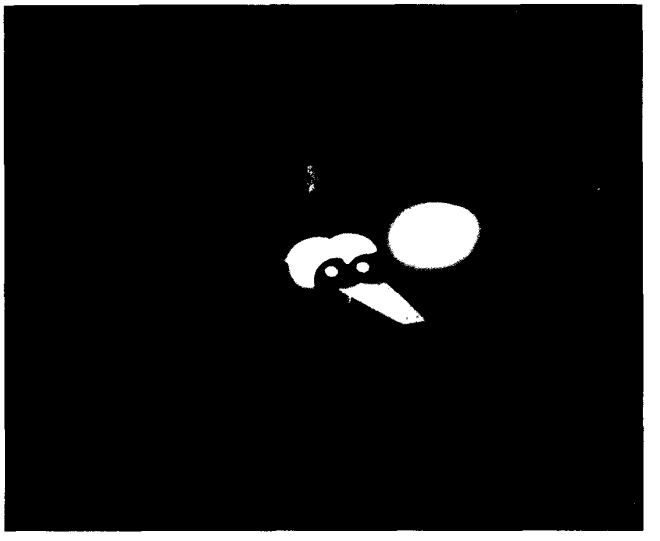
Spacetime Ray Tracing for Animation

Andrew S. Glassner
University of North Carolina at Chapel Hill



We are presenting techniques for the efficient ray tracing of animated scenes. These techniques are based on two central concepts: spacetime ray tracing, and a hybrid adaptive space subdivision/bounding volume technique for generating efficient, nonoverlapping hierarchies of bounding volumes.

In spacetime ray tracing, instead of rendering dynamically moving objects in 3D space, we render static objects in 4D spacetime. To support spacetime ray tracing, we use 4-dimensional analogues to familiar 3-dimensional ray-tracing techniques.

Our new bounding volume hierarchy combines elements of adaptive space subdivision and bounding volume techniques. The quality of the hierarchy and its nonoverlapping character make it an improvement over previous algorithms, because both attributes reduce the number of ray/object intersections that must be computed. These savings are amplified in animation because of the much higher cost of computing ray/object intersections for motion-blurred animation.

We show it is possible to ray trace large animations more quickly with spacetime ray tracing using this hierarchy than with straightforward frame-by-frame rendering.

Ray tracing is a powerful and popular technique for image synthesis. When first introduced for computer graphics,[1,2] ray tracing was comparable in power to scan conversion, but less attractive because of its high computational cost.

Survey

The effects of reflections, refractions, and shadows were estimated by adding recursion to the original ray-tracing algorithm.[3,4] Unfortunately, some notable combinations of these effects were incorrect.

Image synthesis and ray tracing

For example, if a shadow-testing ray encountered a partially transparent sphere, there was no proper single direction in which to send the ray after passing through the sphere's surface. Either this shadow was rendered as though blocked by an opaque object, or the modeler

0272-1716/88/0300-0060$01.00 1988 IEEE

introduced ad hoc techniques into the algorithm to handle particular situations correctly.

A solution to some of these problems was introduced in the form of distributed ray tracing.[5] Whenever there was no single correct value for a ray parameter (such as the direction of the shadow ray discussed above), the domain of useful values was searched for an "appropriate" choice. This choice was made on the basis of the shape of the parameter space being sampled and the expected number of samples to be taken. Soft shadows and antialiasing in all dimensions were now available in a single, conceptually elegant algorithm. A technique for dynamically optimizing the number of rays cast when generating an image was presented by Lee and Uselton.[6]

The ray-tracing algorithm was theoretically unified and extended again by Kajiya.[7] Ray tracing was formalized as a technique for solving the "rendering equation," which describes light distribution and energy balancing in an environment. This work suggested ways to include caustics and diffuse interreflections in a ray-tracing environment.

Unfortunately, a straightforward implementation of ray tracing is prohibitively expensive in computer resources and time. Finding efficient techniques to implement ray tracing is an active research area.

A brief survey of single-image rendering speedup techniques

Efforts to improve the efficiency of the technique have taken place on two major fronts: bounding volumes and space subdivision. Both of these efforts have seen investigation of important subissues: hierarchies for bounding volumes and the style of decimation for space subdivision.

A central idea behind bounding volumes is that it is often cheaper to intersect a ray with several mathematically simple objects than a single complex one. So complex objects are surrounded by simple objects (the bounding volumes), and these are recursively grouped together and enclosed within larger bounding volumes, forming a hierarchy. Rays that miss a bounding volume save a lot of work: they needn't examine any object within. Rays that do strike a bounding volume must then be intersected with everything inside the volume (which might include smaller bounding volumes). Such rays suffer the penalty of having computed the bounding volume intersection; the details of this intersection are useless except to signal that the internal objects must be tested. Bounding volume approaches to ray tracing are described in a number of works.[8-14]

A different approach to speeding up ray tracing is called space subdivision. The central idea here is to decimate space into a collection of disjoint simple volumes (often boxes), which are chosen so that each encloses only a small number of objects. When a ray enters a given box, it is intersected only with the objects within that box. If no objects are hit within the box, the ray moves to the next box on its path and repeats the procedure. Several approaches that use space subdivision have been published.[15-18]

Both techniques address the issue of rendering a single image. In this article we propose combining these methods and extending them into the realm of animated sequences.

A hybrid technique combining adaptive space subdivision and bounding volumes

In this section we present a technique for the creation of efficient bounding volume hierarchies. The technique is a hybrid of adaptive space subdivision and bounding volume techniques.

The advantages of bounding volume techniques lie in their ability to easily avoid computing ray-object intersections for all objects within a bounding volume not penetrated by a particular ray. If the volume is entered, then all of its immediate children must be intersected. If the bounding volumes can overlap, then it is not sufficient simply to proceed with the nearest of these children, since the nearest bounding volume may not contain the nearest object.[9] In this context, the biggest drawback to bounding volume techniques is that sometimes ray-object intersections are ignored; such computations (which may be very expensive for complex objects) are unnecessary. A recent paper[14] presents some techniques for measuring and building a hierarchy, but the definition and construction of good hierarchies is still poorly understood.

