PERFORMANCE ENHANCEMENTS OF MONTE CARLO PARTICLE TRACING ALGORITHMS FOR LARGE, ARBITRARY GEOMETRIES

2 authors, including:

P.J. Burns
Colorado State University
48 PUBLICATIONS 701 CITATIONS

SEE PROFILE

PERFORMANCE ENHANCEMENTS OF MONTE CARLO PARTICLE TRACING ALGORITHMS FOR LARGE, ARBITRARY GEOMETRIES

Charles N. Zeeb and Patrick J. Burns
Department of Mechanical Engineering
Colorado State University
Fort Collins, Colorado 80523
Phone: (970) 491-5778
FAX: (970) 491-1958
czeeb@lamar.colostate.edu and pburns@colostate.edu

ABSTRACT

Drawing on techniques used in the computer graphics field of ray tracing, an efficient algorithm for tracing particles in large, arbitrary geometries containing nonparticipating media is presented. An efficient intersection algorithm for arbitrary triangles and/or convex planar quadrilaterals is discussed in detail. Several techniques used in ray tracing to limit the number of surfaces tested are discussed and the method of uniform spatial division (USD) is implemented. The "mailbox" technique is also covered. To determine the efficiency of the intersection algorithm, and USD, timing results are presented for a number of different spatial divisions for four geometries containing between one thousand and five thousand surfaces each. For USD, speedups in tracing time exceeding a factor of eighty are observed.

1 INTRODUCTION

The modeling of radiative heat transfer is a particularly challenging subject. Since radiation involves "action at a distance," every surface and volume of media in a geometry can affect every other surface and volume. Geometries can be very complex. Furthermore, surface properties are often a function of angle and media properties often are a function of wavelength. Few methods are versatile enough to handle problems this complex. One proven method is the Monte Carlo method (Haji-Sheik, 1988; Modest, 1993).

In radiative Monte Carlo, results are obtained by tracing a statistically large number of bundles or "photons" of energy. Statistical relationships are used to model the emission, absorption, reflection, transmission and scattering of these bundles. Although the method is very versatile, it is also very computationally intensive, since most simulations involve tracing millions or even billions of photons. While Monte Carlo methods are becoming more feasible with today's more powerful computers, it is still very important that Monte Carlo tracing be as efficient as possible.

The main focus of this paper is the modeling of complex geometries using the Monte Carlo method. Most published radiative Monte Carlo work employ simple geometries such as slabs and cubes; there are few published routines to handle arbitrary, complex geometries. Chin et al. (1989; 1992) have covered several Monte Carlo issues, particularly applying Monte Carlo to finite element meshes. Farmer (1995) has also discussed using Monte Carlo to simulate arbitrary geometries constructed using finite element meshes. In addition, Henson et al. (1996) do discuss some techniques for improving the speed of Monte Carlo routines. Several generalized radiative Monte Carlo programs do exist, for example, TSS (Panczak, 1989) and MATRAD (Koeck, 1988). Still, except for the above references, we have found nothing published about these and other such codes' algorithms.

One generalized algorithm discussed in detail has been implemented in the code MONT3D (Maltby, 1987; Burns et al., 1990; Maltby and Burns, 1991; Zeeb et al., 1999), which simulates radiative transfer in geometries with nonparticipating media. The code has been used extensively for more than a decade by Lawrence Livermore National Laboratory (LLNL) and other sites. The output of the code is a radiative exchange factor matrix output by MONT3D which is used as input to thermal analysis codes, particularly TOPAZ3D (Shapiro, 1985). Geometries with 14,000 or more surfaces have been modeled and the current trend is to model even more complex geometries. The code has been independently validated theoretically and verified experimentally.

This paper discusses the latest improvements made in the photon tracing algorithm. This paper differs from most previous works in that techniques from the field of computer graphics are incorporated. As pointed out by Rushmeier (1993), while much of the early work in computer graphics simply borrowed from heat transfer, computer graphics has matured sufficiently that some of the techniques used in computer graphics can be used to improve heat transfer calculations - especially true for ray tracing. While Henson et al. (1996), and particularly Panczak (1989) have also addressed this topic, this paper covers

many aspects not touched in those papers. Since Monte Carlo calculations are computationally intensive, the emphasis here is to present an algorithm that allows arbitrarily complex geometries but is computationally efficient.

Timing results are shown for four geometries ranging from one thousand to five thousand surfaces which demonstrate the efficiency of the new algorithm, particularly the USD algorithm. This in-depth study allows recommendations to be made about optimally applying USD to large radiative geometries.

2 SURFACE GEOMETRIES

For the algorithms described below, it is assumed the geometry is defined in a global Cartesian coordinate system from node points that are input by the user or generated using a grid generation program such as TrueGrid (XYZ, 1997). Surfaces are defined by specifying four node points and an unique surface number. The types of surfaces modeled are either triangles or convex quadrilaterals as shown in Fig. 1. Surfaces must be planar. If the four nodes of a quadrilateral are not coplanar to within a small tolerance, the quadrilateral is divided into two planar triangles. Results for the two triangles are combined before output so the division is transparent to the user, providing a simple way to handle non-planar surfaces deriving from round-off error or mismatched nodes. Mismatched nodes can occur when a complex geometry is generated in several parts and then "patched" together. The orientation of the surfaces follows the right-hand rule, so the surface normal, which determines the direction of emission, always points outward as the surface is traversed in the direction of increasing node number. Surfaces are one-sided; the "back" side, which is opposite the surface normal, does not emit or interact with photons.
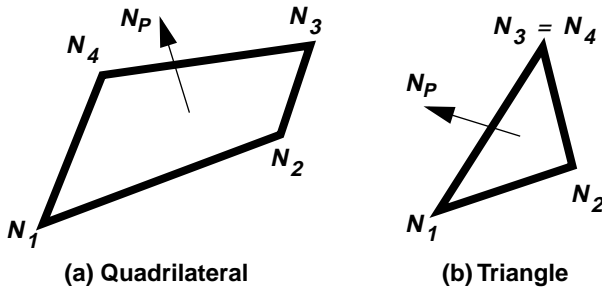

(a) Quadrilateral   **(b) Triangle**

Figure 1 Radiating Surface Geometries

3 CALCULATING THE INTERSECTION DISTANCE TO A PLANE

In all Monte Carlo simulations, most of the work is spent determining which surface a photon hits. This calculation is the starting point of our discussion due to its importance and also because it is the basis for all intersection routines. For our algorithm, there are two parts to the calculation: 1) finding the distance to the plane that contains the polygon, and 2) determining if the intersection point with the plane is

inside the polygon. The first part is covered in this section; the second part is covered in the next section. The previous version of the tracing algorithm used an intersection routine with many similarities to this one. The major difference is that while the current algorithm focuses on calculating the distance to intersection, t_i, the previous algorithm focuses on the point of intersection, R_i, which requires more calculation.

