TM-31

# DEPTH BUFFERING DISPLAY TECHNIQUES FOR CONSTRUCTIVE SOLID GEOMETRY

by

Jaroslaw R. Rossignac
Aristides A. G. Requicha

PRODUCTION AUTOMATION PROJECT

September 1985

Technical Memorandum No. 31

PRODUCTION AUTOMATION PROJECT
Department of Electrical Engineering
College of Engineering & Applied Science
The University of Rochester
Rochester, New York 14627

TM–31

# DEPTH BUFFERING DISPLAY TECHNIQUES FOR CONSTRUCTIVE SOLID GEOMETRY

by

Jaroslaw R. Rossignac*
Aristides A. G. Requicha

September 1985

The work reported in this paper was supported by the National Science Foundation under Grants ECS–81–04646 and ECS–84–03882, and by companies in the Production Automation Project's Industrial Associates Program. Any opinions, findings, conclusions or recommendations expressed or implied are those of the authors and do not necessarily reflect the views of the N.S.F. or the Industrial Associates of the P.A.P.

* Rossignac is now with the IBM T. J. Watson Research Center, Yorktown Heights, NY.

## Abstract

Shaded displays of solids can be generated directly from CSG representations by a simple new algorithm that requires neither face/edge/vertex data nor intersection computations, and is a good candidate for hardware implementation.

## Introduction

The emerging technology of solid modelling is playing a crucial role in the evolution of CAD/CAM and robotic systems towards a higher level of automation and integration [Requicha & Voelcker 1982, 1983]. Shaded, color displays provide realistic visual feedback to (human) users of solid modelling systems, and shading is one of the currently most popular applications of solid modelling. (In fact, shading and solid modelling are often erroneously equated.)

Figure 1 summarizes the principal techniques for generating shaded images in solid modellers. Rectangles in the figure depict representations, and circles depict algorithms or processors. Five representations are shown (see [Requicha 1980] for basic notions in solid representation).

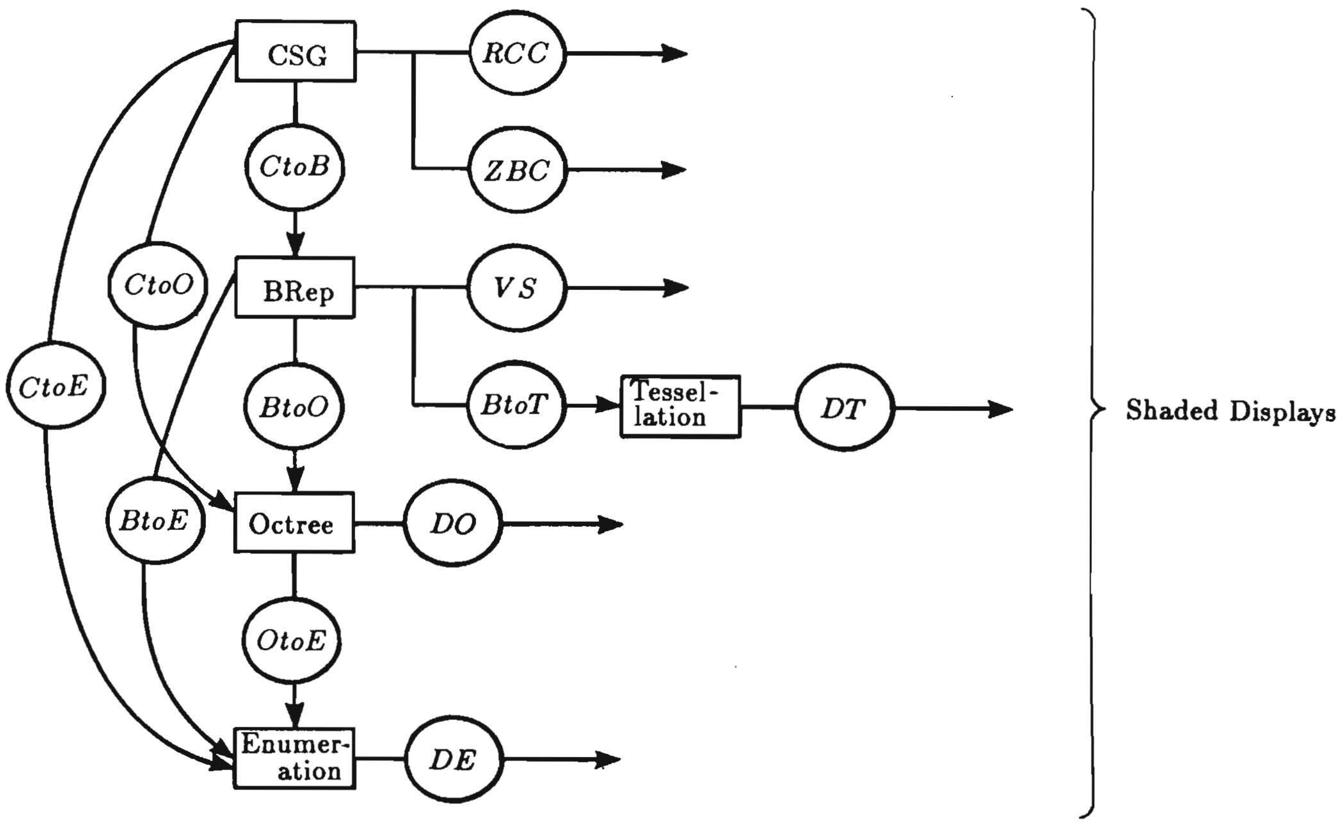


*Figure 1*

Representation conversion and shading of solids.

- *CSG (Constructive Solid Geometry)* — Trees whose internal nodes represent Boolean operations (*i.e.*, regularized set operations [Requicha 1980]) and rigid motions, and whose leaves represent solid primitives. Typically, primitive solids are blocks, cylinders, spheres, cones, and tori.

- *BReps (Boundary Representations)* — Graphs whose nodes represent faces, edges and vertices, and whose arcs represent adjacency and incidence relations.

- *Tessellations* — BReps whose faces are simple convex polygons, often triangles (in which case the representations are called *triangulations*).

- *Octrees* — Hierarchical structures that reflect the recursive subdivision of solids into variably–sized cubes [Meagher 1982].

- *Spatial Enumerations* (also called voxel arrays) — Arrays or lists that correspond to the uniform subdivision of solids into equally–sized cubical volume elements, called *voxels*.

Known algorithms for converting between these representations are also shown in Figure 1.

- *CtoB* converts CSG representations into BReps, and is usually called a *boundary evaluation* algorithm [Requicha & Voelcker 1985].

- *BtoT* tessellates a BRep, approximating curved faces, if any, by simple convex polygons (see *e.g.* [Wördenweber 1983]).

- *CtoO* and *BtoO* convert, respectively, CSG and BReps into octrees. A *CtoO* algorithm based on the notion of cell classification is discussed in [Lee & Requicha 1982] and has been used for several years in the PADL–2 system [Brown 1982] for computing mass properties of solids. A *BtoO* algorithm is decribed in [Tamminen *et al.* 1984]

- *CtoE, BtoE,* and *OtoE* convert, respectively, CSG, BReps and octrees into spatial enumerations. In essence they implement *point membership classification*, *i.e.*, they determine, for each point in a 3–D grid, whether the point is inside, on the boundary, or outside of a solid [Requicha & Voelcker 1977, Tilove 1980, Lee & Requicha 1982]. Point membership classification is straightforward, and special–purpose hardware for converting CSG to spatial enumerations has been designed [Kedem & Hammond 1985].

Passing now to shading algorithms, the most important are shown on the right side of Figure 1.

- *DE* and *DO* display enumerations and octrees. Importantly, both have been implemented in special–purpose hardware. Thus, a voxel machine is running at the University of Pennsylvania [Goldwasser 1984], and octree machines are commercially available.

- Shaded displays can be generated from BReps by a variety of *visible surface (VS)* algorithms [Newman & Sproull 1979, Foley & van Dam 1982]. These algorithms are particularly simple when the BReps are tesselations, and special purpose *tiling engines* for displaying triangulations and other tesselations are commercially available. Several on-going research projects on custom VLSI chips for high–speed display also are oriented to tesselations [Demetrescu 1985, Gharachorloo & Pottle 1985, Poulton *et al.* 1985].

- Shading can be accomplished directly from CSG by means of *ray casting* algorithms for CSG *(RCC)*, or *depth buffer* (also called *Z–buffer*) algorithms for CSG *(ZBC)*. Ray casting algorithms for CSG were pioneered by the Synthavision$^{TM}$ system [Goldstein & Nagel 1971], and are used in several modellers, *e.g.*, PADL–2 and GMSolid [Boyse & Gilchrist 1982, Roth 1982]. Many variants of ray casting for CSG exist — see *e.g.* [Atherton 1983, Bronsvoort *et al.* 1984, Okino *et al.* 1984, Wang *et al.* 1984]. Z–buffer algorithms for CSG are new, insofar as we know, and are the main contribution of this paper. (The "extended Z–buffer" algorithms of [Okino *et al.* 1984, Wang *et al.* 1984] are quite different from the algorithm to be described below, and are best classified, in our opinion, as ray casting variants.)