The other popular speedup technique is space subdivision. Many space subdivision schemes use rectangular prisms (called cells) for the unit element of space. The hierarchy created by adaptive space subdivision techniques is excellent: No cells at any given level overlap, and cells are dense only where the database is dense. On the other hand, rectangular prisms can perform poorly as bounding volumes compared to sets of slabs and other techniques.

To summarize, bounding volumes offer tight bounds but poor hierarchies, while adaptive space subdivision offers poor bounds but very good hierarchies; the approaches are complementary in their strengths and weaknesses. Our technique is to use the excellent hierarchy created by space subdivision as a guide to control the structure of the tighter bounding volume hierarchy. We will now present an overview of the algorithm, and then discuss some variations.

The general theme of our approach can be summarized as constructing a bounding volume hierarchy in the order "space subdivision down, bounding volumes up." For simplicity, we will often refer to a bounding volume simply as a "bound."

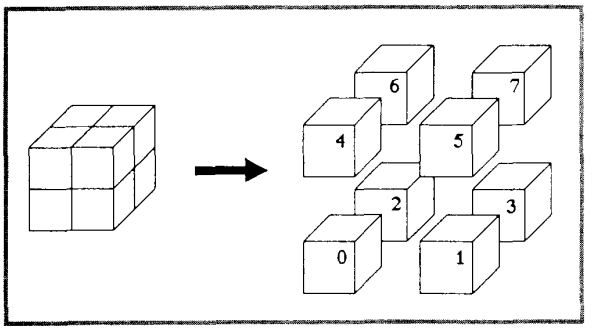


Figure 1. The subdivision of a rectangular prism into eight smaller prisms. On the left is the prism showing the locations of the cutting planes. On the right is an exploded view of the subdivided prism showing the labels of the eight smaller prisms.

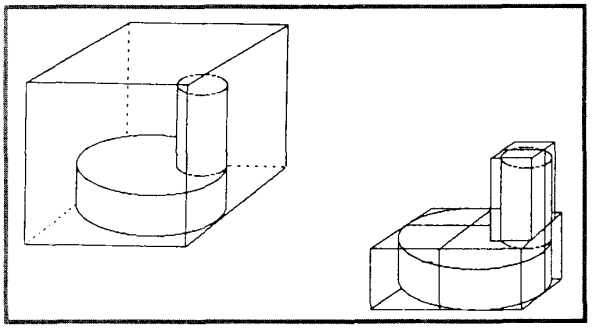


Figure 2. On the top is an object surrounded by a single bounding volume (for clarity the bounding volumes are rectangular prisms in this figure). On the bottom is the same object after the surrounding bounding volume has been subdivided. Note that the new, smaller bounding volumes are contained within, but not equal to, the smaller prisms created by the subdivision of the original bounding volume.

We begin by finding an enclosing box for the entire database, including light sources and the eye. We then evaluate a subdivision criterion (discussed below) for that box and its contents. If we decide to subdivide, then that box is split into eight new, smaller boxes, as shown in Figure 1. It is important to note that these boxes do not overlap. We then examine each new box in turn, determining which objects within the parent box are also within each child. We then evaluate the subdivision criterion for each child box, and recursively apply the subdivision procedure for each box that must be split. The recursion terminates when no boxes need to be subdivided.

This concludes the "space subdivision down" step. We now build the bounding volume hierarchy as we return from the recursive calls made by the space subdivision process. Each node is examined, and a bounding volume is built which encloses all the objects contained within that node, *within the bounds of that node.* One way to visualize this process is to consider building a bounding volume for all objects within a node, and then clipping that volume to lie within the walls of the space subdivision box, as shown in Figure 2 (note that implementations may use a simpler and more direct method). As we work our way back up the space subdivision tree, we build bounding volumes that contain the bounds and primitives of the child boxes at each node. We note that if a cell has only one child, then we may replace that cell by its child to improve efficiency when rendering.

This completes the "bounding volume up" step. The result is a tree of bounding volumes that has both the nonoverlapping hierarchy of the space subdivision technique and the tight bounds of the bounding volume technique. Thus the new hierarchy shares the strengths of both approaches while avoiding their weaknesses. The result is that when we trace a ray, we can always examine the nearest bounding volume at all levels in the hierarchy. If we find an intersection in that volume then we can immediately stop. Figure 3 summarizes the hierarchy creation process.

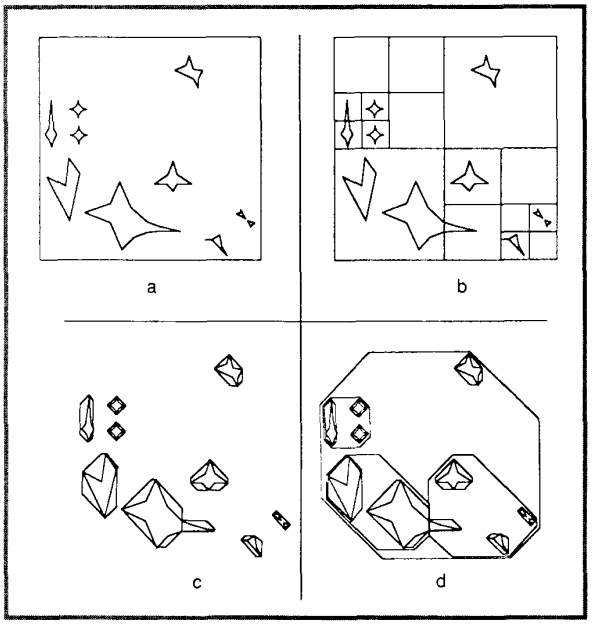


Figure 3. (a) shows a scene of 10 objects. (b) shows an adaptive space subdivision grid placed on those objects, subdividing any cell that contains more than two objects. (c) shows octagonal bounds (four slabs) placed around each primitive. (d) shows the final bounding hierarchy formed by the bounds in (c) and the subdivision tree in (b).

