Exact Ray Tracing of CSG Models by Preserving Boundary Information

Geoff Wyvill and Andrew Trotman

Abstract

We question the idea that regular sets and regularized operations are necessary in Constructive Solid Geometry. We argue that dangling points, lines and planes have their counterparts in the real world and their correct treatment leads to a more robust approach to solid modeling.

New algorithms avoid the use of small corrections (fudge factors) to detect coincident surfaces. Instead, a strictly logical scheme preserves inside/outside information as a ray progresses through the environment. This maintains consistency at every stage and guarantees a correct interpretation of the model.

Keywords: CSG, Regular sets, Ray tracing, Transparency, Rounding errors.

Introduction

Constructive Solid Geometry (CSG) is usually regarded as mainly applicable to engineering problems and this view is reinforced by the choice of illustration one sees in textbooks and papers on the subject. In the preface to his book, Martti Mäntylä (1988) observes, "In addition to design and manufacturing, solid modeling has a role in a number of other applications ...". His examples, however, are all drawn from engineering. Mortensen (1985) shows some pictures of flames from particle systems but all of his solid modeling examples come from mechanical engineering.

At the University of Otago, we use a CSG system for animation and as a testbed for research ideas in Computer Graphics. We render all our pictures by ray tracing. In this environment, we create solid models for clients, but the clients are animators and artists rather than engineers (Fig. 1). For this reason, we are more interested in the colors and surface properties of models than most designers of CSG systems may be. There is relatively little literature on ray tracing of solid models. Mäntylä lists over 100 references to the CSG literature but he mentions only four papers relating to ray tracing. Yet this is an area where there are still significant problems to be solved.

Fig. 1, *The Rose,* CSG is not just for engineers.

Constructive Solid Geometry (CSG) represents solids as sets of points in 3D space. But addition and subtraction of such sets can create dangling points, lines and surfaces with no thickness, so many systems use regularized sets (Tilove 1980). Regular sets, by definition, do not include surface points and from a purely mathematical standpoint, their use appears to solve the problem of generating a class of unwanted artifacts in CSG.

But from a practical point of view, there is something very unsatisfactory about the way in which they must be implemented. Ultimately, we decide whether a point is inside, outside or on the border of an object by means of a numerical comparison and this is subject to rounding errors. If you study the nature of these errors, it quickly becomes apparent that no form of number representation will eliminate them and systems that depend on recognizing border points, actually regard as equal, numbers that are closer than some small quantity — a fudge factor!

Several recent papers, e.g. Segal (1990), have addressed this problem by specifying what the error or tolerance is and we have no argument with these methods in the context of constructing polyhedral models. But when ray tracing, it is not necessary to identify vertices as exactly lying on edges or faces, and it becomes possible to use the full accuracy of the number system. Mike Muuss (1990) has described a boundary structure to represent objects that are not 3-manifolds. So perhaps there is some interest in a move away from regular sets.

The remainder of this paper is divided into four sections. First we describe regular sets, the reason for using them and the arguments against. Then we describe some of the problems inherent in rendering CSG by ray tracing. We present our new algorithm and show that it deals correctly with the problem cases and we conclude with a brief discussion and summary.

1.0 Regular Sets

The need for regular sets is described by Tilove (1980). Figure 2 shows two objects, A and B, that share a common edge. In 3D they share a face. The object, A – B, is identical to A except that a part of its *surface* is missing. The object, A - (A - B), is an infinitesimally thin surface only. The idea of regular sets is to throw away all such surfaces and declare them, a-priori, to be not part of any object. Similarly, the object, (A - B) + (B - A), is almost the same as A + B. Only an infinitesimally thin interior surface is missing. The regularization routine should fill in this missing surface.

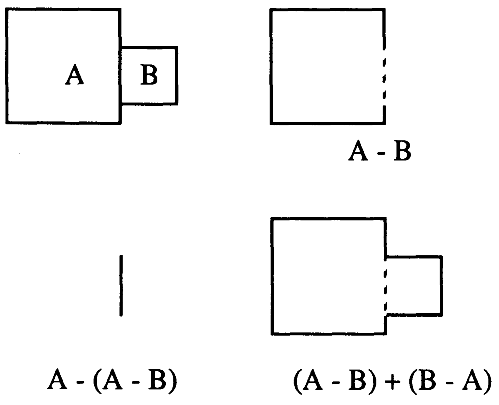


Fig. 2, Non regular sets.

Because objects of no thickness cannot exist in the real world, it is sometimes argued that the regularization of sets produces a more realistic model. However, there is a sense in which such objects do exist. Suppose, for example, we grind a steel plate flat on a surface grinder. Then we braze onto the surface, a small, flat piece of copper. If we attempt to grind the copper away without touching the steel, we will always leave a little behind. If we regrind the plate to the original dimensions, we will leave behind an area where there is some copper and some bare steel on the surface. The shapes of these areas will depend on the errors inherent in any machining operation.