Intersections with planar surfaces are common in ray tracing, so the formula for the intersection with a plane is covered in many ray tracing tutorials. A particularly good discussion is given by Haines (1989). Defining the origin of the photon as $R_0 = (X_0, Y_0, Z_0)$ and its unit direction vector as $E = (E_X, E_Y, E_Z)$, the equation for the position of the photon, R, is given by:

$$R = R_0 + Et \qquad (1)$$

where t is the distance the photon has travelled. The equation for the plane which contains the polygon is given by:

$$Ax + By + Cz - D = 0 \qquad \text{where } A^2 + B^2 + C^2 = 1 \qquad (2)$$

The unit surface normal for the plane (and the polygon in the plane), N_p, is equal to (A, B, C) and D is the distance from the origin of the system, $(0,0,0)$, to the plane. Inserting Eq. (1) into Eq. (2) and solving for t_i, the distance to the intersection with the plane,

$$t_i = \frac{-(AX_0 + BY_0 + CZ_0 - D)}{AE_X + BE_Y + CE_Z} = \frac{D - (N_p \bullet R_0)}{N_p \bullet E} = \frac{v_0}{v_E} \qquad (3)$$

N_p and D are stored for each surface during preprocessing, and R_0 and E are calculated every time a photon is emitted, reflected, or transmitted.

The calculation above can be done most efficiently in steps.
1. Calculate v_E.
2. If v_E is greater than or equal to 0, the photon is incident from the back of the surface (see Fig. 2 (a)), the intersection point is rejected and no further calculation for this surface is needed.
3. Calculate v_0.
4. If v_0 is greater than or equal to 0, the photon must travel backwards from the photon origin to strike the surface (see Fig. 2 (b)). Therefore, the intersection is rejected and no further calculation for this surface is needed.
5. Calculate t_i using Eq. (3).

Step 2 is referred to as "backface culling." As mentioned in Section 2 and shown in Fig. 2, only the "front" of a surface emits or intersects photons and the "back" side is inside the surface itself. In a properly defined geometry, no photons will intersect the back side, so all interactions with that side are ignored. Steps 1 and 2 only require

= The Emitting/Receiving Side of the Surface

= Backwards Travel of the Photon

(a) Hit "Back" of Surface

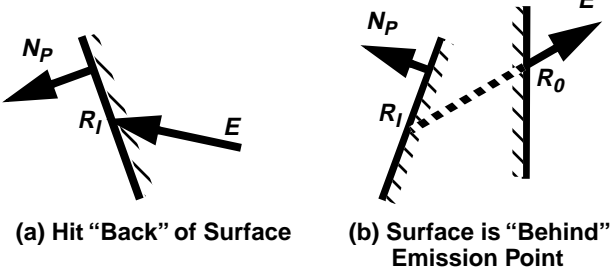**(b) Surface is "Behind" Emission Point**

Figure 2 Conditions to Invalidate a Potential Intersection

three multiplies, two additions, and one compare, and, on average, cause about half the surfaces tested to be rejected.

If transmittance is modeled, an extra step must be added to the algorithm. When a photon is emitted (or reflected) from a surface, that particular surface is never selected as the next intersecting surface because it will always fail the backface culling test. On the other hand, if the photon is transmitted, then that surface will always pass that test. Furthermore, due to round-off error, the transmitting surface may also pass through step 4 and appear to have a very small, positive t_i value. Therefore, if transmission is modeled, an extra check is required to make sure that the previous intersecting surface itself is not chosen as a valid intersection point. Since transmission is rarely modeled, this test is usually done last.

4 POINT-IN-POLYGON TEST

Once a valid distance, t_i, is found from step 5, a check must be made to ensure that the point is within the polygon. A rather complete study of point-in-polygon strategies has been done by Haines (1994). Since our polygons are convex and have only three or four sides, the exterior-edges algorithm has been chosen for the point-in-polygon test. Within the uncertainty in Haines' test results, this test is as fast as any other, and was chosen over the others he discussed because it requires less storage and is easier to implement.

In the exterior-edges test, the half-plane test is performed on all edges of the polygon. The half-plane test requires a "bounding" plane for each edge that is perpendicular to the polygon's surface and includes that edge. From Eq. (2), the "bounding" plane is defined by the normal $N_{PH} = (A_H, B_H, C_H)$ which points into the surface, and the distance, D_H. For the intersection point, R_i, if (Jeger and Eckmann, 1967):

$$A_H X_i + B_H Y_i + C_H Z_i > D_H \qquad (4)$$

then the point is outside that edge of the polygon and is rejected. The test is performed for each edge of the polygon. As soon as the point

fails any test, it is rejected. The constants A_H, B_H, C_H, and D_H are calculated for each surface in the preprocessing (input) stage.

Haines (1989, 1994) suggests to project the polygon and the test point into two dimensions. This saves a floating point add and multiply for each half-plane test. The simplest way to project the problem into two dimensions is to discard one of the X, Y, or Z coordinates. The area of the polygon is not preserved but the topology is. The best coordinate to throw away is the one whose magnitude in the polygon's surface normal, N_p, is the greatest. N_{PH} and D_H must be calculated in this two-dimensional plane using the new two-dimensional coordinates. While the current version of the intersection algorithm implements this two-dimensional test, the previous version uses the three-dimensional form.