If no intersection is found, we then proceed to the next bounding volume, using either the bounding volume or

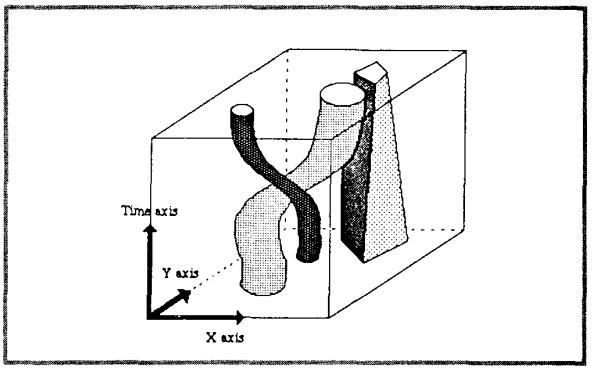


Figure 4. This diagram shows a 3D spacetime. The two space axes are labeled X and Y. Each slice of this spacetime volume parallel to the X and Y axes selects the world at a particular instant of time. Inhabitants of this world would experience the flow of time if they were moving along the time axis at a steady rate.

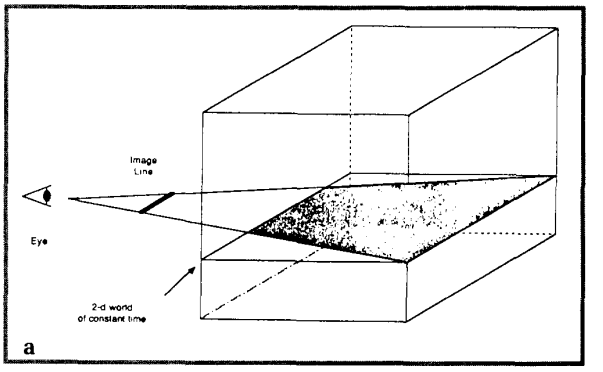
space subdivision structure to guide the ray propagation. It is important to note that since our bounding volumes might not completely enclose their objects, we must check that the intersection is indeed within the limits of the current bound.

There are many ways to apply adaptive space subdivision and bounding volumes; we will briefly mention some of the variations. The space subdivision cells may be axis-oriented,[15] or oriented arbitrarily in space.[13] The adaptive subdivision may be performed by equal cuts in all directions,[15] in a BSP methodology,[17] or with the median cut algorithm[19] based on the distribution of objects in the cell. The subdivision criteria may be based on the amount of projected "void area,"[10] the object count in a cell,[15] or on the density ratio of the total volume enclosed by the objects to the total volume of the cell.[20] The last technique is useful when working with "intelligent" objects, which may represent themselves with different bounding volumes, depending on the level of the bounding hierarchy.

The bounding volume construction may use rectangular boxes,[13] polyhedrons,[10] parallel slabs,[9] or surfaces of revolution.[21] Because the bounds constructed at each cell are a union of the bounds of all child cells and primitive objects in that cell, the style of bound at each cell may be different, enabling one to "tune" the bounds of each object individually.

Spacetime ray tracing

The central idea in our solution to the ray tracing of animated sequences is to consider the time-varying geometry of the 3D database as a static structure in 4D spacetime.[22] Since many people find it difficult to visualize 4D spaces directly, we will approach the 4D spacetime algorithm by analogy with 3D spacetime.



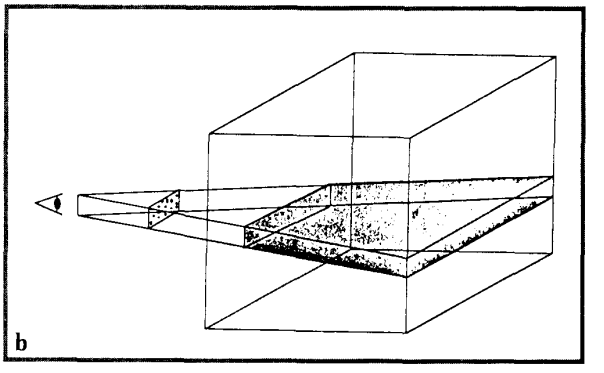


Figure 5. (a) To ray trace a frozen instant of 3D spacetime, we choose a 2D slice along the space axes. This entire slice has the same time value. We then project this 2D world onto a 1D image line, with the observer at the apex of this 2D viewing pyramid (a triangle). (b) To approximate motion blur, different samples in the image are taken at different times. This has the effect of thickening our sampling plane into a sampling volume.

3D spacetime

Three-dimensional spacetime can be thought of as a 3D space, containing a 2D space translated continuously in time.[23] Figure 4 shows a 2D world (a section of a plane), changing with time. In an animated sequence, objects will move about in this 2D space as time progresses.

Most of the rendered animation we usually produce consists of the projection of worldly 3D objects onto a 2D image plane. In a world of one less dimension (the 2D world of Figure 4), we render 2D objects onto a 1D image line, as shown in Figure 5a. Let's say we want to use ray tracing to produce a movie of this changing 2D world. Our rays that sample the 3D spacetime may start at any point (say an intersection with an object) and move in any direction. If we want to include motion blur in our movie, then these rays may also start at any time during a frame, as shown in Figure 5b.

The question now becomes one of quickly tracing the 3D ray in spacetime, intersecting it with each 2D object in its spacetime path. Each intersection of a ray and object is denoted by the three coordinates (X,Y,T). Each (X,Y) location in pure space is called a space point. Each (X,Y,T) location in spacetime is called a spacetime event.[24]

To make our movie of the motion depicted in Figure 4, we could simply shoot rays from the eye, at various times and in various directions, into the 3D spacetime structure and try intersecting the rays with each object, searching for the first event along the ray's path. This naive approach would be very expensive computationally. Alternatively, we can adapt the bounding volume hierarchies described in the previous section. Instead of building spatial bounding volumes in 3D space, we will build spacetime bounding volumes in 3D spacetime. As long as we know the 3D spacetime structure, we can rename one of the axes in the 3D space algorithm as time. When we subdivide along the time axis, we are actually now subdividing the amount of time for which this bounding volume encloses its child objects.

One way to see this is to envision Figure 1 as bounding volume for a 2D object in 3D spacetime. With this interpretation, nodes 0 through 3 now contain the first half of the time interval, and nodes 4 through 7 contain the second half (visualize the time axis as moving from the bottom to the top of the page). Now we can restrict our intersection tests only to those objects that occupy the same region of space and time that the ray is sampling.

Subdivision in higher dimensions

So far we have looked at a 3D spacetime containing a 2D world, rendered onto a 1D image line. The techniques discussed above extend easily into a 4D spacetime of three spatial dimensions plus time. Higher dimensions are also straightforward, and may be useful in situations where objects change along dimensions that are being sampled other than just space and time, such as wavelength.

Animation in 4D spacetime

When we pierce a spacetime bounding volume with a 4D ray, we don't yet actually have the ray/object intersection event. Since collections of objects and other bounding volumes may reside within a single bounding volume, we must look into the volume and test the ray against its contents. Because objects may move in complicated ways over time, we feel that 4D ray tracing is best handled by an object-oriented environment, which allows intelligent objects to perform their own intersections. After describing such an environment, we will describe how to achieve the same function (though with greater effort) from data-driven animation in a procedural environment, such as a traditional keyframe animation system.

Bounding volumes and intersection events from intelligent objects

In our technique the bounding volumes are created by the objects themselves, in response to requests by the hierarchy construction preprocessor.[25] Requests consist of asking an object for its enclosing spacetime volume within some region of spacetime. Many objects can easily respond with one of the bounding volumes discussed earlier.

An advantage of this intelligent object approach (such as described by Amburn, et al.[26]) is that objects can determine their own most efficient representations. For example, a group of stars may represent itself by a single bounding volume when the subdivision begins. When the bounding volume requests enclose smaller spacetime volumes, the star group may improve its representation by describing itself as several smaller clusters, returning several bounding volumes instead of one. Another advantage is the simplicity of the program itself, and the ease of adding new objects. Objects are also able to respond to requests to intersect themselves with a particular 4D ray, returning the first such event along the ray if one exists.

If the application environment of the ray tracer does not support such intelligent objects, then the work of building bounding volumes and finding intersection events must be made by the animation manager. For example, a keyframe animation system would need to construct the bounding volumes for objects in given ranges of space and time according to the interpolation techniques it used to build the animation. When building a particular bounding volume, such a system needs to examine the object carefully throughout the time duration of the request to which it is responding. In a complicated animation system objects may move and change in complicated ways; one must be careful to insure that each bound completely encloses the object for the entire time interval. Similar care must also be taken when determining intersection events.

Summary of the 4D spacetime algorithm

The creation of a piece of animation begins with a preprocessing step. This step recursively builds an adaptive space subdivision tree on the static 4D spacetime structure of the moving objects. When we return back up the tree, we build bounding volumes that enclose the objects at each cell.

To render this structure we fire 4D rays into the bounding volume hierarchy. Because of the nonoverlapping nature of the hierarchy, we are guaranteed that we may choose the nearest bounding volume at every level. If we strike an object in this nearest volume, then we need not also test other, further volumes.

Motion blur due to camera motion is naturally accommodated by using different starting times and positions for the primary rays.

The sources of efficiency

The spacetime rendering algorithm is efficient for animation for the same reason that space subdivision and bounding volumes are efficient for single frames. Consider that in ray tracing, a ray must find its nearest object: This requires searching the entire database. Space subdivision sorts the database almost completely in a preprocessing step. Now each ray need only search the objects in a given volume. Any bounding volume hierarchy in fact does the same thing, although the sorting may be more complicated. With these techniques, each ray needs only to search through a small number of objects, each with a high probability of intersection. In single-frame techniques the bulk of the searching was distributed to the preprocessing sort that built the hierarchy of space enclosures.

Space subdivision and bounding volumes speed rendering by sorting the database once at the start of the frame, instead of for every ray. The technique introduced in this article speeds rendering by using a single, nearly complete spacetime sort instead of many space sorts. It performs this one sort at the start of the animation, instead of for every frame.

Another important source of efficiency is the reduction in the number of object transformation calculations that must be performed. When we shoot rays at different times to approximate motion blur, those rays intersect the objects in the database at different times. To intersect the ray and each object properly, the object must be transformed to the correct position, orientation, and shape for that time. If the object motion is complex, the transformation may include deformations and other sophisticated changes. These transformations may be very expensive to compute. Because our bounding volume hierarchy is nonoverlapping, we avoid computing intersection events along hierarchy descent paths that don't lead to the first intersection. This reduction in the number of intersections that we must compute can become significant for complex object transformations in dense regions of the database.

Implementation

The algorithm generating the hierarchy of spacetime bounding volumes requires a technique of bounding volumes and a technique of adaptive space subdivision. In our implementation, we chose slab bounding volumes[9] and equal subdivision.[15] Both algorithms are easily extended to work in spacetime instead of just space.

Spacetime rays are represented by a pair of 4D events giving the origin and direction of the ray. When rendering at normal scales, the time component of the direction vector may be set to 0, implying that the light ray has infinite speed. At extremely large and small scales, we may instead set the time component to a value consistent with the speed of light in the database. With suitable enhancements to the ray-tracing geometry, we can then handle relativistic effects.[24]

In the 3D environment, a good set of bounding planes consists of the seven planes generated from the three axes and the eight octants they form.[9] The three axes give rise to three principal planes (each containing one unique pair of axes: XY, XZ, or YZ), plus four auxiliary planes that each diagonally slice two octants.

In 4D we have four principal axes, which cut spacetime into 16 subspaces, which we call hexants. Eight of these hexants contain the first half of the time interval, while the other eight cover the latter half of the time interval. We now have four principal planes (containing XYZ, XYT, XZT, YZT), plus eight auxiliary planes that diagonally slice half of the hexants, for a total of 12 planes.

Our principal planes have normals:

(1,0,0,0) (0,1,0,0) (0,0,1,0) (0,0,0,1)

Note that these slabs are required if we are using axis-oriented subdivision, since they form the walls that separate adjacent cells. The auxiliary planes have normals

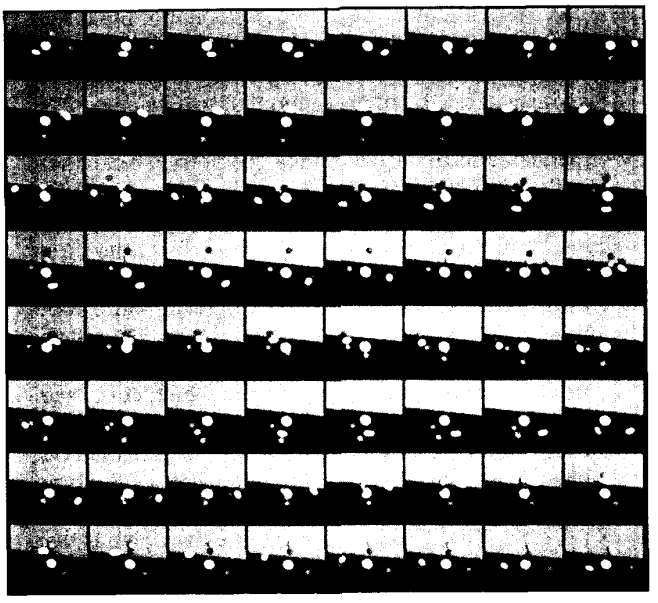
(.5,.5,.5,.5) (.5,.5,.5,−.5) (.5,.5,−5,.5)
(.5,.5,−.5,−.5)

(.5,−.5,.5,.5) (.5,−.5,.5,−.5) (.5,−.5,−.5,.5)
(.5,−.5,−.5,−.5)

Using planes formed by these normals we effectively enclose each spacetime path in a convex, bounding polyhedron formed by the intersection of 12 slabs, each composed of two parallel planes. We may add additional spacetime slabs to those above if we desire even tighter bounding volumes.

The cost of the 4D spacetime ray/slab intersection is virtually the same as for the 3D case. The difference is an extra pair of multiplies and additions once per ray per normal to compute both of the dot products of the spacetime normal with the ray origin and direction.[9] But we consider that cost negligible, since it is amortized over the life of the ray.

Our system is implemented in the C programming language under Unix, which is not the most natural environment for object-oriented programming. We thus use indirect procedure calls and consistent methodology to achieve an object-oriented flavor in the system. For example, our objects are able to respond to messages requesting bounding volumes within a given 4D box, intersections with a given ray, and intersection comple-



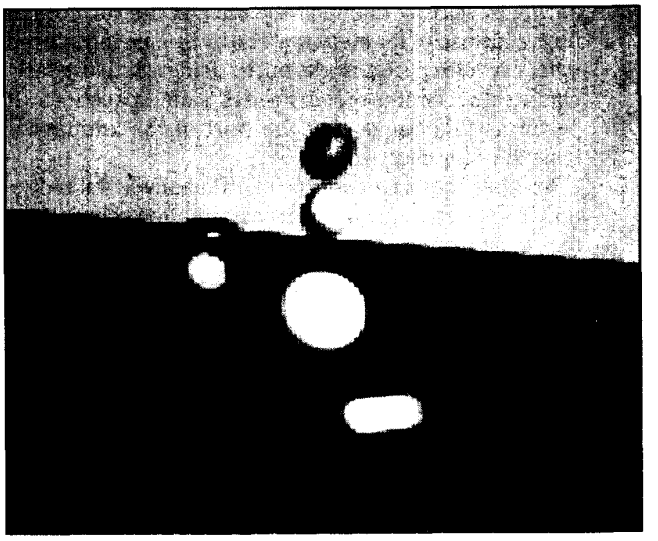


Figure 6. The upper photo shows a cyclic 64-frame animation of six small spheres ("electrons") spinning in different speeds in complicated motion around a larger central sphere (the "nucleus"). The lower photo is an enlargement of the 25th frame, showing the effect of motion blur on the small balls and their shadows.

tions (e.g., determining surface normal). Our objects also perform a variety of householding tasks such as maintaining their own motion paths, managing time-varying surface deformations and texturing, and so on.

Results

Figures 6, 7, and 8 show three animations we have produced with these techniques. Because of the limitations of the print medium, we present the animations by grids of frames equally spaced in time. Read the animation grids as you would read a book: starting at the upper left, moving left to right and top to bottom.

Figure 6 shows 64 samples from a cyclic animation of an atomic model. Six spheres spin about each other and about a central nucleus in a complicated ballet. Motion blurring is evident in the faster moving balls and their shadows.

Figure 7 shows 16 samples from a cyclic animation of a group of spheres, moving on the surface of an octagonal prism. This figure was generated with four samples per pixel, distributed in space and time.

Figure 8 is a 64-frame synopsis of the short film *Dino's Lunch*.

Tables 1 and 2 summarize the statistics we have measured for Figures 6 and 8. All frames were generated at 126-by-126 pixels. Each frame of Figure 6 contains seven spheres and one polygon, and was sampled with a constant 32 eye-rays per pixel. Each frame of Figure 8 contains 47 spheres and 21 polygons, and was sampled with a constant four eye-rays per pixel. All eye-rays were distributed in the 3D spacetime volume occupied by the frame.

The columns labeled "Frame-by-frame" report the costs for the entire animation when generating purely spatial bounding volumes anew for each frame. These

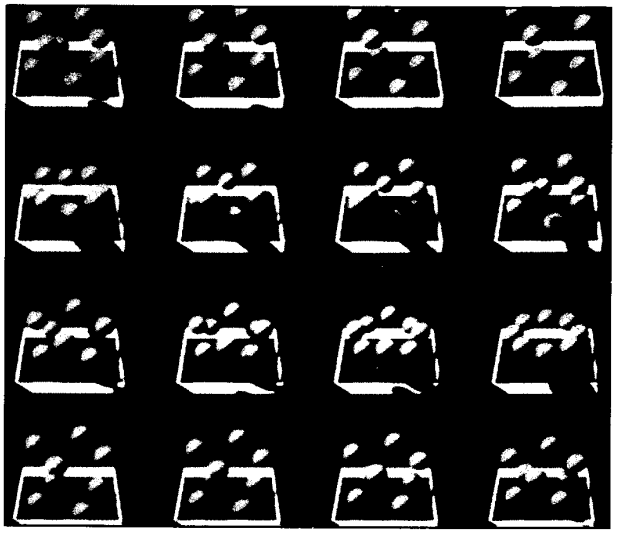
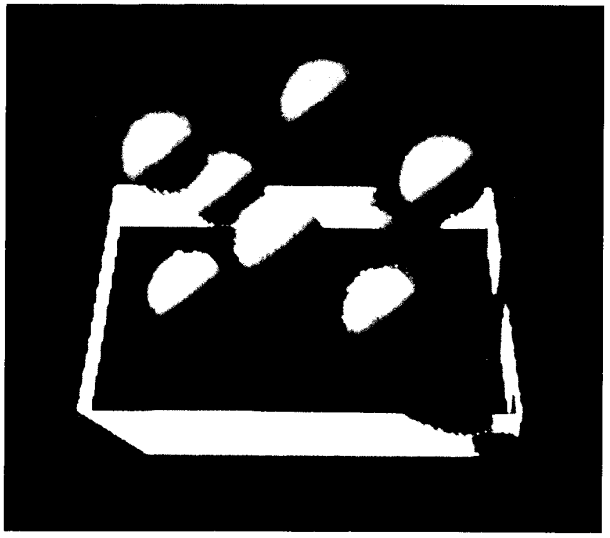


Figure 7. The upper photo shows a cyclic 16-frame animation of eight spheres bounding on the surface of an octagonal prism in a small box. The lower photo is an enlargement of the 12th frame, showing the speckled pattern characteristic of low-density distributed ray tracing with motion blur (each pixel fired four rays).



bounds were taken to encompass the object for the duration of the frame, created and arranged in a hierarchy as in Kay and Kajiya.[9] The column labeled "Spacetime" reports the equivalent costs using the techniques in this paper.

Figures 6, 7, and 8 were all computed using a hybrid subdivision criterion. At the upper levels of the tree, we subdivided until no more than three objects were in a cell. After meeting that criterion, we used a density measure: If the ratio of the volume enclosed by the objects in a cell to the volume of the cell was less than 0.3, the cell was subdivided (we used standard numerical integration techniques[27] to estimate the volumes). The column labeled "Spacetime to frame-by-frame ratio" contains the ratios of the animation totals for the two techniques, and is graphed in Figure 9.

Clock timings are not presented in the tables, since actual rendering times are strongly influenced by programming style and code tuning. Specifically, our code is not optimized for the algorithms in this article, since it performs many other tasks as part of a much larger system.

We have thus normalized all time measurements to an arbitrary unit time. Our unit time was the average time to render one frame of Figure 8. All measurements below the heavy horizontal line in the Tables are reported relative to this time unit. The proper statistics for comparison lie not in the elapsed time, but in the other columns, reporting the number of bounding volumes made and intersected, and the number of ray/object intersections with their accompanying expensive object transformation.

Discussion

The ratios in the comparison columns in Tables 1 and 2 are encouraging. They reveal that even in animations of modest complexity spacetime ray tracing with a hybrid hierarchy yields savings over frame-by-frame rendering.

From Table 1 we see that spacetime ray tracing was able to cut the rendering cost of Figure 6 to about 50 percent of that required by frame-by-frame techniques. Table 2 shows that spacetime ray tracing reduced the cost of Figure 8 to about 80 percent of frame-by-frame methods.

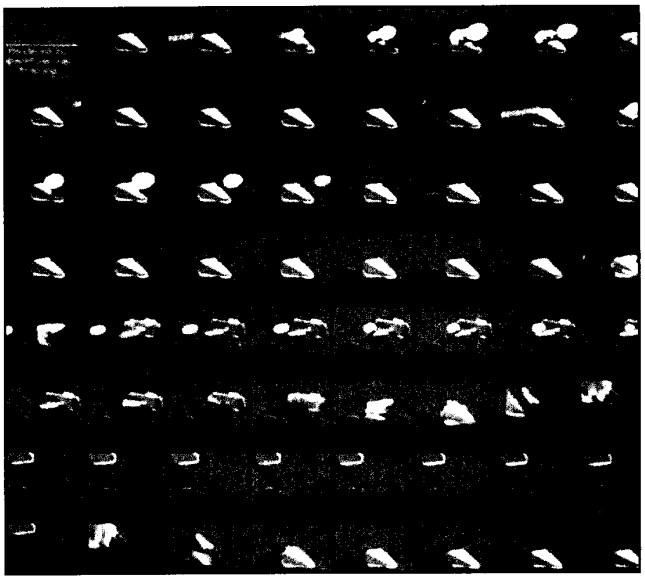


Figure 8. This is a 64-frame sample of the short film *Dino's Lunch*.

Table 1. Rendering statistics for Figure 6.

	FRAME-BY-FRAME		SPACETIME	
	Frame Average	Animation Total	Animation Total	Spacetime to Frame-by-Frame Ratio
# rays	793,637	50,792,795	50,792,794	1.0
# bounding volumes built	7.3	468	217	0.463
# ray/primitive intersections	1,825,365	116,823,428	66,030,634	0.565
# ray/bounding volume intersections	2,857,093	182,854,062	106,664,869	0.583
average primitive intersections per ray	2.3	2.3	1.3	0.565
average bounding volume intersections per ray	3.6	3.6	2.1	0.583
bounding volume hierarchy creation time	0.011	0.710	0.33	0.465
rendering time	34.8	2233	1179	0.768
total animation generation time	35.1	2234	1180	0.528

Table 2. Rendering statistics for Figure 8.

	FRAME-BY-FRAME		SPACETIME	
	Frame Average	Animation Total	Animation Total	Spacetime to Frame-by-Frame Ratio
# rays	22,748	1,455,876	1,455,876	1.0
# bounding volumes built	66	4221	331	.08
# ray/primitive intersections	73,148	4,681,506	4,201,077	0.90
# ray/bounding volume intersections	275,962	17,661,570	12,841,239	0.73
average primitive intersections per ray	3.22	3.22	2.89	0.90
average bounding volume intersections per ray	12.13	12.13	8.82	0.73
bounding volume hierarchy creation time	0.1	6.4	2.2	0.34
rendering time	**1.0**	64	49	0.77
total animation generation time	1.4	90.2	71.1	0.79

Since spacetime ray tracing builds bounding volumes only once at the start of the animation, we would expect that it should generate fewer bounding volumes over the course of the animation than frame-by-frame techniques. The results show that we indeed observed such an effect in both animations. The most important statistic is the number of ray-object intersections; this figure also decreases in spacetime ray tracing, thanks to the additional information provided by the time component in the spacetime bounding volumes.

Thus even in these simple animations, spacetime ray tracing can offer us significant savings in time by reduc-

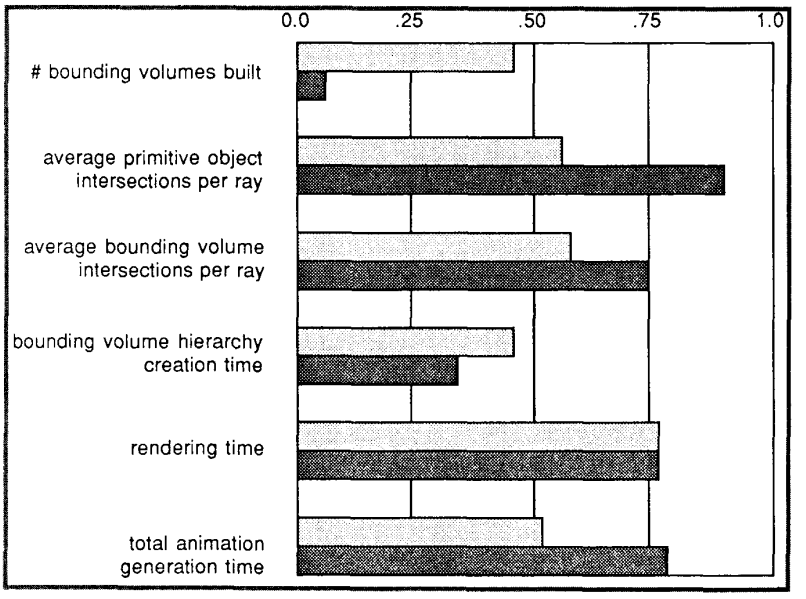


Figure 9. Percentage of work required by new technique relative to frame-by-frame techniques. Light shaded boxes are for Figure 6; dark boxes are for Figure 7.

ing the number of complex intersection operations that must be performed.

We note that several factors strongly affect these statistics, such as the distribution of objects in the scene, the complexity of each ray-object intersection, the complexity of the animation and database transformations, and the length of the animation in frames.

Consider a complex object changing in complex ways over time, such as a boiling fractal volcano with flowing lava. It can be *very* expensive to intersect such an object with a ray at a given time. This is because time-dependent intersections require positioning an object along its motion path, interpolation of all object description parameters, and then construction of the object itself (at least to a level sufficient to reject the ray). Without spacetime bounds all of this work will have to be performed for each object, even for rays that cannot possibly hit the object because they are in the wrong place at the wrong time. Spacetime bounds eliminate the majority of these useless intersection calculations, eliminating also their associated complex object positioning and construction operations.

We therefore expect that complex animation, involving many complex objects in sophisticated motion, will yield substantially higher savings than the examples presented here; indeed, we expect that as the animation grows more complex, the savings will become greater. This is based on the above discussions about the sources of efficiency, and by analogy to the performance of space subdivision and bounding volume algorithms. We are currently planning an elaborate animation called *Dino and the Windmill* (the sequel to *Dino's Lunch*) to test this expectation. *Dino and the Windmill* will also include extensive movement of the lights and camera.

Future work includes lazy evaluations of the bounding hierarchy (constructed of only those bounds needed as the animation progresses). We also plan to synthesize our methods with other multidimensional ray-tracing acceleration techniques, such as that described by Arvo and Kirk.[28]

Summary and conclusion

We have presented techniques for efficient ray tracing of animated scenes. We view the animation problem as a spacetime rendering problem. Thus, instead of rendering dynamically moving 3D objects in space, we render static 4D objects in spacetime. To trace rays in spacetime efficiently, we developed a hybrid technique of adaptive spacetime subdivision and spacetime bounding volumes, which generates an excellent hierarchy of nonoverlapping bounding volumes. The spacetime subdivision is also used during preprocessing to help intelligent objects select the most appropriate bounding volume for differently sized spacetime hypervolumes built as the subdivision progresses. We then trace 4D rays in this static spacetime to find ray-object intersection events.

We are able to ray trace a piece of animation more quickly with this spacetime algorithm and bounding hierarchy than with straightforward frame-by-frame rendering. ■

Acknowledgments

This work was developed and implemented in the Computer Graphics Lab at the University of North Carolina at Chapel Hill.

Thanks go to my advisor, Henry Fuchs, for his support of independent research. The idea of incorporating composite spacetime information into an animation system came out of discussions with Larry Bergman. Both a preliminary and a revised version of this article were carefully reviewed by a host of my fellow students in the Department of Computer Science; my thanks go to Marc Levoy, Pete Litwinowicz, Chuck Mosher, Tom Palmer, and Lee Westover. Kevin Novins of the Department of Radiology also provided insightful comments. Mark Harris and Doug Turner helped with the phrasing of important passages, and Margaret Neal helped make the article cohesive and clear. The observations and suggestions of these volunteer reviewers helped give this article structure and focus. Their friendly companionship in the Lab helped make the work a pleasure. Lakshmi Dasari provided key assistance in the production of the animations.

References

1. A. Appel, "Some Techniques for Shading Machine Renderings of Solids," *Proc. AFIPS Conf.*, Vol. 32, 1968, pp. 37-45.
2. W. Bouknight and K. Kelley, "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources," *Proc. AFIPS Conf.*, Vol. 36, 1970, pp. 1-10.
3. D. Kay, "Transparency, Refraction, and Ray Tracing for Computer Synthesized Images," master's thesis, Cornell University, Ithaca, NY, 1979.
4. T. Whitted, "An Improved Illumination Model for Shaded Display," CACM, June 1980, pp. 343-349.
5. R.L. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," *Computer Graphics* (Proc. SIGGRAPH), July 1984, pp. 137-145.
6. M.E. Lee, R.A. Redner, and S.P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics* (Proc. SIGGRAPH), July 1985, pp. 61-67.
7. J.T. Kajiya, "The Rendering Equation," *Computer Graphics* (Proc. SIGGRAPH), July 1986, pp. 143-150.
8. E.A. Haines and D.P. Greenberg, "The Light Buffer: A Shadow-Testing Accelerator," *CG&A*, Sept. 1986, pp. 6-16.
9. T. Kay and J.T. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics* (Proc. SIGGRAPH), July 1986, pp. 269-278.
10. H. Weghorst, G. Hooper, and D. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM Trans. on Graphics*, Jan. 1984, pp. 52-69.
11. J.T. Kajiya, "New Techniques for Ray Tracing Procedurally Defined Objects," *Computer Graphics* (Proc. SIGGRAPH), July 1982, pp. 245-254.
12. S. Roth, "Ray Casting for Modelling Solids," *Computer Graphics and Image Processing*, Vol. 18., 1982, pp. 109-144.
13. S.M. Rubin and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics* (Proc. SIGGRAPH), July 1980, pp. 110-116.
14. J. Goldsmith and J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing," CG&A, May 1987, pp. 14-20.
15. A.S. Glassner, "Space Subdivision for Fast Ray Tracing," *CG&A*, Oct. 1984, pp. 15-22.
16. M. Dippe and J. Swenson, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *Computer Graphics* (Proc. SIGGRAPH), July 1984, pp. 149-158.
17. M. Kaplan, "Space Tracing: A Constant Time Ray Tracer," SIGGRAPH 85 Tutorial on the State of the Art in Image Synthesis, July 1985.
18. A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: Accelerated Ray-Tracing System," *CG&A*, Apr. 1986, pp. 16-27.
19. P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Computer Graphics* (Proc. SIGGRAPH), July 1982, pp. 297-307.
20. A.S. Glassner, "Spacetime Ray Tracing for Animation," Introduction to Ray Tracing, course notes #13 (SIGGRAPH), ACM, New York, 1987.
21. J.T. Kajiya, "New Techniques for Ray Tracing Procedurally Defined Objects," *Computer Graphics* (Proc. SIGGRAPH) July 1983, pp. 91-102.
22. R. Rucker, The Fourth Dimension, Houghton Mifflin, Boston, 1984.
23. A. Abbott, *Flatland*, Dover Publications, Mineola, N.Y., 1952 (original copyright 1884).
24. P. Bergmann, Introduction to the Theory of Relativity, Dover Publications, Mineola, N.Y., 1975.
25. A.S. Glassner, "Supporting Animation in Rendering Systems, *Proc. CHI+GI*, Canadian Information Processing Soc., Toronto, 1987.
26. P. Amburn, E. Grant, and T. Whitted, "Managing Geometric Complexity with Enhanced Procedural Models," *Computer Graphics* (Proc. SIGGRAPH), July 1986, pp. 189-195.
27. W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes*, Cambridge University Press, N.Y., 1986.
28. J. Arvo and D. Kirk, "Fast Ray Tracing by Ray Classification, *Computer Graphics* (Proc. SIGGRAPH). July 1987, pp. 55-64.

Andrew S. Glassner is a PhD student in Computer Science at the University of North Carolina at Chapel Hill, where he studies algorithms for image synthesis, modeling, and animation. He spent recent summers working on computer graphics at a variety of research institutions, including the Delft University of Technology, Xerox PARC, IBM TJ Watson Research Laboratory, Bell Communications Research, and the New York Institute of Technology.

Glassner writes on computer graphics for both the technical and popular literature. His book for artists, Computer Graphics User's Guide, has recently been translated into Japanese. He is presently working on two new books, describing procedural geometric methods for graphics programmers, and symmetry design for artists. Glassner's current research interests include image synthesis, geometrical methods for designing computer graphics algorithms, and the theory of certain knotwork patterns. He is also interested in other uses of computers for enhancing and supporting creativity, including music, multimedia displays, and interactive fiction, both verbal and visual.

Glassner can be reached at the Department of Computer Science, Sitterson Hall, Box 3175, UNC-CH, Chapel Hill, NC 27599.