INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 37, 3571-3580 (1994)

RAY TRACING WITH A SPACE-FILLING FINITE ELEMENT MESH

JEAN FAVRE AND RAINALD LÖHNER

School of Engineering and Applied Science, The George Washington University, Washington, DC 20052, U.S.A.

SUMMARY

A new ray-tracing technique is presented which does not work with ray–object intersections *per se*, but is based on the traversal of an unstructured tetrahedral mesh providing a convex enclosure of a scene of polyhedral objects. The tetrahedral mesh provides tight bounding and an adaptive subdivision of space. This non-hierarchical data structure is traversed adaptively until one is led directly and unconditionally to the first object intersected. Rendering times are directly related to the average thickness of the enclosing mesh since all tetrahedra are traversed in constant time.

Since the proposed algorithm operates directly with volume elements, it allows for volumetric rendering effects. Volume rendering or anisotropic media can be implemented without any further effort. This is an important advantage as compared to usual techniques, which only operate on surface data.

Timings for several examples show that the use of this type of ray-tracing technique, which is more suitable for general purpose visualization codes than traditional techniques, results in CPU times that are comparable with the best ray-tracing techniques presently used. This is an unexpected and important result, as the vectorization and parallelization of the proposed technique are straightforward, in contrast with traditional ray-tracing techniques.

1. INTRODUCTION

In many situations, an engineer may want to display with a high degree of photo-realism the results from designs (CAD-CAM data) or simulations. One of the ways to achieve such photo-realistic imaging is by the use of ray-tracing techniques. The basic idea is to trace back the light rays from the viewpoint of the observer into the volume of the scene to be rendered. The light rays may impinge on a surface, and reflect multiply with or without attenuation. Several layers of reflections are followed, until the rays either reach a light source, exit the scene 'to infinity', or become too weak due to attenuation. Ray-tracing techniques have been applied extensively during the last decade.[1] The original algorithm used in most ray-tracing implementations was introduced by Whitted,[2] but many acceleration techniques have since been developed to overcome the very high cost incurred by a brute force evaluation of an image. Arvo and Kirk[3] have identified two subsets of acceleration techniques: one can either trace fewer rays, or one can trace faster rays. To achieve the former, 3-D space is usually subdivided into a number of cells which contain a smaller number of objects. The subdivision can be uniform[4] but the user has to trade off traversal time through fine grids versus firing more rays through coarser subdivisions. When the grid spacing is too small, traversing empty space can also become prohibitively expensive.

On the other hand, the subdivision may be done with octrees or BSP trees.[5,6] In all cases, any particular object may end up in more than one cell, resulting in object fragmentation. Extraneous

CCC 0029-5981/94/203571-10$9.00
© 1994 by John Wiley & Sons, Ltd.

Received 9 March 1993
Revised 8 June 1993

work is required to handle pathological cases and additional computations are also necessary to locate the next cell or voxel along the trajectory.

In order to trace faster rays, bounding volumes can be created which surround complex objects with a much simpler geometric object for which intersection calculations are easier.[7,8] When the bounding volume offers a rather tight bound around the object, many expensive intersection calculations can be replaced by the simpler calculations. One can also build a hierarchical set of bounding volumes. However, it can be difficult to create subdivision trees leading to an efficient traversal. The trees must have a low branching ratio to provide a more efficient pruning but it remains difficult to estimate the best traversal order. Furthermore, trees are not unique and it has been shown[5,6] that the order in which the database is created can heavily influence the total rendering time.

In summary, bounding volumes which are object-based can offer arbitrarily tight bounds but hierarchies of extents are difficult to manage and do not prevent the computations of some intersections which are later ignored. Uniform subdivision which is volume-based offers incremental traversal[4] but loose bounding. Non-uniform techniques improve the tightness of the bounds and provide very good hierarchies. Several authors have tried to take advantage of both types of techniques.[9,10] Glassner[9] proposed as a solution a 'tree of bounding volumes that has both the non-overlapping hierarchy of the space subdivision technique and the tight bounds of the bounding volume techniques.'

Given that rays can be thought of as 'particles' or 'photons', an alternative technique to trace rays is to follow them as they traverse 3-D space. The 3-D space is assumed discretized by a finite element or finite volume mesh. Instead of following fluid particles or magnetic field lines[11,12] rays are followed as they traverse the mesh. This allows the incorporation of 3-D volumetric effects, such as attenuation and dispersion of light due to fog or smoke, or the change in colour due to some physical variable of interest. Unlike particle tracing, the main ingredients required for photo-realism are proper boundary conditions at walls. The treatment of walls, on the other hand, can be directly incorporated from traditional ray-tracing techniques.[1] In most simulations, the finite element mesh is already given. If this is not the case, it may be constructed using an automatic gridding technique, such as the advancing front algorithm.[13] The mesh generation is required only once as it is independent of viewing and lighting conditions and it provides a non-uniform unstructured and adaptive spatial subdivision of the volume. The tetrahedral mesh provides an exact fit since each triangle on the objects' boundaries is the base of a tetrahedron. It represents a data structure which has the advantages of tight bounding volumes, non-overlapping hierarchies and provides an adaptive subdivision of space. This type of ray-tracing technique can be used for arbitrarily complex scenes composed of objects with faceted boundaries.

An important question regarding efficiency is whether the proposed technique can compete with traditional ones.[1] To our surprise, it can, as will be shown in Section 4.

An outline of the paper follows. Section 2 provides a summary of the mesh generation technique which yields a non-uniform unstructured and adaptive subdivision of the volume. As opposed to traditional ray-tracing techniques, this data structure is simpler, and yet supports a very efficient algorithm for the traversal of space. In Section 3, we base our discussion on the properties of the linear shape functions of the tetrahedron to provide an algorithm to move the 'particles of light' or 'photons' from one tetrahedron's face to another. By iterating this process, space can be traversed in an adaptive fashion until the light ray hits a surface, where pixel intensities are evaluated. Section 4 presents implementation details and results. Section 5 covers extensions to volumetric data visualization and other graphics techniques. Finally, some conclusions are drawn in Section 6 by comparing this technique with the more traditional ray-tracing techniques and showing its advantages.

2. MESH GENERATION

The proposed ray-tracing algorithm follows rays through a 3-D space discretized by tetrahedra. Therefore, a mesh must be provided. For most simulations, a proper 3-D mesh may be at hand. On the other hand, for CAD-CAM or design/scene display, a grid may have to be built. For this reason, we briefly describe the grid generator used for the present work. Given a surface description in the form of a triangulation, and an outer box enclosing all objects in the scene, the tetrahedral mesh is generated using the advancing front method.[14,13] A 'front' denotes the boundary between the region in space that has been filled with elements and that which is empty. We proceed by filling up the space between the given surface triangulation and the outside boundary given by our all-enclosing bounding box. The main algorithmic steps required to generate a mesh using this method are as follows:

1. Define the spatial variation of element size, stretching and stretching directions for the elements to be created.
2. Define the boundaries of the domain to be gridded. This is typically accomplished by splines in 2-D and surface patches in 3-D.
3. Using the information stored for element creation, set up faces on all these boundaries. This yields the initial front of faces.
4. Proceed to fill up the domain with tetrahedra: given the generation parameters (element size, element stretching and stretching directions) for a given face on the 'front', construct adjacent tetrahedra until the domain is completely filled up with elements.

Obviously, if the surface faces are given, Step 3 is avoided. Moreover, in this particular case the specification of element size and shape in space may be obtained by starting from the surface element size, and enlarging the element size progressively as the volume mesh is constructed.

The generation of the 3-D mesh is required only once. The generated mesh can be stored and reused for many scene renderings. Thus, the relative cost of generating the 3-D mesh may be neglected.

In the present case, it is assumed that the 3-D mesh is provided with the surface triangulation before the ray tracing is started. The ray tracer is used in the post-processing stage of a field solver analysis (heat flow, fluid flow, . . .). Thus, the cost of the mesh generation is really part of the field solver analysis process. The current version of the 3-D mesh generator code runs at about 12 000 tetrahedra/min on an IBM R6000/550 workstation. Thus, it takes about 8·5 min to generate a grid of 100 000 tetrahedra.

3. TRAVERSAL OF THE FINITE ELEMENT MESH

Given the position of an eye point in space, each ray is traced by marching along its direction vector by adaptive increments as we traverse tetrahedral elements of various sizes. For a given position in space and a ray vector, the next tetrahedron pierced by the ray is found. The strategy to find this new element is based on simple properties of linear basis functions.

For a tetrahedron such as the one pictured in Figure 1 with nodes $\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C, \mathbf{x}_D$ and an arbitrary point x_P, we can compute its local co-ordinates ξ, η and ζ as follows:

$$\mathbf{x}_P - \mathbf{x}_A = (\mathbf{x}_B - \mathbf{x}_A)\xi + (\mathbf{x}_C - \mathbf{x}_A)\eta + (\mathbf{x}_D - \mathbf{x}_A)\zeta \tag{1}$$

Introducing the notation $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, we obtain

$$\begin{pmatrix} x_{PA} \\ y_{PA} \\ z_{PA} \end{pmatrix} = G \cdot \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} \tag{2}$$

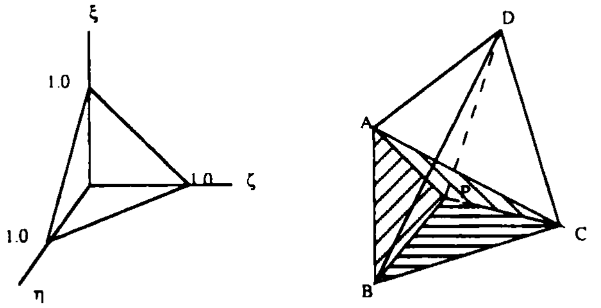

Figure 1. A tetrahedral element mapped into 'natural' co-ordinates ξ, η and ζ

where

$$G = \begin{bmatrix} x_{BA} & x_{CA} & x_{DA} \\ y_{BA} & y_{CA} & y_{DA} \\ z_{BA} & z_{CA} & z_{DA} \end{bmatrix} \tag{3}$$

Thus,

$$\begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} = G^{-1} \cdot \begin{pmatrix} x_{PA} \\ y_{PA} \\ z_{PA} \end{pmatrix} \tag{4}$$

The ξ, η, ζ co-ordinates are related to the element shape functions N_1, N_2, N_3, N_4 by the following formula:

$$\begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix} = \begin{pmatrix} 1 - \xi - \eta - \zeta \\ \xi \\ \eta \\ \zeta \end{pmatrix} \tag{5}$$

The physical nature of the shape functions is identified as the ratio of volumes of tetrahedra based on the internal point $\mathbf{x}_P$ in the total volume (see Figure 1). These volume co-ordinates vary linearly from unity at one node, to zero at the opposite face.[15]

Assuming for now that the ray of light travels in a straight direction, the intersection of the ray vector with the first tetrahedron on the outside boundary is found. Thereafter, the tetrahedral mesh is traversed by successive jumps along the direction vector, intersecting the tetrahedra's faces one after the other until an object face is hit. By doing so, one is able to traverse empty space very quickly while going through large tetrahedra, progressively 'slowing down' as the object boundaries (where the tetrahedra are of smaller size) are approached. The initial intersection problem is of limited complexity because, on the outside boundary of the enclosing volume, the tetrahedra are very large and have very large faces. The convex all-enclosing volume is used as a bounding box and use is made of nearest neighbours to limit the scope of search. In fact, we have noted experimentally that, given the viewing parameters of typical scenes, only a single tetrahedron face projecting over the entire viewing plane was present.

A description of the algorithm used to compute the intersection of the ray vector with a tetrahedron follows.

Ray vectors are followed by jumping from the intersection where the ray enters the tetrahedron at $(x_{\text{in}}, y_{\text{in}}, z_{\text{in}})$ to the intersection where the ray exits the tetrahedron at $(x_{\text{out}}, y_{\text{out}}, z_{\text{out}})$. Expressions for the element shape functions at the two end-points are given by equations (6) and (7):

$$\begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix}_{\text{in}} = \begin{pmatrix} 1 - \xi_{\text{in}} - \eta_{\text{in}} - \zeta_{\text{in}} \\ \xi_{\text{in}} \\ \eta_{\text{in}} \\ \zeta_{\text{in}} \end{pmatrix} \tag{6}$$

and

$$\begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix}_{\text{out}} = \begin{pmatrix} 1 - \xi_{\text{out}} - \eta_{\text{out}} - \zeta_{\text{out}} \\ \xi_{\text{out}} \\ \eta_{\text{out}} \\ \zeta_{\text{out}} \end{pmatrix} \tag{7}$$

We note that with the matrix G defined in equation (3), one has

$$\begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix}_{\text{out}} = \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix}_{\text{in}} + G^{-1} \begin{pmatrix} \mathrm{d}x \\ \mathrm{d}y \\ \mathrm{d}z \end{pmatrix} \tag{8}$$

The vector ($\mathrm{d}x$, $\mathrm{d}y$, $\mathrm{d}z$) represents the displacements along the ray during the time $\mathrm{d}t$ and with a velocity vector equal to the ray direction V.

Equation (8) can be written as

$$\begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix}_{\text{out}} = \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix}_{\text{in}} + G^{-1} V \mathrm{d}t = \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix}_{\text{in}} + \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix} \mathrm{d}t \tag{9}$$

Equation (7) is now equivalent to

$$\begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix}_{\text{out}} = \begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix}_{\text{in}} + \begin{pmatrix} -\alpha_x - \alpha_y - \alpha_z \\ \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix} \mathrm{d}t \tag{10}$$

As mentioned above, on each face of the tetrahedron, there is a shape function which has a zero value. Thus, the four planes defined by the faces have a unique intersection point with the ray vector where one of the shape functions is null. Figure 2 shows an example of such a configuration with P1, P2, P3, P4, the four intersection points (P2 not shown, it is on the horizontal plane defined by $\widehat{ABC}$). We derive the following system of equations which are solved for $\mathrm{d}t$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \alpha_x + \alpha_y + \alpha_z \\ 0 & 1 & 0 & 0 & -\alpha_x \\ 0 & 0 & 1 & 0 & -\alpha_y \\ 0 & 0 & 1 & 0 & -\alpha_z \\ \delta_{1i} & \delta_{2i} & \delta_{3i} & \delta_{4i} & 0 \end{bmatrix} \cdot \begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ \mathrm{d}t_i \end{pmatrix}_{\text{out}} = \begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ 0 \end{pmatrix}_{\text{in}} \tag{11}$$

The four corresponding values of $\mathrm{d}t$, representing displacements along the ray vector, are then found. δ_{ji} is the Kronecker delta and is used on the last row of our matrix equation to represent the equation corresponding to $\mathrm{d}t_i$. $\mathrm{d}t_i$ is the time required to travel to the intersection of the ray with the plane of face i.

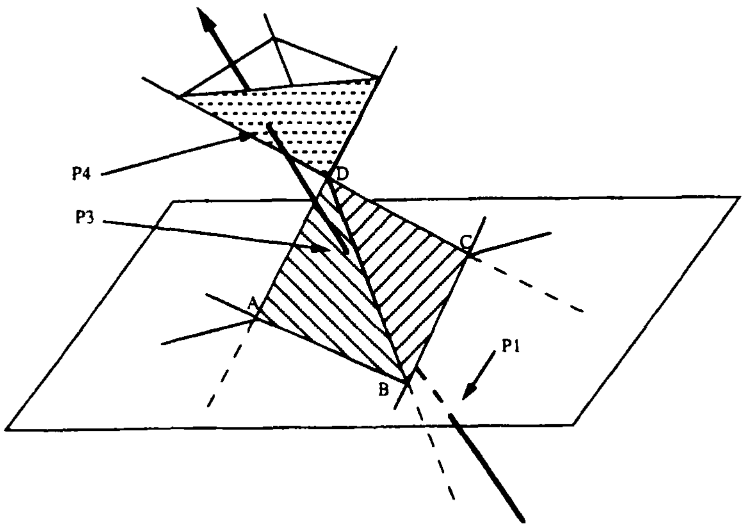

Figure 2. A ray vector intersecting the four zero-valued planes

Solving for the four different values of dt_i yields

$$
\begin{aligned}
dt_1 &= \frac{N_{1_{\text{in}}}}{\alpha_x + \alpha_y + \alpha_z} \\
dt_2 &= \frac{N_{2_{\text{in}}}}{-\alpha_x} \\
dt_3 &= \frac{N_{3_{\text{in}}}}{-\alpha_y} \\
dt_4 &= \frac{N_{4_{\text{in}}}}{-\alpha_z}
\end{aligned}
\qquad (12)
$$

The smallest positive dt_i, which corresponds to the closest intersection point on the tetrahedron's face, is then selected. The 'particle' or 'photon' has thus traversed the tetrahedron, jumping from one face to the other.

Once the intersection where the ray vector exits the current element is found, it is necessary to identify the element that is next. A data structure is required that contains all four neighbours of an element. A $4 \times N$ array—N being the total number of elements—of 'elements-surrounding-elements' named `esuel`[16,11,17] is constructed at the beginning in linear time complexity. The next tetrahedron along the traversal is then given by

`new_element = esuel[i][current_element]`

where `i` is the index of the shape function which has a value of zero on the boundary. The value of `i` corresponds to the index of the smallest dt_i.

4. IMPLEMENTATION AND RESULTS

Each light ray is followed through the tetrahedral mesh until a boundary face is reached, yielding the intersection sought without the need for multiple ray-object intersections. Tetrahedra on the boundary of the 3-D mesh have faces that can either touch the objects or lead to the outside world. The mesh traversal should terminate on these faces. Such conditions are accounted for by writing the face number with a negative sign to replace the next neighbour element in the array esuel. The grid traversal iterates until the next neighbour is found to be a negative number, i.e. a boundary face. With the same framework as the more classical ray tracers, one can now trace secondary rays and shadow rays. Viewing rays are first intersected with the outside boundary triangles using the properties of the shape functions outlined above. An additional array of 'faces-surrounding-faces' is used to take advantage of coherency. Once the intersection is found, the first tetrahedron is identified and the mesh traversal is carried out. Each tetrahedron stores the inverse of the geometry matrix G identified in equation (3).

A comparison of the proposed algorithm with Garrity's method[18] was carried out. Garrity's method intersects the ray vector with all faces of a tetrahedron. Although very straightforward, this method does not seem to have any advantage over the proposed algorithm. We estimated that to store cross-products and point information, he needs 12 bytes per face and 20 bytes per tetrahedron, assuming that he stores pointers to faces and points, although no space and operation count is given in his paper. Not counting 'shared' faces twice, $2 \times N$ faces require a total of $44 \times N$ bytes of storage for an N-element mesh. His calculations of intersection can be done with 30 floating point operations assuming that he knows how to identify the three candidate faces. On the other hand, the technique proposed here only requires the storage of the inverse of the matrix G, plus one pointer to the first node of each tetrahedron. This amounts to $40 \times N$ bytes, slightly less than Garrity's method. The average operation count is 48 floating point operations per traversal. However, each traversal yields not only the closest intersected face, but also the co-ordinates of the point and the values of the shape functions required for interpolation of the field value for pseudo-colouring and in volumetric rendering; the identity of the next cell along the traversal is also given. In Reference 18, the author does not explain how he obtains this very important information. Thus, although our operation count is larger, our method directly yields several other important values.

A comparison of the present implementation with an implementation of the hierarchical bounding volume method of Reference 6 was also performed. The described tests were run on an SGI Crimson with 64 Megabytes of memory. Both ray tracers computed two samples per pixel at a 640×480 resolution, and were run with the same viewing and scene information. Both implementations are written in C and use 64-bit arithmetic.

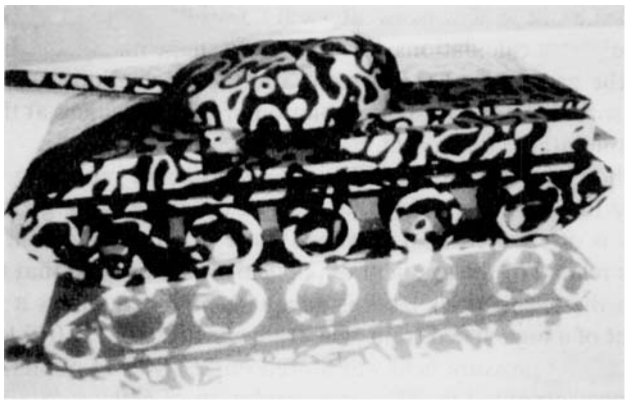

Figure 3. All-terrain camouflaged vehicle

Table I. Rendering times and number of element traversals for different levels of shading

	Traversals of tetrahedra	Rendering time (s)
Viewing rays	10 297 353	82
Viewing rays + reflected rays	19 384 900	156
Viewing rays + shadow rays	34 282 302	237
Viewing rays + reflected rays + shadow rays	57 990 701	412

The first test case consists of an object with 12 925 triangles (see Figure 3). A tetrahedral mesh used for fluid dynamics calculations was used with 43 004 tetrahedra. The scene was lit by two directional light sources which cast shadows. Viewing rays, shadow rays and reflection rays were cast. The proposed algorithm ray-traces the scene in 412 s vs. 424 s for the hierarchical bounding volume method.

An important goal is the design of a ray tracer which can render images in linear time, i.e. in a time proportional to the average thickness of the finite element mesh surrounding the objects. Table I shows that this goal has been reached with the proposed technique, and that rendering times scale linearly with each additional component of the shading equation.

5. OTHER APPLICATIONS

A camouflage texture was computed on the surface of the object in Figure 3. Alternatively, field values such as pressure can be painted on the object. In this case, the major advantage of ray tracing over the typical Gouraud shading used in CAD-CAM post-processing is that colour intensities at pixels are computed with specular highlights, depth attenuation and account for the orientation of the surface with respect to the light sources. This results in a much improved perception of the three-dimensionality of the shape.

The proposed ray-tracing technique can also be used for volumetric rendering. In a typical finite element analysis for flow studies, the output of the simulation will produce both the mesh used for traversal and one or several field values stored on the vertices of the mesh. Equation (10) yields the values of the shape functions at each tetrahedron's face pierced by the ray. In iso-parametric finite element calculations, it is these very shape functions which are used for both the interpolation of the geometry of the element and the calculations of the field values. Thus, once equation (10) is solved, we carry an interpolation of the field values at the vertices to find the value of the field at the intersection point.[15]

The proposed method can also be applied to do a volumetric rendering through an irregular mesh with the ability to merge the polygonal data of the underlying geometric structure. The intensity along a ray is attenuated as it traverses more opaque regions in the mesh. Since the density of the mesh is related to the gradient of the field, one is assured that the sampling done at adaptive intervals on the tetrahedra's faces is adequate. Figure 4 shows a rendering of a high-pressure cloud in front of a high-speed train. The train consisted of 127 049 tetrahedra and 18 720 boundary triangles and the pressure field was stored on 25 664 3-D points within the bounding volume. The image was computed in 209 s at a resolution of 640×480 with one light source.

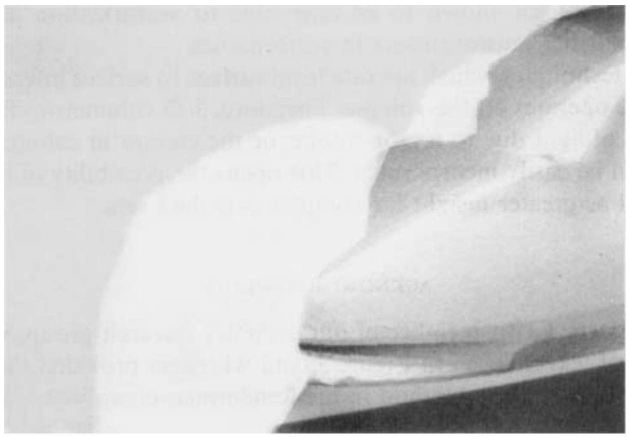

Figure 4. Pressure rendering on a high-speed train

Additionally, one can also modify the spatial characteristics of the medium surrounding the geometry. In addition to atmospheric attenuation modelled in several ray-tracing implementations, one can relax the assumption that the medium is homogeneous. The unstructured grid gives the ability to design a medium with anisotropic properties, by varying both its geometry and the field values stored at its vertices.

6. CONCLUSIONS

A new ray-tracing algorithm has been presented which brings together the advantages of volume subdivision and bounding volume techniques. Use is made of an adaptive unstructured tetrahedral mesh, which is view-independent and allows one to render polyhedral scenes in a time that increases linearly with the number of levels evaluated for pixel intensities. Each level of recursion can be completed in a constant time which measures the average traversal time of the mesh. The tetrahedrization of the domain gives a data structure which is traversed in a time that is independent of the total number of triangles in the scene. As opposed to uniform spatial subdivision, the tetrahedrization allows the generation of large tetrahedra which speed up the traversal of empty space.

Multiple ray–object intersections need never be computed. The finite element mesh is traversed adaptively until the exact object is hit. Thus, computing time is never expended to find ray–object intersections which must later be discarded.

The generation of the 3-D mesh, which provides an adaptive subdivision of the volume, is only required once and is actually part of the pre-processing stage of a field solver. In the case of scientific data, the mesh is provided for free before the ray tracing is started. At the same time, internal features of the volumetric data may be visualized.

At the present time, no effort has been expended to optimize the 3-D grid for ray tracing. The original mesh generation algorithm, which was tuned to flow analysis applications, was used for all examples presented. In fact, the mesh used in the test case of Figure 3 could be optimized with fewer layers.

Even with these suboptimal meshes, the times required by the present ray tracer are comparable with one of the best traditional ray tracers.[1] This is a surprising result. Moreover, particle tracing algorithms have been shown to be amenable to vectorization and parallelization,[19] opening the door to further improvements in performance.

Unlike traditional techniques, which operate from surface to surface intersection, the proposed ray-tracing technique operates on the volume. Therefore, 3-D volumetric effects, such as attenuation and dispersion of light due to fog or smoke, or the change in colours due to any desired physical quantity, can be easily incorporated. This opens the possibility of improved realism for scene display, as well as greater insight for complex 3-D data sets.

ACKNOWLEDGEMENTS

We are indebted to Larry Gritz, member of our graphics research group, who contributed the camouflage procedural texture used in Figure 3, and who also provided the implementation of the hierarchical bounding volume method in his Renderman-compliant ray tracer.

REFERENCES

1. A. S. Glassner, *An Introduction to Ray Tracing*, Academic Press, London, 1989.
2. T. Whitted, 'An improved illumination model for shaded display', *CACM*, **23**(6), 343–349 (1980).
3. See Reference 1, Chapter 6.
4. A. Fujimoto, T. Tanaka and K. Iwata, 'ARTS: accelerated ray-tracing system', *IEEE Comput. Graphics Appl.*, **6**(4), 16–26 (1986).
5. A. S. Glassner, 'Space subdivision for fast ray tracing', *IEEE Comput. Graphics Appl.*, **4**(10), 15–22 (1984).
6. J. Goldsmith and J. Salmon, 'Automatic creation of object hierarchies for ray tracing', *IEEE Comput. Graphics Appl.*, 7(5), 14–20 (1987).
7. T. Kay and J. Kajiya, 'Ray tracing complex scenes', *Comput. Graphics*, **20**(4) (*Proc. SIGGRAPH'86*), 269–278 (1986).
8. H. Weghorst, G. Hooper and D. Greenberg, 'Improved computational methods for ray tracing', *ACM Trans. Graphics*, **3**(1), 315–324 (1984).
9. A. S. Glassner, 'Spacetime ray tracing for animation', *IEEE Comput. Graphics Appl.*, **8**(2), 60–70 (1988).
10. F. W. Jansen, 'Data structures for ray tracing', in L. R. A. Kessener, F. J. Peters and M. L. P. van Lierop (eds.), *Data Structures for Raster Graphics*, Springer, Berlin, 1989.
11. R. Löhner, P. Parikh and C. Gumbert, 'Some algorithmic problems of plotting codes for unstructured grids', *AIAA-89-1981-CP*, 1989.
12. D. Darmofal and R. Haimes, 'Visual feature identification for 3-D data sets', *Paper AIAA-91-1583-CP, 10th AIAA CFD Conf.*, Hawaii, June 1991.
13. R. Löhner and P. Parikh, 'Three-dimensional grid generation by the advancing front method', *Int. j. numer. methods fluids*, **8**, 1135–1149 (1988).
14. R. Löhner, 'Some useful data structures for the generation of unstructured grids', *Comm. Appl. Numer. Methods*, **4**, 123–135 (1988).
15. O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, Vol. 1, 4th edn, McGraw-Hill, New York, 1988, pp. 135–137.
16. K. Koyomada and T. Nishio, 'Volume visualization of 3D finite element method results', *IBM J. Res. Dev.*, **35**(1/2), 12–25 (1991).
17. D. Mavriplis and A. Jameson, 'Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes', paper presented at the *First World Congress of Computational Mechanics*, Austin, Texas, 22–26 September 1986.
18. M. P. Garrity, 'Raytracing irregular volume data', *Comput. Graphics*, **24**(5), 35–40 (1990).
19. R. Löhner and J. Ambrosiano, 'A vectorized particle tracer for unstructured grids', *J. Comput. Phys.*, **91**(1), 22–31 (1990).