5 LIMITING THE SEARCH

Every time a photon is emitted, reflected, or transmitted, the next surface it strikes must be found. Since the time to trace each photon increases linearly with the number of surfaces, it is obvious that great improvements in efficiency can be gained by reducing the number of surfaces that need to be tested. Reducing the number of surface interaction calculations has been a topic of extensive study in the field of ray tracing. A good overview of the general techniques applied to this problem are given by Arvo and Kirk (1989). Of all the techniques reviewed, two general techniques are the most promising: bounding volumes and spatial subdivision. A discussion with a different perspective on applying ray tracing techniques to radiative Monte Carlo can be found in Panczak (1989).

The bounding volume technique reduces the number of intersection calculations by surrounding all the objects in the scene with bounding volumes. The bounding volumes are chosen to be simple objects such as spheres and cubes so that intersection calculations with them are swift. The object or objects inside the bounding volume need only be checked if the bounding volume is intersected. According to Arvo and Kirk (1989), when a hierarchy of bounding volumes is used, the complexity of the intersection calculation is proportional to the logarithm of the number of objects. If the bounding volumes are not used in a hierarchy, the time for the intersection calculations is reduced but is still linear with the number of objects. Since the polygon intersection calculations described above are so simple, bounding volumes should contain collections of polygons. Devising a good way to define bounding volumes for groups of arbitrary surfaces would be difficult and probably not add much to the efficiency of the program.

The other promising technique is three-dimensional spatial subdivision shown in Fig. 3. For this technique, instead of placing bounding volumes around objects, the volume bounding the geometry is partitioned. Space is usually divided into axis-aligned rectangular prisms which are referred to as voxels (a three-dimensional version of a pixel). By aligning the voxel planes with the axes, computations are simplified. The photon is then traced from voxel to voxel. A check is made only inside each voxel to determine if any of the surfaces inside it are intersected. The search stops once the closest intersection within the current voxel is found. This reduces the number of intersection calculations in two ways. First, only surfaces in voxels along the photon path are checked. Second, since voxels are traversed in order, surfaces in voxels further from the origin of the photon are checked only if an intersection is not found in an earlier voxel. For large geometries, the reduction in search time can be very significant.
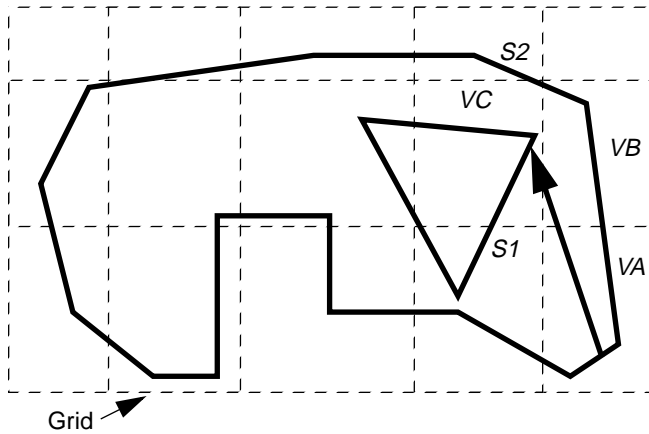
Figure 3 Example of a Non-uniform Grid

It should be noted that if an intersection is found outside the current voxel, it must be rejected, as it is possible that the photon might strike another surface which, while not in the current voxel, contains an intersection point between the current voxel and the rejected intersection point. For example, in Fig. 3, the photon is traced from its point of emission in voxel *VA* through voxel *VB* to its intersection point in voxel *VC*. In voxel *VB*, the potential intersection point on surface *S2* will be found, and must be rejected because the intersection point is outside the voxel. The true intersection point on surface *S1* will be found only after the photon enters voxel *VC*.

Arvo and Kirk (1989) classify spatial subdivision into two general categories: non-uniform and uniform. In non-uniform subdivision, space is discretized into regions of various sizes to allow the voxel density to be greater where the surface density is greater. Two of the most popular examples of this technique are the octree (Glassner, 1984) and the binary space partition (BSP) tree (Kaplan, 1987; Sung and Shirley, 1992). Octrees are created by recursively dividing a rectangular volume around the geometry into eight subordinate octants until the resulting "leaf" voxels meet some criterion for stopping such as a certain maximum number of surfaces per voxel. A BSP tree, on the other hand, is formed by partitioning space at each level of the tree into two pieces using a separating plane. While the planes are often aligned with the coordinate axes, they do not have to be (Chin, 1995).

While both the octree and the BSP tree store the geometry information efficiently, moving from voxel to voxel within the tree requires some calculation. Uniform spatial division (USD) (Fujimoto et al., 1986; Amanatides and Woo, 1987; Cleary and Wyvill, 1988) constitutes another approach. In USD, a regular three-dimensional grid of voxels of uniform size is placed over the geometry. While not dividing the geometry as efficiently as the above two methods, the "next" voxel can be found by fast incremental calculation. More voxels are usually traversed in USD but the cost of traversing the voxels is less.

Other methods do exist, for example, see Sung (1991). His method and most others are just combinations and variations of the octree, BSP tree, and USD methods mentioned above.

Using too many voxels can create inefficiencies. To search the surfaces inside each voxel quickly, a list must be made for each voxel of surfaces completely or partially inside it. As the number of voxels increases, the memory required for these lists and the time required to generate these lists increases. Also, it must be remembered that surface intersections outside the voxel must be rejected. If voxels are too small, surfaces will span several voxels and several intersection calculations for a surface will have to be rejected until the voxel with the intersection is entered. One way to avoid these repetitive calculations is to keep track of past intersection calculations. This is termed the "mailbox" technique and will be discussed below. It should also be noted that if the grid is too fine, time spent traversing the voxels will be significant. This can be a particular problem when using USD because the number of empty voxels generally increases with the number of voxels.