To understand the relevance of this example, we need to examine the errors inherent in a computer representation. To simplify the argument, we will assume that the numbers that represent our coordinate system are fixed point fractions. Then continuous space is represented by a finite grid of points and any position can only be represented by its nearest grid point. The use of floating point numbers does not change this argument. It merely complicates it because the grid no longer has a uniform spacing. If our surfaces are aligned with the coordinate axes, it becomes easy to detect coincidence. But when they are not, we get complications.

Figure 3 shows the objects of Fig. 2 but in a different rotation and with an enlarged grid of representable points shown explicitly. Notice that you cannot place object B with its vertices on grid points so that its edge coincides with that of object A. If you place the objects as shown on the left, then some points close to the surface are clearly outside both A and B. If you place them as on the right, the vertices of B lie inside A.

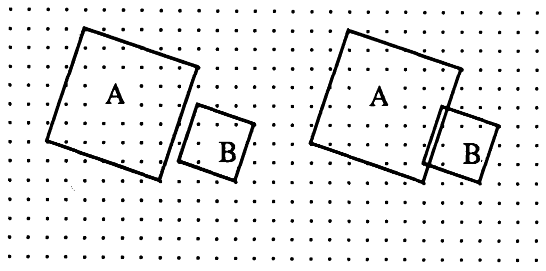


Fig. 3, Coincident surfaces.

In this example, the sides of the two squares have been shown parallel. If the objects are represented by vertex positions, and general transformations are applied to these positions, then the sides or faces will not necessarily be parallel. In a case like that, coincident 'parallel' surfaces can be represented so that on one surface, points can be found that lie inside, outside or on the surface of the other object.

2.0 Ray Tracing

The principle of rendering a CSG model by ray tracing was described by Roth (1982). The intersections each ray makes with the primitive objects are found and these intersection points are ordered by distance from the ray's source. The section of ray between each pair of intersection points can be classified as inside or outside the object and the intersection point at the first inside/outside boundary is the correct one for that ray.

This process finds the correct surface points but can produce some strange effects when surfaces are coincident. Figure 4 shows the effect of what we call ghost surfaces. Two closed cylinders have been built by subtracting plane half spaces from cylindrical half spaces. Then the cylinders have been added to make a simple scene. The top surfaces of the two short cylinders happen to lie in the same plane. Consider what happens when a ray is directed at cylinder B. It intersects with two surfaces, the plane on top of A and the plane on top of B. Since these planes are identical, it should happen that the intersection points are the same. Suppose, the ray tracer happens to deal with the intersection with the surface on A first. The section of ray before the intersection is outside the object and the section of the ray after the intersection is inside. Thus, using Roth's rule, the intersection with the top plane of A can be wrongly identified as the correct intersection. If A and B have different colors, this will give a wrong result.

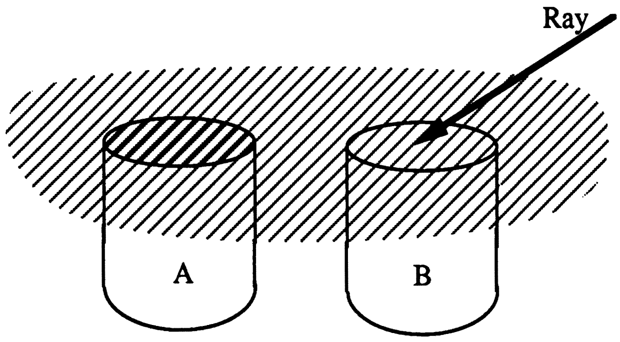


Fig. 4, Ghost surfaces. The ray strikes the top plane of cylinder A and immediately enters B. This intersection can be identified as correct but it carries the color of A.

The correct colors in cases like this can be found by using a more elaborate way to classify points with respect to objects (Wyvill 1988). This classification is not described in detail here, because the algorithm given in Section 3 supercedes it.

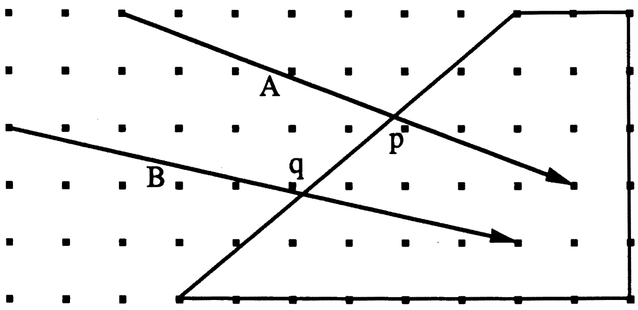


Fig. 5, Two rays intersect a surface. Due to rounding errors, the intersection, p, with ray, A, lies inside the surface. The intersection, q, with ray, B, lies outside.