It is clear from Figure 1 that a designer of modelling systems has many options to provide shading facilities in a solid modeller. Figure 2 shows three of the most interesting. In Figure 2a the input data, which typically describes objects through Boolean operations on previously defined objects, sweeps of 2–D contours, and so on, is translated *(ItoB)* into an equivalent BRep. This is further converted into a tessellation, which is displayed by using a tiling engine. (The architecture of Figure 2a is becoming increasingly popular, and its older version, without tessellation, has been used by several solid modellers.) Displays of existing BReps may be produced quickly because tessellation *(BtoT)* is not very time consuming (and may be available in hardware in the near future), and tiling engines are fast. However, module *ItoB* must produce BReps for objects that are combined by Boolean operations when a user defines new solids, and this can take a substantial amount of time for complex objects. Boundary evaluation and merging algorithms that operate correctly for all input objects [Requicha & Voelcker 1985] are not good candidates for hardware implementation because they are too complicated (but see [Yamaguchi & Tokieda 1985]). Thus, the architecture of Figure 2a suffers from what may be called *the Boolean operation bottleneck.*

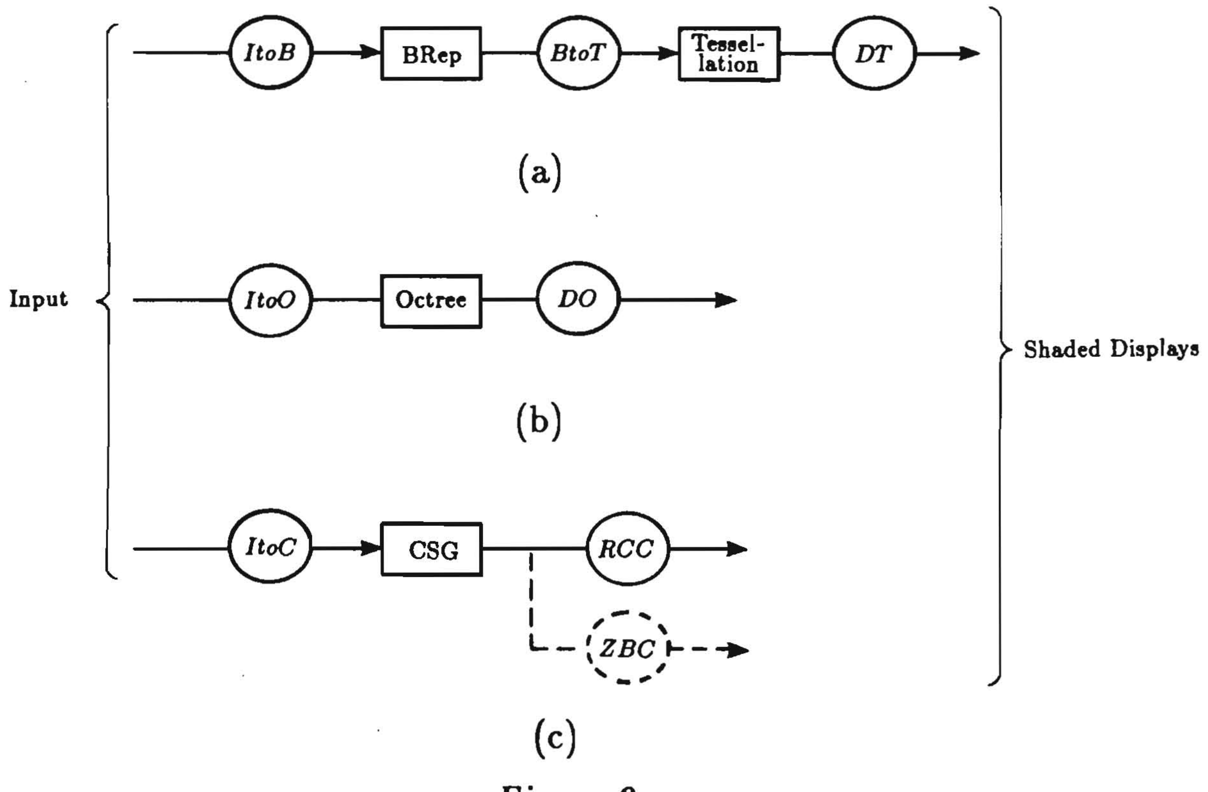


*Figure 2*

Alternative architectures for displaying shaded solids.

The alternatives shown in Figures 2b and 2c avoid this bottleneck. In Figure 2b the input is converted into (usually approximate) octrees, and these are displayed by an octree machine. Boolean operations on octrees can be executed swiftly in hardware (or even in software) and therefore rapid displays of solids that evolve through user editing should be achievable. (Insofar as we know, octree machines have not yet been used in CAD/CAM, possibly because of their current price.)

Figure 2c shows in solid lines the approach adopted in most of the current CSG–based modellers. Objects are defined primarily through Boolean operations, and therefore the translator *ItoC* need not perform any expensive geometric computation to convert the input into CSG trees. Ray casting in software is relatively slow, but many speed–ups are available (see *e.g.* [Roth 1982]). Importantly, ray casting machines for CSG have been designed and are expected to be demonstrated soon [Kedem & Ellis 1984, Sato *et al.* 1985]. The major claim of this paper is that Z–buffer algorithms for CSG are competitive with ray casting, and therefore the alternative shown in dashed lines in Figure 2c is viable and sometimes preferable.

*   *   *

## A Z–buffer algorithm for CSG

Depth– or Z–buffer algorithms that operate on BReps, and especially on polygonal nets, are well-known in computer graphics [Newman & Sproull 1979, Foley & van Dam 1982]. The basic algorithm uses two arrays $I[x, y]$ and $Z[x, y]$, with entries for each pixel in a screen. $I[x, y]$ is the intensity buffer, and $Z[x, y]$ is the depth– or Z–buffer. In pseudo–code the algorithm may be expressed as follows.

ALGORITHM 1

```
for each $(x, y)$ do
  $Z[x, y] \leftarrow$ BigNumber
  $I[x, y] \leftarrow 0$   { or background intensity }
  end   { Initialization loop }
for each face $F$ of solid $S$ do
  for each point $\mathbf{p}$ in a dense grid on $F$ do
    $(x, y) \leftarrow$ ProjectOnScreen($\mathbf{p}$)
    $d \leftarrow \|\mathbf{v} - \mathbf{p}\|$      { $\mathbf{v}$ is the viewpoint }
    if $d < Z[x, y]$ then
      $Z[x, y] \leftarrow d$
      $\mathbf{n} \leftarrow$ Normal to $F$ at $\mathbf{p}$
      $I[x, y] \leftarrow$ CompIntens($\mathbf{p}$,$\mathbf{n}$, LightSources)
      end   { if }
    end   { Scanning loop }
  end   { Main face loop }
Display entire array $I[x, y]$
```

In words: Scan each face; check the distance between each point **p** and the viewpoint **v**; if this distance $d$ is less than that stored in the appropriate Z-buffer location, write $d$ onto the Z-buffer, compute the intensity at **p** (which depends on the normal **n** to the solid and on the light sources), and update the intensity buffer; at the end of the scan the intensity buffer contains the correct image values. Typically, the frame buffer of the display terminal is used to store the intensity array, and therefore the display can be updated incrementally by overwriting the existing value in the frame buffer whenever a new intensity for a pixel is computed.

This algorithm is relatively slow when implemented in software. But it is very simple, and therefore can be implemented in hardware; in fact, current tiling engines use depth–buffering techniques.

Figure 3 illustrates the need for depth testing. Both $\mathbf{p}_1$ and $\mathbf{p}_2$ project on the same pixel, but only $\mathbf{p}_1$ is visible. The intensities that correspond to these two points are different, because the faces $F_1$ and $F_2$ make different angles with the direction of incident light.

