) + (Density r p objB);
Density r p (Product objA objB) = (Density r p objA) * (Density r p objB);
Density r p (Difference objA objB) = (Density r p objA) - (Density r p objB);
Density r p (DensityHalfspace plane)
  | distance > r = 0.0
  | distance < -r = 1.0
  | otherwise = truncatedSphereVolume
where
  distance = DistanceOfPointFromPlane p plane /* In direction of plane normal */
  truncatedSphereVolume = (1 - alpha)^2 * (2 + alpha) / 4 /* volume of sphere inside halfspace */
  alpha = distance / r

```

Figure 4. Functional code for point classification and density evaluation.

2.2, the density operators are equivalent to the standard set operators and the unsmoothed object is equivalent to a standard CSG polyhedron. The Density function then has the role of performing a convolution of that polyhedron's density with the simple spherical filter as defined by the integral in equation (1).

If the density tree is a density halfspace, the integral in equation (1) is equivalent to determining what fraction of the volume of a sphere centred at (x,y,z) lies inside the halfspace. The standard solid geometry formula for the volume of a spherical cap of height h and radius r is

$V = \frac{1}{3}\pi h^2(3r - h)$. Dividing by the volume of the sphere, and substituting $h = r - d$, where d is the distance of the point (x,y,z) from the plane of the halfspace, with a positive sign outside the plane and a negative sign inside, gives the result

$$\rho(x,y,z) = \begin{cases} 0 & \alpha \geq 1 \\ 1 & \alpha \leq -1 \\ (1 - \alpha)^2(2 + \alpha)/4 & \text{otherwise} \end{cases} \quad (2)$$

where $\alpha = d/r$

If the density tree is the sum or difference of two other density trees, the integral in equation (1) equals the sum or difference of the two subtree integrals, so the Density function correctly handles Sum and Difference nodes.

However, the case of products of density trees, equivalent to set intersection, is handled by a major and possibly rather startling approximation: the convolution integral of a product of two volume density functions is obtained by multiplying the two separate convolution integrals. This would seem to return very poor results under some situations. For example, the density due to the product of two abutting antiparallel halfspaces would evaluate to 0.25 on the common plane rather than the expected answer of 0.0. All is not lost however, since both those density values are classified as outside the object, so the PointInObject function ultimately returns the right answer. The real question is: what are the properties of the 0.5 isosurface obtained using this approximation, as compared to the 0.5 isosurface obtained with true convolution? For such purposes, the approximation has the following useful properties:

- It is exactly correct at a Product node when the spherical filter lies entirely outside either of the two subtrees, since then one of the density functions is zero at all points within the filter volume and the result is zero.
- It is exactly correct at a Product node when the spherical filter lies entirely inside the unsmoothed solid volume of either of the two subtrees, since one of the two density functions then equals 1 at all points within the filter volume.
- It is exactly correct along the common edge of two perpendicular intersecting halfspaces, where the sphere has exactly one quarter of its volume inside the intersecting halfspaces and half its volume inside each halfspace.
- It is similarly correct at the common point of three mutually perpendicular intersecting halfspaces.

Properties (a) and (b), plus continuity arguments, ensure that with a radius of zero the isosurface equals the surface of the unsmoothed polyhedron, and as the radius increases from zero, the isosurface departs smoothly from the original surface. Properties (c) and (d) suggest that the approximate isosurface should be fairly close to the true convolution surface around square edges and corners.

To indicate the nature of the convolution approximation, Figure 5 shows three ways of rounding the edge generated by two intersecting halfspaces, for three different intersection angles. The heavy line is the rounded corner produced by the algorithm in this paper. The shaded region with a dotted boundary shows a cylindrical fillet with radial lines to its centre; the cylinder radius has been chosen so that the cylinder meets the halfspace planes along the same lines as the true convolution surface. The true convolution surface itself is virtually indistinguishable from the cylindrical fillet for the 90° and 135° cases, but is shown as a separate dashed line in the 30° case.

All three methods are virtually equivalent for right-angled edges. For acute angled edges the rounding produced by the algorithm in this paper is somewhat more elliptical ("pointier") than the true convolution surface which in turn is more elliptical than the cylindrical surface. For both right-angled and acute angled edges, the curved surface meets the planar surface at the same points for all three blending methods. For obtuse-angled edges, however, the region of curvature produced by our algorithm extends further than one radius from the corner, and in the limit, with a pair of identical intersecting halfspaces, the entire isosurface is inside the planes by about one quarter of the filter radius.

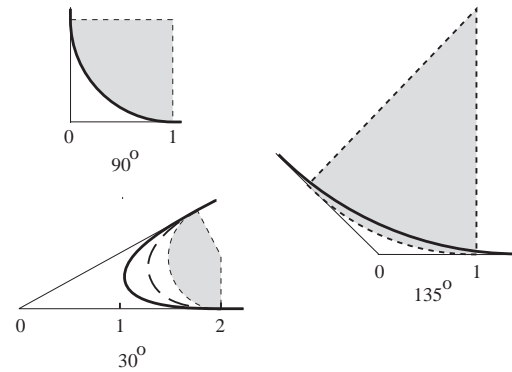


Figure 5: Rounded intersecting halfspaces at angles of 90°, 30° and 135°.

While the shapes of the rounded intersections are entirely acceptable for our purposes, the surface displacement in the case of duplicated intersecting halfspaces may seem slightly disturbing, even though the situation is improbable. The explanation is best understood in the context of the following alternative interpretation of density algorithm of Figure 4.