Although some comparisons of the two methods have been done, it is not clear which of the methods is the most efficient; different geometries have been used for each test. Since geometries can vary widely, each method does better on some geometries than others. Sung and Shirley (1992) have found that axis-aligned octrees and BSP trees give similar performance. Fujimoto et al. (1986) found that, for the sample problems he did, USD was an order of magnitude faster than octrees. Both the octree (Panczak, 1989; Chin et al., 1992) and USD (Koeck, 1988) methods have been implemented in radiative heat transfer Monte Carlo codes, but no comparison of the benefits of the two methods has been made.

The general consensus about USD is that it works well but requires a lot of memory. In fact, this is one of the major complaints against the method (Sung and Shirley, 1992). Sung (1991) compared USD to various octree methods. In the one case where there was enough memory for the number of voxels USD required, it outperformed all other methods.

In this work, USD is chosen for two reasons. First, a version of the USD called the Margolies algorithm (Maltby, 1987; Burns et al., 1990; Maltby and Burns, 1991) has already been implemented in MONT3D. This algorithm has been used for years, has worked well, and has proven very robust. Still, there is room for improvement. As pointed out above, photons under USD usually traverse more voxels than in the octree and BSP methods. Therefore, for the USD method to be more efficient than the octree and BSP tree method, the algorithm that determines the next voxel to enter must be very efficient. The next section describes an improved algorithm that is quite different from the one used earlier. Again, the major difference is that the previous algorithm stressed calculating the intersection point with the voxel boundaries, while the new algorithm only calculates the intersection distance to the voxel boundaries.

Secondly, as long as enough voxels are used to keep the average and maximum number of surfaces per voxel low, USD appears to be the best algorithm. As shown in the results below, with more memory capacity available today, USD is definitely feasible. While Sung (1991) has shown that USD can be inefficient for geometries with extremely uneven distributions of surfaces, it is not expected that these types of geometries are common in radiation problems. For that to occur, the size of the surfaces would have to vary by several orders of magnitude.

The Margolies algorithm is actually a variant of USD because it does not require uniform voxels; it also allows the use of a non-uniform grid in which the grid is still divided into rows and columns, but the spacing between *X*, *Y*, and *Z* divisions is variable. An example is shown

in Fig. 3. More about the advantages and disadvantages of non-uniform grids will be discussed below.

6 THE VOXEL TRACING ALGORITHM

This section outlines the algorithm for tracing using uniform or non-uniform grids. A less detailed description of the use of USD in radiative Monte Carlo is given by Koeck (1988). As shown in Fig. 3, whether the grid is uniform or non-uniform, the grid has $NGCMAX_i$ voxels along each axis and $NGCMAX_i + 1$ grid planes along each axis where i equals $\{1, 2, 3\}$ for the $\{X, Y, Z\}$ axes, respectively.

The key to the speed of the algorithm is that since the voxel boundaries are aligned with the axes, it is very easy to determine the next voxel to be entered. Although there are six sides to a voxel, only three sides, indicated by photon direction, have to be checked. Also, since all the voxel sides intersect, the side the shortest distance from the photon's origin is guaranteed to be the side that is intersected. Furthermore, due to the fact that the voxel sides are aligned with the axes, DG_i, the distance to intersect a voxel side along each axis can be derived from the "distance to a plane" equation in Section 3, and has the following form:

$$DG_i = (GC_{i,j} - R_{0,i})/E_i \qquad (5)$$

where j is the index of the next voxel plane the photon will cross along the ith axis, and GC is the coordinate along the ith axis for that plane. If the grid is uniform, then DGB_i, the distance between grid boundaries along an axis, is constant. Therefore, for uniform grids, while the first value of DG_i must be calculated using Eq. (5), for subsequent voxels DG_i can be updated by using the equation (DGB_i may be negative):

$$DG_i = DG_i + DGB_i \qquad (6)$$

The basic algorithm is simple.
1. Determine the emitting voxel cell and first $GC_{i,j}$ values. For non-uniform grid divisions, these are found by bisection and for uniform grid divisions by direct interpolation.
2. Calculate DG_i for each axis at the photon emission point.
3. Determine the minimum **DG** value.
4. Search for the shortest distance to intersection within the voxel. All surfaces even partially inside the voxel must be checked. To increase the efficiency of this search, the search is done over a precomputed list which specifies the surfaces in each voxel.
5. If no intersection is found, determine the next voxel to enter by the minimum **DG** value and the direction of **E**, update the value of DG_i along the axis traversed, and go back to step 3.

When testing for intersection points inside a voxel, once a valid intersection point is found, any intersection points further from the emission point than the current one are automatically rejected. If *TOLD*, the current shortest distance to intersection, is stored, the point-in-polygon test may be skipped by any surface with a t_i value greater than *TOLD*. When entering a voxel, *TOLD* should be initialized to the minimum value of **DG** required to exit the voxel (at step 3 above). This will automatically reject any intersection points outside the voxel.

Since there are often many empty voxels, it is important that both the intersection calculations and the voxel traversal algorithm be coded as efficiently as possible. Several authors suggest coding the grid traversal so that it only uses integer arithmetic (Fujimoto et al., 1986; Amanatides and Woo, 1987; Cleary and Wyvill, 1988). However, not only will this give a minimal speed improvement, if any, it also increases the possibility of precision errors. It should be noted that even for moderately sized geometries, photon tracing must be coded in double precision or round-off errors become significant.

As noted above, the use of non-uniform spaced voxels makes the DG_i calculation less efficient. This may make one wonder why non-uniformly spaced grids are desirable. The reason is that if enough is known about the geometry, voxels can be enlarged where there are few surfaces and shrunk where there are many, improving the efficiency of the algorithm. However, the usefulness of this type of grid resizing is limited. Enlarging or shrinking one voxel will affect all voxels along one or more coordinate directions. Furthermore, there is no easy way to choose a non-uniform grid *a priori*. In general, the user would probably get more benefit from an optimized uniform grid code than one that allows non-uniform divisions.