When rounding errors are taken into account, other strange effects can occur. Figure 5 shows two rays, A and B, directed at a plane surface. Representable points are shown by dots. If the intersection routines find the nearest representable point to the ideal intersection, then the intersection, p, with A lies inside the surface while the intersection, q, with B lies outside the surface. This is the usual case and our algorithms must expect it. Notice that these errors are present even though the end points of the rays, and the vertices of the object fall exactly on representable points.

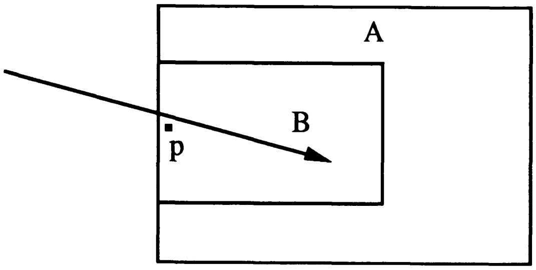


Fig. 6, CSG addition. The intersection of the ray with B is inside A and the intersection with A is inside B, so both are rejected.

Suppose part of a model has genuinely coincident surfaces. This is shown in Fig. 6. Due to rounding errors, the intersection point of a ray with object A can be inside object B while the intersection with B is inside object A. In this case, the algorithm described by Wyvill (1988) fails. Both intersections are thrown away and the ray passes through the surface of the object.

A similar effect can occur in CSG subtraction as shown in Fig. 7. Here the common intersection point is inside the subtracted object and outside the other object. Again, the intersection is missed. There are published algorithms that will not miss surfaces like this, but we know of none that, like Wyvill (1988), also guarantee the correct inheritance of color properties.

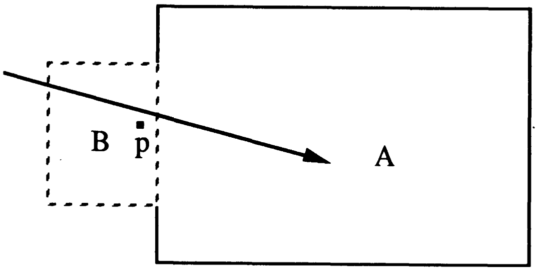


Fig. 7, CSG subtraction, A - B. The intersection of the ray with B is outside A and, therefore, discarded. But the intersection with A is inside B, so both are logically rejected.

Rounding errors can produce strange effects in ray tracing, whether you are using a CSG model or not. In Fig. 8, a ray strikes a reflective surface. Because of rounding errors, the intersection point actually lies *inside* the surface. The reflected ray, therefore, strikes the same surface from inside. Similarly a transmitted ray can start from *outside* the surface and immediately intersect the same surface. We cannot simply ignore a second intersection with the same surface, because there are cases where this does happen: reflections inside a spherical bubble for example.

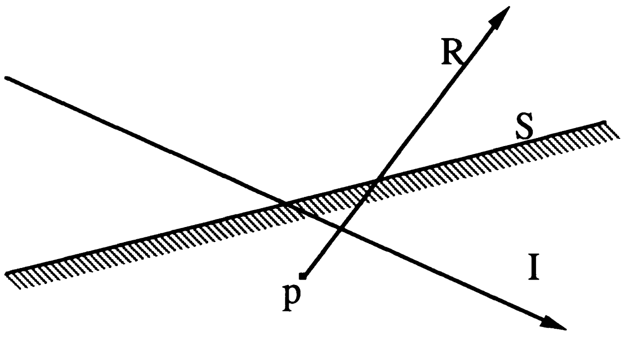


Fig. 8, Ray I intersects the surface, S. The intersection point is represented by p: the nearest grid point. Since p is actually inside S, the reflected ray, R, also intersects S.

Most of these problems can be circumvented by introducing an error limit, ε, within which unequal numbers are regarded as equal. A point is regarded as *on* a surface if it lies within ε of the surface and systems that operate on regular sets use such a tolerance value to avoid inconsistencies. The idea is appealing. Floating point numbers can be represented to about twenty significant figures, while for engineering tolerances, six is plenty. If we set ε to 10^{-16} units, it is vanishingly small compared to the dimensions of our model, yet enormous compared to the likely errors of computation.

In the context of ray tracing, using the error factor, ε, implies changing the way we test to see whether a point lies inside or outside a primitive object. We need to know, in addition, if it is close enough to the surface to be regarded as *on* the surface. A given ray is tested for intersection with the primitives and the intersections are stored in a list, ordered by distance along the ray from the ray origin. If two, or more of these intersection points lie within a distance ε, they can be regarded as coincident and appropriate special action taken (Amanatides 1990).

The problem of Fig. 8 is handled by shifting the intersection point by an amount, ε, along the ray, so that the reflected ray starts on the correct side of the surface. Similarly, a transmitted ray, is started from a false position, ε units along the ray and inside the surface.

There are, however, a number of good reasons for not using an error limit in this way:

1. It is logically unsound. We can predict the circumstances where near coincidence and rounding errors cause problems. We should address those problems directly.