2.3.2 Modelling with smoothed halfspaces

Inspection of the Density function in Figure 4 shows that it is simply an arithmetic expression tree evaluator, in which the leaf nodes, which are density halfspaces,

have values that are functions of r and p . For a given fixed r , the leaf nodes represent convolutionally smoothed halfspaces whose values at a given point p determine the value of the entire arithmetic expression. The density variation due to such a halfspace, as a function of distance from the halfspace, is illustrated in Figure 6. *Our approximate method for convolutional smoothing of polyhedra is thus an **exact** process of approximating polyhedra by density arithmetic on convolutionally smoothed halfspaces.* A product of two identical halfspaces produces the density profile shown by the dashed line in Figure 6, so that displacement of the isosurface away from the plane is to be expected.

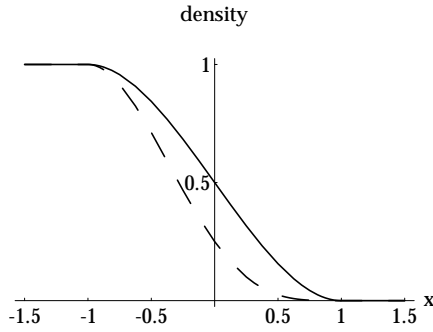


Figure 6. The density variation across a smoothed halfspace and (shown as a dashed line) the density squared.

The view of the algorithm as an approximation to convolutional smoothing is simpler and more natural for most purposes. However, some effects such as the surface displacement just discussed are explicable only in terms of modelling with smoothed halfspaces. The two views are complementary, and throughout the remainder of the paper we use the term *quasi-convolutional smoothing* to embrace both views.

2.4 The Surface Normal

For display purposes we need the surface normal, which is given by the (negative of) the direction of the density gradient $\nabla\rho$. For a smoothed density halfspace, $\nabla\rho$ is obtained by differentiating equation (2). Composite objects are all built by adding, subtracting and multiplying smoothed density halfspaces, each with a density function defined by equation (2). Hence, the density gradient due to a composite object represented as a density tree can be obtained simply by recursively applying the usual differentiation rules for arithmetic expressions.

3 RENDERING

3.1 Introduction

Our goal is to provide more natural looking scene objects for high-quality image rendering purposes, so ray-tracing is the natural choice of rendering method.

Ray traced rendering of a standard CSG object, such as our primary CSG tree, is well understood [16] [1]. Ray tracing implicitly defined surfaces is also well established, both in surface rendering [7][11] and volume rendering [12]. This discussion focuses on the only aspect of the algorithm that is new: identifying the

points where a ray crosses the 0.5 isosurfaces of a density object.

3.2 Restricting the search space

As pointed out by Kalra and Barr [11], it is impossible to reliably ray-trace a completely arbitrary implicit surface: one needs to make use of additional information about the surface in order to bound the regions in which to search for surface crossings. We use the following two facts:

- (a) The smoothed polyhedron's surface always lies within one sphere radius of the unsmoothed surface.
- (b) The filtering process which removes edges and corners also limits the maximum rate of change of the density and its gradient.

To exploit (a), we first trace rays through expanded and contracted versions of the unsmoothed density tree, as illustrated in Figure 7. The ray state variations marked (a) and (b) show the intersection of the ray with the expanded and the contracted object respectively. The combination of the two gives (c), in which isosurface crossings are confined to the shaded regions.

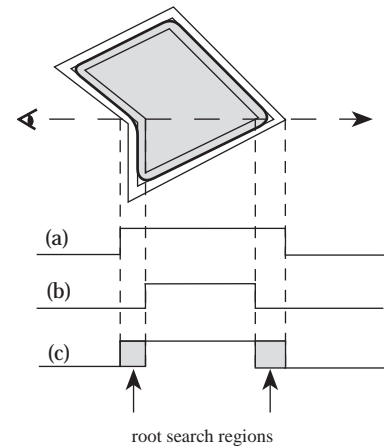


Figure 7. Identifying regions where a ray may cross the 0.5 isosurface.

The expanded and contracted objects are normal CSG models constructed from the original density tree by replacing the arithmetic operators by the normal set operators and by displacing the leaf halfspaces along their surface normals by a suitable distance.

The expansion and contraction distances must be sufficient to ensure that if a point lies inside the contracted approximation it also lies within the true rounded object, and if it lies outside the expanded approximation it also lies outside the true rounded object. If those conditions are satisfied, any isosurface crossings of the rounded object lie in regions that are outside the contracted approximation and inside the expanded approximation. Values of $+r$ and $-r$ would certainly satisfy the required conditions, but much tighter bounds are possible. For example, the isosurface of a rounded convex polyhedron always lies on or within the unrounded polyhedron, so an expansion of $+r$ would be very wasteful in this case.

3.3 Tighter Density Bounds

Let $\rho(\mathbf{x})$ be the density function of a quasi-convolutionally-smoothed object, where \mathbf{x} is the position vector of a point in space. Define an

OuterBound $O(\rho, t)$ of the density with respect to a given non-negative real-valued density threshold t to be any set with the property

$$\mathbf{x} \notin O(\rho, t) \Rightarrow \rho(\mathbf{x}) \leq t \quad (3)$$

Similarly, we can define an *Inner Bound* $I(\rho, t)$ to be any set with the property

$$\mathbf{x} \in I(\rho, t) \Rightarrow \rho(\mathbf{x}) \geq t \quad (4)$$

From these definitions we can derive, as shown in the Appendix, the following three theorems on outer bounds:

- 1) If $\rho(\mathbf{x}) = \rho_1(\mathbf{x})\rho_2(\mathbf{x})$, with $\rho_1, \rho_2 \in [0,1]$, then $S = O(\rho_1, t) \cap O(\rho_2, t)$ is an outer bound of ρ with respect to threshold t .
