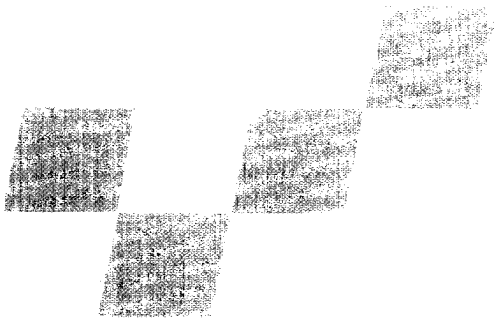


A New Space Subdivision for Ray Tracing CSG Solids

Jung-Hong Chuang
National Chiao Tung University

Weun-Jier Hwang
Telecommunication Laboratories

Ray tracing successfully creates realistic images of CSG solids. A nonuniform space subdivision scheme that reduces intersection computations and point classifications could make this technique even more attractive.

In solid modeling, the two most common schemes for representing solids are constructive solid geometry (CSG) and boundary representation (B-rep). CSG has proved advantageous in several types of computations, such as the construction of models. Other computations, such as rendering, require explicit geometry about the boundary of the CSG solid. Boundary information for the CSG tree can generally be obtained through boundary evaluation, but this is a highly complex task.

Ray tracing is widely recognized as a powerful, effective technique for producing realistic images of 3D scenes especially because it is simple to implement and can model a variety of visual phenomena, such as specular reflection, transparency, and shadows. Unfortunately, ray tracing is extremely time-consuming because of many ray-object intersection computations. Numerous methods to speed up the tracing process have been proposed, such as bounding volume and bounding hierarchy[1] and space subdivision.[2] Space subdivision is preferable to other methods because of its consistent performance in scenes of varying complexity. Different space subdivision schemes are possible, each with advantages and disadvantages, such as binary space partition (BSP) trees, octree decomposition, and uniform subdivision.

Several methods for ray tracing CSG solids based on space subdivision have been proposed.[3-5] Previous studies generally have emphasized deriving suitable space subdivision schemes to reduce the number of intersection computations and to efficiently trace the rays.

Another issue in ray tracing CSG models is the vast number of point classifications, whereby ray-primitive intersection points are classified with respect to the CSG tree to determine if they lie on the boundary of the resulting solid. To speed up point classification, Jansen[6] and Roth[7] proposed methods for general rendering techniques.

In this article, we describe a nonuniform space subdivision scheme that reduces both the number of ray-object intersection computations and point classifications. Our method uses the face planes of the primitives' *S*-bounds,[8] described later, in a bottom-up fashion and produces a subdivision wherein the localized CSG tree in each leaf voxel is greatly minimized. The use of *S*-bounds in the space subdivision effectively reduces the number of intersection computations as well. The reduction of the localized CSG tree in turn further reduces the number of intersection computations and point classifications. We briefly review existing methods for ray tracing CSG solids, describe our proposed space subdivision method, discuss our implementation and compare it to Bouatouch's[4] method, and summarize our test results.

Direct rendering of CSG solids

CSG solids can be rendered directly via ray tracing, for example, or displayed indirectly by first converting CSG trees to B-rep and applying graphics hardware common in modern workstations. Indirect rendering looks attractive because boundary evaluation is more efficient today. Moreover, once the B-rep is available, the viewpoint can be changed without regenerating the B-rep. However, shading techniques typically cannot produce realistic images. Although radiosity has grown more popular for global illumination, its ability to effectively handle specular effects is limited. Ray tracing is thus still important for realistic off-line rendering of CSG solids.

Of the many techniques proposed for direct rendering of CSG solids, most extend rendering algorithms originally developed for computer graphics. These include CSG scan-line algorithms,[9] CSG *z*-buffer algorithms,[10] and CSG ray tracing.[3-5,7] Two concerns arise when using ray tracing to render CSG solids: the complexity of computing ray-object intersections and the need to determine (classify) whether or not the intersection points lie on the resulting solid.

Roth was the first to apply ray tracing to display CSG models.[7] His algorithm projects each subtree's bounding

0272-17-16/95/$4.00 © 1995 IEEE

box to the view plane and reduces the number of intersection computations.

Wyvill used octree decomposition to subdivide the space until each voxel is classified as inside the CSG solid, outside the CSG solid, or of a size less than a prescribed tolerance.[5] Cells that overlap the boundary of the CSG solid are associated with reduced CSG expressions by which the intersection computations and point classifications can be performed.