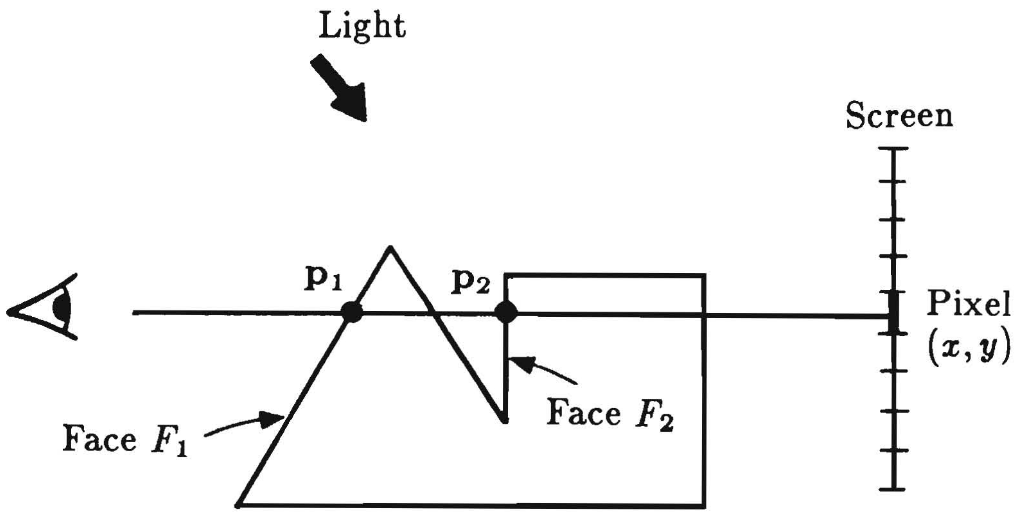


*Figure 3*

Depth testing

Algorithm 1 is not directly applicable to solids represented by CSG, *because explicit representations for faces are not available.* But it can be extended to CSG by exploiting the *generate and classify* paradigm that has been found useful in the design of many CSG–based algorithms [Lee & Requicha 1982, Sarraga 1982, Requicha & Voelcker 1985]. We use the fact that boundaries of solids are monotonically decreasing under Boolean operations; that is, the boundary of $S = A \otimes B$, where $\otimes$ denotes one of the regularized set operators, must be a subset of (the union of) the boundaries of $A$ and $B$. This implies that the boundary of a solid $S$ must be included in the union of the boundaries of all the primitives $P_i$ in the solid's CSG representation. The faces of $S$ are subsets of (unions of) faces of primitives, and these are easily computed from a CSG representation. Scanning the primitives' faces, instead of the actual solid's faces, yields a superset of the points needed by the Z–buffer algorithm. Some of the points must be discarded because they are not on the actual faces. This can be done by *point membership classification*, *i.e.*, by an algorithm that determines whether any given point in space is inside, on the boundary, or outside of a solid. A complete Z–buffer algorithm for CSG is shown below.

ALGORITHM 2

```
for each $(x, y)$ do
  $Z[x, y] \leftarrow$ BigNumber
  $I[x, y] \leftarrow 0$    { or background intensity }
  end   { Initialization loop }
for each face $F$ of each primitive of solid $S$ do
  for each point $\mathbf{p}$ in a dense grid on $F$ do
    $(x, y) \leftarrow$ ProjectOnScreen($\mathbf{p}$)
    $d \leftarrow \|\mathbf{v} - \mathbf{p}\|$      { $\mathbf{v}$ is the viewpoint }
    if $d < Z[x, y]$ then
      CVal $\leftarrow$ ClassPoint($\mathbf{p}$,$S$)
      if CVal = $onS$ then
        $Z[x, y] \leftarrow d$
        $\mathbf{n} \leftarrow$ Normal to $F$ at $\mathbf{p}$
        $I[x, y] \leftarrow$ CompIntens($\mathbf{p}$,$\mathbf{n}$, LightSources)
        end   { Classification test }
      end   { Depth test }
    end   { Scanning loop }
  end   { Main face loop }
Display entire array $I[x, y]$
```

Point membership classification algorithms for CSG are simple and well-known. The following pseudo-code describes a simplified version of the basic algorithm. It assumes that each node $N$ in a CSG tree is represented by a record with three fields: links $N$.Left and $N$.Right to the two subtrees (sons) of $N$, and a field $N$.Op that indicates which operator or primitive solid is represented by the node. When $N$ represents a motion operation, the left subtree of $N$ represents the solid to be moved, and the right subtree is a leaf representing the rigid motion $M$ itself. (Typically, motions are represented by $4 \times 4$ matrices.)

ALGORITHM 3

```
function ClassPoint(p,$S$)
  case $N$.Op of
    Primitive   : return ClassPointWrtPrim(p,$S$)
    Motion Op : return ClassPoint($M^{-1}$(p),$S$.Left)
    Boolean Op: return Combine(ClassPoint(p,$S$.Left),
                                ClassPoint(p, $S$.Right), $S$.Op)
    end    { case }
  end    { ClassPoint }
```

The function ClassPoint recursively classifies a point with respect to each subtree of a Boolean node, and combines the two results (by using look-up tables derived from simple topological considerations [Requicha & Voelcker 1977, 1985, Tilove 1980]). At a motion node the motion $M = S$.Right is inverted, applied to the point, and the transformed point recursively classified with respect to the original solid $S$.Left. The recursion ends at the primitive leaves, where classification is accomplished by a primitive–specific procedure. In most CSG modelers the primitive solids can be expressed as Boolean combinations (usually intersections) of algebraic halfspaces, *i.e.*, of sets of points $\mathbf{p}$ that satisfy an algebraic inequality $f(\mathbf{p}) \leq 0$, where $f$ is a polynomial. ClassPointWrtPrim simply classifies $\mathbf{p}$ against each halfspace by testing the sign of $f(\mathbf{p})$, and combines the results. (In practice a "fuzz factor" $\epsilon$ must be used for comparing real numbers because of round-off errors.) The most expensive computation carried on by Algorithm 3 is polynomial evaluation for classifying a point with respect to primitive halfspaces, and this can be done very quickly.

Observe that point membership classification does not require computing intersections between geometric entities. Intersection computations amount to solving systems of nonlinear equations, and are the most time consuming (and unreliable) components of ray casting and other algorithms for CSG. Z–buffer algorithms for CSG have been used at the University of Rochester for the past few years to produce shaded displays of blended solids, precisely because it is difficult to compute for blending surfaces the intersections that are required by other display algorithms. Figure 4 shows examples of blends displayed through depth buffering. (See [Rossignac & Requicha 1984] for additional examples.)

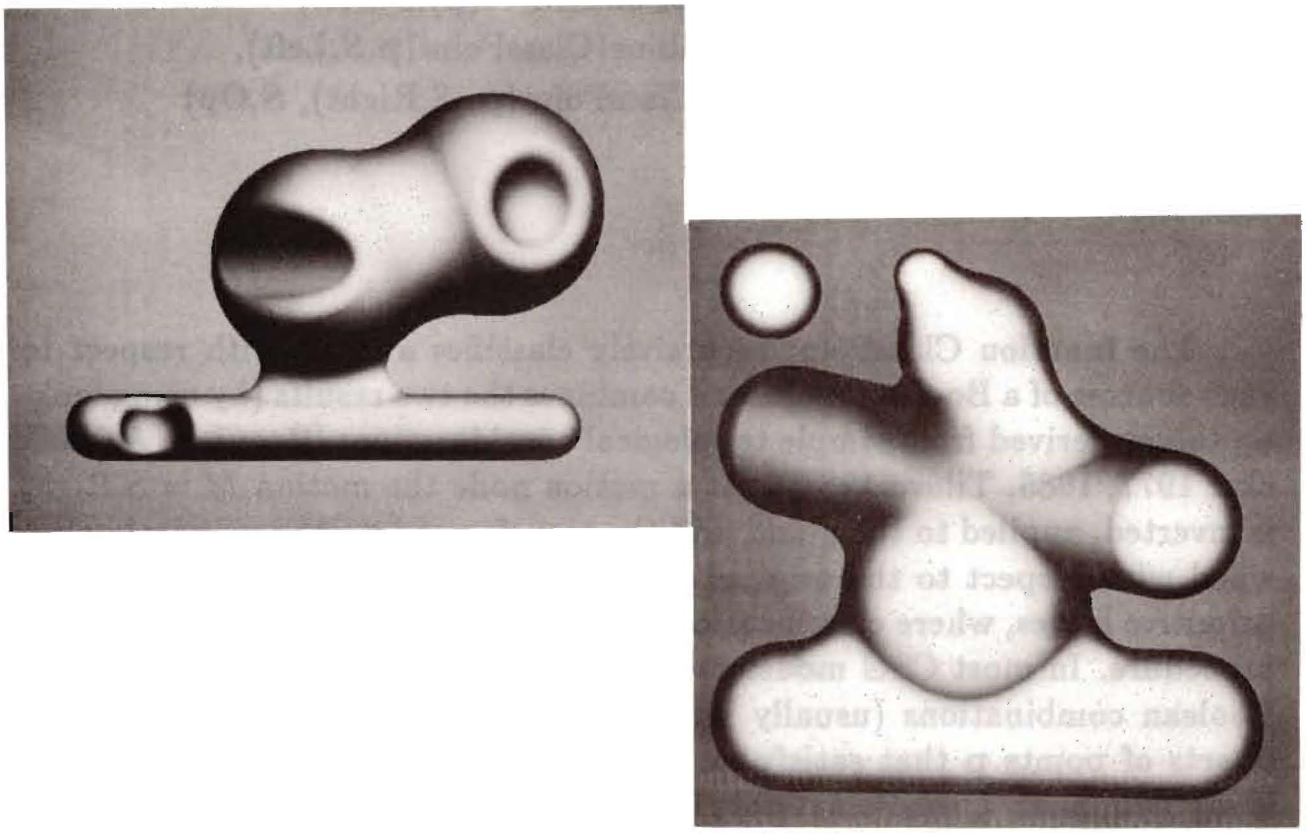


*Figure 4*

Shaded images of blended solids generated by depth buffering.