- 2) If $\rho(\mathbf{x}) = \rho_1(\mathbf{x}) + \rho_2(\mathbf{x})$ then $S = O(\rho_1, t/2) \cup O(\rho_2, t/2)$ is an outer bound of ρ with respect to threshold t .
- 3) If $\rho(\mathbf{x}) = \rho_1(\mathbf{x}) - \rho_2(\mathbf{x})$, with $\rho_1, \rho_2 \in [0,1]$, then $S = O(\rho_1, t) - I(\rho_2, 1-t)$ is an outer bound of ρ with respect to threshold t .

These three theorems allow us to calculate a CSG object guaranteed to include the 0.5 isosurface using the algorithm of Figure 8, with an initial threshold density of 0.5. The function *Displacement*, used in Figure 8, returns the distance of the isosurface of value t from the plane of a density halfspace that has been smoothed with a filter of radius r . In effect, *Displacement* solves the cubic equation (2) for α given ρ , as follows.

Equation (2), when α is between -1 and +1, can be rearranged to give:

$$\alpha^3 - 3\alpha + (2 - 4\rho) = 0$$

This equation has a standard trigonometric solution—see for example [20]. The root of interest in our case is the one in the range -1 to +1, given by

$$\alpha = 2 \cos\left(\frac{\varphi + 4\pi}{3}\right) \quad (5)$$

$$\text{where } \varphi = \arccos(2\rho - 1)$$

```
OuterBounds :: Real DensityTree -> SceneObject;

OuterBounds t (Product objA objB)
= Intersection (OuterBounds t objA) (OuterBounds t objB);

OuterBounds t (Sum objA objB)
= Union (OuterBounds (t/2) objA) (OuterBounds (t/2) objB);

OuterBounds t (Difference objA objB)
= SetDifference (OuterBounds t objA) (InnerBounds (1-t) objB);

OuterBounds t (DensityHalfspace (n,d))
=Halfspace (n, d+(Displacement filterRadius t))
```

Figure 8. The outer bound of a density tree for a given density threshold.

Three further theorems relating to inner bounds, given in Appendix 1, lead to a similar *InnerBounds* function for calculating a CSG object guaranteed to lie entirely within the 0.5 isosurface.

3.4 Root finding

Within the possible isosurface regions along the ray path we use a root finder to locate all points where the density given by the algorithm of Figure 4 equals 0.5. The root finding algorithm, which is given in Figure 9, is a refinement of that of Kalra and Barr [11]: it exploits the band-limited nature of the density function to control and limit recursive subdivision of the root-search region of the ray while guaranteeing that all roots are correctly identified. A ray segment that may contain roots is recursively subdivided until it can be proved to be sufficiently short that it can contain at most one root. A simple *regula falsi* root finder [15], interleaved with binary search to improve worst case performance, is then used on any segment that has endpoints with opposite signs, indicating the presence of a root.

Kalra and Barr's method is applicable to what they call *LG* functions, for which one can determine the Lipschitz bound L and the equivalent bound G on the scalar gradient of the function, both bounds being calculated over a specific ray segment. L is just the maximum absolute value of the gradient of the function (i.e., a bound on how quickly the density can change along the ray) and G is the maximum value of the gradient of the gradient (i.e., a bound on how quickly the gradient itself can change).

The following two sections explain how to calculate *LG* values over a ray segment within a density tree and how to use those values to determine whether a segment needs to be further subdivided.

```
:: RaySegment = (Real, Real)
// start and end distances along ray

:: DensityFunction = (Real -> (Real, Real))
// function from a distance along a ray to density and gradient

:: MLGFunction = (RaySegment -> (Real, Real, Real))
// function to return maximum values of density, density
// gradient, and density gradient within a ray segment

Roots :: MLGFunction DensityFunction RaySegment -> [Real];
Roots mlg f (x1, x2)
  | shortEnough && noRoot = []
  | shortEnough          = RegulaFalsiRoot f (x1,x2)
  = (Roots mlg f (x1, xmid)) ++ (Roots mlg f (xmid, x2))
where
  shortEnough = ShortEnough mlg (x1, f1, f1') (x2, f2, f2')
  noRoot      = f1 * f2 > 0.0
  (f1, f1')   = f x1
  (f2, f2')   = f x2
  xmid        = (x1 + x2)/2
```

Figure 9. The algorithm for identifying all roots (isosurface crossings) in a given ray segment.