7 MAILBOXES

One of the problems with spatial subdivision is that no record is kept of past calculations. If a surface exists in several voxels, the same intersection is calculated repeatedly. To prevent this, "mailboxes" (Amanatides and Woo, 1987; Cleary and Wyvill, 1988; Arvo and Kirk, 1989; Sung, 1991; Sung and Shirley, 1992) are used to store past calculations. Each surface has a "mailbox" that holds the results of the calculations and a photon counter value that indicates the last time those calculations were done. The photon counter is incremented by one each time a photon is emitted, reflected, or transmitted. When the intersection calculation is performed for a surface, the first step is to compare the value of the counter in the mailbox to the current counter. If the two are equal, then the intersection in the mailbox is used, instead of being recalculated.

It should be noted that Sung (1991) found several cases in which mailboxes increased execution time. He notes that mailboxes are efficient only if most objects span more than one voxel. In particular, he suggests that, for the mailbox algorithm to be efficient, the objects in the scene must be larger than the voxels.

The mailbox algorithm is implemented here using two one-dimensional arrays of length number of surfaces, one for the distance to the intersection and the other for the last value of the photon counter. To implement the mailbox algorithm, only minor changes are required in the loop over surfaces in a voxel. The first change is that, for each surface, the first step is to compare the value of the photon counter stored for the surface to the current value. If the counter values do not match, then the algorithm is the same as before, except that the two one-dimensional arrays are updated for each intersection calculation. If the values do match, then the distance to surface has already been calculated. In this case, the distance is compared to *TOLD*. If it is less than *TOLD*, the point-in-polygon test is performed. Otherwise the surface is rejected. It should be noted that if a surface fails any of the tests except the comparison to *TOLD*, then the distance to intersection is set to a

very large number, insuring it fail the comparison to *TOLD* above the next time it is encountered.

The mailbox algorithm was tested on a number of large geometries (1,000 to 5,000 surfaces) and grid resolutions. It was found that the mailbox technique increased the run time in all cases. For the types of simplified surfaces and complex geometries usually modeled by our algorithm, the mailbox technique is not effective.

8 GRID TRACING RESULTS

To assess the improvements mentioned above, four geometries of varying complexity are tested. Several different discretizations are tested, in hopes of determining some guidelines in selecting the optimal grid for a geometry. Besides showing the overall efficiency of the current algorithm compared to the previous one, these large geometries demonstrate the power of USD. Earlier tests of USD with small geometries (143 surfaces or less) found that USD decreased run time by a factor of 2 to 5 (Burns et al., 1990; Maltby and Burns, 1991). As will be shown below, the reductions in run time are much more significant for larger geometries.

8.1 Geometries

Four geometries were used: cham, ampa, ampb, and gun. They are described below.

Cham is a model of the National Ignition Facility (NIF) target chamber containing 1,382 surfaces. It is a medium-sized geometry, shown in Fig. 4. The front section of the sphere is removed to show the inner detail. NIF (http://lasers.llnl.gov/lasers/nif.html) is currently under construction and will contain 192 extremely powerful lasers allowing research in inertial confinement fusion and other related topics. A discussion of the modeling of this chamber by TOPAZ3D is given in Raboin (1998).

Ampa and ampb are geometries representing the NIF laser amplifier assembly as shown in Fig. 5. The front wall and symmetry plane across the top of the geometry have been removed to show the inner
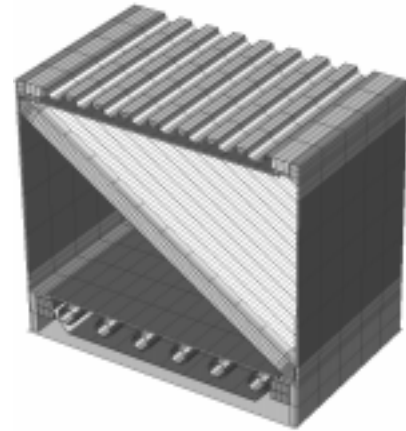

Figure 5 View of the Amplifier Geometry

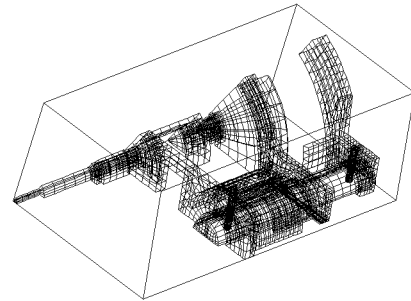**Figure 4 Cut-Away View of the Chamber Geometry**

Figure 6 View of the Gun Geometry

detail of assembly. Ampb, with 4,581 surfaces, is a more detailed representation than ampa, with 3,381 surfaces. More details about the modeling of this geometry are given in Sutton et al. (1998).

Gun, shown in Fig. 6, models the radiation coupling between the outer parts of an electron gun. The gun is enclosed to capture escaping photons. The gun is symmetric and is modeled as a wedge from 0° to 60°. The edges of the wedge are modeled as specular symmetry planes. This geometry has 4, 580 surfaces; one less than ampb.

8.2 Test Description

In all tests below, surfaces are black, as using reflecting or transmitting surfaces would just obscure observations about the photon intersection algorithm. Unless otherwise specified, all times given are solution times.

For testing, the geometries are "gridded" using approximately cubical voxels. A different number of photons are emitted for each geometry depending on the number of original surfaces and the number of photons per original surface as shown in Table 1. The cham and gun geometries have non-planar surfaces that are split into two triangles as

Table 1: Photon Statistics

Geometry	Number of Original Surfaces	Number of Split Surfaces	Total Number of Surfaces	Photons Emitted per Original Surface	Total Number of Photons Emitted (Millions)
cham	1,182	144	1,326	20,000	23.64
ampa	3,381	0	3,381	10,000	33.81
ampb	4,581	0	4,581	10,000	45.81
gun	4,297	283	4,580	10,000	42.95

described in Section 2. Each triangle is treated as a separate surface by the intersection routine, so the efficiency of the intersection routine depends on the total number of surfaces.

All tests were performed on a 233 MHz 604e PowerPC chip running Macintosh OS 8.0. The Absoft f77 compiler was used. Timings among repeated runs differed by 5% at most, and typically by less than 1%. To obtain accurate timing for the photon tracing only, a modified version of the code, which does not write output files, is used for all results.