The Z–buffer display algorithm generates images of good quality for objects with rounded edges and corners, as in Figure 4, but suffers from aliasing near sharp edges. Figure 5 shows why aliasing occurs. Figures 5a and 5b depict two cross-sections of a solid, produced by planes normal to an edge of the solid. Suppose that face scanning generates points $\mathbf{p}_1, \mathbf{p}_2$ that project on a single pixel, and points $\mathbf{p}_3, \mathbf{p}_4$ that project on another single pixel. Points $\mathbf{p}_2$ and $\mathbf{p}_3$ are discarded because they are farther from the viewpoint than, respectively, $\mathbf{p}_1$ and $\mathbf{p}_4$. Observe that, since $\mathbf{p}_1$ and $\mathbf{p}_4$ lie in different faces, the corresponding intensities generally will be different. A similar situation may occur at many crossections along the edge. The result is a jagged appearance for the edge. Aliasing can be reduced by well-known filtering techniques [Newman & Sproull 1979, Foley & van Dam 1982], and is not very important in most CAD/CAM applications.

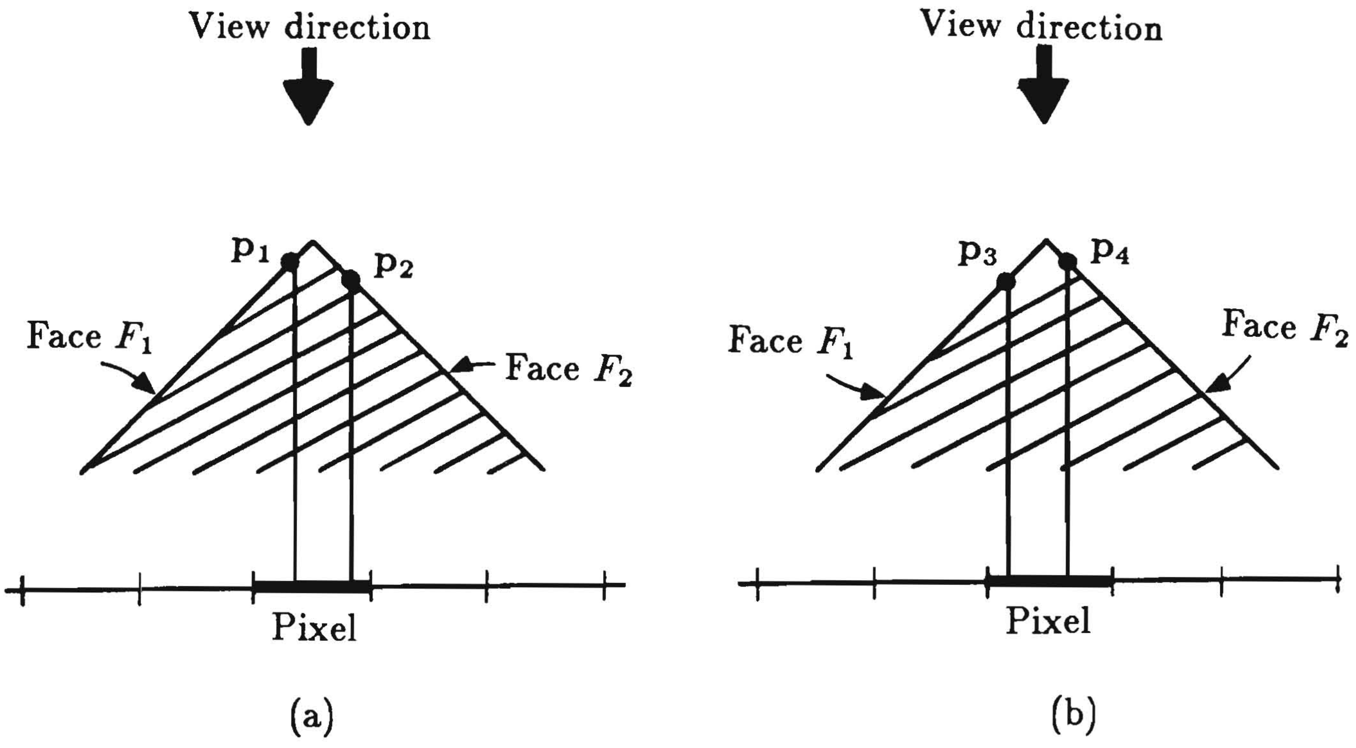


*Figure 5*

Edge aliasing

Algorithm 3 may produce incorrect results when solids being combined by Boolean operations have overlapping boundaries. To resolve so-called "on/on ambiguities" one must augment classification results with *neighborhood* information, as explained in [Requicha & Voelcker 1977, 1985, Tilove

1980]. Neighborhood representation and manipulation complicate the algorithm and slow down its execution. One can trade speed for correctness. Thus, simply ignoring neighborhoods, as in Algorithm 3, produces correct results "most of the time", and occasional errors may be acceptable because image quality usually is not crucial in CAD/CAM applications when fast display generation is needed for user feedback. (Intolerable errors may occur sometimes, for example when displaying a null object defined as $S = A -^* A$.)

Representing and combining neighborhoods for points that lie in the boundaries of several halfspaces (*e.g.*, object's vertices) is nontrivial. (See [Requicha & Voelcker 1985] for details of neighborhood manipulations.) For the purpose of image generation by depth buffering, one can use the following simpler approach, which is guaranteed to produce correct results within the resolution of the screen. (An alternative approach that produces correct results almost always is described below, in the section on "Fast classification".) When a point $\mathbf{p}$ classifies on the boundary of a halfspace $H$, represent its neighborhood by a pair $(H, \mathbf{n})$, where $\mathbf{n}$ is a unit vector passing through $\mathbf{p}$, normal to the halfspace boundary, and pointing towards the outside of the halfspace. If $\mathbf{p}$ is *on* two halfspaces with coincident boundaries it is easy to combine the corrresponding neighborhood representations; in essence, neighborhood combination amounts to determining whether the two normals are parallel or antiparallel (*i.e.*, opposite), and this can be done quickly and easily. But what if a point $\mathbf{p}$ is *on* two halfspaces with distinct boundaries? Simply discard $\mathbf{p}$ and use a new point $\mathbf{p}'$ that projects on the same pixel as $\mathbf{p}$ and also is in the face being scanned. Since $\mathbf{p}$ lies in the curve of intersection of two distinct halfspaces, it is always possible to generate a nearby $\mathbf{p}'$ that does not.

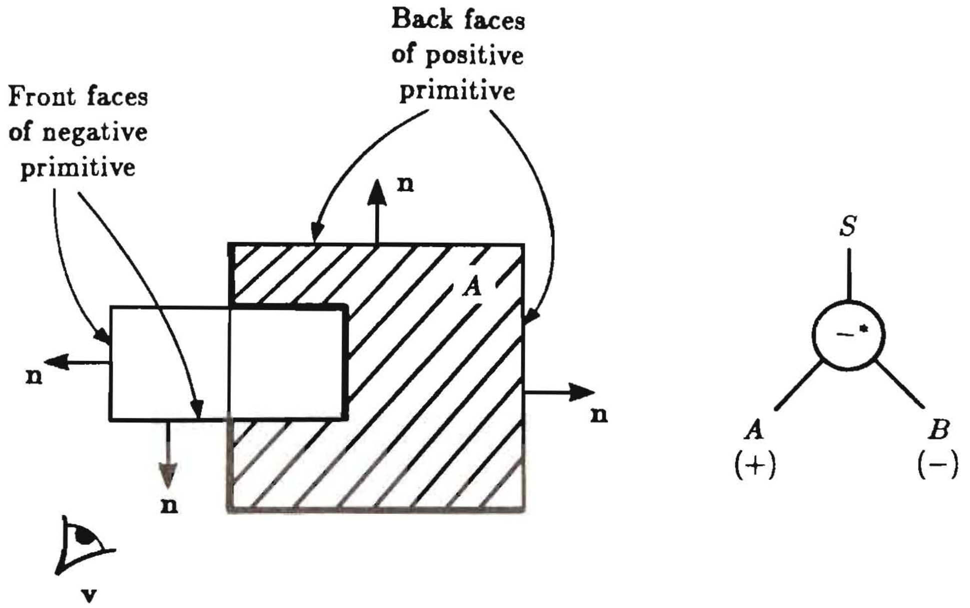
* * *

## Efficiency improvements

The Z-buffer algorithm described above, like most algorithms used in geometric modelling, must be supplemented with efficiency enhancement techniques to achieve reasonable performance when implemented in software. The following are a few of these techniques.