3.4.1 LG values for density trees

Density values in our method are defined by an arithmetic expression tree, with leaves taking the form of equation (2). For a ray segment through a leaf, the value of L is obtained by differentiating (2) at the point where the ray segment comes closest to the halfspace:

$$L = \begin{cases} \frac{3}{4}(1 - \alpha_{\min}^2)|\mathbf{R} \cdot \mathbf{N}| & \alpha_{\min} < 1 \\ 0 & \text{otherwise} \end{cases}$$

where r = filter radius

$$\alpha_{\min} = \min\left(\frac{|d|}{r}\right)$$

d = distance from halfspace

\mathbf{R} = ray direction vector

\mathbf{N} = halfspace normal

A further differentiation gives us a formula for G , which is evaluated at a point where the ray segment is furthest from the halfspace (but is zero if the ray-segment is always further than r from it):

$$G = \frac{3}{2r^2} |\alpha|_{\max} |\mathbf{R} \cdot \mathbf{N}|$$

For a general density tree, the values of L and G are obtained by recursively descending the tree, applying appropriate bounds-combining formulae at each internal node. That involves also calculating a bound M on the density itself. We use the following bounds-combining formulae:

$$L(\rho_1 \pm \rho_2) = L(\rho_1) + L(\rho_2)$$

$$G(\rho_1 \pm \rho_2) = G(\rho_1) + G(\rho_2)$$

$$L(\rho_1 \rho_2) = M(\rho_1)L(\rho_2) + M(\rho_2)L(\rho_1)$$

$$G(\rho_1 \rho_2) = M(\rho_1)G(\rho_2) + M(\rho_2)G(\rho_1) + 2L(\rho_1)L(\rho_2)$$

$$M(\rho_1 + \rho_2) = \min(1, M(\rho_1) + M(\rho_2))$$

$$M(\rho_1 - \rho_2) = M(\rho_1)$$

$$M(\rho_1 \rho_2) = M(\rho_1)M(\rho_2)$$

3.4.2 Using LG values in root-finding

This section is concerned with the function **ShortEnough** in Figure 9, which returns true if a ray segment is sufficiently short that multiple roots cannot exist within the segment; such segments do not need to be subdivided. The method uses the ray segment end-point densities and density gradients (ρ_1, ρ'_1) and (ρ_2, ρ'_2) , together with the LG bounds for the segment, calculated as in the previous section. We improve upon the method of Kalra and Barr [11] by exploiting the known density and gradient values at the end of the ray segment to obtain a tighter criterion than that of Kalra and Barr.

The condition for the existence of multiple roots is derived from the following observations, illustrated in Figure 10:

- The existence of multiple roots implies at least one turning point after the first root and before the last.
- The distance along the ray to get from the start point to the turning point in (a) is at least the greater of $|\rho_1|/L$ or $|\rho'_1|/G$, as shown in Figure 10(a)
- If the signs of ρ_1 and ρ'_1 are the same, the density along the ray must first pass through a turning point and back to the original value (and with the gradient negated) before heading towards the root. This

requires a distance of at least $2|\rho'_1|/G$ in addition to that of (b). See Figure 10(b)

- A lower bound on the distance along the ray to get from the turning point of (a) to the end point of the segment can be determined as in (b) and (c).

From the above four observations, if we denote the distance term in (c) as the “Turn-Around Distance”, or TAD , a necessary condition for the existence of multiple roots in a ray segment of length l is:

$$\max\left(\frac{|\rho_1|}{L}, \frac{|\rho'_1|}{G}\right) + \max\left(\frac{|\rho_2|}{L}, \frac{|\rho'_2|}{G}\right) + TAD(\rho_1, \rho'_1) + TAD(\rho_2, -\rho'_2) \leq l \quad (6)$$

$$\text{where } TAD(\rho, \rho') = \begin{cases} 0 & \text{sign } \rho \neq \text{sign } \rho' \\ 2\frac{|\rho'|}{G} & \text{sign } \rho = \text{sign } \rho' \end{cases}$$

Function **ShortEnough** returns True if equation (6) evaluates to False, or if l is negligibly small.

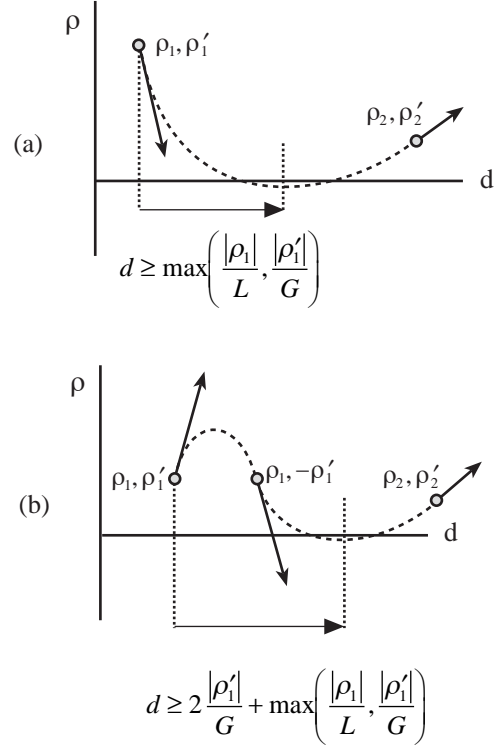


Figure 10. Ray Segment Length Constraints for Multiple Roots.

3.5 Space subdivision

To improve the performance of the ray tracer, we use a Binary Space Partitioning tree (BSP tree) space subdivision scheme, similar in principle to that described by Jansen [10].

We can classify CSG subtrees as “inside”, “outside”, or “straddling” with respect to a partitioning plane provided that the leaves of the tree can be so classified. To make this possible we allow convex intersections of halfspaces, and the density-tree equivalent, to be tagged as *boundable*. The vertices of such convex objects can

be calculated from their constituent halfspaces and used to classify the objects. One must be careful when classifying density trees, since they have non-zero density at distances of up to one filter radius outside their defining planes.