2. Sooner or later, a user will place two surfaces exactly ε units apart. The result is that, due to rounding errors, some points will be recognized as *on* the surface while others will not. All of the original problems will reappear.

3. There are cases where ideal objects become very thin. Figure 9, for example, where two spheres are subtracted. Use of ε can produce a hole in the top. This hole is incorrect and only caused by the use of ε. Without an ε there is no hole.

4. Almost coincident surfaces occur in nature and produce the same effects (machining away the copper on the steel surface). If you put a red surface and a blue surface in the same place, the system might produce a random mixture of these colors. It is much harder to explain why this happens when you deliberately put them ε units apart.

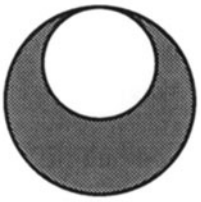


Fig. 9, Object created by subtracting spheres (in cross-section). There is a single point of no thickness at the top. This means that there is a finite hole if a very thin surface is regarded as absent.

3.0 The New Algorithms

The basic idea behind our new algorithms is to keep track of inside/outside information by recording when it changes, i.e. at intersection points. To make this work, in practice, you have to deal with some special cases and trying to explain these makes the main ideas difficult to see. Therefore, we describe the principles first and the special cases afterwards

3.1 Material Properties

The motivation for our earlier paper (Wyvill 1988) was to be able to represent models such as the eggcup in Fig. 10. Layers of different material are added together to make a composite block of material and the final shape is carved from this block. The final pattern on the surface is produced by revealing areas of differently colored volumes of material within the block. If a CSG system allows union of sets, and if the internal colors of materials are regarded as important, then we need a rule to determine which color to use for a volume that is common to the components of a union.

We avoid this ambiguity by using an asymmetric addition operator, $+\!\!\!+$, instead of set union. By definition:

$$a +\!\!\!+ b \equiv (a - b) + b$$

This means that the priority of volume properties is implicitly defined whenever objects are added together. We believe this system is easier to control than other systems such as Salesin's (1990).

When dealing with transparent objects, things get more complicated. At inner surfaces, it is necessary to know the refractive index of the medium the ray is leaving as well as the one it is entering. The rules described in the next section derive this information, unambiguously, from the CSG tree.

3.2 Point Classification

The new algorithm needs to classify each intersection point with respect to a CSG tree. We describe the classifier for completeness, and to provide a context for the rest of the algorithm. The method of classification is not new. It is a minor extension of an algorithm previously published (Wyvill 1988). The result of the classification is a record that describes the state of the ray passing through the given point. The possible classifications are:

- *OUT*
- *IN, medium*
- *BORDER, from, to*

OUT means that the point is outside the object described by the given tree. *IN* means that it is inside. If it is inside, then *medium* is a pointer to a structure that describes the properties of the material surrounding the point. If the classification is *BORDER,* then it is accompanied by two values, *from* and *to,* that are pointers to the descriptions of the material that the ray is leaving and the material that the ray is entering.

The simplest CSG tree of all is a *leaf node* that describes a single primitive object and its material properties. In our system, these are half spaces defined by functions (Wyvill 1986). When a point is tested with respect to a primitive, the only possible values are *IN* and *OUT*. In the case of *IN, medium* is set to the appropriate material. The value *BORDER* is treated specially. It is assigned only to a point, with respect to a particular instance of a primitive, when that point is created as the result of an intersection test with the primitive. The primitive intersection function also finds the surface normal and determines whether the ray is entering or leaving the primitive. If the ray is entering the primitive, then *to* is set to the material of the primitive and *from* is set to *empty*. If the ray is leaving the primitive, then *from* is set to the material of the primitive and *to* is set to *empty*.

Each intersection point is described by a structure that includes a pointer to the *leaf node* with which it was intersected. So, to classify an intersection point with respect to a *leaf node,* we first check to see if this is the particular *leaf node* for which the point has the property *BORDER*. If it is that node, then we have classified the point. Otherwise, we call the appropriate function to classify the point as *IN* or *OUT*. Notice that a point can be classified as *BORDER* with only one *leaf node*.

To classify a point with respect to a general node of the CSG tree, we first classify the point (recursively) with respect to the sub-trees and then combine the classifications according to a set of rules that define the behaviour of that node. The tables below describe these rules for the asymmetric addition node and the subtraction node. The special material *empty* is denoted by Ø.

+ right node value left node value	*IN, r*	*OUT*	*BORDER, c, d*

IN, l	*IN, r*	*IN, l*	if *c* = Ø, *BORDER, l, d*
			if *d* = Ø, *BORDER, c, l*
			else *BORDER, c, d*

OUT	*IN, r*	*OUT*	*BORDER, c, d*
BORDER, a, b	*IN, r*	*BORDER, a, b*	

Table 1, Combination rules for asymmetric addition node

− right node value left node value	*IN, r*	*OUT*	*BORDER, c, d*

IN, l	*OUT*	*IN, l*	if *c* = Ø, *BORDER, l,* Ø
			if *d* = Ø, *BORDER,* Ø*, l*
			else *OUT*

OUT	*OUT*	*OUT*	*OUT*
BORDER, a, b	*OUT*	*BORDER, a, b*	

Table 2, Combination rules for subtraction node

3.3 Inside / Outside Determination

The problems described in Section 2 all have a root cause in that the state of inside/outside information has become inconsistent. In the real world, we cannot go from A to B without crossing the intervening space. But in the false world of computer representation, we allow a ray to proceed from outside an object to inside without going through an intersection.

We can enforce consistency by insisting that, for each leaf node, the state of the ray, *IN* or *OUT*, can change only when an intersection has been found. In simple ray tracing, every primary ray emanates from an eyepoint and every secondary ray is spawned from an intersection point on an earlier ray. The rays, therefore, form a tree structure in the scene, and it is all connected. Suppose we determine the state, *IN* or *OUT*, of the eyepoint with respect to all the *leaf nodes* in the scene. As we traverse the graph represented by the rays, we update this information when, and only when, a ray intersects the boundary of a primitive. When the classifier wants the status of a *leaf node*, it is found stored in the *leaf node* structure. Only the initial *IN* and *OUT* values need be determined from the geometrical test.

3.4 The Ray Tracer

We find all the intersection points along the ray and order them by distance from the eye. Each intersection, in turn, is checked for validity (see below) and then classified with respect to the CSG tree, until one is found with the classification *BORDER, x, y*. This is the correct intersection point and the ray is crossing from medium, x into y. If x = y, the intersection can be discarded.

As each intersection point is classified, its *leaf node* is updated because the ray has either entered or left so that further points along the ray will be classified correctly.

Transmitted rays, after calculation of refraction direction, are spawned from the intersection point which will be regarded as inside the surface just intersected. Reflected and light-source-seeking rays are also spawned from the intersection point, but the *leaf node* has already been updated. This means that the reflected and light-seeking rays are, logically speaking, spawned from outside the surface.

Let us see how this process handles each of the special cases. Firstly, ghost surfaces are eliminated by the ordered tree traversal performed by the classification routine. Figure 11 shows the tree corresponding to the objects in Fig. 4. The ray shown at the top makes intersections with the top surfaces of *A* and *B*, but the classification process quickly eliminates the intersection with *A* because it is *OUT* with respect to the cylinder.

In the case of coincident surfaces, we will have two (or more) intersections that are in the same place, or nearly the same place. Take, for example, the problem of Fig. 6. A single surface is intersected by a ray and the intersection point, p, lies inside the surface. Let us suppose that this surface represents the surface of $A+\!\!+B$, where the sub-objects *A, B*, share this surface. Because the intersection points are identical, the intersection with *A* is really inside *B*, but if we process the intersection with A first, we will be logically outside B because we have not yet crossed that boundary. The intersection with B is also inside A, but if we process B first, we will be logically outside A. In either case, the ray tracer takes the correct action and gives us a single, valid intersection.

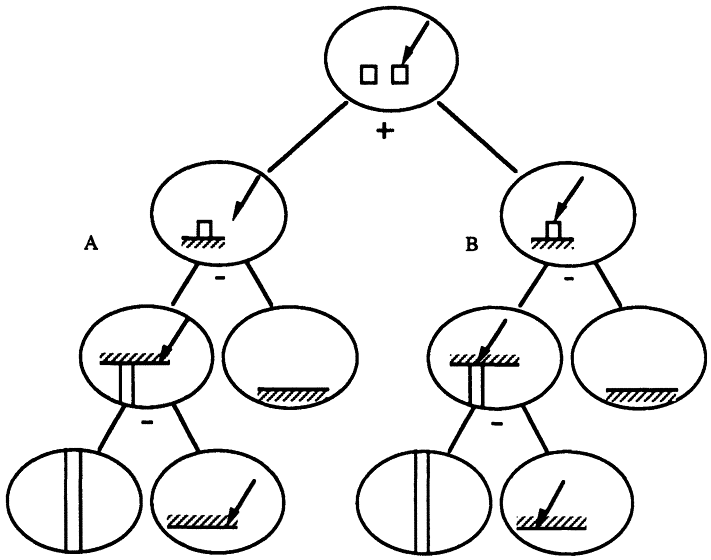


Fig. 11, Elimination of ghost surfaces.

If the surfaces of *A* and *B* are not quite parallel, the intersection with *A* can be inside or outside *B,* while the intersection with *B* can be inside or outside *A*. This gives four distinct cases which are handled consistently by the table. When the surfaces are exactly coincident, we will always strike the same surface first. Whether it is A or B will depend on the ordering of the CSG tree and the way our sorting algorithm handles identical points. When the surfaces are almost identical, rounding errors can result in some rays being identified as hitting A while some strike B. This is in accord with our physical example of grinding the steel and copper. In no case is an intersection missed.

Figure 7 is more complicated. There are four distinct cases:

i Intersection with A outside B, intersection with B outside A.
ii Intersection with A inside B, intersection with B inside A.
iii Intersection with A inside B, intersection with B outside A.
iv Intersection with A outside B, intersection with B inside A.

It can be seen, from Fig. 12, that all four cases represent possible, legitimate solids. This means that ordinary inside/outside tests would inevitably lead to a wrong interpretation of Fig. 7. In Fig. 7, the intersection with object A would be treated as case ii in Fig. 12 because the intersection is within object B. The intersection with object B in Fig. 7 would be treated as case i in Fig. 12. The reason for the wrong interpretation is that the inside/outside test has been inconsistent. If we encounter the intersection leaving B first, then we can argue that the intersection with A, encountered later, cannot be inside B because we know that we have left B. The new algorithm either finds the intersection with A before it leaves B, or else finds the intersection with B before it enters A. Either way, a single intersection is found.

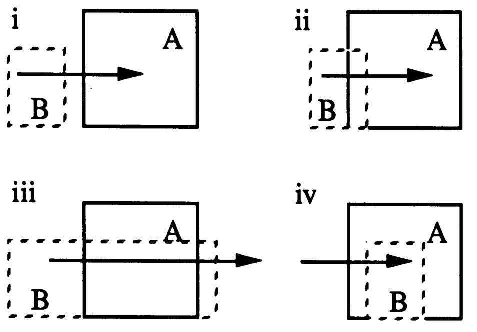


Fig. 12, Four cases of intersection relations for A - B.

3.5 Validity Check

Because the ray tracer carries the logical inside/outside information for every *leaf node*, we no longer need to adjust intersection points to avoid spurious intersections with reflected or transmitted rays. In the case of Fig. 8, the reflected ray, *R*, starts inside the surface due to rounding errors, but it is still logically outside. The intersection of *R* with *S* can be rejected because we are *leaving* a *leaf node* when we are already out of it.

We say this intersection is invalid. We recognize it, unambiguously, because the primitive intersection routine has recorded whether we are entering or leaving, and the *leaf node* tells us whether we are currently *IN* or *OUT*. When an intersection is rejected because it fails this validity check, the *leaf node* is not updated because there has been no change in its status.

3.6 When multiple surfaces cannot be avoided

There is one remaining problem with coincident, transparent surfaces. After the first intersection has been found, the intersection point may lie inside one or more additional surfaces. If we simply cast the transmitted ray, it will intersect immediately with the additional surfaces and these intersections will be behind the ray origin. See Fig. 13.

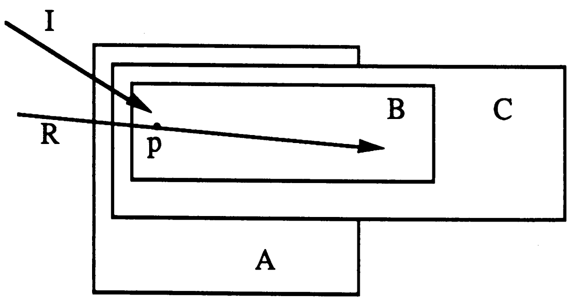


Fig. 13, Coincident surfaces. Incident ray, I, intersects with A, but due to rounding errors the intersection, p, is inside B and C. The refracted ray, R, makes valid intersections with B and C, but these are behind p.

There is no way to distinguish these surfaces from any others behind the ray origin so we cannot permit general intersection points to be behind. The solution is to identify these surfaces as we process the original ray.

Whenever the classifier returns *BORDER x, y,* we look ahead at the next intersection, if any, on the list. We perform a geometrical inside/outside test to see whether the *current* intersection point lies beyond the surface of the next intersection. If it does, then we have detected a coincident surface, and we must pass through it before spawning the transmitted ray. The correct action, in this case, is not clear. By definition, we are passing through an infinitesimally thin surface. Should we generate another reflected ray? In our present implementation, we compromise. The coincident surfaces are crossed, in order, until either the ray is stopped by an opaque surface or the next surface is beyond the original intersection point.

This creates a sequence of values, from the classifier, of the form:

BORDER a_1, a_2 *BORDER* a_2, a_3 .... *BORDER* a_{n-1}, a_n

Of course, this list will be longer than two or three only in pathological cases. If all the surfaces are transparent, or if some of the surfaces are subtracted, the incident ray will continue and, in principle, we could generate reflections from each interface. What we do at the moment, is to treat the whole sequence as a single intersection: *BORDER* a_1, a_n.

3.7 The Implementation

The ray tracer relies on being able to classify the origin of the ray with respect to all the objects in the scene. This is done separately for each ray. Initially the ray is cast into the scene and all intersection points with all objects are inserted into a list. This list is then sorted by distance along the ray. To classify the eye point with respect to the primitives, the *IN* or *OUT* status of each object with intersection points behind the eye is reversed as the ray progresses through each of its intersection points up to the eye.

For primitives that are not closed, it is necessary to determine unambiguously what happens at the first intersection point. The ray could be either entering or leaving the object. We choose a reference point some large distance along the ray, before the intersection point, and determine, numerically, whether the point is inside or outside the primitive. Provided that the reference point is not close to any intersection point, this result will be consistent.

Currently, we use a scheme of uniform space division (Fujimoto 1986) where the world is divided into voxels that are either uniform or active. Uniform voxels are empty or full of material belonging to only one *leaf node*. Active voxels contain boundaries, and these are represented by a reduced CSG tree (Wyvill 1986). Ghost surfaces, as in Fig. 4, cannot contribute valid intersections, so a voxel containing only ghost surfaces would be uniform. We traverse the voxels using a variant of Cleary's algorithm (Cleary 1988). If our space division routines have done their job we can guarantee that every voxel contains all the information necessary to represent the scene within the voxel. We can now consider each voxel on its own, as if it were the whole scene being rendered. Every time a ray enters a new voxel the ray is checked for intersections with all the primitives and any valid intersection point within the voxel is a true intersection point between the ray and scene.

4.0 Results

Figure 14 shows some simple objects ray traced without avoiding the rounding error problems and a correct version using the new ray tracer. Each object has been added to itself to create coincident surfaces. Some rays penetrate to the blue background, while some find the correct surface but are then shadowed by the surface they have just intersected.

Figure 15 shows a spherical lamina. A sphere has been subtracted from itself, creating exactly matching surfaces, and a plane subtracted from the top. This produces coincident intersection points wherever a ray strikes the sphere. The sorting routine always finds the positive sphere first so the second intersection is still inside the subtracted one. The effect of this is that the lamina is visible only from the outside. Although rays pass through the back surface, it is visible in the reflective backplane.

Figure 16 is a simple scene that includes a transparent object and Fig. 17 shows a more complicated scene containing over 900 primitive objects with reflection and texturing.

5.0 Discussion

It could be argued that we have used regular sets in that our classifier recognizes only inside and outside points except for the intersection itself. We have disallowed coincident surfaces as detected by each ray, but we are capable of detecting the surface of an object subtracted from itself, although the resulting object has no thickness.

There are many places where the algorithm can be improved. For example, it is probably not necessary to perform a complete, top down tree traversal at each intersection. Only one surface has been intersected, and it should be possible to use this information to find the change in current medium. The new ray tracer does a little more work in finding intersections that get discarded after reflection. However, it avoids the calculations associated with error limits and fudge factors, and where repeated inside/outside tests used to be needed, it uses stored flags.

5.1 Conclusion

Scenes can be rendered directly from CSG models without using regularized operators and without resorting to fudge factors to decide when boundaries coincide.

A practical version of the ray tracer works at similar speeds to a conventional algorithm using error limits, in a system where space division is used.

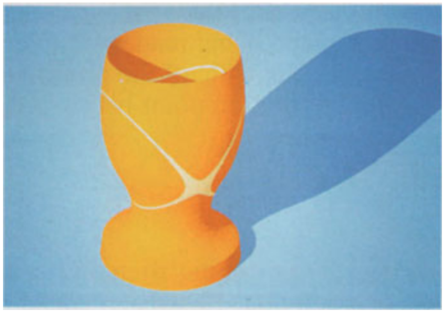


Fig. 10. Computer generated design for an eggcup turned from layered wood.

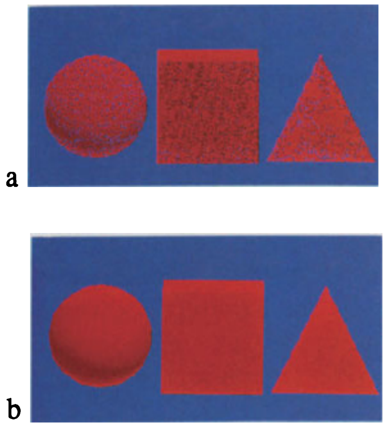


Fig. 14. (a) Errors produced by coincident surfaces, (b) The errors corrected with the exact ray tracer.

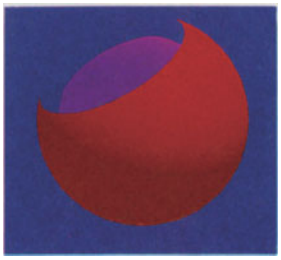


Fig. 15. Spherical lamina with reflective backplane. The object is invisible from the inside.

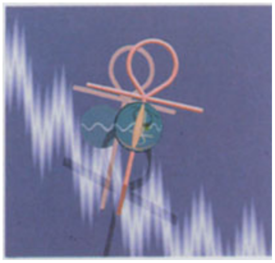


Fig. 16. Example with transparency.

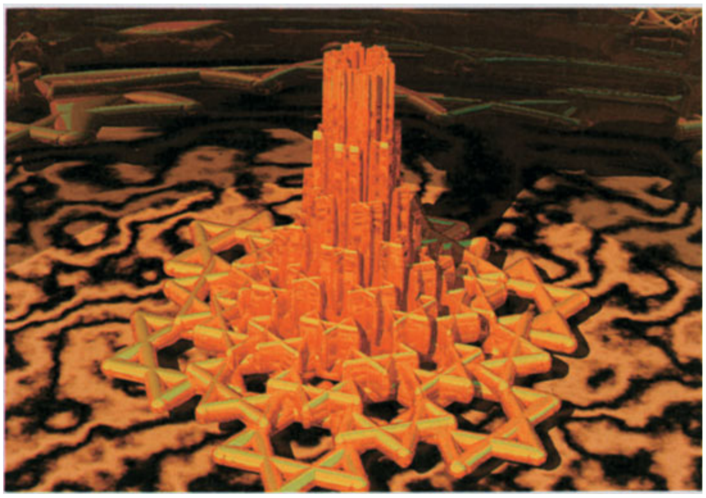


Fig. 17. *Masterpiece* a complex design rendered with the new algorithm.

This approach should be particularly useful for building ray tracing hardware, because it enables the full accuracy of the computer's number system to be used. With modern workstations, there is little advantage in using reduced precision arithmetic, because the floating point hardware returns full precision results more quickly than special software. But a hardware ray tracer could gain some advantage by working in fixed point and limited precision.

Acknowledgements

We wish to thank the other members of our graphics research group for proofreading and help with bug fixes. Figure 1, *The Rose,* was designed by Stuart Smith and Fig. 17, *Masterpiece,* by Craig McNaughton. This research has been funded by the University of Otago. We are also grateful to École Polytechnique Fédérale de Lausanne where the final copy and photographs were completed.

References

Amanatides J, Mitchell DP (1990) Some Regularization Problems in Ray Tracing, Proc. Graphics Interface '90, 221-228.

Cleary JG, Wyvill G (1988) Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Division, *The Visual Computer* 4(2): 65–83.

Fujimoto A, Tanaka T, Iwata K (1986) ARTS: Accelerated Ray-Tracing System, *IEEE Computer Graphics and Applications* 6(4): 16–26.

Mäntylä M (1988) *An Introduction to Solid Modeling,* Computer Science Press, Rockville, MD.

Mortenson ME (1985) *Geometric Modeling,*Wiley, New York.

Muuss MJ, Butler LA (1990) Boolean Operations on Boundary Representation Solids Using n-Manifold Geometry, Proc. Ausgraph '90, 291–299.

Roth SD (1982) Ray Casting for Modeling Solids, *Computer Graphics and Image Processing* 18(2): 109–144.

Salesin D, Stolfi J (1990) Rendering CSG Models with a ZZ-Buffer, Proc. SIGGRAPH '90, *Computer Graphics* 24(4): 67– 76.

Segal M (1990) Using Tolerances to Guarantee Valid Polyhedral Modeling Results, Proc. SIGGRAPH '90, *Computer Graphics* 24(4): 105–114.

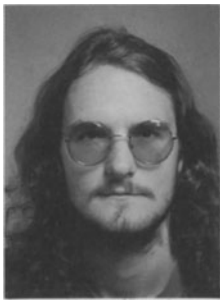
Tilove RB (1980) Set Membership Classification: A Unified Approach to Geometric Intersection Problems, *IEEE Transactions on Graphics,* C-29, 10: 874–883.

Wyvill G, Sharp P (1988) Volume and Surface Properties in CSG, Proc. CGI '88, *New Trends in Computer Graphics,* Springer-Verlag, 257–266.

Wyvill G, Kunii TL, Shirai Y (1986) Space Division for Ray Tracing in CSG, *IEEE Computer Graphics and Applications* 6(4): 28–34.

Geoff Wyvill graduated in physics from Jesus College, Oxford, and started working with computers as a research technologist with the British Petroleum Company. He gained MSc and PhD degrees in computer science from the University of Bradford where he lectured in computer science from 1969 until 1978. He is currently senior lecturer in computer science at the University of Otago. He is on the editorial board of The Visual Computer and is a member of SIGGRAPH, ACM, CGS and NZCS.

Address: Department of Computer Science
University of Otago Box 56
Dunedin, New Zealand



Andrew Trotman is a graduate student at Otago University. His research interests include constructive solid geometry and computer animation. He completed a BA degree in computer science in 1988 and he is a student member of ACM and SIGGRAPH.

Address: Department of Computer Science
University of Otago Box 56
Dunedin, New Zealand