**Elimination of invisible faces.** Most of the visible surface algorithms for BReps immediately discard "back faces", *i.e.*, faces whose points $\mathbf{p}$ satisfy $\mathbf{n}.(\mathbf{v}-\mathbf{p}) < 0$, where the dot denotes inner product, $\mathbf{v}$ is the viewpoint, and $\mathbf{n}$ is the normal to the face at $\mathbf{p}$ and pointing towards the exterior of the solid. Curved faces must be split at the profile, or silhouette edges, to ensure that all the points in each of the resulting face segments have normals that are consistently oriented towards or away from the viewpoint.

Because object faces are unavailable in a CSG representation, back face elimination in CSG requires a different approach. Consider a solid $S$ and let $P$ be one of the primitives in $S$'s CSG tree. A primitive face of $P$ is a front face of $P$ if the normal directed towards the exterior *of the primitive* points towards the viewpoint; otherwise it is a back face of $P$. Examine the unique path from the root of the CSG tree to $P$ and count the number of times the path branches to the right at difference operators. If this number is even (or zero) the primitive is *positive*, otherwise it is *negative*. In the Z-buffer algorithm, discard the back faces of positive primitives, and the front faces of negative primitives; only the remaining primitive faces are potentially visible. This procedure is illustrated by a very simple example in Figure 6.



Back faces of positive primitive
Front faces of negative primitive
n
n
n
n
A
v
S
−*
A
(+)
B
(−)

*Figure 6*

The back faces of positive primitives and front faces of negative primitives are invisible. Potentially visible faces are shown by thicker lines.

**Point rejection by depth testing.** Depth testing is much cheaper than point membership classification, and therefore should be performed first, as shown in Algorithm 2. If the potentially visible primitive faces are sorted front to back and processed in sorted order, it is likely that a visible point be found in one of the first faces, and points in subsequent faces be rejected by depth testing only, without incurring additional classification costs.

**Optimal primitive face scanning.** Ideally, face scanning should generate exactly one point for each pixel in the projection of a primitive face on the screen. Insufficient point density produces "holes" in the Z-buffer; hidden faces or background incorrectly appear through these holes within the images of an object's true faces (see Figure 7). Too high a density is wasteful of computing resources.

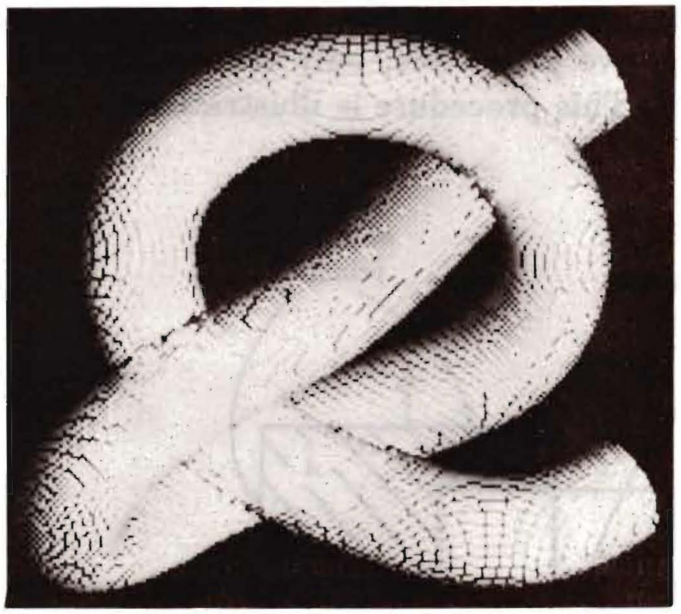


*Figure 7*

An image containing "black holes" due to insufficient scanning density.

For simple, *e.g.* planar or cylindrical, faces one can use standard scan conversion techniques [Newman & Sproull 1979, Foley & van Dam 1982], or solve the algebraic equation of a surface to find the 3-D point in a face that corresponds to an $(x, y)$ pixel (which amounts essentially to ray casting with respect to a halfspace). For complex faces it is more fruitful to use the parametric equations $F(u, v)$ of the faces, and simple estimates for increments $\Delta u, \Delta v$; this generally produces suboptimal scans but avoids expensive scanning computations.

**Fast classification.** Many techniques are known for increasing the average performance of classification and other geometric algorithms. Typically they exploit *locality* of geometric computations, and use object enclosures, sweeping (also known as "scan line") algorithms, spatial grids, and other *spatial directories* — see *e.g.*, [Roth 1982, Tamminen *et al.* 1984, Tilove 1981b] and references therein.

A new method for speeding–up classification by using so–called active zones [Rossignac & Voelcker 1985] can increase substantially the performance of Z–buffer algorithms for CSG. A so–called $I$–zone is associated with each primitive $P$ in the CSG representation of a solid $S$. An $I$–zone is represented by a modified subset of the solid's CSG tree. The intersection of $\partial P$, the boundary of a primitive $P$, with its $I$–zone contains the contribution of $P$ to the boundary of $S$, and therefore those portions of $\partial P$ outside the $I$–zone may be discarded. We refer the reader to [Rossignac & Voelcker 1985] for a detailed explanation of the theory and algorithms. Here we give a simple example, and note that the theory implies that no classification is needed when objects are defined solely by union operations, because all the $I$–zones are empty. Consider the object of Figure 8a, defined by the CSG tree of Figure 8b, which represents a combination of five rectangular primitives. The $I$–zone of $A$ in this example is simply $C$. In the Z–buffer algorithm it suffices to scan only $C \cap \partial A$, the subset of the faces of $A$ within the $I$–zone, and to classify the points generated only with respect to $C$. Therefore classification with respect to $D$ and $E$ can be avoided entirely.

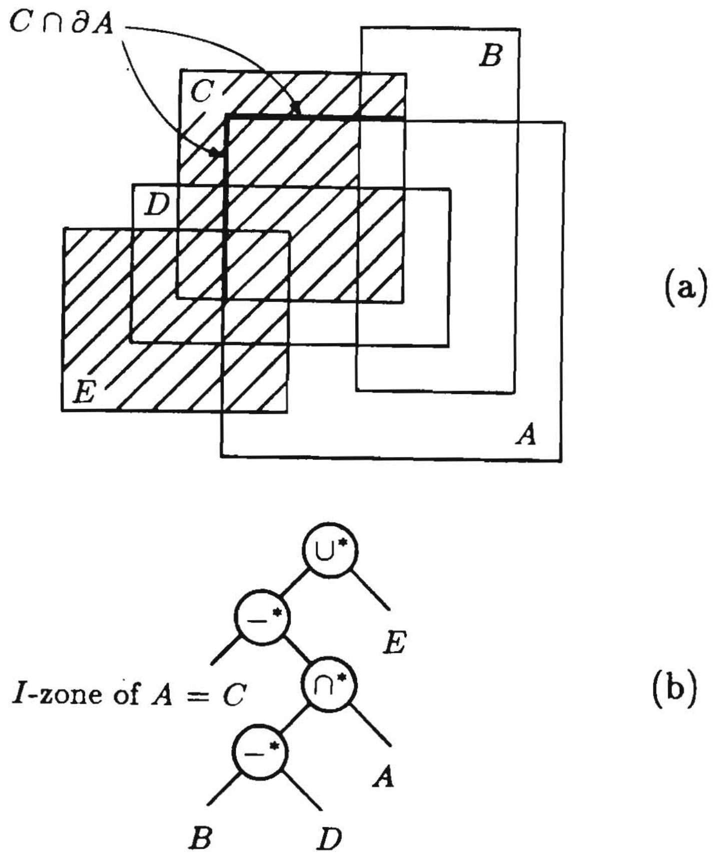


*Figure 8*

An example of classification using $I$-zones.

Classification calculations for depth buffering algorithms can be simplified as follows. Consider a *binary* classifier that outputs *true* when a point $\mathbf{p}$ is inside or on the boundary of $S$, and *false* when $\mathbf{p}$ is outside of $S$. (That is, it computes the characteristic function of the set $S$.) Classification with respect to halfspaces, and hence with respect to primitives, requires less comparisons in this approach, and (ignoring neighborhoods) the procedure for combining classification results simply applies to them the logical operators *or, and*, and *not and.* If the binary classification of a point $\mathbf{p}$ with respect to a solid $S$ is *true* the point need not be on the boundary of $S$. This causes no errors in a depth buffering algorithm because there must be some visible point $\mathbf{q}$ on the boundary of $S$ that is in front of $\mathbf{p}$, and therefore $\mathbf{p}$ will eventually by rejected by a depth test — see Figure

9. If binary classification is used together with the ordering of depth tests discussed earlier, few unnecessary classifications are likely to be performed.

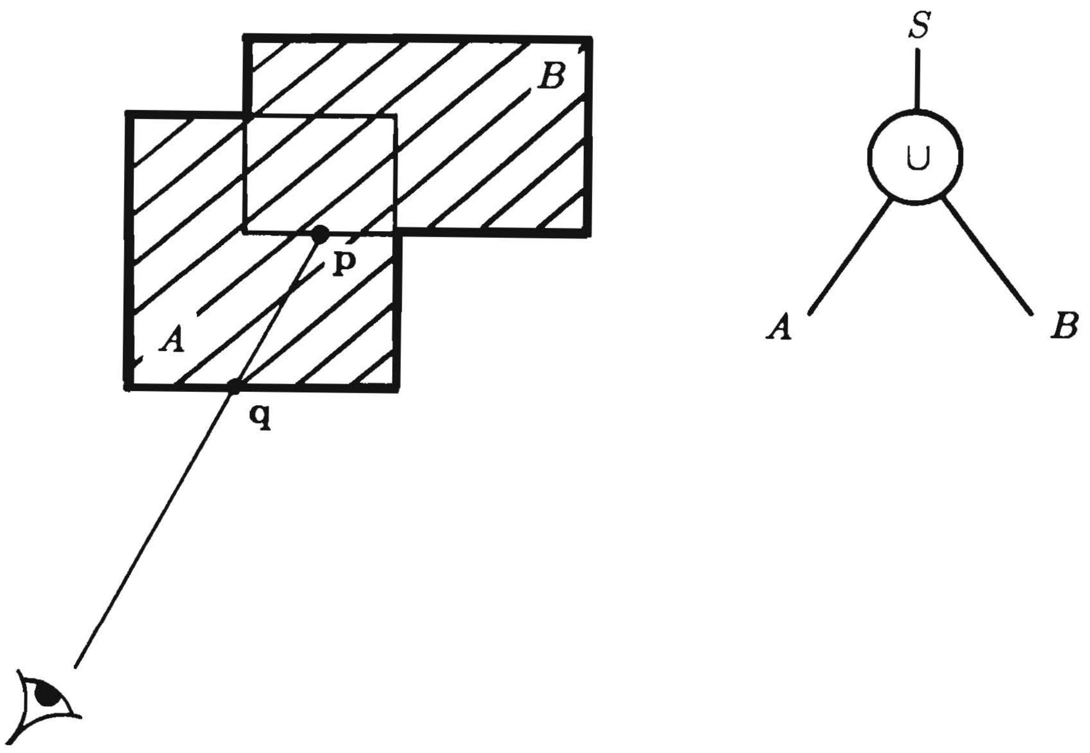


*Figure 9*

The binary classification of **p** with respect to $S = A \cup B$ is *true* and **p** is not on the boundary of $S$. But **p** will be rejected when its depth is compared with that of **q**.

We noted earlier that neighborhood manipulations may be ignored in point membership classifiers used in Z–buffer algorithms if correctness for all inputs is not very important, and we also described a relatively simple and correct approach. The following is an alternative approach, which is fast and fails very rarely. Suppose that a point **p** is *on* two subsolids $A$ and $B$. To classify **p** with respect to $S = A \otimes B$ we select a point $\mathbf{p}'$ at a small distance $\delta$ behind **p**, and classify $\mathbf{p}'$ with respect to $S$. If $\mathbf{p}'$ is *on* $S$ or *on* two subsolids of $S$, ignore $\mathbf{p}'$, select another point, and classify it. If $\mathbf{p}'$ is *in* $S$ then **p** is *in* or *on* $S$; in either case, **p** can be added to the Z–buffer. If $\mathbf{p}'$ is *out* of $S$ then **p** is *out* of $S$ or *on* a back face of $S$; in either case **p** can be discarded. Figure 10 illustrates various possibilities. The procedure has an intrinsic "resolution" of $\delta$. It is clear from Figure 10e that incorrect answers may be produced near certain edges or if the object has a wall of thickness less than $\delta$ in the vicinity of **p**. Since **p**$'$ can be chosen so as to avoid *on/on* cases, its classification can be accomplished by using $I$–zone techniques without neighborhoods.

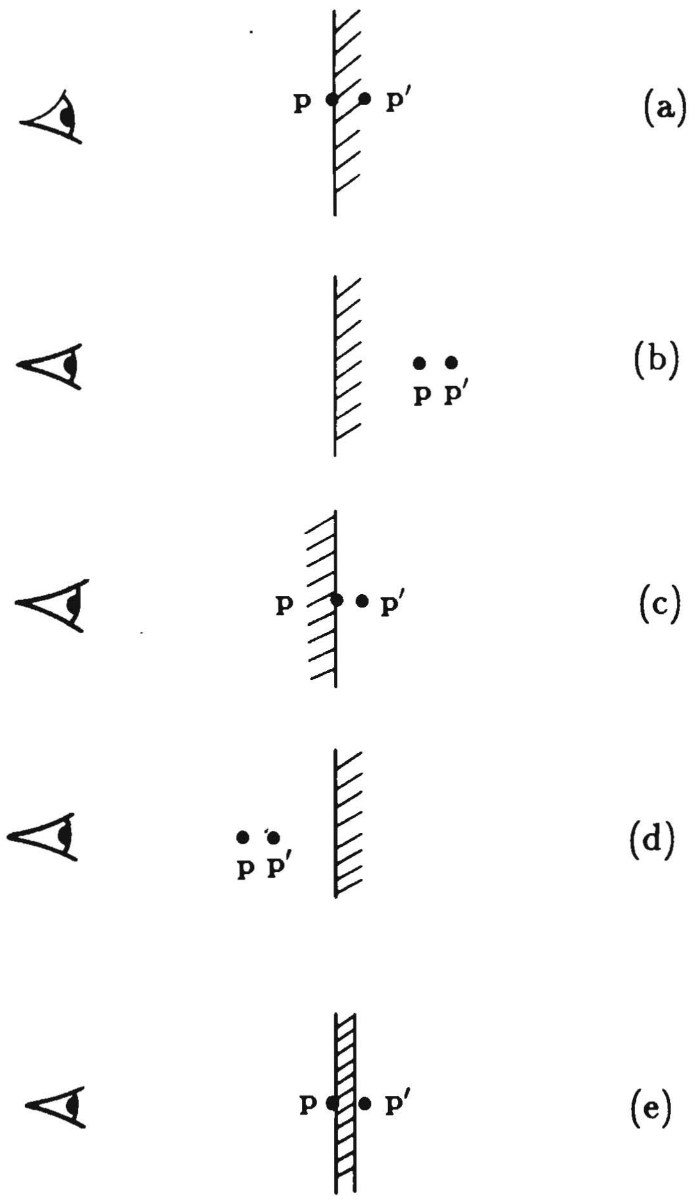


*Figure 10*

Inferring the classification of **p** by classifying a point **p**$'$ behind **p**.

**Sampling and recursive refinement.** The running time of Z-buffer algorithms is directly proportional to the number of points generated and tested. This number can be decreased substantially through sampling techniques similar to those used in ray casting algorithms for CSG [Roth 1982]. Thus, scan faces at a coarser resolution and interpolate results when, say, four adjacent points in the same primitive face are found to have the same classification. If classification values for adjacent points are not the same, subdivide by decreasing the sampling interval and recurse. Sampling may cause errors when objects have features that are small compared with the sampling interval, but has dramatic effects on performance.

**Use of generators.** The curves of constant $u$ and $v$ parameters in a parametrically defined primitive face $F(u,v)$ are called *generators.* They are linear or circular for the faces of the common primitives — blocks, cylinders, cones, spheres, and tori. Instead of classifying points on a face, one can classify entire generators and then consider only points that lie in the "on" subsets of the generators, and therefore are "on" the solid as well. This replaces classification of many points with classification of the generators in which they lie, and may lead to significant speed-ups. However, classification of generators requires the computation of intersections between generators and surfaces, and therefore negates one of the advantages of depth buffering algorithms.

* * *

## Comparison with ray casting

Ray casting and the Z-buffer algorithm introduced in this paper both operate directly on CSG representations, and therefore avoid the Boolean operation bottleneck. To compare the two approaches let us first review the basic ray casting algorithm.

ALGORITHM 4

```
for each Pixel in Screen do
  $R \leftarrow$ CreateRay(ViewPoint,Pixel)
  $RwrtS \leftarrow$ ClassLine($R, S$)
  if $RinS = \emptyset$ then $I \leftarrow 0$    { or background intensity }
  else $\mathbf{p} \leftarrow$ FirstPoint($RinS$)
    $\mathbf{n} \leftarrow$ Normal to $S$ at $\mathbf{p}$
    $I \leftarrow$ CompIntens($\mathbf{p}$, $\mathbf{n}$, LightSources)
    end    { else clause }
  Display($I$,Pixel)
  end    { do loop }
```

In words: cast a ray $R$ between the viewpoint and each pixel; classify it with respect to the solid $S$; extract the first point $\mathbf{p}$ of $RinS$, the subset of $R$ that classifies inside of $S$; compute the intensity $I$ at $\mathbf{p}$ by using an illumination model, and output $I$ to the screen. Typically, relatively simple illumination models are used in CAD/CAM (but see [Clark 1985]) and the most expensive computation in Algorithm 4 is line/solid classification — function ClassLine — which involves intersecting the line with the faces of all the primitives in the CSG tree of $S$.

Depth buffering has a major advantage over ray casting: point membership classification is inherently simpler and faster than line membership classification. In particular, no intersection calculations are required for classifying points. Intersection procedures for complex primitives are difficult to write, costly to execute, and sometimes numerically unreliable.

Which of the two shading algorithms is faster? Ignoring speed-ups, ray casting classifies $s$ rays, where $s$ is the number of screen pixels, with respect to a solid having $p$ primitives. The worst case complexity of line classification is quadratic on $p$ [Tilove 1981a], and therefore the complexity of ray casting is proportional to $s.p^2$. In a worst case each face of the object covers almost all the screen and the Z–buffer algorithm must generate in the order of $s$ points per primitive face. Since the number of faces is in the order of $p$ and point classification is linear in $p$ [Tilove 1981a], the overall complexity of depth buffering also is proportional to $s.p^2$. This worst–case analysis is too crude to be practically useful, but it suggests that ray casting and depth buffering have comparable performances, which differ mainly in multiplicative constants.

Intuitively one expects Z–buffering to be faster than ray casting when there are many complicated primitives (*e.g.* tori), for which intersection calculations are expensive. This expectation is confirmed by the experimental data summarized in Table 1. The table shows the ratio of the running times of ray casting and Z–buffer algorithms for the test objects shown in Figures 11, 12, and 13. The Z–buffer algorithm used in the tests was not coded for speed. It uses a binary classifier with $I$–zones and no neighborhoods, but has no other speed–ups. The ray caster is part of the PADL–2 system, and uses spatial directories, a special–purpose classification procedure, sampling, and several other efficiency–enhancement techniques. We turned off sampling in the PADL–2 ray caster to provide a fairer comparison, since sampling can also be used in depth buffer algorithms. The data show that even a non–optimized Z–buffer shader is competitive with current ray casters. Depth buffering is much faster than ray casting for solids whose faces lie in complex surfaces and project on a small area of the screen. For example, the object of Figure 13 is rendered 25 times faster by Z–buffering than by ray casting.

| Test Object | ZB/RC |
|---|---|
| A | 1.1 |
| B | .47 |
| C | .04 |

*Table 1*

Ratio of Z–buffer (ZB) to ray casting (RC) execution times for test objects.

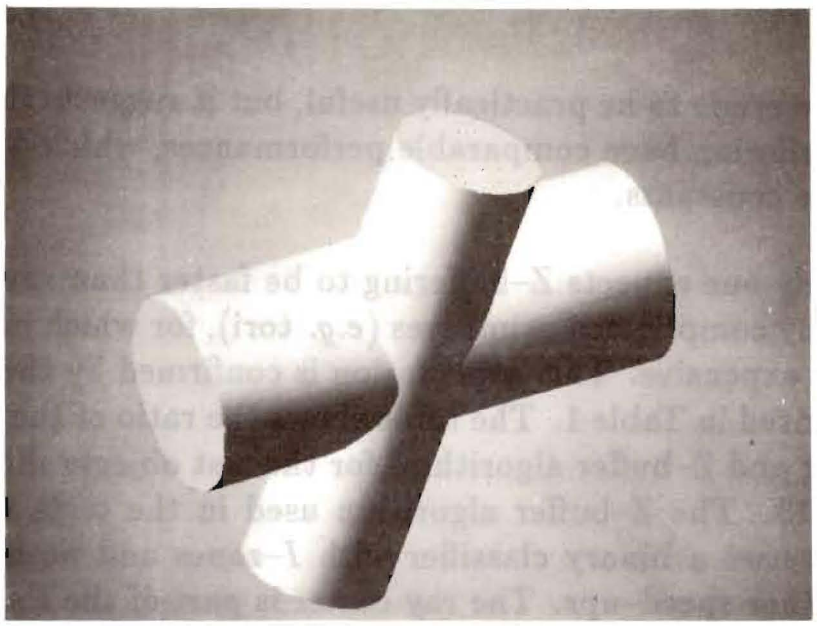


*Figure 11*
Test object A

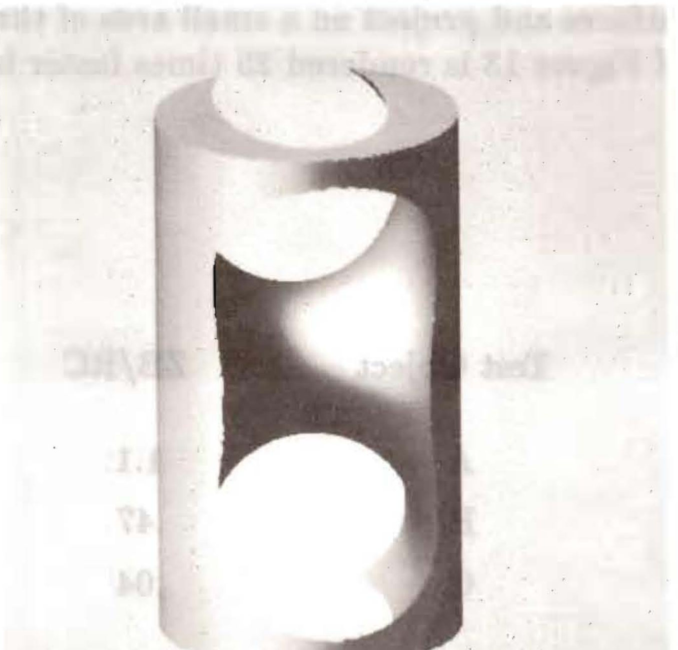


*Figure 12*
Test object B

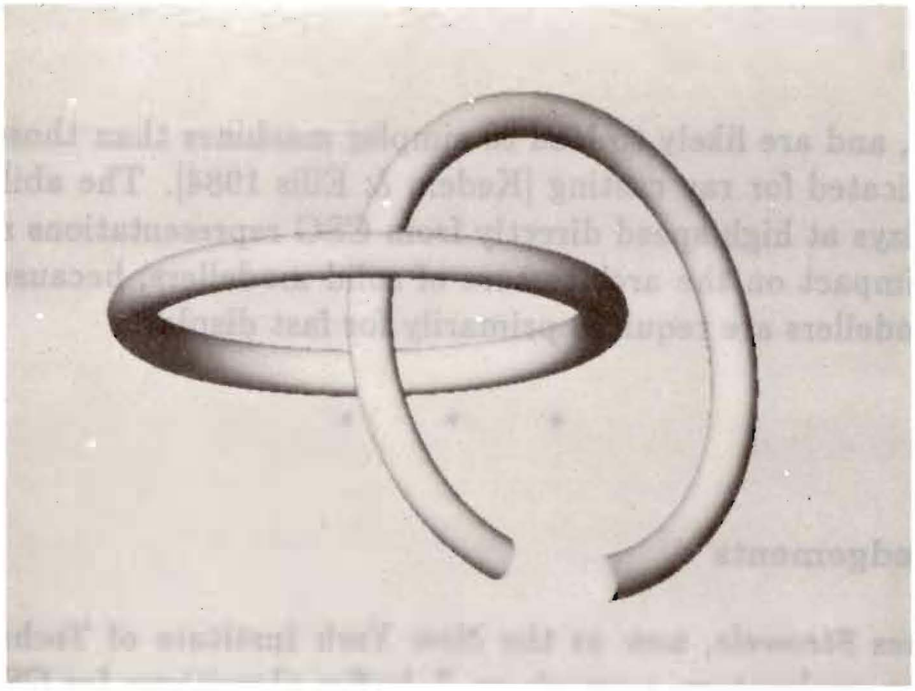


*Figure 13*

Test object C

## Conclusions

The depth– or Z–buffer shading algorithm presented in this paper operates directly on CSG representations. It does not require BRep information, *i.e.*, explicit face, edge and vertex representations, and therefore avoids expensive boundary merging and evaluation computations for displaying objects defined through Boolean operations.

The algorithm's performance compares favorably with that of ray casting, which is the shading algorithm used by most of the current CSG–based modellers. Depth buffering is easier to implement than ray casting. While ray casting requires computing intersections of lines with surfaces by solving nonlinear systems of equations, Z–buffering only requires polynomial evaluations to classify points with respect to halfspaces.