4 RESULTS

4.1. Images

Figure 12 shows the rounding effects produced by different combinations of normal CSG objects and density CSG objects; slightly simplified definitions for the objects, starting at the back left, are given in Figure 11. In the definition of obj3, note the use of an extra clipping plane to satisfy the density difference requirement of Section 2.2. Simply subtracting box1 would also be valid, but produces an “overhang” in the $y = 1$ plane, around the hole.

Figure 13 shows the effects of different rounding radii on the “rounded difference” object (obj4) of Figure 12. It can be seen that the method provides a smooth transition between pure polyhedral modelling and “blobby” or “soft object” modelling [2] [22] [3].

Figure 14 shows the use of small amounts of rounding to produce a softer more-natural look. The front stapler is geometrically the same as the back stapler but with rounding enabled, and various rounding radii used on different subcomponents. As well as softening the appearance, the rounding eliminates the Mach band along the middle of the side face of the upper portion of the stapler, and helps delineate component boundaries, particularly where the base touches the ground plane and at the bottom of the steel staple holder.

obj1	=	SetDifference UnitCube box1;
obj2	=	SetDifference (Rounded r UnitCube) box1;
obj3	=	SetDifference UnitCube (Product (topClip (Rounded r box2)));
obj4	=	Rounded r (Difference UnitCube box1);
obj5	=	Union (Rounded r UnitCube) (Rounded r box3);
obj6	=	Rounded r (Sum (UnitCube box3));
where		
r	=	0.07;
box1	=	Shift (0.25,0.8,0.25) (Scale (0.5,0.5,0.5) UnitCube);
box2	=	Shift (0.25,0.8,0.25) (Scale (0.5,1.0,0.5) UnitCube);
box3	=	Shift (0.25,1.0,0.25) (Scale (0.5,0.5,0.5) UnitCube);
topClip	=	DensityHalfspace ((0,1,0) ,1);

Figure 11 : Scene object descriptions for Figure 12.

4.2 Other effects

It is possible to extend the data structures to allow different smoothing radii on different halfspaces of a polyhedron. Figure 15(a) illustrates an extreme example of this: the unit height object is constructed from the intersection of four planes with a rounding radius of 0.3 and two with a rounding radius of 0.03. This gives the appearance of greatly different rounding radii on different edges. However, the “rounds” on the end faces are actually almost elliptical with semi-major axis lengths of 0.3 and 0.03. This shape defect highlights the fundamental limitation of quasi-convolutional smoothing as compared to surface blending techniques: lack of precise control of the geometry of rounded edges and corners.

Figure 15(b) shows the result of adding four slightly-overlapping cubical density objects: a bulging “seam” is created, due to density values of 2 and 4 in the region of intersection. Such an effect is probably unintended, and potentially interferes with the correctness of the algorithm by displacing isosurfaces outside the calculated bounding object (section 3.3). It does, however, suggest the possibility of deliberately introducing density variations to model surface detail or texture. This is a topic for future research.

The assumption throughout this paper is that density trees always produce density values in the range 0 to 1. This constraint is not enforced by the program, but is achieved manually during scene modelling by the addition of suitable clipping planes. A completely automatic solution is mathematically trivial—just implement a CSG union $a \cup b$ as a density expression $a + b - a \times b$ —but expensive. Finding a low-cost automatic solution is another topic for future research.

4.3. Performance

A key aspect of the work described in this paper, as compared to that of Colburn [5], is the use of a fast approximation for the convolution integral and the associated density gradient (or isosurface normal).

Our prototype ray tracer is written in the lazy functional language Clean 1.0. On a Macintosh Quadra 700, rendering times for some of the 600×380 pixel images in this paper are shown in Table 1. Also shown are the rendering times using the same program but with all rounded density objects replaced by their polyhedral CSG equivalent. The times should be interpreted with considerable caution for the following reasons.

The use of Clean, as compared to C, typically imposes a run-time penalty of around a factor of three [13]. The penalty may be higher in our context, since various low-level optimization techniques, such as the use of table look-up methods for the calculations of equation (6), were not feasible in the early version of Clean (Clean 0.8) that was used for most of the development. Also, the timings are strongly dependent on the structure of the BSP tree, which is necessarily different for the rounded and non-rounded scenes. The time for a non-rounded rendering of the CSGExamples scene is anomalously high because of a poor BSP tree in this case; altering the heuristics that control BSP tree construction might significantly reduce this time, but increase others. Optimisation of BSP tree construction for ray tracing is not well understood and is a high priority for future work, though we are starting to suspect that BSPtree space subdivision is fundamentally less efficient than a spatial enumeration method like that of Fujimoto et al [6].

Despite the above reservations, comparing the rendering times for rounded images with those for unrounded images should still give a reasonable indication of the inherent speed of the rounding algorithm as compared to normal CSG ray tracing algorithms.

The rounded images cost from 1.5 to 5 times as much to compute as the unrounded equivalents. The cost factor depends primarily on how many pixels in the image lie on rounded surfaces, which is why the variable rounding image is relatively so expensive. A factor of 2 to 3 is

typical for small scenes, with small amounts of rounding. It must be admitted, however, that the tracer is designed for rendering rounded objects, not unrounded objects; higher performance on unrounded objects is undoubtedly achievable, so that the cost factor of 2 to 3 is probably a lower bound.

Fig.	Description	Time (mins)	Time w/o rounding
10	CSG Examples	55	45
11	Variable rounding	105	20
12	Staplers	240	70

Table 1. Image rendering times, without antialiasing. Also shown are the rendering times for identical scenes without rounding.