The so-called *minimal bounding box* of the primitive can be used to prune portions outside the CSG solid.[3,4] Arnaldi subdivided the view plane through a BSP based on the projection of the minimal bounding boxes. He then extended the 2D partition to 3D with further subdivision along the view direction.[3] Bouatouch employed BSP-based subdivision to subdivide the space. He used the number of minimal bounding boxes in a voxel as a stopping criterion.[4]

Costs associated with ray tracing CSG solids using space subdivision include ray-object intersection computation, ray traversal, and point classification.[11] Earlier studies examined the cost effect of different subdivisions on intersection computation and ray traversal, but not on point classification. Point classification is combinatorial in nature and might require neighborhood information in some cases. Moreover, it is an inner loop element in the ray tracing process. Hence, it might consume much of the total processing time, especially for a large CSG tree. Any space-subdivision scheme must take this into account.

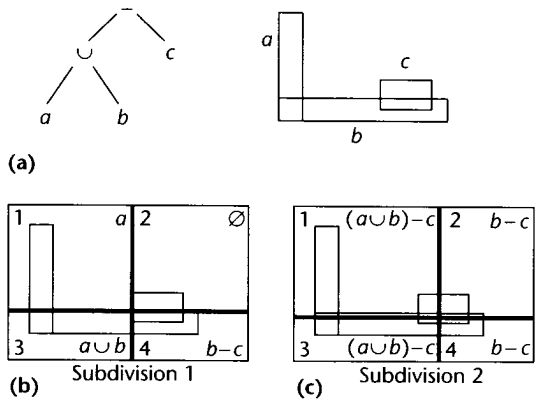
Effects of space subdivision

When CSG solids are ray traced using space subdivision, each voxel is generally assigned a sub-CSG tree, which is the localization of the CSG tree on that voxel.[3,4] Different space subdivisions result in different sub-CSG tree assignments and thus different performance in later intersection computation and point classification.

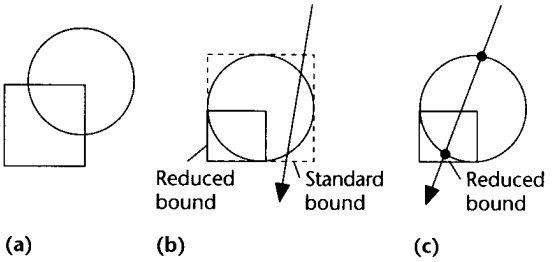
For the CSG solid in Figure 1a, let's examine the sub-CSG tree assignment resulting from the two space subdivisions shown in Figures 1b and 1c. Voxel 1 of subdivision 1 is associated with a tree of single node a, whereas voxel 1 in subdivision 2 has a sub-CSG tree for $(a \cup b) - c$. Consequently, an intersection point in voxel 1 of subdivision 1 must be classified against tree a, while the same intersection in subdivision 2 must be classified against $(a \cup b) - c$. Moreover, the ray intersecting voxel 1 in subdivision 2 must be checked to see if it intersects with primitives a, b, and c, but in subdivision 1 only primitive a in voxel 1 must be checked. Subdivision using the boundary planes in subdivision 1 results in a much simpler sub-CSG tree assignment, which reduces the intersection computations and point classifications.

Bounding box and S-bounds

The bounding box technique is widely acknowledged as useful in reducing the number of ray-object intersections.[7] A primitive's standard bounding box might contain portions outside the resulting CSG solid. If the box can be reduced to exclude these portions, then both the intersection computation and the point classification will be more efficient.



1 For the CSG solid in (a), two different space subdivisions shown in (b) and (c) result in different sub-CSG tree assignments.



2 (a) The primitive and the effects of a ray intersecting the primitive's (b) standard bounding box and (c) reduced bounding box.

As an example, consider a circle differencing a square as shown in Figure 2a. Figure 2b depicts a ray that intersects the circle's standard bounding box but not the reduced bound. Figure 2c shows that only the intersection point lying inside the reduced bound must be classified. Similar effects are achieved by reducing the bounding box for the CSG tree's internal nodes.

The bounding box at the root of the CSG tree can be propagated downward toward the leaves, further refining the primitive's bounding box.[3,4] The *S*-bounds concept uses a similar idea but with additional steps that repeat the upward and downward propagations until the size of the primitive's bounding boxes cannot be reduced further.[8] Initially, each primitive is associated with a bounding box. In the upward propagation, we evaluate the CSG tree based on the primitives' current bounding boxes and derive a new bounding box for each internal node, including the root. In the downward propagation, we propagate the current bounding box at the root downward toward the leaves. At each level we calculate the intersection to further refine the bounding box of each child node. Subsequent upward and downward propagations minimize the size of each node's bounding box, including the root, internal nodes, and leaves.

Subdividing the space based on face planes of the *S*-bounds effectively reduces intersection computations and minimizes the size of the sub-CSG tree assigned to each leaf voxel. Minimizing each leaf voxel's sub-CSG tree can further reduce the number of intersection computations and point classifications. These reductions are especially significant for ray tracing CSG solids containing complex primitives such as tori and sweeping volumes. Traditional space subdivisions (for example, octree and BSP) generally do not consider the geometry of CSG primitives. As a result, the sub-CSG trees can be highly complex given densely distributed primitives.

Ray tracing with a new space subdivision

A nonuniform space subdivision based on S-bounds minimizes the size of the bounding boxes as well as the size of the sub-CSG tree in each voxel. The increased processing cost for ray traversals resulting from nonuniform subdivision can be reduced through a suitable data structure, which we describe later in this section, and can be ameliorated by reduced intersection computations and point classifications.

A new space subdivision algorithm

The nonuniform space subdivision algorithm we propose consists of a preprocessing step and a subdivision step. In the preprocessing step, we perform the S-bounds computation and produce an S-bound for each node of the CSG tree.

The objective of the subdivision step is to subdivide the space so as to minimize the sub-CSG tree assigned to each leaf voxel. The space subdivision yields a *subdivision hierarchy* and assigns a sub-CSG tree to each leaf voxel of this hierarchy. The data structure for the voxel on each level of the hierarchy contains pointers both to its child voxels and to its neighboring voxels. Each leaf voxel of the hierarchy also contains, in addition to pointers, the associated sub-CSG tree.

To construct the subdivision hierarchy of a CSG tree, we subdivide the root's S-bound by upwardly propagating the subdivision hierarchies at both child nodes. This process is applied recursively to each internal node of the CSG tree. Consequently, the subdivision proceeds in a bottom-up fashion, starting from the subdivision hierarchies of primitives (the primitives' S-bounds). The sub-CSG tree associated with each leaf voxel of the subdivision hierarchy is assigned or updated during upward propagation. We obtain the subdivision hierarchy of a CSG tree by calling the recursive routine `Space_Subdivision(node)` with parameter `node` the *root* of the given CSG tree. The input parameter to this routine is a node of the CSG tree.

```
Space_Subdivision(node) {
  if (node is a primitive)
    return primitive's S-bound as
    its subdivision hierarchy and
    the primitive itself as the
    associated sub-CSG tree;
  else {
    left_subdivision_hierarchy
    = Space_Subdivision
      (left_child_node);

    right_subdivision_hierarchy
    = Space_Subdivision
      (right_child_node);

    Merge_Subdivision_Hierarchy(node,
    left_subdivision_hierarchy,
    right_subdivision_hierarchy);
    }
}
```

Here we describe how the `Merge_Subdivision_Hierarchy()` is computed for a CSG node $p = l \otimes r$, where $\otimes$ is any Boolean operator and l and r are left and right children of p, respectively. Let L, R, and P be the S-bounds of l, r, and p, respectively, and S_l and S_r be the subdivision hierarchies of l and r, respectively. The subdivision hierarchy of node p, denoted as S_p, is constructed by subdividing its S-bound P using the subdivision hierarchies S_l and S_r. The following two steps describe this process:

1. *Subdivide S-bound P using the subdivision hierarchy of l.* If $\otimes$ is a union operator, we subdivide S-bound P using the face planes of L and embed the subdivision hierarchy S_l into the leaf voxel that coincides with L, producing the desired subdivision hierarchy S_p. The sub-CSG trees originally associated with the leaf voxel of S_l are retained at the corresponding leaf voxels in S_p. Other leaf voxels of S_p have empty sub-CSG trees assigned to them. If the operator $\otimes$ is an intersection or a difference, the subdivision hierarchy S_p and its associated sub-CSG trees are identical to that of l, since S-bounds L and P have the same extent.
2. *Continue the subdivision of S_p using the subdivision hierarchy of S_r.* For each nonempty leaf voxel V_r in S_r we perform a downward search for leaf voxels of S_p that intersect with V_r, then subdivide each of these leaf voxels, say V_p, using face planes of V_r. For each newly created leaf voxel, if the voxel intersects with V_r, then we assign to it the sub-CSG tree that results from applying $\otimes$ to the sub-CSG trees associated with V_p and V_r, respectively. Otherwise, the voxel inherits the sub-CSG tree associated with V_p. The voxel V_p becomes an internal node of S_p and is not associated with a sub-CSG tree.

The following pseudocode describes the downward search and subdivision in Step 2:

```
Downward_Subdivision(operator ⊗,
    subdivision_hierarchy,
    right_voxel Vr) {
  if (subdivision_hierarchy is a leaf
      voxel) {
    Subdivide the leaf voxel by face
        planes of Vr and assign
        appropriate sub-CSG trees to
        newly created leaf voxels as
        described in Step 2;
    return;}
  for each child voxel D of the
    subdivision_hierarchy do
    if (voxel D intersects with
        right_voxel Vr)
      Downward_Subdivision(
        operator ⊗,
        subdivision_hierarchy of D;
        right_voxel Vr)
}
```

A voxel is subdivided using the face planes of another voxel in a manner similar to BSP; that is, only the

respective halfspace resulting from the previous subdivision is subdivided. As a result, the number of voxels can be reduced and the size of the vacant voxel enlarged. In the resulting subdivision hierarchy of p, each leaf voxel is empty, intersects with primitives of both l and r, or intersects with primitives of only l or only r. Consequently, the sub-CSG tree assigned to each leaf voxel is minimized.

Figure 3 illustrates the subdivision process.

Ray tracing

After space subdivision comes standard ray tracing. For those nonempty voxels the ray encounters, we first compute the intersections of the ray with primitives in the associated sub-CSG tree, discard the intersections outside of the primitives' S-bounds, then sort the intersections along the ray, and finally classify intersections against the sub-CSG tree until we find the nearest intersection. If the nearest intersection is not in the current voxel, the next voxel pierced by the ray is searched. Because the subdivision is nonuniform, locating the next voxel along the ray generally requires a vertical and a horizontal search on the subdivision hierarchy.

To speed up the search, for each face F of leaf voxel V we maintain a pointer pointing to the adjacent voxel on the lowest possible level of the hierarchy that has a face which totally encloses F. When locating the next voxel along the ray, we determine the face at which the ray exits the voxel and then retrieve the face pointer. If the neighboring voxel is a leaf, we have found the next voxel. If the neighbor is an interior node, the point where the ray exits the voxel is computed and used to perform a downward search on the neighbor's subdivision hierarchy.

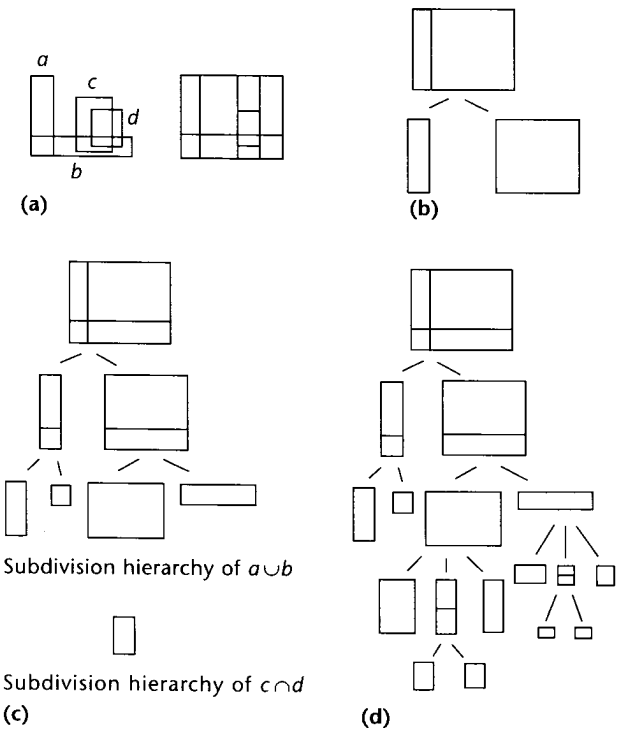
When a ray traverses from one voxel to a neighboring voxel of equal or lower subdivision level, the neighbor is generally a leaf voxel, and no further traversal is necessary. On the other hand, when the ray traverses from a voxel of lower subdivision level to the neighboring voxel, a downward search is required.

In addition to locating the next voxel along the ray, another problem for ray tracing with space subdivision is the fragmentation of the object. An object might overlap several leaf voxels such that it cannot be totally enclosed by any voxel. For such an object, the ray-object intersection computation is generally performed in all intersecting voxels. To eliminate unnecessary computations, the *mail box* concept has been proposed.[3] For each object, both the identity of the most recent ray that intersects the object and the corresponding intersections are recorded in the mail box. Subsequent intersection computations simply check to see if the object has been tested against the current ray and, if so, refer to the intersection data stored in the mail box.

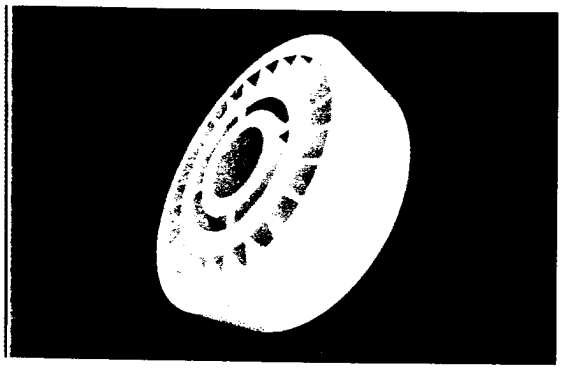
Subdivision level

Traditional space subdivisions typically require a termination criterion such as a prescribed maximum level of subdivision or a maximum number of primitives allowed inside a voxel. In the CSG model, voxels might overlap with many primitives, and, consequently, the number of primitives inside the voxels exceeds the allowable number when the maximum level of subdivision is reached. For such voxels, the associated sub-CSG tree might be complex and contain unnecessary information.

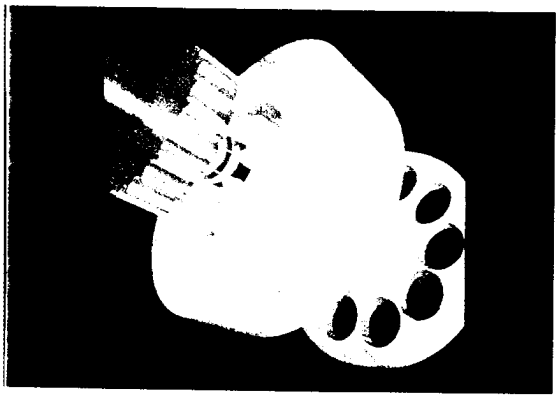
Traditional space subdivisions facilitate intersection computation and ray traversal, but not for point classification. In contrast, in our nonuniform space subdivision scheme the height of the CSG tree affects the subdivision level. Although more subdivisions might



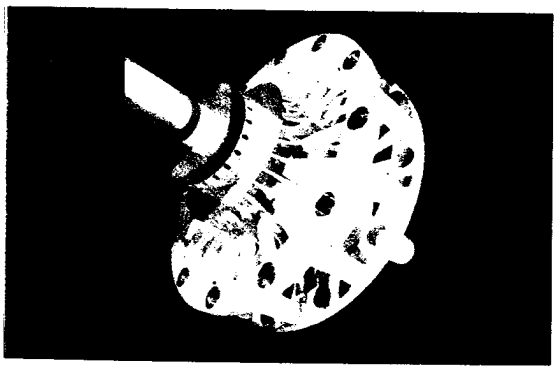
3 Computing the space subdivision hierarchy of $(a \cup b) - (c \cap d)$. The CSG solid and its subdivision are shown in (a). The subdivision hierarchy of $a \cup b$ is depicted in (b) after performing Step 1 of `Merge_Subdivision_Hierarchy()`. The left and right subdivision hierarchies of $(a \cup b) - (c \cap d)$ are shown in the top and bottom portions of (c), respectively. The top portion results from applying `Merge_Subdivision_Hierarchy()` on $a \cup b$. The resulting space subdivision hierarchy of $(a \cup b)-(c \cap d)$ appears in (d).



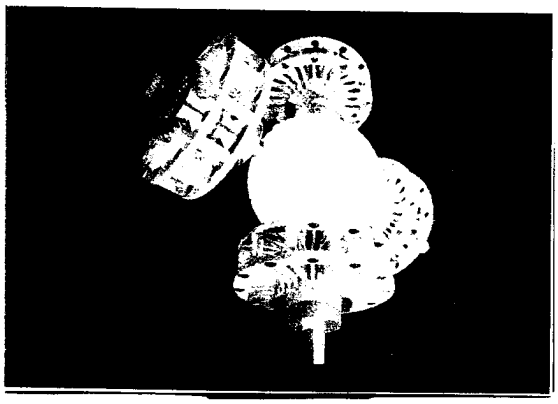
4 Ray-traced image of Model 2.



5 Ray-traced image of Model 3.



6 Ray-traced image of Model 4.



7 Ray-traced image of Model 5.

be required in the area where many Boolean operators are involved, our proposed subdivision generates smaller, more compact sub-CSG trees, significantly reducing the number of intersection computations and point classifications.

Nonetheless, the subdivision hierarchy can be large for a complex CSG solid. For such solids, we must consider the complexity of ray traversal and the storage space necessary for the hierarchy. Although we can speed up ray traversal by setting pointers to neighboring voxels, the storage space required for a hierarchy of high subdivision level could be a problem. To avoid such difficulties, we can terminate the space subdivision midway by setting a minimum size of leaf voxels or a maximum level of subdivision.

Alternatively, we can take advantage of the ratio of surface areas. When the value of `surface_area_ratio`, equal to `size_of_surface_of_subdivided_voxel` divided by `size_of_surface_of_subdividing_voxel`, exceeds a user-defined value, the voxel subdivision terminates; otherwise, the subdivision takes place. Our experimental results show that, when applied to test models, this heuristic criterion significantly reduced the subdivision hierarchy's size. Heuristics of this kind have also been applied in the construction of the hierarchy extent tree.[11]

Experiments and results

We implemented our proposed space subdivision in C++ on a Silicon Graphics Indy workstation with a MIPS 4600PC CPU and 32 Mbytes of RAM. We conducted experiments on five models, described in Table 1. Ray-traced images of four of these models appear in Figures 4, 5, 6, and 7. All ray-traced images are 512 × 512 in resolution.

For comparison purposes, we also implemented the BSP space subdivision proposed by Bouatouch[4] because it uses minimal bounding boxes of primitives for subdivision termination. We believe that ray tracing CSG solids with traditional subdivisions—uniform, octree, or BSP—speeds up significantly when the minimal bounding box or *S*-bounds of the primitive, rather than the primitive itself, determines subdivision termination.

Table 2 compares the performance of the BSP subdivision and the proposed nonuniform space subdivision algorithms. The subdivision time includes *S*-bounds computation, which is relatively short. The minimum bounding box computation by Bouatouch[4] has been replaced by *S*-bounds computation, which generally produces smaller bounding boxes. Moreover, for BSP subdivision, several pairs of the maximum subdivision level and the maximum number of primitives allowed inside each voxel were tested for each model; only the best results were used for comparison. Both subdivisions implement the auxiliary data structure *mail box* to avoid unnecessary ray-object intersection computations.

From Table 2 we see that the gain factor for the numbers of intersection computations and point classifications ranges from 1.70 to 4.54, while the gain factor for processing time ranges from 1.29 to 2.82. Because the BSP method does not subdivide the space according to the effective volume of CSG primitives, the sub-CSG trees can be highly complex where primitives are densely distributed. In these cases, we expect the gain factor to be much higher.

Note that the gain factors could be higher, if the minimum bounding box used in BSP-based subdivision is not replaced by *S*-bounds. Intersection computation time would be significantly reduced in cases involving

Table 1. Number of primitives featured in five test models.

Models	Model 1	Model 2	Model 3	Model 4	Model 5
Number of primitives	30	34	71	78	313

Table 2. Performance comparison of BSP subdivision and nonuniform space subdivision (time in seconds).

Model (l, d)[1]	Space Subdivision[2]	Number of Leaf Voxels	Intersection Number	Intersection G.F.[3]	Classification Number	Classification G.F.[3]	Processing Time Subdivision	Processing Time Synthesis	Processing Time G.F.[3]
Model 1	(3, 3) A	368	904,332	-	788,433	-	0.89	162.19	-
(1, 1)	B	163	503,385	1.80	292,436	2.70	0.05	97.85	1.67
Model 1	(2, 2) A	128	9,595,253	-	5,077,301	-	0.29	1,146.91	-
(1, 4)	B	163	4,963,478	1.93	2,037,985	2.49	0.07	627.81	1.83
Model 2	(3, 2) A	864	2,480,288	-	5,253,956	-	2.91	311.15	-
(2, 1)	B	826	1,457,153	1.70	2,837,061	1.85	0.90	242.45	1.29
Model 3	(3, 3) A	288	5,194,272	-	6,799,685	-	1.56	682.34	-
(2, 1)	B	1,787	1,813,277	2.86	1,498,425	4.54	1.79	255.29	2.66
Model 3	(3, 3) A	288	13,187,402	-	13,092,877	-	1.56	1,455.40	-
(2, 2)	B	1,787	4,741,579	2.78	3,325,635	3.94	1.79	533.00	2.72
Model 4	(3, 2) A	424	2,875,077	-	2,367,595	-	2.23	494.30	-
(2, 1)	B	1,680	1,177,205	2.44	888,869	2.66	1.84	197.18	2.49
Model 4	(3, 3) A	400	7,485,665	-	5,718,938	-	2.14	1,032.00	-
(2, 2)	B	1,680	3,282,658	2.28	2,227,521	2.57	1.84	410.28	2.51
Model 5	(4, 3) A	1,928	4,725,284	-	3,573,122	-	15.20	1,055.10	-
(1, 2)	B	12,659	2,016,291	2.34	1,313,283	2.72	28.60	350.47	2.82

[1] The (l, d) represents the number of light sources and the depth of ray tracing for the models tested.

[2] A is the BSP subdivision by Bouatouch[4]; B is our proposed subdivision. The numbers preceding A represent pairs of maximum subdivision level and maximum number of primitives.

[3] Gain factor.

complex primitives such as tori and sweeping volumes. Although we did not extensively test all models for them, the gain factors should remain roughly constant on applying the secondary ray of various depths. Our experiments on Model 2 show a relatively small gain factor because most of its primitives are not axis-aligned. This problem is typical for space subdivision based on axis-aligned bounding boxes.

Nonuniform space subdivision time is generally proportional to the number of primitives. For the models we tested, the time for subdivision is relatively small compared with that for synthesis. We expect that the increased subdivision time needed for large CSG trees can be greatly ameliorated by reduced synthesis time because sub-CSG trees become more localized as more subdivisions are performed.

We also examined the effect of `surface_area_ratio` on several models in terminating voxel subdivision midway. Table 3 (on the next page) illustrates our results. The heuristic effectively reduced the number of leaf voxels with a relatively small efficiency loss. Observe that the number of leaf voxels is significantly reduced when `surface_area_ratio` equals 0.95.

Conclusion

The nonuniform space subdivision scheme we have proposed for ray tracing of CSG solids greatly reduces the number of ray-object intersection computations and point classifications. We have performed several experiments and observed that the gain factor on processing time over the BSP-based space subdivision method[4] ranges from 1.29 to 2.82. For CSG solids with complex primitives, such as tori and sweeping volumes, the reductions achieved with this subdivision scheme lead to significantly improved efficiency.

The algorithm first performs S-bounds computation on a given CSG tree and then subdivides the S-bound at the root by upwardly propagating the subdivision hierarchies at both of its child nodes. This process is performed recursively for each internal node; thus, the subdivision proceeds in a bottom-up fashion, starting with the primitives' S-bounds.

Terminating the proposed space subdivision midway is possible by setting a minimum size of leaf voxels. We tested a heuristic termination criterion that utilized the ratio of surface areas and significantly reduced the subdivision hierarchy's size. Nonuniform space subdivision implemented without early termination is usually more computationally efficient than when terminated midway; however, subdivision with early termination is necessary when computer memory is limited.

Because our proposed subdivision largely depends on the CSG expression, and because the expression is not unique for a CSG solid, more research is necessary to understand how the choice of the CSG expression affects the performance of this algorithm. ■

Table 3. Effect of `surface_area_ratio` on subdivision termination (time in seconds).

Model (*l*, *d*)[1]	surface_area_ratio	Number of Leaf Voxels	Number of Intersections	Number of Classifications	Subdivision Time	Synthesis Time
Model 2	NA[2]	826	1,457,153	2,837,061	0.90	242.45
(2, 1)	0.95	131	2,538,463	3,924,578	0.11	243.02
	0.90	107	2,839,382	4,147,851	0.09	252.66
Model 3	NA[2]	1,787	1,813,277	1,498,425	1.79	255.29
(2, 1)	0.95	751	3,015,476	2,324,237	0.47	288.27
	0.90	732	3,396,218	2,595,657	0.44	294.65
Model 3	NA[2]	1,787	4,741,579	3,325,635	1.79	533.00
(2, 2)	0.95	751	7,940,186	5,448,535	0.47	593.23
	0.90	732	8,823,721	6,069,345	0.44	611.07
Model 4	NA[2]	1,680	1,177,205	888,869	1.84	197.18
(2, 1)	0.95	588	2,093,746	1,829,567	0.51	239.16
	0.90	583	2,106,021	1,835,696	0.51	238.90
Model 4	NA[2]	1,680	3,282,658	2,227,521	1.84	410.28
(2, 2)	0.95	588	5,848,183	4,592,871	0.51	506.84
	0.90	583	5,871,167	4,606,391	0.51	506.34
Model 5	NA[2]	12,659	2,016,291	1,313,283	28.60	350.47
(1, 2)	0.95	5,361	3,717,191	3,148,116	6.47	487.81
	0.90	5,298	3,749,548	3,180,649	6.39	494.33

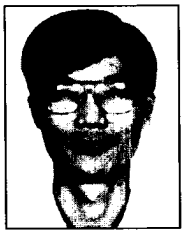
[1] The (*l*, *d*) represents the number of light sources and the depth of ray tracing for the models tested.
[2] The `surface_area_ratio` strategy was not applied.

Acknowledgment

This work was supported by the National Science Council of the Republic of China under grant NSC 82-0408-E-009-209. When this article was written, Weun-Jier Hwang was at National Chiao Tung University.

References

1. T.L. Kay and J.T. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics*, Vol. 20, No. 4, Aug. 1986, pp. 269-278.
2. A.S. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, Vol. 4, No. 10, Oct. 1984, pp. 15-22.
3. B. Arnaldi, T. Priol, and K. Bouatouch, "A New Space Subdivision Method for Ray Tracing CSG Modelled Scenes," *Visual Computer*, Vol. 3, 1987, pp. 98-108.
4. K. Bouatouch et al., "A New Algorithm of Space Tracing Using A CSG Model," *Proc. Eurographics*, Eurographics Assoc., Amsterdam, The Netherlands, 1987, pp. 65-78.
5. G. Wyvill, T.L. Kunii, and Y. Shirai, "Space Division for Ray Tracing in CSG," *IEEE CG&A*, Vol. 6, No. 4, April 1986, pp. 28-34.
6. F.W. Jansen, "Depth-Order Point Classification Techniques for CSG Display Algorithms," *ACM Trans. Graphics*, Vol. 10, No. 1, Jan. 1991, pp. 40-70.
7. S.D. Roth, "Ray Casting for Modeling Solids," *Computer Graphics and Image Processing*, Vol. 18, No. 2, 1982, pp. 109-144.
8. S. Cameron, "Efficient Bounds in Constructive Solid Geometry," *IEEE CG&A*, Vol. 21, No. 4, July 1991, pp. 68-74.
9. X. Pueyo and J.C. Mendoza, "A New Scan Line Algorithm for the Rendering of CSG Trees," *Proc. Eurographics*, Eurographics Assoc., Amsterdam, The Netherlands, 1987, pp. 347-361.
10. J.R. Rossignac and A.A.G. Requicha, "Depth-Buffering Display Techniques for Constructive Solid Geometry," *IEEE CG&A*, Vol. 6, No. 9, Sept. 1986, pp. 29-39.
11. J.D. MacDonald and S.K. Booth, "Heuristics for Ray Tracing Using Space Subdivision," *Visual Computer*, Vol. 6, 1990, pp. 153-166.



Jung-Hong Chuang *is an associate professor of computer science and information engineering at National Chiao Tung University, Taiwan, Republic of China. His research interests include geometric and solid modeling, computer graphics, and visualization. Chuang received a BS degree in applied mathematics from National Chiao Tung University in 1978 and MS and PhD degrees in computer science from Purdue University in 1987 and 1990, respectively.*

Weun-Jier Hwang *is an assistant research engineer at the Telecommunication Laboratories, Taiwan, Republic of China. His research interests include computer graphics and networks. Hwang received a BS in transportation and management in 1990 and an MS in computer science and information engineering in 1992, both from National Chiao Tung University.*

Readers may contact Chuang at Dept. of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan, Republic of China, e-mail jhchuang@csunix.csie.nctu.edu.tw.