8.3 Determining the Optimal Grid

While we have used USD in the past, there has been no in-depth study of its application to large-scale geometries. The purpose of this section is determine the efficiency of USD for large geometries and determine guidelines in selecting grids.
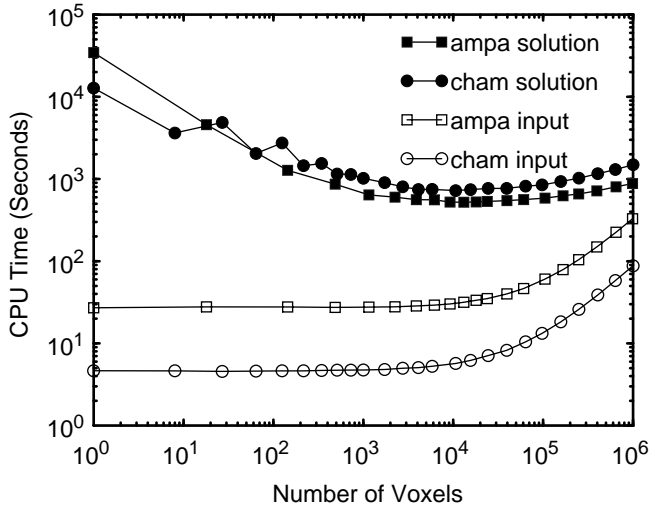
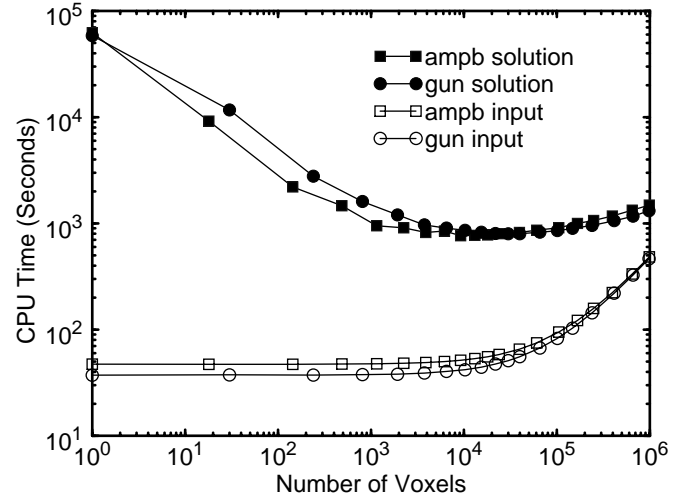**Figure 8 Results for the Ampb and Gun Geometries**

Figure 7 Results the for Ampa and Cham Geometries

Execution times versus numbers of voxels for the current algorithm for the four geometries are given in Figs. 7 and 8. The curves marked "solution" are the solution times for the algorithm and those marked "input" are the input times for the algorithm. The curves are very flat around the optimal grid. In fact, it is hard to specify one grid as "optimal." As stated above, the uncertainty in the results is about 5%. For each geometry, there are several points that are within 5% of the minimum value, yielding a range. Several statistics for the optimal grid range are given in Table 2. The results "min" and "max" are for the minimum and maximum size grid in the optimal range. The optimal grid range is quite large; the maximum grid size is around three to four times the minimum grid size. While the optimal grid varies with geometry, these results suggest that for geometries of 1,000 to 5,000 surfaces, a good first estimate of the optimal grid is 15,000 voxels.

Since determining the next voxel to enter requires less calculation than the surface intersection calculations, the optimal grid favors fewer surfaces per voxel over fewer empty voxels. For geometries tested, 37% to 60% of the voxels are empty when execution time is within 5% of the optimal time.

7

Table 2: Grid Statistics

Geometry	Number of Voxels	Surfaces per Voxel	Number of Empty Voxels	Surfaces per Non-empty Voxel
cham min	4,096	2.40	1,521	3.82
cham max	15,625	1.52	7,192	2.82
ampa min	9,216	2.44	3,942	4.26
ampa max	39,546	1.10	23,578	2.73
ampb min	9,216	2.66	4,114	4.80
ampb max	23,958	1.53	13,476	3.50
gun min	15,360	1.52	5,880	2.46
gun max	65,910	0.887	30,084	1.63

Table 3: Current Algorithm Statistics

Geometry	CPU Time Per Photon (X10^{-6} Seconds)			Speedup Ratios (No Grid/Optimal)		α_o
	"Optimal Grid" Solution Time	"No Grid" Solution Time	Time Not Tracing	Solution Time	Tracing Time	
cham	30.6	541	5.51	17.7	21.4	0.574
ampa	15.4	1,020	5.68	66.1	104	0.429
ampb	16.7	1,360	5.72	81.4	123	0.429
gun	18.6	1,370	5.72	73.3	106	0.447

The input times for the code are shown in Figs. 7 and 8. The input time grows quickly as the number of voxels becomes large. For the geometries tested this does not present a problem, as in the optimal grid range, the input time is insignificant relative to the solution time. Furthermore, production runs are much longer than the ones for these tests. Usually, more photons are emitted per surface, particularly as the number of surfaces increases. Also, tracing times are longer since reflection exists. As the number of surfaces modeled is increased further, this may be a problem, but only future testing can determine this.

As noted in Table 1, each geometry emits a different number of photons, so comparing the results can be difficult. For this reason, Table 3 provides results in time per photon. The "optimal grid" solution time is the shortest execution time for that geometry. The "no grid" solution time is the solution time when no grid is used. The "no grid" case is the same as specifying only one grid cell or voxel, and includes the insignificant overhead of initializing the voxel tracing routine once per photon. The other columns in the table are described below.

The "time not tracing" is the time spent by the solution phase in activities other than tracing, including specifying the photon emission point and direction, determining if the photon is absorbed, reflected, or transmitted and the overhead of the algorithm. If the surfaces are not all black, this would also include determining the photon's new direction after being transmitted or reflected.

The "time not tracing" was obtained by running the codes with the intersection routines commented out. While it is difficult to assess how accurately this measures the time not tracing, similar estimates were obtained by profiling the code on a RS/6000 workstation.