5 DISCUSSION AND CONCLUSIONS

We have described a new method for defining and rendering polyhedra with smoothly rounded edges and corners. The goal was to provide more natural looking polyhedral scene objects, for which purposes geometric precision is not generally an issue. The method achieves a similar effect to that of the surface blending sometimes performed during modelling, but with several advantages:

- (a) it handles any arbitrary polyhedral topology, such as vertices with many incident edges
- (b) the extra cost in scene description memory of a smoothed polyhedron over an unsmoothed polyhedron is negligible: one extra floating point number, as compared to the large number of polygon mesh elements used to approximate surfaces generated by surface blending algorithms.
- (c) when both modelling and rendering are taken into account, the method is much simpler to implement, involving only a fairly straightforward extension to a normal CSG ray tracer.

The amount of smoothing can be set separately for each scene component, and further CSG operations can then be performed using the smoothed components, so that a wide range of effects is possible. Using small smoothing radii gives a softer more natural look to polyhedra, and delineates their boundaries more clearly. Larger radii lead ultimately to “soft object” modelling effects.

Quasi-convolutional smoothing does not compete with surface blending techniques where geometric precision is an issue, for example in engineering design. The only parameter the user can vary is the rounding radius, so the precise shape of the rounding surface is not controllable.

Rendering is by ray tracing and typically from 2 to 5 times as long as for an equivalent scene without any rounding—the factor increases as the visible effects of the rounding increase. During modelling, the underlying CSG model, with rounding ignored, could be converted to a boundary representation in the usual way [18] [17] for fast polygon display.

As described, our method uses a simple spherical version of a “box” filter. In a signal-processing context, this is undoubtedly an inferior low pass filter as compared to weighted filters such as a Gaussian. In our context, however, we doubt whether the choice of filter is very relevant, since its only effect is to alter the shape of the

actual blend surface. Nonetheless, our method can easily be altered to use an arbitrary spherical filter, using table-look methods for obtaining densities, gradients etc. It can be noted that Colburn[4][5] and Bloomenthal and Shoemake [3] both used Gaussian filters in order to exploit their separability property, rather than for signal-processing reasons.

6 FURTHER WORK

The system outlined in this paper opens up several interesting avenues for further research, particularly into:

- (a) The exact mathematical properties of the quasi-convolutional surfaces
- (b) The full extent of the modelling potential of the method
- (c) The effects of using a differently shaped or weighted filter than the simple constant-value sphere assumed throughout this paper
- (d) Polygonization of the rounded surfaces to allow high speed rendering
- (e) Extension of the method to allow smoothing of non-polyhedral objects

ACKNOWLEDGEMENTS

I am most grateful to Andrew MacGibbon for modelling the stapler in Figure 14, implementing the woodgrain texture, and for helping with most of the image rendering. My thanks also to Peter Shirley for his useful comments on an early draft of the paper and to Peter Kulka for useful comments on a later draft. The first stages of the work were carried out at the Cornell Program of Computer Graphics; my sincere thanks to Donald Greenberg for his support during this period. Lastly, my thanks to the Auckland University Research Committee for funding the equipment.

APPENDIX

Theorems on Density Bounds

Given here are the proofs of the theorems used in section 3.3, plus equivalent theorems for use in calculating an Inner Bound CSG object. The definitions for Inner and Outer Bounds have already been given in equations (3) and (4).

Theorem 1. If $\rho(\mathbf{x}) = \rho_1(\mathbf{x})\rho_2(\mathbf{x})$, with $\rho_1, \rho_2 \in [0,1]$, then $S = O(\rho_1, t) \cap O(\rho_2, t)$ is an outer bound of ρ with respect to threshold t .

Proof.

$$\begin{aligned}
 \mathbf{x} \notin S &\Rightarrow \mathbf{x} \notin O(\rho_1, t) \text{ or } \mathbf{x} \notin O(\rho_2, t) \\
 &\Rightarrow \rho_1(\mathbf{x}) \leq t \text{ or } \rho_2(\mathbf{x}) \leq t \\
 &\Rightarrow \rho(\mathbf{x}) \leq t \text{ (since } \rho_1, \rho_2 \in [0,1]) \\
 &\Rightarrow S \text{ is an outer bound of } \rho \text{ with respect to } t \\
 &\text{(from (3))}
 \end{aligned}$$

Theorem 2. If $\rho(\mathbf{x}) = \rho_1(\mathbf{x}) + \rho_2(\mathbf{x})$ then $S = O(\rho_1, t/2) \cup O(\rho_2, t/2)$ is an outer bound of ρ with respect to threshold t .

Proof.

$\mathbf{x} \notin S \Rightarrow \mathbf{x} \notin O(\rho_1, t/2) \text{ and } \mathbf{x} \notin O(\rho_2, t/2)$
 $\Rightarrow \rho_1(\mathbf{x}) \leq t/2 \text{ and } \rho_2(\mathbf{x}) \leq t/2$
 $\Rightarrow \rho(\mathbf{x}) \leq t$
 $\Rightarrow S \text{ is an outer bound of } \rho \text{ with respect to } t$
 (from (3))

Theorem 3. If $\rho(\mathbf{x}) = \rho_1(\mathbf{x}) - \rho_2(\mathbf{x})$, with $\rho_1, \rho_2 \in [0, 1]$, then $S = O(\rho_1, t) - I(\rho_2, 1 - t)$ is an outer bound of ρ with respect to threshold t .

Proof.

