Polygonization of Quasi-Convolutionally Smoothed Polyhedra

Burkhard Wünsche and Richard Lobb
University of Auckland

April 7, 1997

Abstract

The smoothing of a polyhedral model is an important task in Computer Graphics. Richard Lobb [Lob96] introduced quasi-convolutional smoothing, a fast rounding scheme approximating convolutional smoothing. For a fast interactive display of the model its surface must be polygonized. We introduce here Triage Polygonization, a new fast polygonization method for quasi-convolutionally smoothed polyhedra. The polygonization method exploits the property that quasi-convolutionally smoothed polyhedra usually have predominantly planar surfaces with only edges and corners rounded.

A quasi-convolutionally smoothed polyhedron is represented implicitly as a density field isosurface. Triage Polygonization subdivides the density field in a BSP-like manner and classifies the resulting cells as inside, outside, or intersected by the isosurface. Planar surface areas usually lie on the boundary of cells and are extracted directly from the subdivided density field with minimal fragmentation. For cells intersected by the isosurface a more general polygonization is performed. For quasi-convolutionally smoothed scenes with a small rounding radius Triage Polygonization is 20–30 times faster and outputs only 1–2% of the polygons of the Marching Cubes algorithm without compromising the approximation. The approach taken for Triage Polygonization can be extended to related problems.

1 Introduction

A large proportion of computer graphics scenes are modeled with polyhedra. However, natural polyhedra usually have smoothed edges and corners connecting adjacent planes. Many methods exist to replace sharp edges and corners with rounded surfaces, but most prove computationally difficult. Lobb [Lob96] introduced quasi-convolutional smoothing as a solution. In his method polyhedral objects are represented by CSG-like structures with arithmetic operators as internal nodes rather then set membership operators. An object is rounded by approximating the process of true convolutional filtering. The smoothed object's surface is defined as an isosurface in a density field.

Quasi-convolutionally smoothed objects have some special properties. The most important is that they usually consist of large planar surfaces connected by relatively small areas of smooth curved surfaces.

In Lobb's original work scenes were rendered by ray-tracing. The results are good but may easily need several hours computation time. Especially during the modeling process this is not acceptable. A solution is to polygonize the scene. Existing polygonization algorithms, however, do not exploit the special properties of quasi-convolutionally smoothed objects and hence lead to fragmentation and unnecessarily long computation time.

Triage Polygonization is a fast, high-quality polygonization method especially designed for quasi-convolutionally smoothed objects. It first generates a special BSP-like space subdivision of the object. Potential regions of curvature are identified and most planar regions of the rounded object are extracted in a subsequent step. A subspace polygonization process polygonizes the remaining parts of the object surface. The method is fast and results in only a small fragmentation of the surface.

2 Quasi-convolutional smoothing

2.1 Rounding Methods

Real world objects that are approximately polyhedral usually have rounded edges and corners. In modeling such objects, the rounded surfaces are not usually functionally important. In CAD/CAM technology these surfaces are called *blends*. The principle difficulty is to shape and position blends so as to achieve tangency to primary (unrounded) surfaces. A good overview of blending methods is given by Hoschek and Lasser [HL92].

Most existing blending methods are complex and often require user interaction. Furthermore they have practical limitations when more than three surfaces influence the shape of the blend at any given point. Colburn [Col90] presents a solution by introducing a spherical test volume with a radius equal to the desired blend radius. The surface of the smoothed object is 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 1 shows the two dimensional equivalent: a polygon and its smoothed version.

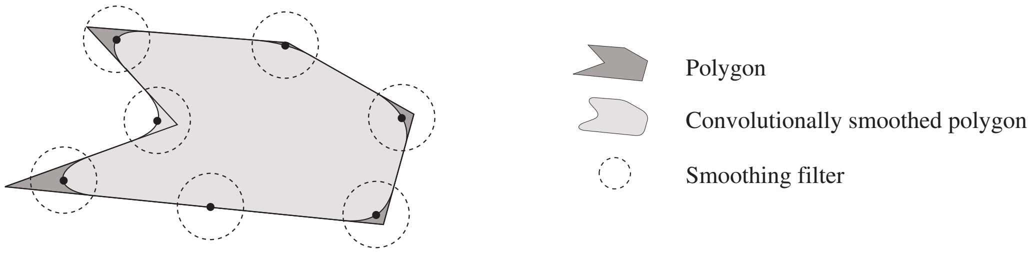

Figure 1: A convolutionally smoothed polygon.

Note that now the shape of the blend at any given point is influenced only by the portion of the unblended model that falls inside the sphere. The process can be understood as a low pass filtering with a spherical filter of radius r and is mathematically expressed by a convolution.

2.2 Quasi-convolutional smoothing

Though the true convolutional smoothing process is easy to understand and mathematically simple, it is computationally expensive to find points on the object surface. To render or to polygonize the smoothed object a large number of surface points must be found. Also for a non-convex object the surface of the smoothed volume can lie outside the volume. This makes it difficult to know where to search for the surface of the rounded object.

Lobb addresses these problems with an approximation to convolutional smoothing called quasi-convolutional smoothing. Quasi-convolutional smoothing requires that a polyhedral object be represented as a Constructive Solid Geometry (CSG) tree whose internal nodes denote regular set operations [Req80] and whose leaves are convex polyhedral solids. The convex polyhedral primitives are themselves regarded as intersections of halfspaces defined by all the polyhedral faces (Figure 2). The whole object is defined by applying the set operations recursively to the child objects. Lobb defines a global rounding radius r for the polyhedron and replaces the set operations union, intersection, and set difference by the arithmetic operations addition, multiplication, and subtraction. He then convolves only the leaf halfspaces with a spherical filter of radius r.

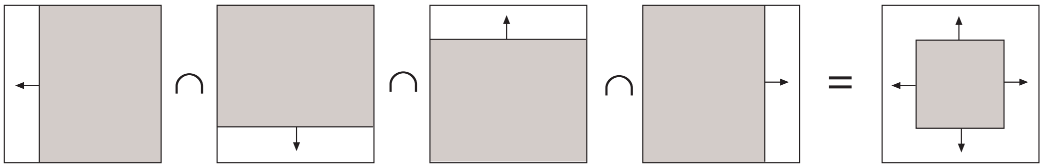

Figure 2: A square modeled as the intersection of four halfspaces. In 3-space six halfspaces define a cube.

The *density field* $\rho^r_H(p)$ of a smoothed halfspace H^r is defined as the convolution of the halfspace with a spherical filter of radius r. The halfspace is regarded as a density field with a value 1 inside the halfspace and a value 0 outside H. The value of the convolution at any point p is then equal to the volume of the sphere centered at p intersected with the halfspace, for which the answer is

$$\rho^r_H(p) = \begin{cases} 0 & \alpha \geq 1 \\ 1 & \alpha \leq -1 \\ (1-\alpha)^2 * (2+\alpha)/4 & \text{otherwise} \end{cases} \tag{1}$$

$$\text{where } \alpha = \frac{d}{r}$$

and d is the distance of point p to the halfspace H.

The density field $\rho^r_{obj}(p)$ of a quasi-convolutionally smoothed CSG object can than be defined as

$$\rho^r_{Obj}(x) = \begin{cases} \rho^r_H(p) & \text{if obj is a halfspace } H \\ \rho^r_A(x) + \rho^r_B(x) & \text{if } obj = A \cup B \\ \rho^r_A(x) * \rho^r_B(x) & \text{if } obj = A \cap B \\ \rho^r_A(x) - \rho^r_B(x) & \text{if } obj = A \setminus B \end{cases} \tag{2}$$

where "\" denotes the set difference.

Finally the surface of the quasi-convolutionally smoothed object Obj^r is all points $x \in R^3$ for which

$$\rho^r_{Obj}(x) = 0.5$$

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, and
2. the set difference can be applied only to objects that completely intersect.

For a discussion of these constraints and other properties of quasi-convolutionally smoothed objects refer to [Lob96].

Figure 3 shows in (a) the product of two unsmoothed halfspaces. The resulting density field has a density of one for all points inside both halfspaces and a density of zero otherwise. Part (b) of the figure shows the product of two convolutionally smoothed halfspaces. The 0.5 isosurface of the resulting density field (the black line) forms a quasi-convolutionally smoothed version of the object in (a). The surface is virtually indistinguishable from that obtained by true convolutionally smoothing. For a discussion of the properties of quasi-convolutional smoothing see the original paper [Lob96].

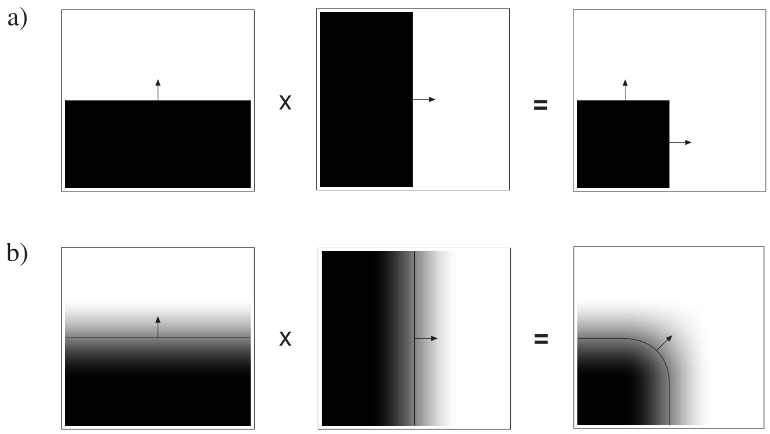

Figure 3: The product (intersection) of two unsmoothed (a) and smoothed halfspaces (b).

3 Polygonization Methods for Implicit Surfaces

Our polygonization problem is to approximate the isosurface of the density field of the quasi-convolutionally smoothed polyhedron. Many published methods exist for finding a polygonal approximation to such an implicitly defined surface. A practical introduction to polygonization methods for implicitly defined surfaces is provided by Bloomenthal who also gives working C code [Blo94]. Ning and Bloomenthal give a good evaluation of polygonization algorithm [NB93]. Gelder and Wilhelms [vGW94] give a thorough discussion of design objectives of isosurface algorithms, isosurface generation and solving of ambiguities.

The well known Marching Cubes method of Lorensen and Cline [LC87] involves creating an array of cubes and evaluating the density field at each vertex. For each cube that contains the isosurface the method constructs a polygonal approximation to the surface, using a precomputed table of 15 topologically distinct high-low patterns of cell vertices. This table lookup method is devised for speed, at the occasional expense of a correct topology. In the original implementation the authors did not recognize ambiguities.

Düurst [Düu88] showed that this could yield a discontinuity between cells.

Wyvill et al. [WMW86] also use a cubical cell array. Their method disambiguates by using the facial average value.

Ambiguities can be resolved implicitly by decomposition into simplices. This approach is taken by Koide et al. [KDK86] and Doi and Koide [DK91]. They decompose a cell into tetrahedra and interpolate linearly on each tetrahedral edge.

All the above methods are fast but assume no knowledge about the implicitly defined surface. In order to detect fine surface detail the above methods need to use a fine sampling grid, which results in a high fragmentation of planar surface areas.

Adaptive methods give less fragmentation. Bloomenthal [Blo88, BW90] extends the method of Koide et al. to an adaptive subdivision based on cubes. Hall and Warren [HW90] use an adaptive subdivision technique which maintains a tetrahedral honeycomb at all times. Adaptive sampling means that in areas of high curvature the sampling rate is increased. However, in order not to miss small surface details a fine initial mesh is still required.

Alternatively a fine sampling grid can be used and then a mesh optimization method can be employed to find large planar or nearly planar surface areas (e.g. [HH93, KT96]). Montani et al. [MSS94] uses a discretized Marching Cubes algorithm such that nearly planar faces are coplanar and can be easily merged. However, it is clear that it is more efficient to extract directly large planar surface areas.

Our approach solves these problems by detecting areas of curvature from the definition of the quasi-convolutionally smoothed objects. Planar surface areas are extracted without fragmentation, which results in a higher speed and far fewer polygons as compared to the above methods.

4 Triage Polygonization

4.1 Overview of Triage Polygonization

Triage Polygonization is defined in three steps, illustrated in Figure 4. The figure shows a simple scene constructed by quasi-convolutionally smoothing the set difference of a large and a small cube. The resulting object is shown in part (h) of the figure.

The first step of Triage Polygonization is a polyhedral subdivision of the density field into regions completely inside and completely outside the isosurface (low and high cells, respectively). Some regions can not be classified and might be intersected by the isosurface. They are called unclassified cells. Figure 4 (a) – (c) show the low, high, and unclassified cells, respectively, of the resulting subdivision. Note the small size of the unclassified cells in part (c) of the figure as compared to the size of the object. Only for these cells must an implicit surface polygonization be performed.

The second step of Triage Polygonization is the extraction of *tree polygons*, which separate low from high cells. These are shown in (d). Back facing polygons are illuminated only with an ambient light source and are hence shaded in dark red.

The third and last step of Triage Polygonization, the *subspace polygonization*, is the polygonization of the density field inside unclassified cells The resulting *subspace polygons* are shown in (e).

The complete polygonization output by Triage Polygonization is given in (f) flat shaded, in (g) as a wire-frame representation and in (h) Gouraud shaded. In Figure 4 (g) and in all subsequent wire-frame representations the back-facing polygons are removed.

4.2 Polyhedral Subdivision of Space

The density field ρ_H^r of a quasi-convolutionally smoothed halfspace partitions $\mathbb{R}^3$ into four parts (Figure 5). We denote with h 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 $-r$, as shown in the figure. Then for all points outside h_r the density field is constant zero. Similarly inside h_{-r} the density field is constant 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 all points on h_{-r} have a density value of one, all points on h_r have a density value of zero, and all points on h have a density value of 0.5.

Figure 6 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,

Figure 4: The three steps of Triage Polygonization: first the polyhedral subdivision partitions a density field into low cells (a), high cells (b), and unclassified cells (c). The second step extracts tree polygons (d), which separate low cells and high cells. Finally subspace polygons are obtained by applying a subspace polygonization to the unclassified cells in (c). The final polygonization is shown flat shaded in (f), as a wire-frame representation with removed back-faces in (g), and Gouraud shaded in (h).

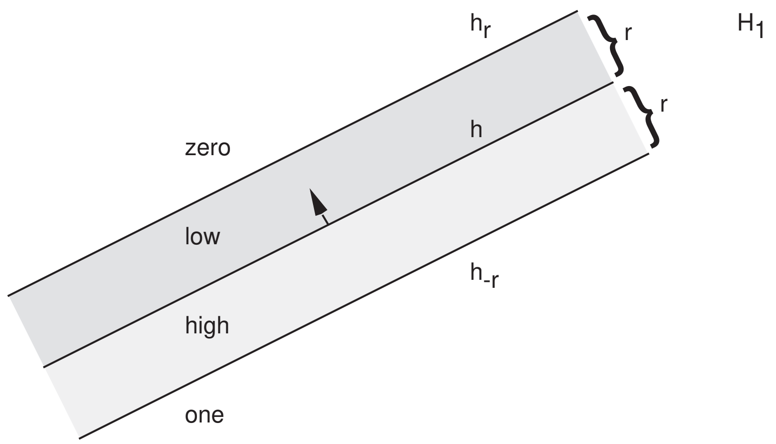

Figure 5: A quasi-convolutionally smoothed halfspace partitions the euclidean space into four regions.

respectively[1].

According to equation 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 $h_{2,-r_2}$. Whole regions can be classified in a similar manner as follows.

4.2.1 Density Classification

In Figure 6 the planes h_1 and h_2 and the corresponding halfspace planes displaced by the rounding radius partition $\mathbb{R}^3$ into 16 regions. By using interval arithmetic [Duf92, Sny92] it is possible to compute the density values for a whole region of the density field. As an example consider region R_1 bounded by the planes h_1, h_{1,r_1}, h_2, and $h_{2,-r_2}$.

[1] We have chosen two different rounding radii for the halfspaces. This extension is explained in section 4.5.

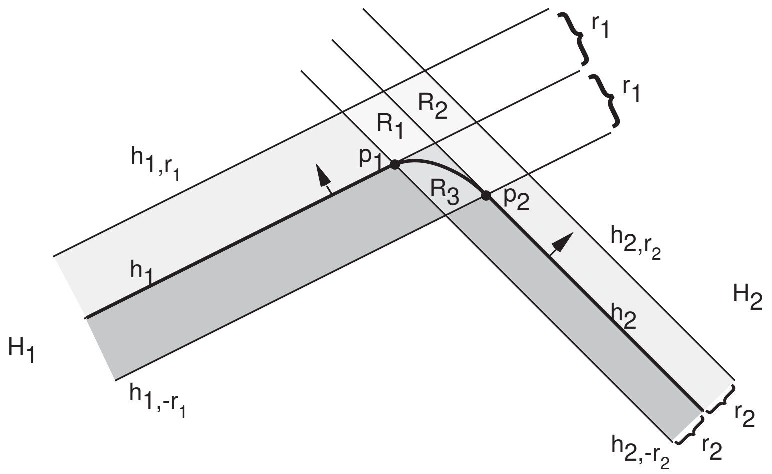

Figure 6: Intersection of two halfspaces.

Since R_1 is bounded by the planes h_1 and h_{1,r_1} the contribution of the density field of $H_1^{r_1}$ to this region varies between zero and 0.5. We express this as

$$\rho^{r_1}_{H_1}(R_1) = (0.0, 0.5)$$

Similarly, since R_1 is bounded by h_2 and $h_{2,-r_2}$, we have

$$\rho^{r_2}_{H_2}(R_1) = (0.5, 1.0)$$

and obtain

$$\rho_{H_1 \cap H_2}(R_1) = \rho^{r_1}_{H_1}(R_1) *_{[]} \rho^{r_2}_{H_2}(R_1) = (0.0, 0.5) *_{[]} (0.5, 1.0) = (0.0, 0.5)$$

where $*_{[]}$ is the multiplication operator for two intervals. Hence all density values in the region R_1 lie in the (open) interval $(0.0, 0.5)$ and therefore the region R_1 lies outside the 0.5 isosurface.

The above interval computations can be simplified by replacing intervals with density classes. We denote with *low* any subinterval of the half-open interval $[0.0, 0.5)$ and with *high* any subinterval of the closed interval $[0.5, 1.0]$. All other intervals are called *unclassified*. An unclassified interval is a subinterval of $[0.0, 1.0]$. For computational efficiency we introduce a *zero* interval

$[0.0, 0.0]$ and a *one* interval $[1.0, 1.0]$. Note that the constraints placed on the CSG object in section 2 ensure that the density field is between 0 and 1 everywhere.

Table 1 summarizes the rules for the multiplication of two density classes. Similar tables can be defined for addition and subtraction.

*	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

Table 1: Multiplication of density classes

Note that by replacing the intervals with density classes some information is lost. However, we found that for our objective (the detection of regions intersected by the 0.5 isosurface) this information loss was insignificant and therefore we preferred this simpler approach.

4.2.2 BSP Trees

A polyhedral subdivision suitable for the polygonization process must identify planar and curved regions of the surface. Figure 5 gave an example of how such a subdivision is achieved for the quasi-convolutionally smoothed intersection of two halfspaces. The process can be generalized for an arbitrary quasi-convolutionally smoothed object.

To do this we first introduce a special type of *Binary Space Partitioning* (BSP) tree to partition and classify a density field into density classes. The fundamental methodology underlying BSP trees is spatial partitioning [FKB80, Nay81]. Planes are used to recursively subdivide $\mathbb{R}^3$ to create a disjoint set of 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.

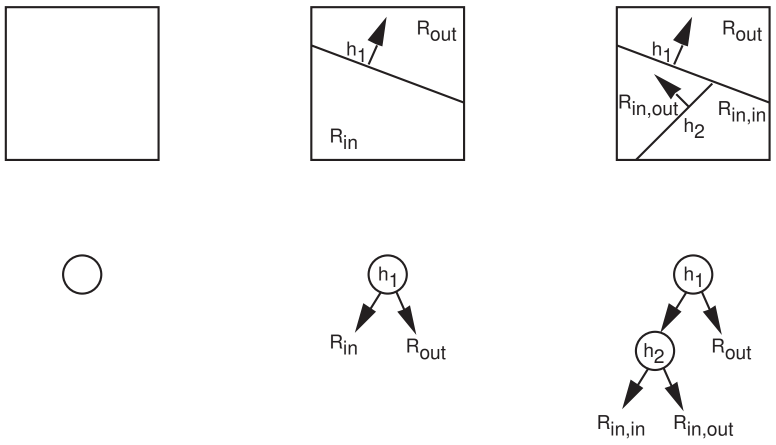

Figure 7: Constructing a BSP tree.

The construction of a BSP tree is illustrated in Figure 7. One begins with a region of space R, chooses some plane h that intersects R, and then uses h to induce a binary partitioning of R. If H_{in} and H_{out} denote the inside and outside open halfspaces of h two new regions are derived: $R_{in} = R \cap H_{in}$ and $R_{out} = R \cap H_{out}$.

Both children can in turn be partitioned, and so on, to produce a binary tree of regions. We make the convention that R_{in} and R_{out} are always the left and right child (*IN tree* and *OUT tree*), respectively, of the current node. We introduce a *density BSP tree*, simply called BSP tree from now on, that differs from the standard definition in that the resulting cells are classified into density classes.

Our polyhedral subdivision algorithm transforms the density field of a quasi-convolutionally smoothed CSG object into a BSP tree. The algorithm, which is similar to the standard algorithm for the conversion of a CSG object into a BSP tree (e.g. [TN87, NAT90]), is discussed in section 4.2.4.

4.2.3 Replacing Polyhedral Primitives with Classified Cells

Since we want to achieve a partitioning of the density field of the whole quasi-convolutionally smoothed CSG object we first partition the density field of the convex polyhedral primitives, which are the building bocks for more complex polyhedra. Every such primitive is replaced by a union of classified cells that together represent the non-zero density field of the primitive. Again the process is best explained by an example.

Figure 8 shows a quasi-convolutionally smoothed object (a) defined as a rounded set difference of two cuboids (b). Each cuboid has a density field (c). In order to subdivide these density fields into non-zero regions we define for every halfspace plane of the primitive object the corresponding inner and outer halfspace planes (d), which partition the density fields of the polyhedral primitives as shown in (e).

A primitive object is an intersection of halfspaces. As with the intersection of two halfspaces in Figure 6, we classify each region with respect to each halfspace into zero, low, high, or one and then compute its density class by multiple applications of table 1. The definition of the partition guarantees that the density class in the centre of a region is equal to the density class of the whole region. Figure 9 gives pseudocode for this algorithm.

Note that a region lying outside any outer halfspace plane has a zero density class. It is therefore 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 (function `ClassifyAgainstAllPlanes`). For example, if at least one density class is zero then the product is zero.

4.2.4 Polyhedral Subdivision with a BSP Tree

Figure 10 gives the algorithm to transform the whole CSG object into a BSP tree (Figure 8 (g)). 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 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 non-primitive 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 using the algorithm of Figure 11.

Figure 8: A quasi-convolutionally smoothed object (a) is defined by a CSG object with a rounding attribute (b). The halfspace planes of the primitive objects are displaced by the rounding radius (d) and are used to subdivide the density field of the corresponding primitive (c). The resulting object (e) can be transformed into a BSP tree (g) classifying the density field of the quasi-convolutionally smoothed CSG object (f).

```
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
      classifiedCell = classifiedCell ∪ (cell,densityClass)
   return classifiedCells
```

Figure 9: 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))
```

Figure 10: 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.

4.3 Tree Polygons

The BSP tree partitions the density field of a quasi-convolutionally 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 and a high cell is 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 [TN87] and is given in Figure 12.

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 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 which has already 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 quasi-convolutionally smoothed object is continuous and therefore no part of the 0.5 isosurface neighbors a zero or a

```
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
```

Figure 11: 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 = SelectedBitsOfPoly ( 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
```

Figure 12: Extracting tree polygons from a BSP tree

one cell[2].

4.4 Subspace Polygonization

After extracting the tree polygons it is known that the remaining object surface lies inside or on the unclassified cells. 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 13 illustrates this process.

4.4.1 Computing intersection points

A set of points on the isosurface is formed by computing for every face of an unclassified cell the intersection points of its edges 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. The intersection point is found by a rootfinder using a regula falsi method with interleaved binary search.

[2] This is not true for the final implementation, where we achieve a fast clipping of the object by clipping the density field.

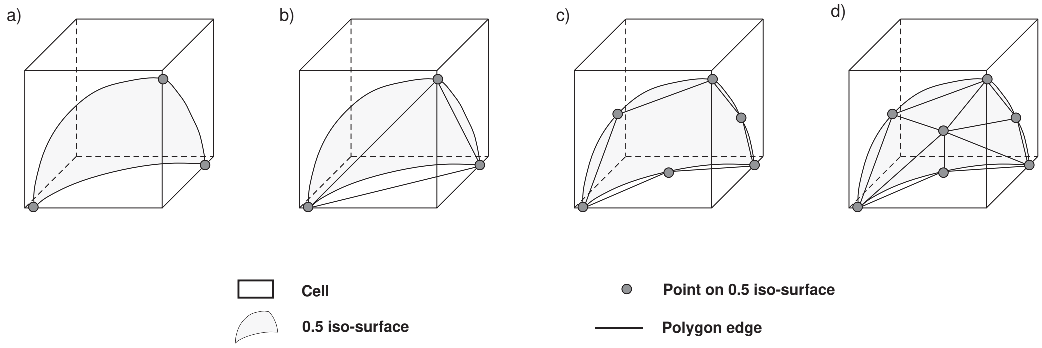

Figure 13: The subspace polygonization is performed in four steps. (a) The isosurface intersections with the edges are found. (b) The intersections are connected to create one or more polygons. (c) The polygon edges are refined by adding new vertices. (d) The topological polygon(s) are subdivided into planar polygons.

4.4.2 Connecting Intersection Points

Having determined the intersections of all edges of the cell with the isosurface, the intersection of the isosurface with cell faces of the cell is approximated 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. To resolve ambiguities observe that only neighboring intersection points can be connected (otherwise the isosurface would be self-intersecting or folded). 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 Figures 14 (a) and (b). Figure 14 (c) shows that this approach, like all polygonization methods, can in principle yield the wrong topology. However, the inherent smoothness of the quasi-convolutionally smoothed density field makes such an outcome very unlikely.

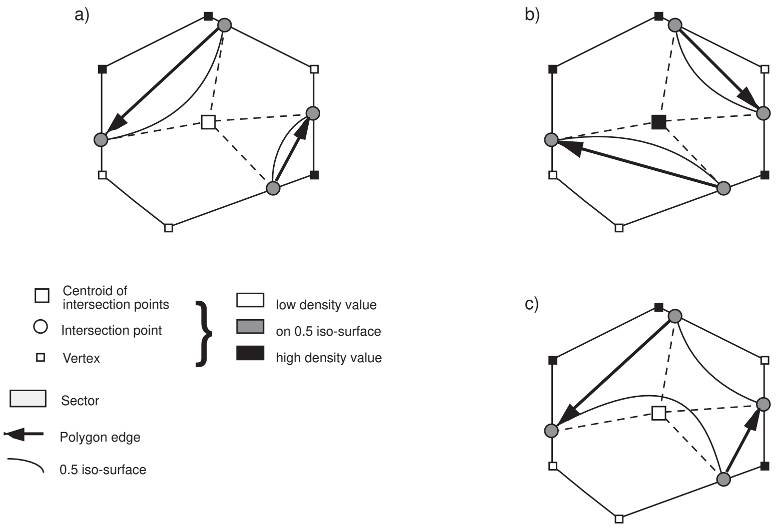

Figure 14: Resolving edge connection ambiguities using the centroid of the intersection points.

Once all edges have been determined for a cell, one or more topological polygons are formed by connecting the edges. Those polygons approximate the isosurface intersection with the cell.

4.4.3 Refining Edges

As can be seen from figure 4, 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 desireable. 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 (Figure 15).

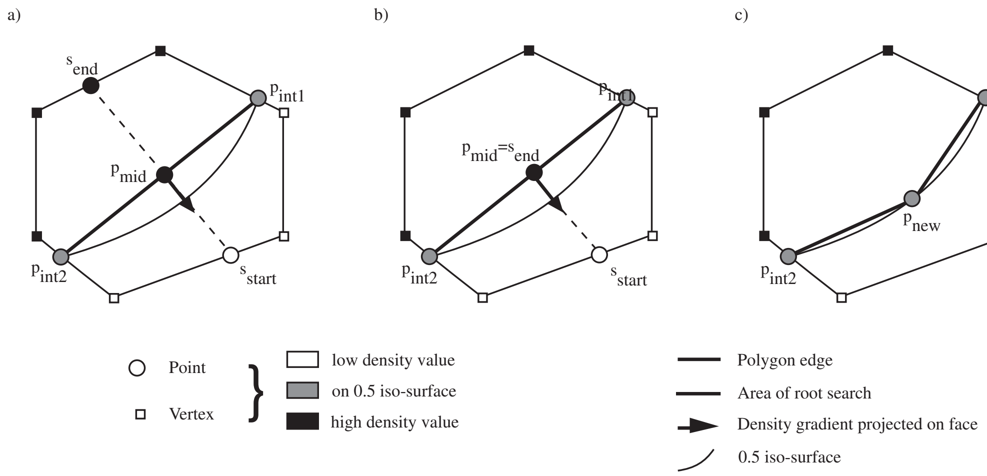

Figure 15: The line along the density field gradient through p_{mid} (a) defines a linear search space for the edge refinement (b). The refined edge is shown in (c).

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} (Figure 15 (a)). 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 perform a root search between that point and p_{mid} (Figure 15 (b)). 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 no isosurface intersection exists and do not refine the edge.

4.4.4 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. The polygonal approximation of the isosurface is improved by moving the centroid in the direction of the density gradient until the 0.5 isosurface intersection is found (Figure 13). 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.

4.5 Improvements

4.5.1 Local Density Field

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 quasi-convolutionally smoothed density field, involving only those halfspaces that have a non-constant 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 field grows linearly in the size of the object.

4.5.2 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 show [Wün96] that the density field is then a convex swept surface. Figure 16 shows that for these cells an improved subspace polygonization is given as part of the convex hull of all isosurface intersection points. An efficient algorithm, which finds polygons of maximum size, is given in [Wün96].

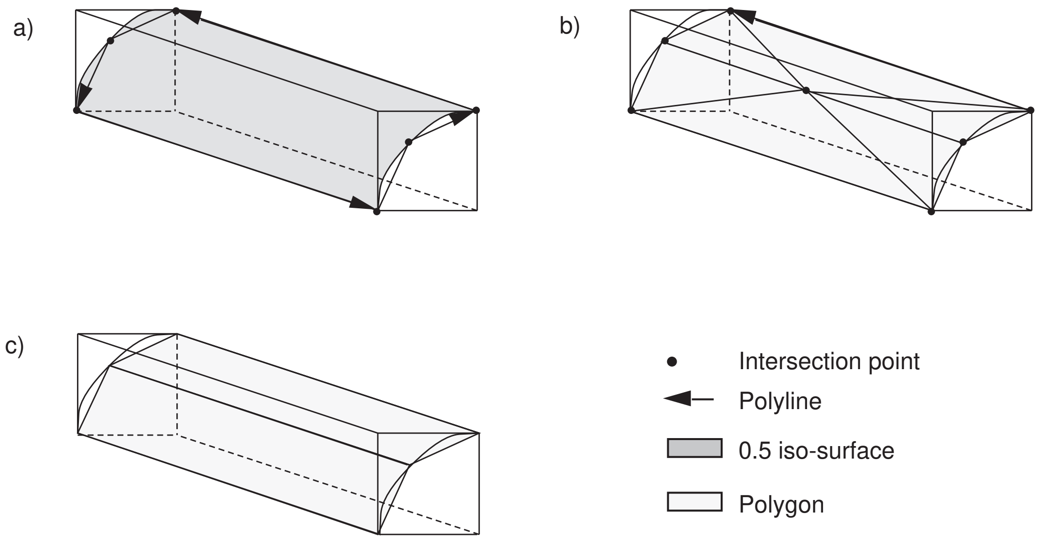

Figure 16: (a) Cell containing a quasi-convolutionally smoothed intersection of two halfspaces. (b) The subspace polygonization leads to fragmentation. (c) Desired result of the subspace polygonization.

4.5.3 Variable Rounding Radius

Another desirable feature is to define different rounding radii for the different edges of an object. Though this effect is not possible for quasi-convolutionally smoothed objects, a similar effect can often be obtained by defining different smoothing radii for the halfspaces which form a polyhedral primitive. This technique was used to produce the cylindrical metal pins with smoothly flattened ends in Figure 18 (see also the enlargements).

5 Rendering

For Gouraud or Phong shading the vertex normals of the polygons are required. 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

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

The density gradient is computed by differentiating the arithmetic tree defining the density field (see section 2.2).

6 Results

We implemented our algorithm in the functional language CLEAN 1.0, on a POWER MACINTOSH 9500/120. The polygonized scenes were rendered using QUICKDRAW3D.

6.1 Images

Figure 17 shows a hole punch modelled as a simple unsmoothed CSG object with polyhedral primitives. By specifying a few rounding radii we obtain a much better looking smoothed hole punch shown in Figure 18. 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 corner is represented with two long rectangles and 6 triangles, respectively. The Gouraud shaded picture in Figure 18 (b) shows that the produced polygons are sufficient to achieve the visual impression of a smoothed surface.

Figure 17: The unsmoothed “Hole Punch” scene. The result is shown as a wire-frame representation (a) and Gouraud shaded (b).

Figure 18: The “Hole Punch” scene polygonized with Triage Polygonization. The result is shown as a wire-frame representation (a) and Gouraud shaded (b).

The enlargements of Figure 18 depict the punch, part of the hinges, and some metal pins in detail. The 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. An efficient implementation is achieved by directly clipping the density field of an object before polygonization. Details are given in [Wün96].

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.

Figure 19 and 20 show a model of a stapler before and after applying our algorithm, respectively. Note that wherever possible the polygonization method finds long triangles and rectangles. Also it can be seen that very thin objects such as the side plates of the hinge are polygonized without problems.

Figure 21 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 depicts a clipped quasi-convolutionally smoothed small cube subtracted from a bigger unsmoothed cube. We model set operations on polygonized objects by merging BSP trees [TN87, NAT90].

The bottom enlargements of Figure 21 (a) and (b) give an example of a concave three plane corner. The corner results from a quasi-convolutionally smoothed union of a big cube and a small cuboid. It can be seen that the corner is nicely polygonized with only 26 triangles.

6.2 Comparison with the Marching Cubes algorithm

The Marching Cubes algorithm is a popular method for implicit surface polygonization, and 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 quasi-convolutionally smoothed scene.

Figure 22 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

Figure 19: The unsmoothed “Stapler” scene. The result is shown as a wire-frame representation (a) and Gouraud shaded (b).

Figure 20: The “Stapler” scene polygonized with Triage Polygonization. The result is shown as a wire-frame representation (a) and Gouraud shaded (b).

Figure 21: The "CSG Example" scene polygonized with Triage Polygonization. The result is shown as a wire-frame representation (a) and Gouraud shaded (b).

in part (a) was polygonized with Triage Polygonization whereas for (b) the Marching Cubes algorithm was used. Note that our algorithm achieves a good polygonization for all objects, independent of the rounding radius. The polygonization for the objects with small rounding radius can be considered as 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 quasi-convolutionally smoothed objects.

Figure 22: Triage Polygonization and the Marching Cubes algorithm applied to the "Variable Radius" scene. Part (a) shows the result of Triage Polygonization as a wire-frame representation (left) and Gouraud shaded (right). Part (b) shows the same representations for the result of the Marching Cubes algorithm.

We found that on average Triage Polygonization is about 20–30 times faster than the Marching Cubes algorithm and outputs only a fraction ($\approx$ 1–

2%) of its number of polygons. The results also confirm that Triage Polygonization produces arbitrarily complex polygons of vastly different size, whereas the Marching Cubes algorithm produces only about equally-sized triangles. Also note that Triage Polygonization yields for a rounding radius of zero the b-rep of the unsmoothed object, whereas the Marching Cubes algorithm can not polygonize sharp edges at all.

The Marching Cubes algorithm becomes superior if the rounding radius reaches about a quarter of the object size (the bottom middle object in Figure 22). In that case, however, the object no longer fulfills our design objective that it is predominantly planar.

6.3 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 which suggest an average time complexity of about $O(n^{1.3})$, where n is the number of halfspaces in the unsmoothed CSG object. This result could be improved to $O(n \log n)$ by using the BSP tree algorithm of Naylor, Amanatides, and William [NAT90]. Note that this complexity is quite different from that for a conventional polygonization method, such as the Marching Cubes algorithm. For this algorithm the time complexity is $O(m^3)$, where m is the sample resolution.

As a result our algorithm can become less efficient for a complex object with hundreds of vertices if compared to the Marching Cubes approach with a low resolution. On the other hand our approach always guarantees that rounded edges and corners are found, which is not the case for most conventional algorithms. Also note that very complex objects are unlikely in the intended application. Scenes like the stapler and the hole punch, with at most a few tens of vertices per smoothed component, cause no problem.

6.4 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. That situation is rare with a Marching Cubes algorithm but common with ours, since our partitioning planes lie on the unsmoothed object's faces. Extending the algorithm to handle such vertices as special cases is complex. We have chosen instead largely to avoid this problem by classifying vertices against a displaced $0.5+\epsilon$ 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 those cases are easily identified and the missing polygons can be generated in a postprocessing step. See [Wün96] for details. We use a value for ϵ of 0.001.

The more general issue of ensuring continuity of the polygonized isosurface is discussed at length in [Wün96]. Our algorithm yields a continuous surface if no vertices lie exactly on the displaced isosurface and if the polyhedral subdivision of space has the honeycomb property[3]. While binary space partitioning does not naturally lead to a honeycomb, it is possible to maintain a honeycomb property by use of a data structure, such as the hash table used by Wyvill et al. [WMW86], 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, and we have been unable to implement complete sharing of face and edge information effciently. As a consequence our implementation is occasionally unable to generate a continuous surface, at which point it aborts with an error message. The situation is rare and we have for example generated an animation of a robot figure being gradually smoothed with a larger and larger radius filter until it vanishes completely. Nonetheless, we acknowledge that difficulties remain, and these are a topic for future research.

7 Conclusion

This paper has presented Triage Polygonization, a new polygonization method specifically designed for quasi-convolutionally smoothed objects. Triage Polygonization performs best for quasi-convolutionally smoothed objects smoothed with a rounding radius small in comparison to their size. Such objects have

[3] 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.

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. We believe that for the above case 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).

8 Future Work

As mentioned in subsection 6.4 several problems still exist with our method. We would like to implement the algorithm in an imperative language (C/C++) using a data structure which allows us to generate a BSP tree modified to maintain a honeycomb property.

An even better quality of polygonization should be achieved 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 Hall and Warren [HW90] or Bloomenthal [Blo88, BW90] could be employed.

We have designed Triage Polygonization to polygonize quasi-convolutionally 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 [Dan97].

References

[Blo88] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer-Aided Geometric Design*, 5(4):341 – 355, November 1988.

[Blo94] Jules Bloomenthal. An implicit surface polygonizer. In Paul S. Heckbert, editor, *Graphic Gems*, volume IV, chapter IV.8. Academic Press, Cambridge, MA 02139, 1994.

[BW90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109 – 116, March 1990. Special Issue on 1990 Symposium on Interactive 3D Graphics.

[Col90] Steve Colburn. Solid modeling with global blending for machining dies and patterns. SAE technical paper series, SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001 U.S.A., April 1990. 41st Annual Earthmoving Industry Conference.

[Dan97] Peter John Dansted. Convolutional smoothing of polyhedra. Master's thesis, University of Auckland, 1997.

[DK91] A. Doi and A. Koide. An efficient method of triangulating equivalued surfaces by using tetrahedral cells. *IEICE Trans. Commun. Elec. Inf. Syst.*, E-74(1):214 – 224, January 1991.

[Duf92] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics*, 26(2):131 – 138, July 1992.

[Düu88] Martin J. Düurst. Additional reference to "marching cubes". *Computer Graphics*, 22(2):72, April 1988. Letter.

[FKB80] H. Fuchs, Z. Kedem, and B.Naylor. On visible surface generation by a priority tree structure. *Computer Graphics*, 14(3):124–268, June 1980.

[HH93] P. Hinker and C. Hansen. Geometric optimization. In G. M. Nielson and D. Bergeron, editors, *Proceedings of Visualization '93*, pages 189 – 195, Los Alamitos, California, 1993. IEEE, Computer Society Press.

[HL92] Josef Hoschek and Dieter Lasser. *Fundamentals of Computer Aided Geometric Design*, chapter 14, pages 572 – 601. AK Peters Ltd., Wellesley, MA 02181, second edition, 1992.

[HW90] Mark Hall and Joe Warren. Adaptive polygonization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, 10(5):33 – 42, November 1990.

[KDK86] A. Koide, A. Doi, and K. Kajioka. Polyhedral approximation approach to molecular orbit graphics. *J. Molec. Graph.*, 4:149 – 156, 1986.

[KT96] Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications*, 16(3):64–77, May 1996. ISSN 0272-1716.

[LC87] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163 – 169, July 1987. Proceedings of SIGGRAPH.

[Lob96] Richard Lobb. Quasiconvolutional smoothing of polyhedra. *The Visual Computer*, 12(8):373 – 389, 1996.

[MSS94] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In D. Bergeron and A. Kaufman, editors, *Proceedings of Visualization '94*, pages 281 – 286. IEEE, Computer Society Press, 1994.

[NAT90] Bruce F. Naylor, John Amanatides, and William Thibault. Merging BSP trees yields polyhedral set operations. *Computer Graphics*, 24(4):115 – 124, August 1990.

[Nay81] Bruce F. Naylor. *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes.* PhD thesis, University of Texas, Dallas, Texas, May 1981.

[NB93] Paul Ning and Jules Bloomenthal. An evaluation of implict surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33 – 41, November 1993.

[Req80] Aristides A. G. Requicha. Representation for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437 – 464, December 1980.

[Sny92] John M. Snyder. Interval analysis for computer graphics. *Computer Graphics*, 26(2):121 – 130, 1992.

[TN87] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21(4):153 – 162, July 1987. Proceedings SIGGRAPH '87.

[vGW94] Allen van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337 – 375, October 1994.

[WMW86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. A data structure for soft objects. *The Visual Computer*, 2(4):227 – 234, August 1986.

[Wün96] Burkhard C. Wünsche. A fast polygonization method for quasi-convolutionally smoothed polyhedra. Master's thesis, University of Auckland, August 1996. URL: http://www.cs.auckland.ac.nz/~bwue001/Thesis/thesis.pdf.