The speedup ratios listed in the table are the solution or time spent tracing for the "no grid" case divided by the same result for the optimal grid case. Comparing the "time not tracing" to the optimal solution time, it can be seen that only 63% to 82% of the optimal solution time is spent in photon tracing. This represents quite an improvement over the "no grid" solution where 99% or more of the time is spent in photon tracing. The "time not tracing" appears to be a weak function of the number of surfaces.

Table 3 indicates that the speedup can exceed a factor of 80 for the entire solution phase and over a factor of 120 for the intersection calculations. However, it would be helpful to quantify the speedup as a function of the number of surfaces. Cleary and Wyvill (1988) have done an in-depth theoretical analysis of a USD algorithm using a mailbox scheme. They have found that the run time is a complex function that depends on the number of surfaces, the average times for four different

Table 4: Previous Algorithm Statistics

Geometry	CPU Time Per Photon (X10^{-6} Seconds)		Optimal Algorithm Ratios (Old/New Algorithm)	
	"Optimal Grid" Solution Time	Time Not Tracing	Solution Time	Tracing Time
cham	53.0	5.31	1.73	1.90
ampa	22.9	5.38	1.48	1.79
ampb	25.7	5.43	1.54	1.84
gun	29.5	5.43	1.59	1.87

parts of the algorithm, the mean area of the objects, and the mean circumference of the objects.

A simplified model that is often used in these cases is the time per photon, t_p, given by:

$$t_p = a N_s^{\alpha} + b \qquad (7)$$

where N_s is the number of surfaces. For the "no grid" case, α is 1 since all N_s surfaces have to be checked. If we assume a is constant for a geometry and that b is equivalent to the "time not tracing" per photon, then α_o, the value of α for the optimal grid can be determined. As the table shows, α_o lies between 0.43 and 0.57. This is consistent with estimates of the dependence of optimal grid tracing time on N_s derived from an analytical perspective [Burns and Pryor, 1999].

8.4 Memory Requirements

Although photon tracing is often performed on large geometries with thousands of surfaces, its memory requirements are usually not prohibitive. Ignoring the memory required for the Margolies grid algorithm which will be discussed below, the storage required is on the order of the number of surfaces. While it is true that the final output, the exchange number matrix, is number of surfaces squared in size, only a small block of rows of that matrix are stored in memory at any one time. For the geometries used in this study of 14,080 nodes, 4,581 surfaces and 6 surface materials types, excluding the storage for the exchange matrix and the uniform grid, only 1.7 megabytes of storage is required. Storing 200 rows of the exchange matrix in memory at a time requires about another 3.5 megabytes. If the block size is decreased, the code must write its results to disk more frequently, thereby increasing I/O time. For the Margolies grid algorithm, the extra storage required is around 0.1 megabytes for 10,000 voxels and around 0.7 megabytes for 100,000 voxels.

8.5 Previous Algorithm Results

To gauge the effectiveness of the algorithm discussed in this paper, comparisons are made to the previous version of the algorithm (Maltby, 1987; Burns et al., 1990; Maltby and Burns, 1991) which is publicly available (Maltby et al., 1994; ftp://lamar.colostate.edu/pub/czeeb/

monte). Several runs are performed and some of the results for the older version of the algorithm are given in Table 4. The speedup ratios listed in Table 4 are the solution time or time spent tracing for the old algorithm optimal grid case divided by that for the new algorithm optimal grid case. Gathering data from all the runs done, the new algorithm is 33% to 45% faster than the old algorithm. Interestingly, the time not tracing is slightly better for the old algorithm, as new features added to the code slow it down slightly. The improvements to the voxel tracing algorithm seem to be just as important as the improvements to intersection calculations. The timing curves for the old algorithm are similar in shape to the curve for the new algorithm results and exhibit very similar optimal grid ranges. As mentioned in Section 3, the main difference between the algorithms is that for both surfaces and voxels the current algorithm focuses calculating the distance to intersection, t_i, instead of the point of intersection, $\boldsymbol{R_i}$.

9 CONCLUSIONS

An in-depth study has been completed of a Monte Carlo photon tracing algorithm for large geometries with arbitrary planar surfaces in nonparticipating media. Methods from the computer graphics field of ray tracing have been reviewed and implemented. An efficient algorithm for determining intersections has been presented. Furthermore, an assessment of ways to further increase the efficiency of the algorithm has been conducted. Uniform spatial division (USD) was chosen as the most promising technique, based upon its simplicity and effectiveness. The mailbox technique was found in all cases to increase execution times, and is therefore not recommended. Tests were performed on four geometries containing between 1,000 and 5,000 surfaces each. For these geometries, the following results were obtained. USD yielded speedups in run time of factors as great as 81. While the optimal subdivision varies with geometry, execution time varies slowly with number of voxels (grid cells). Good results are obtained with 15,000 voxels. The memory requirements for USD were found to be slight; less than a megabyte of memory was required to store the grid variables for the optimal grids for all geometries tested. The memory requirements for the rest of the algorithm were also found to be slight, between 1.7 and 5.5 megabytes was required for the largest geometry tested. The current photon tracing algorithm discussed in this paper is found to be 33% to 45% faster than the previous algorithm.

REFERENCES

Amanatides, J., and Woo, A., 1987, "A Fast Voxel Traversal Algorithm for Ray Tracing," In *Proceedings of EUROGRAPHICS '87*, G. Marechal, ed., Elsevier Science Publishers B. V., North-Holland, pp. 3-9.

Arvo, J., and Kirk, D., 1989, "A Survey of Ray Tracing Acceleration Techniques," In *An Introduction to Ray Tracing*, A. Glassner, ed., Academic Press, San Diego, CA, pp. 201-262.

Burns, P. J., Maltby, J. D., and Christon, M. A., 1990, "Large-Scale Surface to Surface Transport for Photons and Electrons via Monte Carlo," *Computing Systems in Engineering*, Vol. 1 No. 1, pp. 75-99.

Burns, P. J., and Pryor, D. V., 1999, "Surface Radiative Transport at Large Scale via Monte Carlo," Vol. 9 of *Annual Review of Heat Transfer*, Begell House, New York, NY. (in press).