$\mathbf{x} \notin S \Rightarrow \mathbf{x} \notin O(\rho_1, t) \text{ or } \mathbf{x} \in I(\rho_2, 1 - t)$
 $\Rightarrow \rho_1(\mathbf{x}) \leq t \text{ or } \rho_2(\mathbf{x}) \geq 1 - t$
 case 1: $\rho_1(x) \leq t \Rightarrow \rho(x) = \rho_1(x) - \rho_2(x) \leq t$
 case 2: $\rho_1(x) > t, \rho_2(x) \geq 1 - t$
 $\Rightarrow \rho(x) = \rho_1(x) - \rho_2(x) \leq 1 - (1 - t)$
 $\therefore \mathbf{x} \notin S \Rightarrow \rho(\mathbf{x}) \leq t$
 $\Rightarrow S \text{ is an outer bound of } \rho \text{ with respect to } t$
 (from (3))

Theorem 4. If $\rho(\mathbf{x}) = \rho_1(\mathbf{x}) + \rho_2(\mathbf{x})$, with $\rho_1, \rho_2 \geq 0$, then $S = I(\rho_1, t) \cup I(\rho_2, t)$ is an outer bound of ρ with respect to threshold t .

Proof.

$\mathbf{x} \in S \Rightarrow \mathbf{x} \in I(\rho_1, t) \text{ or } \mathbf{x} \in I(\rho_2, t)$
 $\Rightarrow \rho_1(\mathbf{x}) \geq t \text{ or } \rho_2(\mathbf{x}) \geq t$
 $\Rightarrow \rho(\mathbf{x}) \geq t$
 $\Rightarrow S \text{ is an inner bound of } \rho \text{ with respect to } t$
 (from (4))

Theorem 5. If $\rho(\mathbf{x}) = \rho_1(\mathbf{x})\rho_2(\mathbf{x})$ then $S = I(\rho_1, \sqrt{t}) \cap I(\rho_2, \sqrt{t})$ is an outer bound of ρ with respect to threshold t .

Proof.

$\mathbf{x} \in S \Rightarrow \mathbf{x} \in I(\rho_1, \sqrt{t}) \text{ and } \mathbf{x} \in I(\rho_2, \sqrt{t})$
 $\Rightarrow \rho_1(\mathbf{x}) \geq \sqrt{t} \text{ and } \rho_2(\mathbf{x}) \geq \sqrt{t}$
 $\Rightarrow \rho(\mathbf{x}) \geq t$
 $\Rightarrow S \text{ is an inner bound of } \rho \text{ with respect to } t$
 (from (4))

Theorem 6. If $\rho(\mathbf{x}) = \rho_1(\mathbf{x}) - \rho_2(\mathbf{x})$, with $\rho_1, \rho_2 \geq 0$, then $S = I(\rho_1, t) - O(\rho_2, 0)$ is an outer bound of ρ with respect to threshold t .

Proof.

$\mathbf{x} \in S \Rightarrow \mathbf{x} \in I(\rho_1, t) \text{ and } \mathbf{x} \notin O(\rho_2, 0)$
 $\Rightarrow \rho_1(\mathbf{x}) \geq t \text{ and } \rho_2(\mathbf{x}) = 0$
 $\Rightarrow \rho(\mathbf{x}) \geq t$
 $\Rightarrow S \text{ is an inner bound of } \rho \text{ with respect to } t$
 (from (4))

REFERENCES

- [1] Amanatides, J. and Mitchell, D.P., "Some Regularization Problems in Ray Tracing," in *Proceedings of Graphics Interface '90*, May 1990, pp. 221–228.
- [2] Blinn, J.F., "A Generalization of Algebraic Surface Drawing," *ACM Transactions on Graphics*, vol. 1, no. 3, 235–256, July 1982.
- [3] Bloomenthal, J. and Shoemake, K., "Convolution Surfaces," in *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, no. 4, July 1991, pp. 251–256.
- [4] Colburn, S., *Method for Global Blending of Computer Modeled Solid Objects using a Convolution Integral*, United States Patent No. 4,791,583, Dec. 1988.
- [5] Colburn, S., "Solid Modeling with Global Blending for Machining Dies and Patterns," *SAE Technical Paper Series*, #900878, 1990.
- [6] Fujimoto, A., Tanaka, T., and Iwata, K., "Arts: Accelerated Ray-Tracing System," *IEEE Computer Graphics and Applications*, 16–26, Apr. 1986.
- [7] Hanrahan, P., "Ray Tracing Algebraic Surfaces," in *Computer Graphics (SIGGRAPH '83 Proceedings)*, vol. 17, no. 3, July 1983, pp. 83–90.
- [8] Hoffmann, C. and Hopcroft, J., "The potential method for blending surfaces and corners," in *Geometric Modeling: Algorithms and New Trends*, Farin, G., Ed. SIAM, Philadelphia, 1987, pp. 347–365.
- [9] Hudak, P., Peyton-Jones, S., Wadler, P., Boutel, B., Fairbairn, J., Fasel, J., Hammand, K., Hughes, J., Johnsson, T., Kieburtz, D., Nikhil, R., Partain, W., and Peterson, J., "Report on the Programming Language Haskell," *ACM SigPlan Notices*, vol. 27, no. 5, 1–164, 1992.
- [10] Jansen, F., "Data Structures for Ray Tracing," in *Data Structures for Raster Graphics (Workshop proceedings)*. *Eurographics Seminars*, Springer-Verlag, 1986, pp. 57–73.
- [11] Kalra, D. and Barr, A.H., "Guaranteed Ray Intersections with Implicit Surfaces," in *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, no. 6, July 1989, pp. 297–306.
- [12] Levoy, M., "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, vol. 9, no. 3, 245–261, July 1990.
- [13] Plasmeijer, R. and van Eekelen, M., *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.
- [14] Plasmeijer, R. and van Eekelen, M., "Concurrent Clean Language Report (Version 1.0)," Tech. Rep., University of Nijmegen, April 1995.
- [15] Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., *Numerical Recipes in C: The Art of Scientific Computing* (2nd ed.). Cambridge: Cambridge University Press, 1992.

- [16] Roth, S.D., "Ray Casting for Modelling Solids," *Comput. Graphics and Image Process. (USA)*, vol. 18, 109–144, Feb. 1982.
- [17] Tawfik, M.S., "An Efficient Algorithm for CSG to B-rep Conversion.," in *Proc. Symp. on Solid Modelling Foundations and CAD/CAM Applications (Austin, Texas)*, June 1991, pp. 99-108.
- [18] Thibault, W.C. and Naylor, B.F., "Set Operations on Polyhedra Using Binary Space Partitioning Trees," in *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, July 1987, pp. 153–162.
- [19] Turner, D.A., "Miranda: a Non-strict Functional Language with Polymorphic Types," in *Proc. Conference on Functional Programming Languages and Computer Architecture (Nancy, France). LNCS, No. 201.*, Springer-Verlag, 1985, pp. 1-16.
- [20] Weast, R.C. (ed.), *CRC Handbook of Tables for Mathematics*. The Chemical Rubber Company, 1970.
- [21] Woodward, J.R., "Blends in Geometric Modelling," in *Proc. 2nd IMA Conference on the Mathematics of Surfaces (Cardiff, Wales)*, Oxford University Press, Oxford, UK, Sept. 1986, pp. 1-43.
- [22] Wyvill, B., McPheeters, C., and Wyvill, G., "Data Structures for Soft Objects," *The Visual Computer*, vol. 2, no. 4, 227-234, 1986.

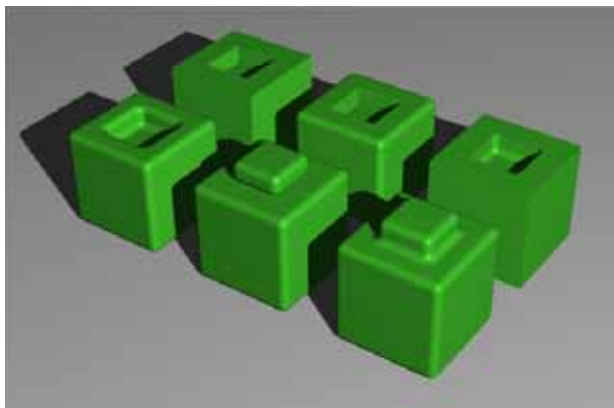


Figure 12. Illustrating the use of different set operations on rounded objects.

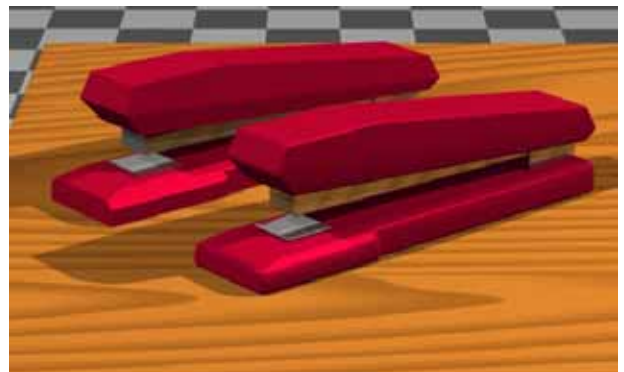


Figure 14. The use of rounding to achieve a softer more natural appearance. The two staplers are defined by the same CSG operations, but the front stapler has had various different roundings used on its sub-components.

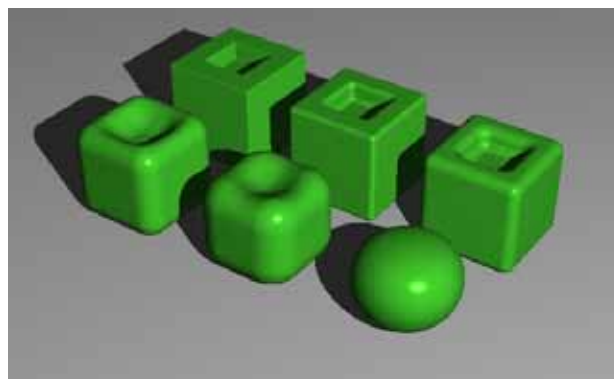


Figure 13. Illustrating the effects of increasing the radius of the smoothing filter on the object obj4 from Figure 12. Successive radii values, from the back left corner, are: 0.01, 0.05, 0.1, 0.2, 0.3, and 0.5.

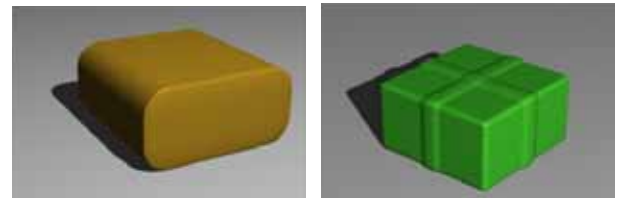


Figure 15. Some other effects. (a) shows the use of different roundings on different planes of a cuboid. (b) shows the result of illegally summing four cubes with non-empty intersections.