The main components of the basic Z–buffer algorithm for CSG are face scanning, depth testing in the Z–buffer, and point membership classification. Suboptimal face scanning is relatively straightforward, hardware implementations of Z–buffers exist, and VLSI chips for point membership clasification have been designed [Kedem & Hammond 1985]. Therefore depth–buffer algorithms for CSG are very good candidates for VLSI implementation, and are likely to lead to simpler machines than those currently being fabricated for ray casting [Kedem & Ellis 1984]. The ability to generate displays at high speed directly from CSG representations may have a profound impact on the architecture of solid modellers, because BReps in current modellers are required primarily for fast display.

* * *

## Acknowledgements

Jacques Stroweis, now at the New York Institute of Technology, did some of the exploratory research on Z–buffer algorithms for CSG in 1983, as an M.Sc. project in Electrical Engineering at the University of Rochester.

* * *

## References

[Atherton 1983] P. R. Atherton, "A scan–line hidden surface removal procedure for constructive solid geometry", *ACM Computer Graphics*, Proc. SigGraph '83, vol. 17, no. 3, pp. 73–82, July 1983.

[Boyse & Gilchrist 1982] J. W. Boyse and J. E. Gilchrist, "GMSolid: Interactive modeling for design and analysis of solids", *IEEE Computer Graphics & Applications*, vol. 2, no. 2, pp. 27–40, March 1982.

[Bronsvoort *et. al.* 1984] W. F. Bronsvoort, J. J. van Wijk, and F. W. Jansen, "Two methods for improving the efficiency of ray casting in solid modelling", *Computer Aided Design*, vol. 16, no. 1, pp. 51–55, January 1984.

[Brown 1982] C. M. Brown, "PADL-2: A technical summary", *IEEE Computer Graphics & Applications*, vol. 2, no. 2, pp. 69-84, March 1982.

[Clark 1985] A. L. Clark, "Roughing it: realistic surface types and textures in solid modeling", *ASME Computers in Mechanical Engineering*, vol. 3, no. 5, pp. 12-16, March 1985.

[Demetrescu 1985] S. Demetrescu, "High speed image rasterization using scan line access memeories", in H. Fuchs (Ed.), *1985 Chapel Hill Conference on VLSI.* Rockville, MD: Computer Science Press, 1985, pp. 221–243.

[Foley & van Dam 1982] J. D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics.* Reading, MA: Addison–Wesley, 1982.

[Gharachorloo & Pottle 1985] N. Gharachorloo and C. Pottle, "SUPER BUFFER: a systolic VLSI graphics engine for real time raster image generation", in H. Fuchs (Ed.), *1985 Chapel Hill Conference on VLSI.* Rockville, MD: Computer Science Press, 1985, pp. 285–305.

[Goldstein & Nagel 1971] R. A. Goldstein and R. Nagel, "3–D visual simulation", *Simulation*, vol. 16, no. 1, pp. 25–31, 1971.

[Goldwasser 1984] S. M. Goldwasser, "A generalized object display processor architecture", *IEEE Computer Graphics & Applications*, vol. 4, no. 10, pp. 43–55, October 1984.

[Kedem & Ellis 1984] G. Kedem and J. L. Ellis, "Computer structures for curve–solid classification in geometric modelling", Tech. Memo. No. 51, Production Automation Project, Univ. of Rochester, May 1984.

[Kedem & Hammond 1985] G. Kedem and S. W. Hammond, "The point classifier: A VLSI processor for displaying complex two dimensional objects", in H. Fuchs (Ed.), *1985 Chapel Hill Conference on VLSI.* Rockville, MD: Computer Science Press, 1985.

[Lee & Requicha 1982] Y. T. Lee and A. A. G. Requicha, "Algorithms for computing the volume and other integral properties of solids: II — A family of algorithms based on representation conversion and cellular approximation", *Comm. ACM*, vol. 25, no. 9, pp. 642–650, September 1982.

[Meagher 1982] D. Meagher, "Geometric modeling using octree encoding", *Computer Graphics & Image Processing*, vol. 19, no. 2, pp. 129–147, June 1982.

[Newman & Sproull 1979] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics.* New York: McGraw–Hill, 2nd ed., 1979.

[Okino *et. al.* 1984] N. Okino, Y. Kakazu, and M. Morimoto, "Extended depth–buffer algorithms for hidden–surface visualization", *IEEE Computer Graphics & Applications*, vol. 4, no. 5, pp. 79–88, May 1984.

[Poulton *et al.* 1985] J. Poulton, H. Fuchs, J. D. Austin, J. G. Eyles, J. Heinecke, C. Hsieh, J. Goldfeather, J. P. Hultquist, and S. Spach, "PIXEL–PLANES: Building a VLSI–based graphic system", in H. Fuchs (Ed.), *1985 Chapel Hill Conference on VLSI.* Rockville, MD: Computer Science Press, 1985, pp. 35–60.

[Requicha 1980] A. A. G. Requicha, "Representations for rigid solids: Theory, methods, and systems", *ACM Computing Surveys*, vol. 12, no. 4, pp. 437–464, December 1980.

[Requicha & Voelcker 1977] A. A. G. Requicha and H. B. Voelcker, "Constructive solid geometry", Tech. Memo. No. 25, Production Automation Project, Univ. of Rochester, November 1977.

[Requicha & Voelcker 1982] A. A. G. Requicha and H. B. Voelcker, "Solid modelling: A historical summary and contemporary assessment", *IEEE Computer Graphics & Applications*, vol. 2, no. 2, pp. 9–24, March 1982.

[Requicha & Voelcker 1983] A. A. G. Requicha and H. B. Voelcker, "Solid modelling: Current status and research directions", *IEEE Computer Graphics & Applications*, vol. 3, no. 7, pp. 25–37, October 1983.

[Requicha & Voelcker 1985] A. A. G. Requicha and H. B. Voelcker, "Boolean operations in solid modelling: Boundary evaluation and merging algorithms", *Proc. IEEE*, vol. 73, no. 1, pp. 30–44, January 1985.

[Rossignac & Requicha 1984] J. R. Rossignac and A. A. G. Requicha, "Constant-radius blending in solid modelling", *ASME Computers in Mechanical Engineering*, vol. 3, no. 1, pp. 65–73, July 1984.

[Rossignac & Voelcker 1985] J. R. Rossignac and H. B. Voelcker, "Redundancy and null object detection in constructive solid geometry", Tech. Memo. No. 52, Production Automation Project, Univ. of Rochester, 1985 (forthcoming).

[Roth 1982] S. D. Roth, "Ray casting for modeling solids", *Computer Graphics & Image Processing*, vol. 18, no. 2, pp. 109–144, February 1982.

[Sarraga 1982] R. F. Sarraga, "Computation of surface areas in GMSolid", *IEEE Computer Graphics & Applications*, vol. 2, no. 7, pp. 65-70, September 1982.

[Sato *et al.* 1985] H. Sato, M. Ishii, K. Sato, and M. Ikesaka, "Fast image generation of constructive solid geometry using a cellular array processor", *ACM Computer Graphics*, Proc. SigGraph '85, vol. 19, no. 3, pp. 95-102, July 1985.

[Tamminen *et. al.* 1984] M. Tamminen, O. Karonen, and M. Mäntylä, "Ray–casting and block model conversion using a spatial index", *Computer Aided Design*, vol. 16, no. 4, pp. 203–208, July 1984.

[Tilove 1980] R. B. Tilove, "Set membership classification: a unified approach to geometric intersection problems", *IEEE Trans. on Computers*, vol. C-29, no. 10, pp. 874-883, October 1980.

[Tilove 1981a] R. B. Tilove, "Line/polygon classification: a study of the complexity of geometric computation", *IEEE Computer Graphics & Applications*, vol. 1, no. 2, pp. 75-88, April 1981.

[Tilove 1981b] R. B. Tilove, "Exploiting spatial and structural locality in solid modelling", Tech. Memo. No. 38, Production Automation Project, Univ. of Rochester, October 1981.

[Wang *et. al.* 1984] K. K. Wang, S. F. Shen, C. Cohen, C. A. Hieber, and A. I. Isayev, "Computer–aided design and fabrication of molds and computer control of injection molding", Progress Report. No. 10, Injection Molding Project, College of Engineering, Cornell Univ., pp. 242–302, January 1984.

[Wördenweber 1983] B. Wördenweber, "Surface triangulation for picture production", *IEEE Computer Graphics & Applications*, vol. 3, no. 8, pp. 45–51, November 1983.

[Yamaguchi & Tokieda 1985] F. Yamaguchi and T. Tokieda, "A solid modeler with a 4 x 4 determinant processor", *IEEE Computer Graphics & Applications*, vol. 5, no. 4, pp. 51–59, April 1985.

* * *