Chin, J. H., Panczak, T., and Fried, L., 1989, "Finite Element and Raytracing in Coupled Thermal Problems," In *Numerical Methods in Thermal Problems, Vol. VI, Proceedings of the Sixth International Conference*, R. W. Lewis, and K. Morgan, ed., Pineridge Press, Swansea, U.K., pp. 683-701.

Chin, J. H., Panczak, T., and Fried, L., 1992, "Spacecraft Thermal Modelling," *International Journal for Numerical Methods in Engineering*, Vol. 35, pp. 641-653.

Chin, N., 1995, "A Walk through BSP Trees," In *Graphic Gems V*, A. W. Paeth, ed., AP Professional, San Diego, CA, pp. 121-138.

Cleary, J. G., and Wyvill, G., 1988, "Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision," *The Visual Computer*, Vol. 4, Springer-Verlag, pp. 65-83.

Farmer, J. T., 1995, *Improved Algorithms for Monte Carlo Analysis of Radiative Heat Transfer in Complex Participating Media*, Ph.D. Dissertation, University of Texas at Austin, Austin, TX.

Fujimoto, A., Tanaka, T., and Iwata, K. 1986, "ARTS: Accelerated Ray Tracing System," *IEEE Computer Graphics and. Applications,* Vol. 6 No. 4, pp. 16-26.

Glassner, A. S., 1984, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and. Applications,* Vol. 4 No. 10, pp. 15-22.

Haines, E., 1989, "Essential Ray Tracing Algorithms," In *An Introduction to Ray Tracing*, A. Glassner, ed., Academic Press, San Diego, CA, pp. 33-77.

Haines, E., 1994, "Point in Polygon Strategies," In *Graphic Gems IV*, P. S. Heckbert, ed., AP Professional, San Diego, CA, pp. 24-46.

Haji-Sheikh, A., 1988, "Monte Carlo Methods," Chapter 16 in *Handbook of Numerical Methods in Heat Transfer*, Minkowycz et al., ed., John Wiley & Sons, New York, pp. 673-722.

Henson, J. C., Malalasekera, M. G., and Dent, J. C., 1996, "Heat Transfer in Three-Dimensional, Nonhomogeneous Participating Media," *ASME Proceedings of the 31st National Heat transfer Confer-*

ence - Volume 3 - Solution Methods for Radiative Heat Transfer in Participating Media, HTD-Vol. 276, ASME, New York, pp. 25-34.

Jeger, M., and Eckmann, B., 1967, *Vector Geometry and Linear Algebra (For Engineers and Scientists)*, Interscience Publishers, New York.

Kaplan, M. R., 1987, "The Use of Spatial Coherence in Ray Tracing," In *Techniques for Computer Graphics*, D. F. Rogers, and R. A. Earnshaw, ed., Springer-Verlag, New York, NY, pp. 174-193.

Koeck, C., 1988, "Improved Ray Tracing Technique for Radiation Heat Transfer Modelling," *Proceedings of the 3rd European Symposium on Space Thermal Control & Life Support Systems*, Noordwijk, The Netherlands, (3-6 Oct. 1988) ESA SP-288, pp. 255-260.

Maltby, J. D., 1987, *Three-Dimensional Simulation of Radiative Heat Transfer by the Monte Carlo Method*, M.S. Thesis, Colorado State University, Fort Collins, CO.

Maltby, J. D., and Burns, P. J., 1991, "Performance, Accuracy and Convergence in a Three-Dimensional Monte Carlo Radiative Heat Transfer Simulation," *Numerical Heat Transfer, Part B; Fundamentals*, Vol. 16, pp. 191-209.

Maltby, J. D., Zeeb, C. N., Dolaghan, J., and Burns, P. J., 1994, *User's Manual for MONT2D - Version 2.6 and MONT3D - Version 2.3*, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO.

Modest, M. F., 1993, *Radiative Heat Transfer*, McGraw-Hill, St. Louis.

Panczak, T. D., 1989, "A Fast, Linear Time, Monte Carlo Radiation Interchange Program Utilizing Adaptive Spatially Coherent Subdivision," In *Numerical Methods in Thermal Problems, Vol. VI, Proceedings of the Sixth International Conference*, R. W. Lewis and K. Morgan, ed., Pineridge Press, Swansea, U.K., pp. 701-712.

Raboin, P. J., 1998, "Computational Mechanics Moves Ahead," *Science and Technology Review*, May, Lawrence Livermore National Laboratory, UCRL-52000-98-5, pp. 12-19.

Rushmeier, H. E., 1993, "Computer Graphics Techniques for Computing Radiation Heat Transfer," *Proceedings of the NFS Joint Workshop on Radiative Heat transfer in Highly Coupled Physical Systems*, University of Texas at Austin, Austin, TX, October 4-8.

Shapiro, A. B., 1985, "TOPAZ3D - A Three-Dimensional Finite Element Heat Transfer Code," Lawrence Livermore National Laboratory, UCID-20484.

Sung, K., 1991, "A DDA Octree Traversal Algorithm for Ray Tracing," In *Proceedings of EUROGRAPHICS '91*, F. H. Post, and W. Barth, ed., Elsevier Science Publishers B. V., North-Holland, pp. 73-85.

Sung, K., and Shirley, P., 1992, "Ray Tracing with a BSP Tree," In *Graphic Gems III*, D. Kirk, ed., AP Professional, San Diego, CA, pp. 271-274.

Sutton, S., Erlandson, A., London, R., Manes, K., Marshall, C., Petty, C., Pierce, R., Smith, L., Zapata, L., Beullier, J., Bicrel, B., 1998, *Thermal Recovery of the NIF Amplifiers*, Lawrence Livermore National Laboratory, UCRL-JC-124528 Rev 1.

XYZ Scientific Applications, Inc., 1997, *TrueGrid Manual - Version 1.4.0,* XYZ Scientific Applications, Inc., Livermore, CA

Zeeb, C. N., Burns, P. J., Branner, K., and Dolaghan, J., 1999, *User's Manual for MONT3D - Version 2.4*, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO.