The Visual Computer

Triage polygonization of rounded polyhedra

Burkhard Wünsche, Richard Lobb

University of Auckland,
Department of Computer Science, Private Bag 92019,
Auckland, New Zealand
E-mail: {burkhard,richard}@cs.auckland.ac.nz

This paper describes a method for rounding and polygonizing edges and corners of polyhedral models to produce softer, more natural looking objects. The underlying rounded surface is assumed to be defined by "quasiconvolutional smoothing", and the focus is on polygonizing the surface rapidly and efficiently. A binary space partitioning (BSP) tree is used to classify space in the vicinity of a polyhedron to identify the curved regions. Planar surfaces are extracted as single polygons, and extra polygons are introduced only at rounded edges and corners. The result is a high-quality polygonization of the rounded polyhedron with none of the fragmentation problems of more general polygonization methods.

Key words: Polygonization – Polygonal models – Rounding – BSP trees – CSG models

Correspondence to: B. Wünsche

1 Introduction

Many common objects are approximately polyhedral in shape. However, computer models of such objects often look artificial because of the unnaturally sharp edges and corners of true polyhedra. Human modellers sometimes overcome this problem by explicitly trimming planar surfaces and then adding smooth blends between adjacent faces. Although this can produce excellent results, it is rather time consuming. Furthermore, the underlying blending process is computationally difficult or even intractable with some configurations. A good overview of blending methods is given by Hoschek and Lasser (1992).

Colburn (1990) introduced *convolutional smoothing* as a modelling technique in mechanical engineering. His method uses a spherical test volume or *filter* with a radius equal to the desired blend radius. In the simplest version of this method (namely an unweighted filter), the surface of a smoothed object is 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 solid and half is outside. Figure 1 illustrates the process in two dimensions, showing a polygon and its smoothed version.

Convolutional smoothing is easy to specify: only a single blending radius is required to round an arbitrarily complex solid. However, it is in general hard to implement. Even in the simple unweighted filter case, in which the convolution integral reduces to the volume of the intersection of a sphere and a solid, the geometry is difficult or intractable. Colburn (1990) approximates the convolution calculation by using an octree model of the solid. Dansted (1997) shows how an exact convolution can be computed for polyhedral solids and unweighted filters, but the computation is expensive. Lobb (1996) introduces a fast approximation to convolutional smoothing of polyhedra called *quasiconvolutional smoothing*. Polyhedra are represented by constructive solid geometry (CSG) trees with halfspaces at the leaves. True convolutional smoothing is applied to the leaves only, and the resulting density fields are combined by means of arithmetic operators in lieu of set operations. Lobb's method is the one chosen for use in this paper, and it is discussed in more detail in the next section.

All these convolutionlike methods define the surface of the rounded object by an implicit function, i.e., the surface is all points (x, y, z) such that

The Visual Computer (1999) 15:36–54

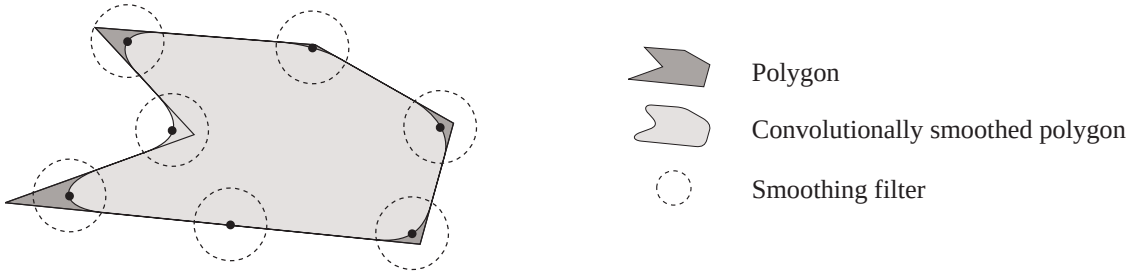


Fig. 1. Convolutional smoothing of a polygon

$f(x, y, z) = k$ for some function f and isosurface level k. The problem then is how to render that surface. Colburn uses ray casting to find surface points from which a polygon mesh is constructed. Dansted uses a space-subdivision method to obtain a polygonization. Lobb directly ray-traces his models using an implicit surface algorithm specially adapted to quasiconvolutionally smoothed models. Ray tracing is expensive, and is certainly not suitable for interactive use during modelling. The two polygonization methods are also relatively expensive and output large numbers of polygons.

The most common method of polygonizing implicit surfaces is the marching cubes method or some variant (Wyvill et al 1986; Lorensen and Cline 1987; Doi and Koide 1991). Such methods subdivide the space containing the surface into a uniform cell grid, and use a table lookup to approximate the surface through each cell. The cell size must be sufficiently small to capture the smallest features of interest in the object. In the case of slightly rounded polyhedra, the cell size needs to be comparable to the rounding radius of the edges and corners. Edges are then represented by many small cell-sized polygons rather than by a few long, thin polygons. Worse still, the planar portions of the object that are unaffected by the rounding are also fragmented into large numbers of tiny polygons. Regular subdivision methods are thus unsuitable for use with polyhedron rounding.

Adaptive subdivision methods (Bloomenthal 1988, 1994; Hall and Warren 1990) give less fragmentation. These methods adaptively reduce the scale of the subdivision mesh in regions of high curvature. However, in order not to miss small surface details, a fairly fine initial mesh is still required, which in turn leads to significant fragmentation of planar faces.

The only completely general way of obtaining a high-quality polygonization in the presence of small surface details is to use a fine sampling grid, which produces large numbers of small polygons, and then use a mesh optimization postprocess to merge adjacent planar or nearly planar polygons (Hinker and Hansen 1993; Kalvin and Taylor 1996; Gieng et al 1998). Such methods would certainly work on the implicit surfaces of rounded polyhedra. However, some degree of unnecessary fragmentation is inevitable because general methods cannot exactly fit subdivision grids to the arbitrarily oriented polyhedra. Furthermore, optimizing large meshes of small polygons is bound to be expensive; it is clearly more efficient to compute an efficient polygonization directly if possible. A survey of polygonization methods and optimization techniques is given by Wünsche (1997).

This paper presents triage polygonization, which is a fast, high-quality polygonization method especially designed for quasiconvolutionally smoothed objects. It generates a binary space partitioning (BSP) tree subdivision of space around the object. The tree allows planar regions of the rounded polyhedron to be extracted directly, each as a single polygon. Regions containing curved surfaces are also identified and polygonized efficiently in a step called *subspace polygonization*.

2 Quasiconvolutional smoothing

The triage polygonization method introduced in this paper is specifically designed for use with quasiconvolutionally smoothed polyhedra. This sec-

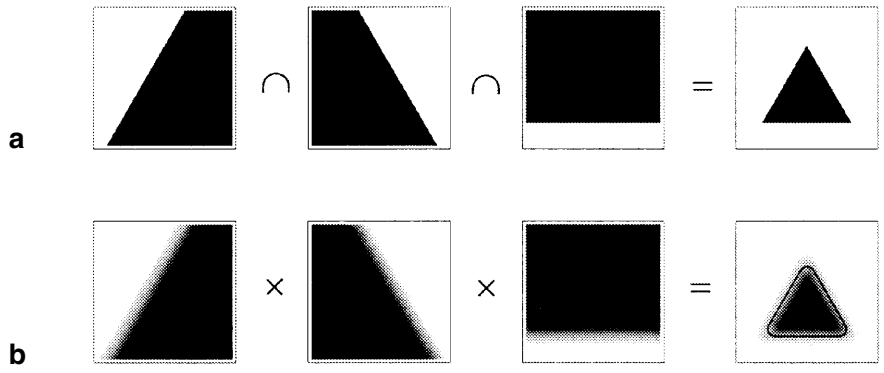


Fig. 2a, b. Rounding a triangle: **a** the triangle is represented as the intersection of three halfspaces; **b** the rounded triangle is obtained by smoothing and then multiplying the individual halfspaces. The solid line in the bottom right figure shows the 0.5 isocontour that defines the boundary of the rounded triangle

tion summarizes the relevant aspects of quasiconvolutional smoothing. For full details, refer to the original paper (Lobb 1996).

Quasiconvolutional smoothing requires that the polyhedron to be rounded be represented as a constructive solid geometry (CSG) tree with halfspaces at the leaves. A convex solid is trivially represented as the intersection of the halfspaces defined by its faces (Fig. 2a). More complex solids are constructed by unions, intersections and differences of convex solids. The rounded polyhedron is then defined as the 0.5 isosurface of the field obtained by:

1. Convolutionally smoothing all the leaf half spaces.
2. Combining the resulting density fields by means of the arithmetic operations of multiplication, addition and subtraction in lieu of the CSG operations of intersection, union, and set difference.

This process is illustrated in two dimensions in Fig. 2b.

The halfspace is regarded as a density field with a value 1 inside the halfspace and a value 0 outside it. The values of 1 and 0, rather than the more common $+1$ and -1, are chosen so that set intersection can be mimicked by multiplication, as discussed later. The value of the convolution at any point p is equal to the volume of a sphere with radius r centered at p intersected with the halfspace, for which the answer is

$$\rho_H^r(p) = \begin{cases} 0 & \alpha \geq 1 \\ 0 & \alpha \leq -1 \\ (1-\alpha)^2 * (2+\alpha)/4 & \text{otherwise,} \end{cases} \quad (1)$$

where $\alpha = \frac{d}{r}$ and d is the distance of point p to the halfspace H.

The density field $\rho_{pbj}^r(p)$ of a quasiconvolutionally smoothed CSG object can then be defined as

$$\rho_{obj}^r(x) = \begin{cases} \rho_H^r(p) & \text{if } obj \text{ is a halfspace H} \\ \rho_A^r(x) + \rho_B^r(x) & \text{if } obj = A \cup B \\ \rho_A^r(x) * \rho_B^r(x) & \text{if } obj = A \cup B \\ \rho_A^r(x) - \rho_B^r(x) & \text{if } obj = A \backslash B, \end{cases} \quad (2)$$

where "\" denotes set difference.

Finally, the surface of the quasiconvolutionally smoothed object is all points $x \in R^3$ for which

$$\rho_{obj}^r(x) = 0.5.$$

For this scheme to be a workable approximation to convolutional smoothing, the original CSG object is restricted in two important ways:

1. The union operation can be applied only to non-intersecting objects.
2. The set difference can be applied only to objects that completely intersect.

We furthermore assume that the density is always between zero and one. For a discussion of these

constraints and other properties of the method, refer to Lobb (1996).

3 Triage polygonization

3.1 Overview of triage polygonization

Triage polygonization consists of three distinct steps.

1. *Polyhedral subdivision of space.* In this step the density field of the quasiconvolutionally smoothed object is partitioned into convex regions called *cells*, each of which is one of the following:
   (a) A "low" cell, meaning that the density at all points in the cell is below the 0.5 isosurface level.
   (b) A "high" cell, meaning that the density at all points is above the 0.5 isosurface level.
   (c) "Unclassified", meaning that the cell probably includes some densities above the isosurface level and some below. These cells contain the curved surfaces of the rounded object.
2. *Extraction of planar regions.* In this step we identify and output polygonal faces that are shared by both high cells and low cells. These polygons, called *tree polygons*, essentially cover the planar regions of the surface of the rounded polyhedron, i.e., all those surface points that are common to both the rounded and the unrounded polyhedron.
3. *Subspace polygonization.* In this step we output a polygonization of the isosurface passing through each of the unclassified cells. These output polygons essentially cover the curved regions of the surface of the rounded polyhedron.

3.2 Polyhedral subdivision of space

3.2.1 Density classification

The density field ρ_H^r of a quasiconvolutionally smoothed halfspace partitions $\mathbb{R}^3$ into four parts. Let h denote the halfspace plane of halfspace H, and define two parallel planes h_r (*outer halfspace plane*) and h_{-r} (*inner halfspace plane*) at distances of r and $-r$ from h, respectively. Then, for all points outside h_r, the density field is constantly zero. Similarly, inside h_{-r} the density field is constantly one. Between the two planes are a low region (density values smaller than 0.5) and a high region (density values greater than or equal to 0.5) separated by the halfspace plane h. Note that points that lie exactly on the planes h_r, h, and h_{-r} have density values of zero, 0.5, and one, respectively.

Figure 3 shows the density field of the intersection of two halfspaces. The two halfspaces H_1 and H_2 are smoothed with rounding radii r_1 and r_2, respectively. (We have chosen two different rounding radii for the halfspaces. This extension is explained in Sect. 3.5.) From Eq. 2, the resulting density field is the product of the density fields of the halfspaces. For example, the density value at the point p_1 is computed as

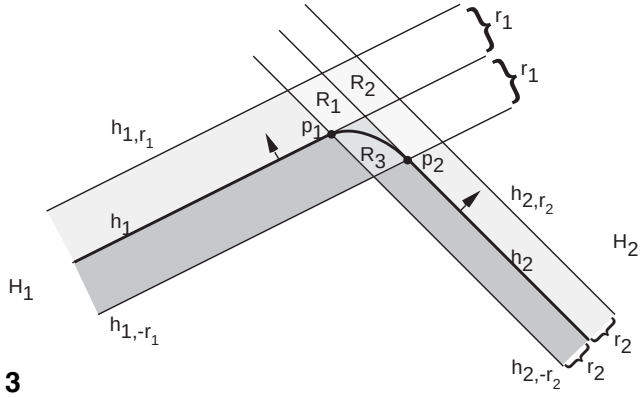
$$\rho_{H_1 \cap H_2}(p_1) = \rho_{H_1}^{r_1}(p_1) * \rho_{H_2}^{r_2}(p_1) = 0.5 * 1 = 0.5$$

since p_1 lies on both the planes h_1 and $h_{2,-r_2}$.

The planes H_1 and H_2, together with their associated inner and outer halfspace planes, partition the density field over $\mathbb{R}^3$ into 16 regions. By using interval arithmetic (Duff 1992; Snyder 1992), we can obtain bounds on the density values over each region. As an example, consider region R_1 is bounded by the planes h_1, h_{1,r_1}, h_2, and h_{2,r_2}. Since R_1 is bounded by the planes h_1 and h_{1,r_1}, the contribution of the density field of H_1 to this region varies between zero and 0.5. Similarly, since R_1 is bounded by h_2 and h_{2,r_2}, the density contribution of H_2 varies between 0.5 and one. Hence, all density values in the region R_1 lie in the product of these two intervals, namely the (open) interval (0.0, 0.5), and therefore the region R_1 lies outside the 0.5 isosurface.

A similar calculation for region R_3 in Fig. 3 yields the result that all density values in the region lie in the interval (0.25, 1), so potentially this region *can* contain the 0.5 isosurface.

These interval computations can be simplified by replacing intervals with density classes. A single smoothed halfspace partitions space into regions with classes zero, one, low, or high. We introduce a new density class *unclassified* to denote any other classification. Table 1 summarizes the rules for the multiplication of two density classes. Similar tables can be defined for addition and subtraction.



3

Fig. 3. Intersection of two halfspaces

Fig. 4. A quasiconvolutionally smoothed object (**a**), its defining CSG object (**b**), and the density fields of the two primitives (**c**). The halfspace planes of the primitive objects, together with their inner and outer halfspace planes, partition the density fields into convex cells (**d**), which are then classified (**e**). Merging the primitive density fields using a BSP tree approach provides a partitioning for the density field of the whole object (**f**)

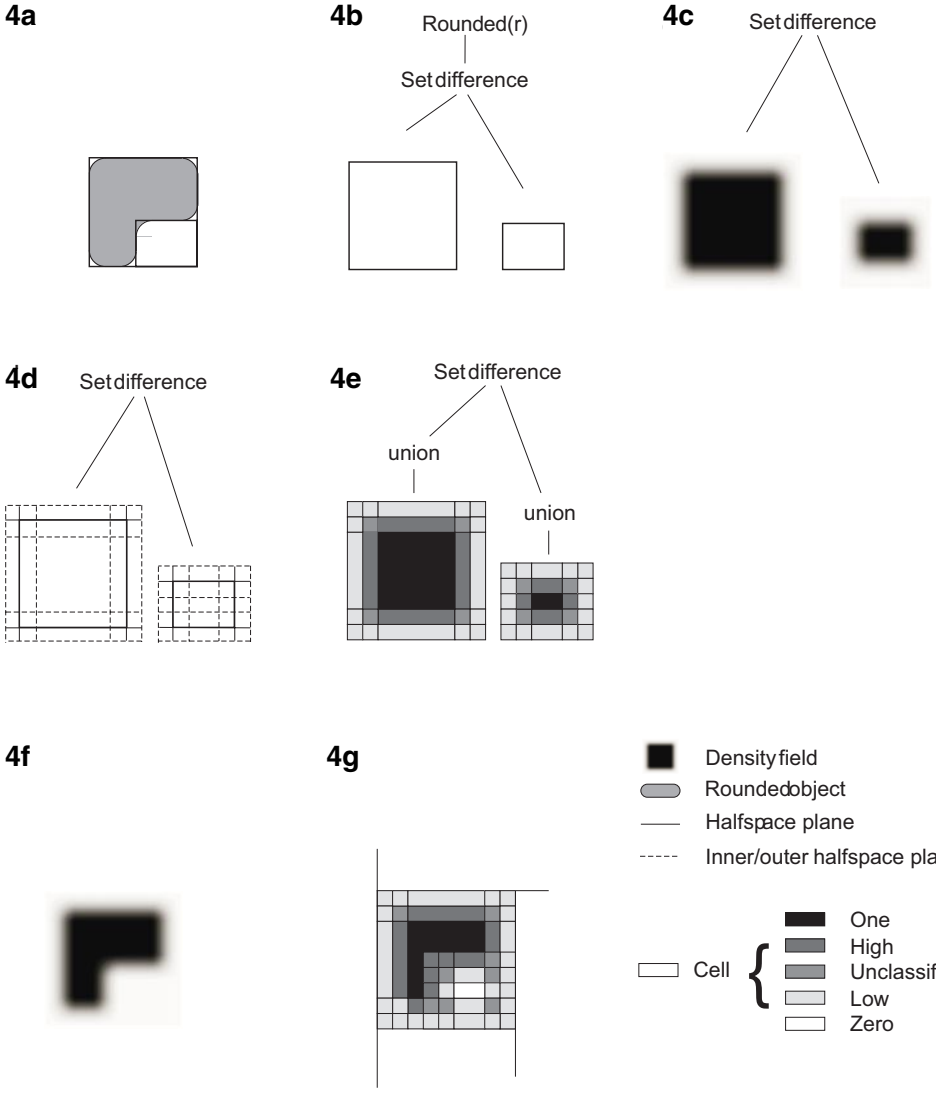


Table 1. Multiplication of density classes

	Zero	Low	Unclassified	High	One
Zero	Zero	Zero	Zero	Zero	Zero
Low	Zero	Low	Low	Low	Low
Unclassified	Zero	Low	Unclassified	Unclassified	Unclassified
High	Zero	Low	Unclassified	Unclassified	High
One	Zero	Low	Unclassified	High	One

Replacing the intervals with density classes does result in some loss of information. However, for our purposes (the detection of regions intersected by the 0.5 isosurface), this information loss is insignificant.

3.2.2 Replacing polyhedral primitives with classified cells

According to Sect. 2, the polyhedron to be rounded is represented as a CSG tree with halfspaces at the leaves. We now restrict that definition so that we deal only with bounded polyhedra constructed from convex polyhedral primitives. Rounding such a CSG tree by quasiconvolutional smoothing results in an arithmetic tree that combines the density fields created by rounding each convex polyhedral primitive. The first step in our algorithm is thus to replace each primitive by a union of classified cells that together represent the nonzero density field of the primitive. We call the resulting modified tree a *density CSG tree*. Figure 4 illustrates the process. It shows a quasiconvolutionally smoothed object (Fig. 4a) defined as a rounded set difference of two cuboids (Fig. 4b). In order to subdivide the density functions of the individual primitives (Fig. 4c) into nonzero regions, we define, for every halfspace plane of the primitive object, the corresponding inner and outer halfspace planes (Fig. 4d). This creates a set of polyhedral regions or cells. As with the intersection of two halfspaces in Fig. 3, we classify each cell with respect to each halfspace into zero, low, high, or one and then compute its density class by multiple applications of Table 1.

Figure 5 gives the pseudocode for the algorithm that classifies each cell. Note that a region lying outside any outer halfspace plane has a zero density class, so it is sufficient to subdivide the polyhedron defined as the intersection of all outer halfspaces. Note, too, that the density class of a cell is the product of its classifications in all different halfspaces. Hence its density class can be computed by counting the number of times it is classified by the planes into each of the density classes. For example, if at least one density class is zero, then the product is zero.

3.2.3 Polyhedral subdivision with BSP trees

The next step is to merge all the classified cells at the leaves of the density CSG tree into a unified polyhedral subdivision of space around the original polyhedron, as hinted at in Fig. 4g. We use BSP trees for this purpose.

The fundamental methodology underlying BSP trees is spatial partitioning (Fuchs et al 1980; Naylor 1981). Planes are used to recursively subdivide $\mathbb{R}^3$ to create a set of disjoint convex cells. Each cell is then designated as either interior or exterior to the set. The boundary need not be represented explicitly as it is derivable from the cells. We introduce a *density BSP tree*, simply called BSP tree from now on, which is an ordinary BSP tree in which the resulting leaf cells are classified into density classes.

Our polyhedral subdivision algorithm transforms the density field of a density CSG tree into a BSP tree. The algorithm is similar to the standard algorithm for the conversion of a CSG object into a BSP tree (Thibault and Naylor 1987; Naylor et al 1990). Figure 6 gives the algorithm. If the CSG object is not a primitive, we transform the left child object recursively to a BSP tree and insert the right child object according to the corresponding set operation. A primitive density CSG object, i.e., a classified cell, is transformed into a linear tree where the enclosed region has the same density class as the cell and the outside regions are classified as zero. A nonprimitive CSG object is converted to a BSP tree by recursively converting the left subtree of the object to a BSP tree and then inserting into that the right CSG subtree. The insertion is under control of the CSG set operation that uses the algorithm of Figure 7.

```
function ClassifiedCells (r :Real /* rounding radius */, planes :List of Plane /* halfspace planes */ ) :List of ClassifiedCell
    /* Subdivides non-zero density field of a polyhedral primitive into classified cells */
        outerPlanes=TranslateInNormalDirection(r, planes)
        innerPlanes=TranslateInNormalDirection(−r, planes)
        boundingPolyhedron=Intersection of halfspaces of outerPlanes
        cells=Subdivide( boundingPolyhedron, planes∪innerPlanes)
        classifiedCells=∅
        For each cell ∈ cells do
            (n_zero, n_low, n_high, n_one)=ClassifyAgainstAllPlanes (cell, planes)
            if n_zero 0 then densityClass=zero
            elsif n_low 0 then densityClass=low
            elsif n_high == 0 then densityClass=one
            elsif n_high == 1 then densityClass=high
            else densityClass=unclassified
            classifiedCells=classifiedCells∪(cell,densityClass)
        return classifiedCells
```

Fig. 5. Transforming a polyhedral primitive into classified cells

```
function CSG2BSP(obj:CSGObject):BSPTree
    /* Transforms a CSG object into a BSP tree */
    if obj is a classified cell then
        return LinearBSPTree(obj.densityClass, obj.faceList)
    else /* obj is not a classified (convex) cell */
        return InsertCSGinBSP(obj.right, CSG2BSP(obj.left), obj.op)
    function LinearBSPTree(class:DensityClass, faces:List of Faces):BSPTree
    /* Computes linear BSP tree for a convex CSG object */
    if faces is empty then return MakeLeaf(class)
    else
        plane=Plane of HeadOfList(faces)
        inTree=LinearBSPTree(class, TailOfList( faces))
        return MakeNode(plane, inTree, MakeLeaf(zero))
```

Fig. 6. Transforming a CSG object into a BSP tree

Function InsertCSGinBSP involves splitting the CSG object by the partitioning plane of the root of the BSP tree. The resulting portions are then recursively inserted in the two subtrees of the BSP tree. If the CSG object reaches a leaf node, it is transformed recursively into a BSP tree and the density classes of the BSP tree are updated according to the density class of the leaf node and the involved set operation.

3.3 Tree polygons

The BSP tree partitions the density field of a quasiconvolutionally smoothed object into cells with density classes zero, low, unclassified, high, and one. It can be seen that the density field on a face between a low cell and a high cell is a constant 0.5. These faces are therefore part of the polygonized surface and can be extracted directly. The algorithm for this step is similar to the surface extraction step for BSP trees (Thibault and Naylor 1987) and is given in Figure 8.

For each partitioning plane, we get a candidate tree polygon by intersecting a bounding box with the partitioning plane. The bounding box must enclose all high and one density class regions and can be computed efficiently from the primitives of the original CSG object. The definition of our BSP tree guarantees that all unsmoothed parts of the object's surface lie on a partitioning plane; that is, a halfspace plane of the original CSG object.

For each BSP node, the candidate polygon is pushed down the IN and OUT trees to find the bits of it facing a high cell on their inside and a low cell on their outside. Note that it is sufficient to push the face down the subtrees because the candidate face is obtained from a bounding box that has al-

```
function InsertCSGinBSP(obj:CSGObject, tree:BSPTree, op:SetOperation):BSPTree
    /* Insert a GSG object into a BSP tree using a given set operation */
    if tree is not a leaf
        (inObj,outObj)=SplitCSGObj(tree.plane, obj)
        inTree = InsertCSGinBSP(inObj, tree.inTree, op)
        outTree = InsertCSGinBSP(outObj, tree.outTree, op)
        return MakeNode(tree.plane, inTree, outTree)
    else /* tree is a leaf with a density class */
        if op == ∪
            if tree.densityClass == one then return tree /* no change */
            if tree.densityClass == zero then return CSG2BSP(obj)
            return UpdateDensityClasses(CSG2BSP(obj), tree.densityClass, op)
        if op == ∩
            if tree.densityClass == zero then return tree /* no change */
            if tree.densityClass == one then return CSG2BSP(obj)
            return UpdateDensityClasses(CSG2BSP(obj), tree.densityClass, op)
        if op == \
            if tree.densityClass == zero then return tree /* no change */
            return UpdateDensityClasses(CSG2BSP(obj), tree.densityClass, op)
    function UpdateDensityClasses(tree:BSPTree, class:DensityClass, op:SetOperation):BSPTree
    /* Perform for the density classes of all cells of the given tree a set operation with the given density class */
    if tree is a leaf
        then newTree = MakeNode(Apply(op, tree.densityClass, class))
    else
        inTree = UpdateDensityClasses(tree.inTree, class, op)
        outTree = UpdateDensityClasses(tree.outTree, class, op)
        newTree = MakeNode(tree.plane, inTree, outTree)
    return newTree
```

Fig. 7. Insert a CSG object into a BSP tree

```
function TreePolygons(tree:BSPTree, boundingBox:Polyhedron):List of Polygons
    /* Compute all tree polygons, i.e., polygons separating low from high cells, for a given BSP tree and a given
bounding box of its non-zero cells /*
    if tree is a leaf then return ∅
    else
        (inBox, outBox) = SplitBoundingBox(boundingBox, tree.plane)
        candidatePoly = Intersection(boundingBox, tree.plane)
        resultPolys = ∅
        highOnInsidePolys = SelectedBitsOfPoly(candidatePoly, High, inTree)
        for each polygon p in highOnInsidePolys
            lowOnOutsidePolys = SelectedBitsOfPoly(p, Low, outTree)
            resultPolys = resultPolys∪lowOnOutsidePolys
        highOnInsidePolys = SelectedBitsOfPoly(FlippedFace(candidatePoly), High, outTree)
        for each polygon p in highOnInsidePolys
            lowOnOutsidePolys = electedBitsOfPoly (p, Low, inTree)
            resultPolys = resultPolys ∪ lowOnOutsidePolys
        polysInInTree = TreePolygons(tree.inTree, inBox)
        polysInOutTree = TreePolygons(tree.outTree, outBox)
        return resultPolys ∪ polysInInTree ∪ polysInOutTree
    function SelectedBitsOfPoly(poly:Polygon, class:DensityClass, tree:BSPtree):List of Polygons
    /* Inserts the poly into the tree and returns all bits that reach a leaf with the specified density class /*
    if tree is a leaf then
        if tree.class == class then return poly
        else return ∅
    else
         (inBit, outBit) = SplitPolygon(poly, tree.plane)
        retainedInBits = SelectedBitsOfPoly(inBit, class, tree.inTree)
        retainedOutBits = SelectedBitsOfPoly(outBit, class, tree.outTree)
        return retainedInBits ∪ retainedOutBits
```

Fig. 8. Extracting tree polygons from a BSP tree

ready been clipped on the partitioning planes of all parent nodes of the current BSP node.
The same process is executed for the flipped candidate face. The density field outside the flipped candidate face is now given by the IN tree and the density field inside the face is given by the OUT tree. After finding the tree polygons of a BSP node, the algorithm is called recursively for the subtrees of the node until a cell is reached.
Note that the density field of a quasiconvolutionally smoothed object is continuous, and therefore no part of the 0.5 isosurface neighbors a zero or a one cell. (This is not true for the final implementation, which achieves a fast clipping of the object by clipping the density field.)

3.4 Subspace polygonization

It is known that the remaining object surface lies inside or on the unclassified cells after the tree polygons have been extracted. An explicit representation of the cell boundaries can be obtained either by a postprocessing step or by maintaining the cell's boundary as an intrinsic component of the BSP tree (Naylor et al 1990). We polygonize the 0.5 isosurface inside an unclassified cell by approximating it as follows:

1. Compute the points on the isosurface where the cell edges intersect the isosurface.
2. Connect the intersection points to form one or more topological polygons.
3. Refine each edge of the polygon(s) by finding an additional point on the isosurface near each edge midpoint.
4. Subdivide the topological polygon(s) into planar polygons.

Figure 9 illustrates this process.

3.4.1 Computing intersection points

We form set of points on the isosurface by computing the intersection points of the edges of every unclassified cell with the 0.5 isosurface. Since we assume a smooth surface, an intersection point exists if the end points of an edge lie on different sides of the isosurface. A rootfinder finds the intersection point using a regula falsi method with an interleaved binary search.

3.4.2 Connecting intersection points

Having determined the intersections of all edges of the cell with the isosurface, we approximate the intersection of the isosurface with faces of the cell by connecting appropriate pairs of intersection points by straight lines lying in the faces of the cell.
Since each BSP tree contains arbitrarily shaped convex polyhedral cells, the number of isosurface intersection points with a given face is unlimited. For more than two intersection points, the connection is ambiguous. Observe that only neighboring intersection points can be connected (otherwise the isosurface would be self-intersecting or folded) to resolve ambiguities. The density class of the centroid of the intersection points belonging to a cell face is used to resolve the ambiguity. A pair of consecutive intersection points around a face is connected by a polygon edge if and only if the density class of the centroid differs from that of the cell face vertices lying between the intersection points. The direction of the edge is chosen so that points inside the isosurface have a high density value.
This is illustrated in Fig. 10a and b. Figure 10c shows that this approach, like all polygonization methods, can, in principle, yield the wrong topology. However, the inherent smoothness of the quasiconvolutionally smoothed density fields makes such an outcome very unlikely.
Once all edges have been determined for the isosurface through a cell, the edges are connected to form one or more topological polygons. These polygons approximate the isosurface intersection with the cell.

3.4.3 Refining edges

As can be seen from Fig. 3, a single BSP tree cell typically encompasses the entire curvature of a rounded edge. As described so far, the algorithm would replace a simple rounded edge with a single polygon. A better approximation is clearly desirable. Surprisingly, we have found that just two polygons, Gouraud shaded, generally produce an entirely acceptable effect. We subdivide each edge of a topological polygon into two, introducing a new vertex that lies on the isosurface. The new point is initially created at the midpoint of the edge and is then displaced along the line of the density field gradient until the isosurface is found (Fig. 11).

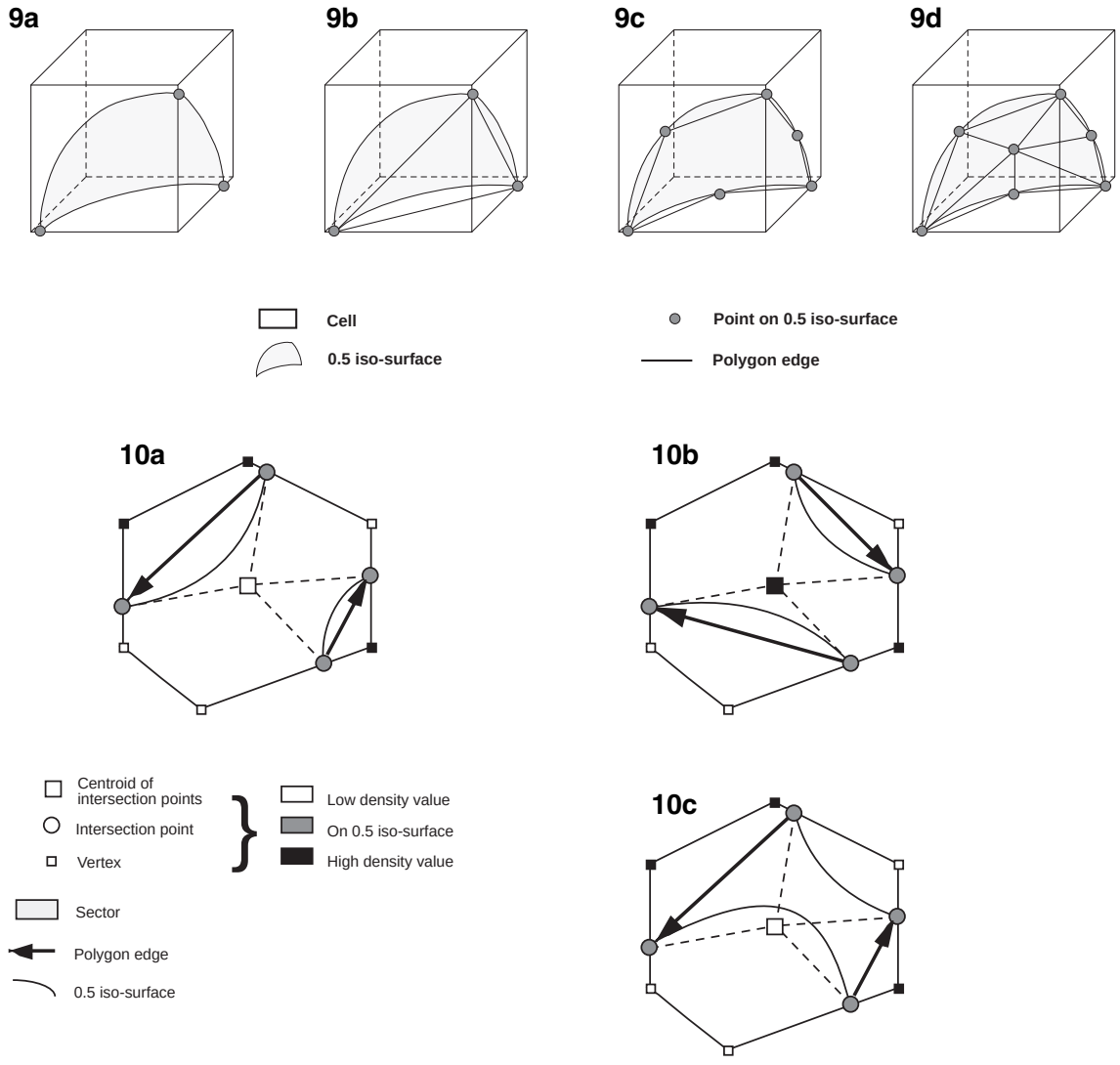


Fig. 9a–d. The subspace is polygonized in four steps: **a** the isosurface intersections with the edges are found; **b** the intersections are connected to create one or more polygons; **c** the polygon edges are refined by adding new vertices; **d** the topological polygon(s) are subdivided into planar polygons

Fig. 10a–c. Using the centroid of the intersection points to resolve edge connection ambiguities

Since the density gradient in p_{mid} usually does not lie in the face plane, we take instead its projection $\nabla_{proj_F}\rho$ on the face given by

$$\nabla_{proj_F}\rho = \nabla\rho - (n_F \cdot \nabla\rho)\, n_F,$$

where n_F is the face normal.
We intersect the line along the density field gradient with the face edges and compare the density classes of the intersection points s_{start} and s_{end} with the density class of p_{mid} (Fig. 11a). The two lines $\overline{s_{start}p_{mid}}$ and $\overline{s_{end}p_{mid}}$ are candidates for a root search. If either s_{start} or s_{end} is on the side of the isosurface opposite to p_{mid}, we search for a root between that point and p_{mid} (Fig. 11b). If both the former points are on the side of the isosurface opposite to p_{mid}, we search in the direction of the density field gradient. Otherwise we assume that

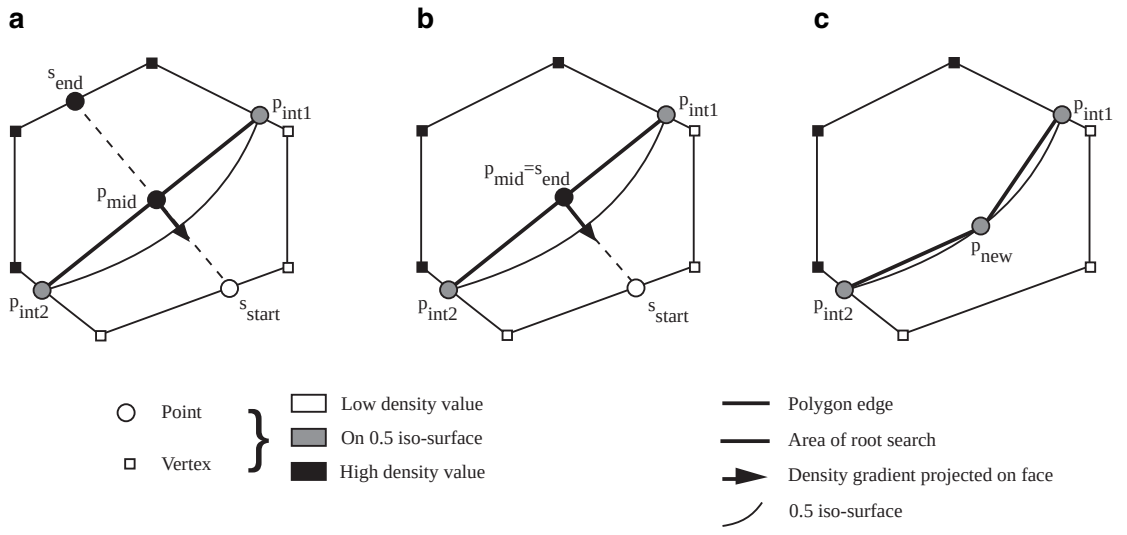


Fig. 11. The line along the density field gradient through p_{mid} (**a**) defines a linear search space for the edge refinement (**b**). The refined edge is shown in (**c**)

no isosurface intersection exists and do not refine the edge.

3.4.4 Flattening topological polygons

The last step of the subspace polygonization divides each topological polygon into planar polygons by connecting each vertex to the centroid of the topological polygon, thus triangulating the topological polygon. We can improve the polygonal approximation of the isosurface by moving the centroid in the direction of the density gradient until the 0.5 isosurface intersection is found (Fig. 9). This again is a root search. The search space is restricted to the volume of the cell to ensure that the approximation to the isosurface stays within the cell.

3.5 Improvements

3.5.1 Local density dield

The subspace polygonization involves repeated evaluation of the density field at points inside the unclassified cell. To make this calculation more efficient, a local density field is defined for each unclassified cell. This is effectively a pruned version of the original quasiconvolutionally smoothed density field, involving only those halfspaces that have a varying contribution to the densities within the cell. The local density field can be computed during the computation of density classes with a table similar to Table 1. We found that the local density fields usually have a constant size, whereas the global density fields grows linearly with the size of the object.

3.5.2 Intersection of two halfspaces

We inspected the results of our polyhedral subdivision algorithm and found that on average 30% of the unclassified cells contained a local density field from the intersection of two halfspaces. It can be shown (Wünsche 1996) that the density field is then a convex swept surface. Figure 12 shows that an improved subspace polygonization is given for these cells as part of the convex hull of all isosurface intersection points. An efficient algorithm that finds polygons of maximum size is given in Wünsche (1996).

3.5.3 Variable rounding radius

Another desirable feature is to define different rounding radii for the different edges of an object.

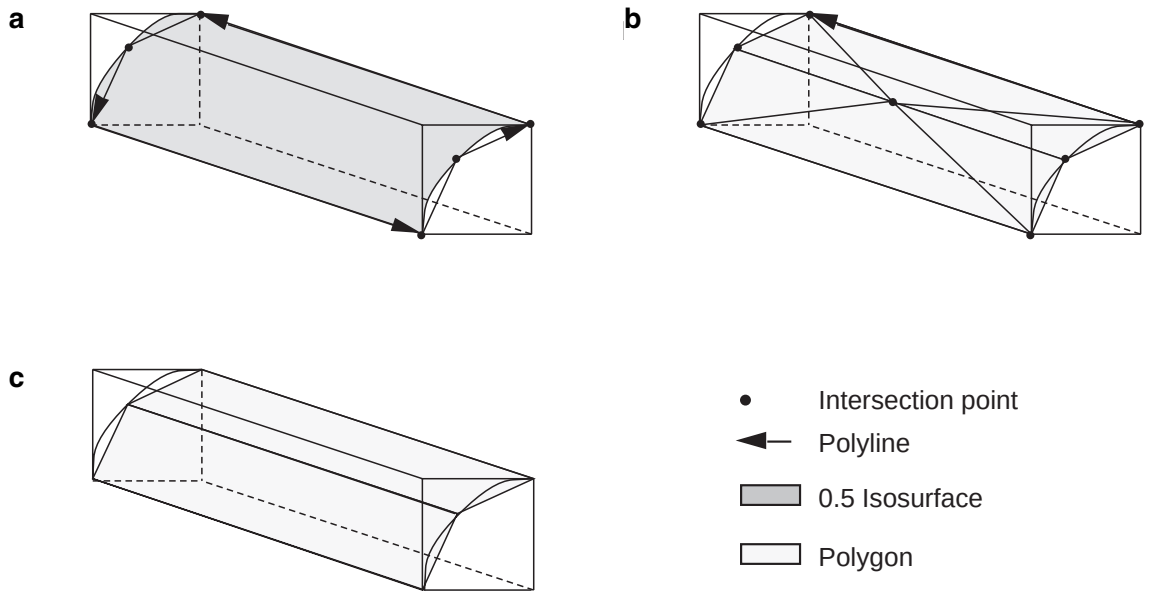


Fig. 12. **a** Cell containing a quasiconvolutionally smoothed intersection of two halfspaces; **b** the subspace polygonization leads to fragmentation; **c** the desired result of the subspace polygonization

Though this effect is not possible for quasiconvolutionally smoothed objects, we can often obtain a similar effect by defining different smoothing radii for the halfspaces that form a polyhedral primitive. This technique was used to produce the cylindrical metal pins with smoothly flattened ends in Fig. 13 (see also the enlargements).

4 Rendering

The vertex normals of the polygons are required for Gouraud or Phong shading. The vertex normals of tree polygons, which always describe a planar area of the isosurface, are given simply by the surface normal of the polygon. The vertex normal $\vec{n}_{p_i}$ of a vertex p_i of a subspace polygon is given by the gradient of the density field at the vertex

$$\vec{n}_{p_i} = -\frac{\nabla\rho(p_i)}{\|\nabla\rho(p_i)\|}.$$

We compute the density gradient by differentiating the arithmetic tree defining the density field (see Sect. 2).

5 Results

We implemented our algorithm in the functional language CLEAN 1.0 on a Power Macintosh 9500/120. The polygonized scenes were rendered with QUICKDRAW3D.

5.1 Images

Figure 13a and c shows a hole punch modeled as a simple unsmoothed CSG object with polyhedral primitives. By specifying a few rounding radii, we obtain a much better looking smoothed hole punch as shown in Fig. 13b and d. The base of the hole punch is rounded with a smoothing radius considerably smaller than the base itself. As a result, triage polygonization extracts most of the object's surface as large rectangles. A smoothed edge and a corner are represented with two long rectangles and six triangles, respectively. The Gouraud shaded picture in Fig. 13d shows that the produced polygons are sufficient to achieve the visual impression of a smoothed surface.
The enlargements of Fig. 13 depict the punch, part of the hinges, and some metal pins in detail. The

Fig. 13. The unsmoothed (**a, c**) and smoothed (**b, d**) "hole punch" scene. The results are shown as wire-frame representations (**a, b**) and Gouraud shaded (**c, d**)

Fig. 14. The unsmoothed (**a**) and smoothed (**b**) "stapler" scene. The results are shown as wire-frame representations (**a, b**) and Gouraud shaded (**c, d**)

punch is modeled as a rounded cuboid. Note that the punch pin has a sharp edge at the top. We achieve this very easily by applying a clipping plane to the rounded cuboid. We achieve an efficient implementation by directly clipping the density field of an object before polygonization. Details are given in Wünsche (1996).

The two metal pins at the bottom right corner of the enlargement are constructed from halfspaces with different rounding radii. This gives the impression of a cylinder with a smoothly flattened end. Observe that the cylindrical part of a smoothed metal pin is approximated with rectangles, whereas the more complicated end of a pin is represented by triangles.

A model of a stapler is shown before (Fig. 14a, c) and after (Fig. 14b, d) applying our algorithm, respectively. Note that, wherever possible, the polygonization method finds long triangles and rectangles. It can also be seen that very thin objects, such as the side plates of the hinge, are polygonized without problems.

Figure 15 shows a scene modeled as a union of six objects, each derived by applying various combinations of rounding operations, and a set operation to a cube and a small cuboid. Two interesting cases are shown as enlargements. The top enlargements of both parts of the figure depict a clipped quasiconvolutionally smoothed small cube subtracted from a bigger unsmoothed cube. We model set operations on polygonized objects by merging BSP trees (Thibault and Naylor 1987; Naylor et al 1990).

The bottom enlargements of Fig. 15a and b give an example of a concave three plane corner. The corner results from a quasiconvolutionally smoothed union of a big cube and a small cuboid. It can be seen that the corner is nicely polygonized with only 26 triangles.

5.2 Comparison with the marching cubes algorithm

The marching cubes algorithm is a popular method for implicit surface polygonization, and it provides a good basis for comparison with our new method. To achieve comparable visual quality, we applied the marching cubes algorithm with a grid size of half the rounding radius of the quasiconvolutionally smoothed scene.

Figure 16 shows the results of both algorithms for a "variable radius" scene, which shows an object constructed as a set difference of a cube and a small cuboid smoothed with several different rounding radii. The object in Fig. 16a and c was polygonized with triage polygonization, whereas, for the object in Fig. 16b and d, the marching cubes algorithm was used. Note that our algorithm achieves a good polygonization for all objects, independently of the rounding radius. The polygonization for the objects with small rounding radius can be considered optimal. For the objects rounded with a rather large smoothing radius, some "bands" are visible where the object is polygonized more finely. This is due to small cells in the polyhedral subdivision of the density fields defining the quasiconvolutionally smoothed objects.

We found that triage polygonization is about 20–30 times faster than the marching cubes algorithm on average and it outputs only a fraction (≈1%–2%) of its number of polygons. The results also confirm that triage polygonization produces arbitrarily complex polygons of vastly different sizes, whereas the marching cubes algorithm produces only about equally sized triangles. Also note that triage polygonization yields the b-rep of the unsmoothed object for a rounding radius of zero, whereas the marching cubes algorithm cannot polygonize sharp edges at all.

The marching cubes algorithm becomes superior if the rounding radius reaches about a quarter of the object size (the middle object of the bottom row in Fig. 16b and d). In that case, however, the object no longer fulfills our design objective that it is predominantly planar.

5.3 Complexity

The complexity of triage polygonization is governed by the subspace polygonization step, which is a binary space partition. We have made measurements on several scenes that suggest an average time complexity of about $O(n^{1.3})$, where n is the number of halfspaces in the unsmoothed CSG object. Using the BSP tree algorithm of Naylor et al. (1990) could improve this result to $O(n \log n)$. Note that this complexity is quite different from that for a conventional polygonization method, such as the marching cubes algorithm, for which the time complexity is $O(m^3)$, where m is the sample resolution.

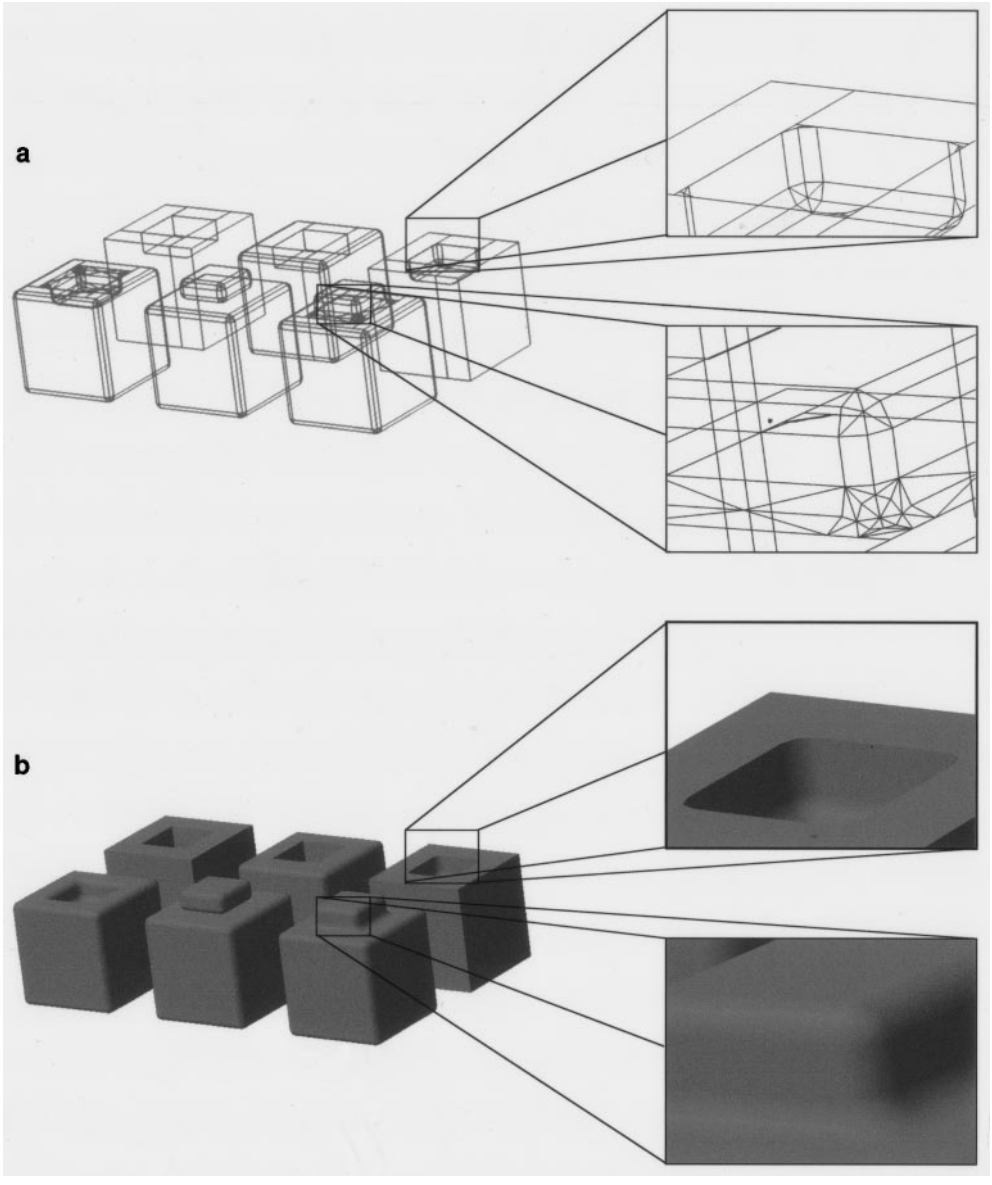


Fig. 15. The "CSG example" scene polygonized with triage polygonization. The result is shown as a wire-frame representation (**a**) and Gouraud shaded (**b**)

In principle, our algorithm could become less efficient if a single complex object were being rounded or if a low sampling resolution were used. However, rounding is usually applied to fairly simple objects that are components of more complex objects. Low sampling resolution cannot generally be used, since fine structures or high-curvature edges cannot then be represented.

5.4 Known problems

As with all geometric algorithms, numerical robustness is an issue. The subspace polygonization stage depends on classifying cell vertices as above or below the isosurface, which leads to the question of how to treat vertices that lie directly on the isosurface. This situation is rare with a marching cubes algorithm, but common with ours, since

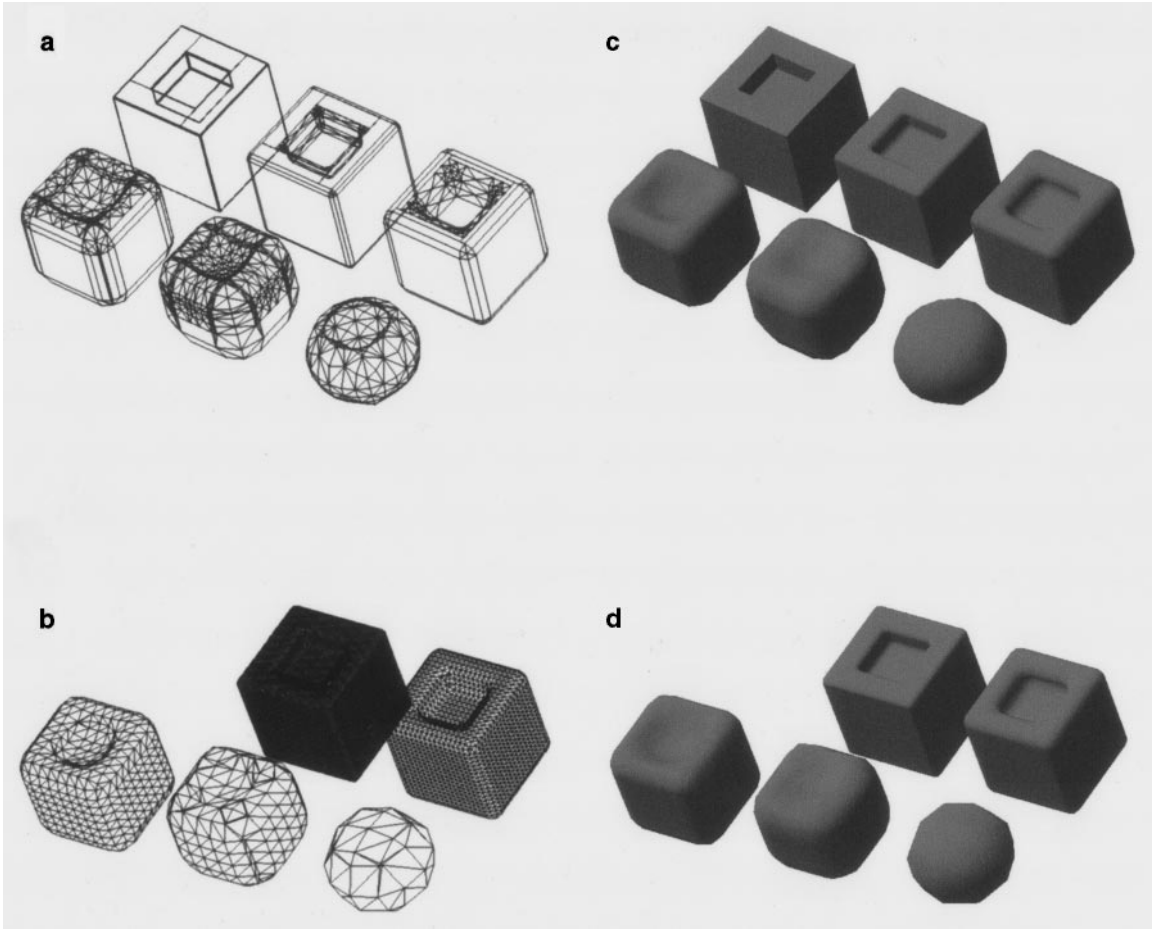


Fig. 16. Triage polygonization and the marching cubes algorithm applied to the "variable radius" scene: the result of triage polygonization as a wire-frame representation (**a**) and in Gouraud shading (**c**), the same representation for the result of the marching cubes algorithm (**b, d**)

our partitioning planes lie on the unsmoothed object's faces. Extending the algorithm to handle such vertices as special cases is more complex and requires maintaining a honeycomb property. (A honeycomb is a polyhedral subdivison of space in which each internal face of each polyhedral cell is entirely shared by exactly one other cell.) This requires a data structure, such as the hash table used by Wyvill et al. (1986), that enforces sharing of edge and face information. Unfortunately, the language we have used for our implementation, CLEAN 1.0, lacks a working array data type or pointers, and we have been unable to implement complete sharing of face and edge information.
We have chosen, instead, largely to avoid the problem by classifying vertices against a displaced $0.5 + \varepsilon$ isosurface. We determine which edges intersect the displaced isosurface, but then compute the actual 0.5 isosurface during root searching. This strategy can lead to holes in the isosurface in certain cases, but such cases are easily identified, and the missing polygons can be generated in a postprocessing step. We use a value of 0.001 for ε. Details, and the more general issue of ensuring continuity of the polygonized isosurface, are discussed at length in Wünsche (1996).

6 Conclusion

This paper has presented triage polygonization, a new polygonization method specifically designed

for quasiconvolutionally smoothed objects. Triage polygonization performs best for quasiconvolutionally smoothed objects smoothed with a rounding radius small in comparison to their size. Such objects have predominantly planar surfaces with only edges and corners rounded. Triage polygonization extracts planar surfaces by means of a BSP tree with minimal fragmentation and approximates most rounded edges and corners with a nearly minimal number of polygons. For such objects, triage polygonization is superior to all general polygonization methods for implicit surfaces known to us.

Triage polygonization also performs well for strongly rounded objects and in such cases its performance is similar to general polygonization methods.

Triage polygonization is invariant under affine linear transformation and the quality of the polygonization is independent of the rounding radius (if it is reasonably small).

7 Future work

As mentioned in Sect. 5.4, several problems still exist with our method. We would like to implement the algorithm in an imperative language (C/C++), using a data structure that allows us to generate a BSP tree modified to maintain a honeycomb property.

We should be able to achieve an even better quality of polygonization by making the refinement process for edges and topological polygon dependent on the curvature of the surface. An adaptive refinement process similar to that suggested by Bloomenthal (1988) could be employed.

We have designed triage polygonization to polygonize quasiconvolutionally smoothed objects. However, it should be adaptable to other polyhedral smoothing schemes based on implicit surfaces. In particular, we would like to investigate its use with true convolutional smoothing (Dansted 1997).

References

Bloomenthal J (1988) Polygonization of implicit surfaces. Comput Aided Geom Des 5:341–355

Bloomenthal J (1994) An implicit surface polygonizaer. In: Heckbert PS (ed) Graphics Gems IV, Chap. 8. Academic Press, Cambridge

Colburn S (1990) Solid modeling with global blending for machining dies and patterns. Proceedings of the 41st Annual Earthmoving Industry Conference, SAE technical paper series, SAE International, Warrendale, Pa

Dansted PJ (1997) Convolutional smoothing of polyhedra. Masters's Thesis, Department of Computer Science, University of Auckland, Auckland, New Zealand

Doi A, Koide A (1991) An efficient method of triangulating equivalued surfaces by using tetrahedral cells. IEICE Trans Commun Elec Inf Syst E-74:214–224

Duff T (1992) Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. SIGGRAPH Comput Graph 26:131–138

Fuchs H, Kedem Z, Naylor BF (1980) On visible surface generation by a priority tree structure. SIGGRAPH Comput Graph 14:124–133

Gieng TS, Hamann B, Joy KI, Schussman G, Trotts IJ (1998) Constructing hierarchies for triangle meshes. IEEE Trans Visualization Comput Graph 4:145–161

Hall M, Warren J (1990) Adaptive polygonization of implicitly defined surfaces. IEEE Comput Graph Appl 10:33–42

Hinker P, Hansen C (1993) Geometric optimization. Comput IEEE Proceedings of Visualization '93, pp 189–195

Hoschek J, Lasser D (1992) Fundamentals of Computer Aided Geometric Design, Chapt. 14, pp 592–601, AK Peters Ltd, Wellesley, MA 2nd edition

Kalvin AD, Taylor RH (1996) Superfaces: polygonal mesh simplification with bounded error. IEEE Comput Graph Appl 16:64–77

Lobb R (1996) Quasiconvolutional smoothing of polyhedra. Visual Comput 12:373–389

Lorensen W, Cline H (1987) Marching cubes: a high resolution 3D surface construction algorithm. Comput Graph (SIGGRAPH'87 Proceedings), 21:163–169

Naylor BF (1981) A priori based techniques for determining visibility priority for 3D scenes. PhD Thesis, University of Texas, Dallas, Tex

Naylor BF, Amanatides J, Thibault W (1990) Merging BSP trees yields polyhdral set operations. SIGGRAPH Comput Graph 24:115–124

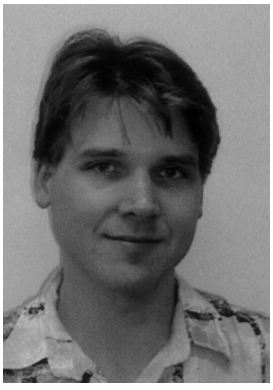
Snyder JM (1992) Interval analysis for computer graphics. SIGGRAPH Comput Graph 26:121–130

Thibault WC, Naylor BF (1987) Set operations on polyhedra using binary space partitioning trees. Comput Graph (SIGGRAPH'87 Proceedings), 2:153–162

Wyvill G, McPheeters C, Wyvill B (1986) Animating soft objects. Visual Comput 2:235–242

Wünsche BC (1996) A fast polygonization method for quasiconvolutionally smoothed polyhedra. Master's Thesis, Department of Computer Science, University of Auckland, Auckland, New Zealand

Wünsche BC (1997) A survey and analysis of common polygonization methods & optimization techniques. Mach Graph Vision 6:451–486



BURKHARD WÜNSCHE received a Vordiplom in Computer Science from the University of Kaiserslautern, Germany, in 1993 and an MSc in Computer Science from the University of Auckland, New Zealand, in 1996. He is currently a PhD student in Computer Science at the University of Auckland, working on visualization methods for use with bioengineering models. His current research interests include computer graphics, scientific visualization, geometric modeling, and bioengineering.

RICHARD LOBB is a Senior Lecturer in the Department of Computer Science at the University of Auckland, New Zealand. His research interests are in computer graphics and visualization. He received an MSc in Physics in 1970 and a PhD in Radio Science in 1975, both from the University of Auckland.