Visualization of molecular dynamics via ray-tracing and animation in a vectorized environment

by G. N. Williams
E. L. Nelson
D. M. Barnett
K. Parmley

Scientific visualization methodologies are being utilized increasingly in attempts to understand physical phenomena via mathematical and simulation model results. Presented herein are the results of a visualization project which produced a vectorized, high-resolution, ray-traced animation of the dynamics of a protein molecule. The resulting animation was recorded on 35-mm film, with a resolution of 1024 × 1024 pixels with 24 color bits. Run-time statistics were also collected which relate image generation parameter ranges and interdependencies.

Introduction

"The purpose of computing is insight, not numbers."
Richard Hamming [1]

"The goal of visualization is to leverage existing scientific methods by providing new scientific insight through visual methods."
B. H. McCormick et al. [1]

The above quotations are truly relevant in today's world of scientific data. Current data acquisition capabilities and supercomputational modeling capabilities are generating data faster than contemporary interpretative methodologies can manage. As a step toward the solution of this problem, visualization capabilities must be significantly enhanced, or the scientific community will—sooner rather than later—be buried by the mountains of numbers generated by its own software.

The scientific visualization project reported in this paper was initiated as an experiment to evaluate the

interpretative capabilities of the data/information interface, and to investigate the computational efficiency of specific imaging algorithms. In particular, the objectives of this research were to utilize visualization methodologies to enhance the understanding of molecular dynamics, and to study the performance of scalar and vector versions of ray-tracing image synthesis software.

The results of the research include a film entitled *A Molecular Voyage*. This animation sequence is composed of ray-traced images which were recorded on 35-mm film. Each image resolution is 1024 × 1024 pixels, with 24 color bits each.

The paper addresses the following topics:

- The enzyme, *porcine pancreatic elastase*.
- Choreography and animation techniques.
- Image generation via ray-tracing.
- Scalar/vector statistics for ray-tracing algorithms.

The enzyme

Enzymes are a class of proteins that regulate the chemical activity in cells. Every enzyme serves as a catalyst for a particular chemical reaction. The molecules that participate in the reaction regulated by the enzyme are called the enzyme's *substrates*. The substrates can bind to an area on the surface of the enzyme called the *active site*. Once the substrates are bound to the active site, chemical bonds can form or break within them. The altered molecules are then released from the enzyme, which remains chemically unchanged.

The specificity of an enzyme for a particular chemical reaction is a result of the specificity of the active site in its ability to bind with particular substrates. The three-dimensional structure of the active site determines which molecules may bind to it as substrates, in the same way that a lock requires a specific key. Thus, an appreciation of the structure of an enzyme is important in understanding how it functions.

The dynamics of an enzyme are as important as its structure. Enzymes undergo conformational changes that alter the charge distribution at the active site, bringing about the chemical changes that occur in the substrates.

Computer animation is a useful tool for studying structure and change, in contexts ranging from chemistry to cosmology. Processes occurring in picoseconds or millennia, across nanometers or light years, can be scaled into a size and time frame appreciable by the human observer.

Working with biochemistry researchers at Texas A&M University, we obtained results of atomic-level molecular simulations that depict the structure and dynamics of the enzyme *porcine pancreatic elastase*. With these data, the two-part film *A Molecular Voyage* was created.

Figure 1

Still photograph of the active site of *porcine pancreatic elastase*, taken from Part 1 of *A Molecular Voyage*.

Part 1 depicts the active site of the enzyme. The camera flies around and through the atoms, which are modeled as semitransparent spheres. The radius of each sphere is determined by the van der Waals radius of the atom it represents. Red spheres represent oxygen atoms, blue spheres represent nitrogen, and black spheres represent carbon. **Figure 1** is a photograph of the active site. **Figure 2** identifies the amino acid residues depicted in Figure 1. Another view of the active site is shown in **Figure 3**.

In Part 2, the whole enzyme is shown, without the hydrogen atoms. In addition, the molecule is animated using the results of the simulation of its conformational changes. **Figure 4** is a photograph of the entire molecule.

Choreography and animation

Animation is the creation of a sequence of images that gives the illusion of motion or "life" by modeling the changes in a set of objects over time. Any property of an object can be animated: typically position and orientation, but also color, reflectance, transparency, texture, shape, etc. In addition, the position, color, intensity, and other properties of a light source can be animated.

The world depicted in *A Molecular Voyage* consists of an elastase molecule, 600 randomly distributed background spheres, the virtual camera (i.e., the view reference point and the normal to the viewplane,

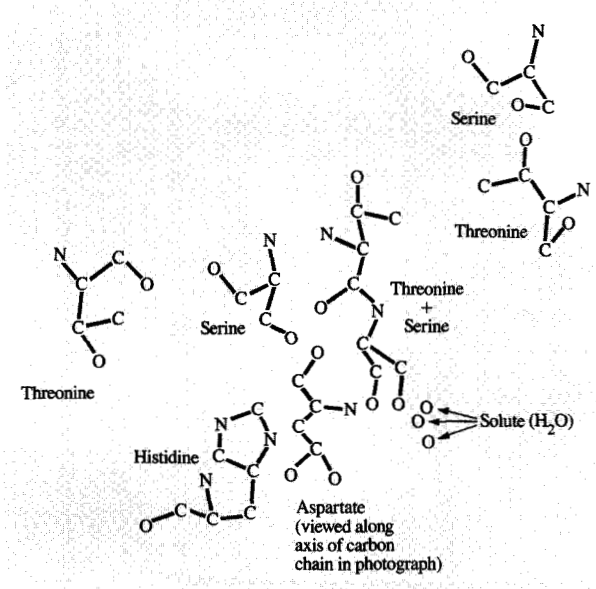

Figure 2

Schematic of the active site of *porcine pancreatic elastase,* as viewed in Figure 1, showing amino acid residues.

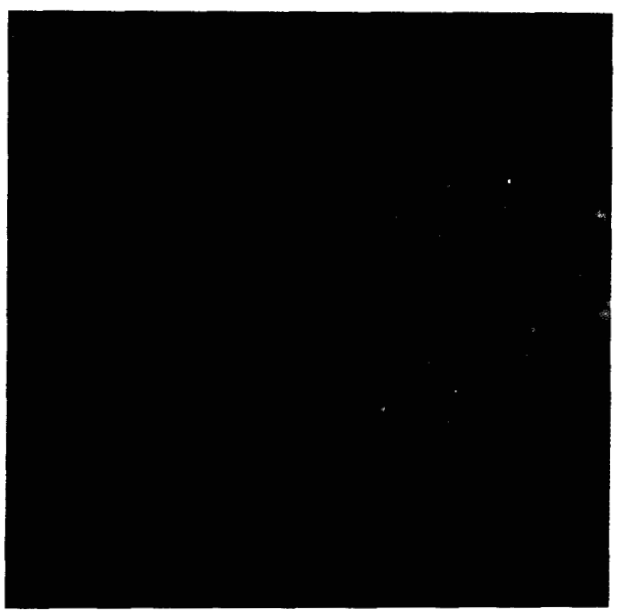

Figure 3

Still photograph of the active site of *porcine pancreatic elastase,* taken from Part 1 of *A Molecular Voyage.*

hereafter referred to as the viewpoint and view direction, respectively), and three light sources: one above and one below the molecule and one that follows the camera, or the "miner's light."

The main animation tasks in making *A Molecular Voyage* were to realistically depict the conformational changes of elastase and to move the camera smoothly around and through the molecule.

Key-framing was used to animate the position and orientation of the camera. The camera position was interpolated by a combination of natural cubic splines and interactive sketching. Camera orientation was specified by selecting one of two functions: (1) tracking the molecule or (2) looking forward along the camera path. Interpolation was accomplished by blending the two functions to ease the transition from one view direction to the other. The techniques presented for animating the camera could be applied to any object, although a different representation of orientation may be appropriate for objects other than cameras.

The camera position is specified in rectangular coordinates. For each key-frame, the animator gives the (x, y, z) coordinates of a control point and a frame number. The interpolation software then calculates the coordinate values for each frame between two key-frames, so that the camera arrives at each control point at the specified time.

Linear interpolation is not satisfactory here, because it leaves discontinuities in the first derivative at each control point, resulting in noticeable jerkiness of motion. Commonly, spline functions are used to guarantee second-order continuity (i.e., continuity of position, velocity, and acceleration) throughout an object's path. Natural cubic splines [2] were used to interpolate the camera position.

Splines are sensitive to changes in the spacing of control points along the abscissa. Since key-frames are not necessarily spaced evenly in time, splining position as a function of time (or frame number) causes a vexing problem: Changing the timing can change the camera path. Changing the frame number of a key-frame so that an object will arrive at the control point earlier or later can cause the object to take a different path to the control point. It is desirable to make the path geometry independent of the timing of the motion, while ensuring second-order continuity of both the path and the object's motion along the path.

A double-interpolation method [3] was used to separate the path from the timing. First, the key-frames are numbered $K = 0, 1, 2, \cdots, n$. The control point positions $P = (x, y, z)$ are then splined as functions of key-frame numbers. Unless key-frames are added or deleted, the shape of the path does not change.

The second interpolation step is to spline the key-frame number K as a function of frame number t. This curve determines the timing of the camera's motion.

Frame numbers can then be changed to speed up or slow down the object without affecting the path taken.

Because $P(K)$, which equals $[x(K), y(K), z(K)]$, is a set of natural cubic splines, the camera's path has second-order continuity. Furthermore, it is easy to show that the composite function $P[K(t)]$, defining the object's motion over time, also has second-order continuity. The result is smooth, seamless animation.

One caution is that the timing curve $K(t)$ should be nondecreasing. A decrease in $K(t)$ would cause the object to move backward along its path.

Camera orientations can be specified by Euler angles (pitch, roll, yaw), quaternions [4, 5], or axis angle and roll angle. It has been demonstrated that quaternions have many advantages for interpolating orientations of objects when the main requirement is that rotations be done smoothly, and the orientation at frames other than key-frames is not particularly important. However, in animating a camera, additional requirements may be imposed. For instance, in *A Molecular Voyage* there are two major modes of camera orientation: tracking the molecule and looking forward along the camera's path.

To implement the tracking and forward-looking modes of camera operation, the axis-angle and roll-angle representation seems most natural. In the tracking mode, the axis (or view vector) is the vector from the camera position to the center of the enzyme. In the forward-looking mode, the view vector is the tangent to the camera path. This tangent vector was approximated by taking the vector from the current position to the next position. In both modes, the default roll angle is zero; i.e., the "up" direction for the camera is in the plane of the view vector and the y-axis.

Rather than specifying an arbitrary orientation at each key-frame, the user selects a mode of camera operation. If a key-frame and its successor both specify tracking mode, the normalized view vector at each point on that segment of the path is

$$V(t) = \frac{O(t) - C(t)}{|O(t) - C(t)|}, \tag{1}$$

where t is the frame number, $O(t)$ is the position of the object of interest, and $C(t)$ is the position of the camera.

If a key-frame and its successor both specify forward-looking mode, the normalized view vector for that segment of the path is

$$V(t) = \frac{C(t+1) - C(t)}{|C(t+1) - C(t)|}, \tag{2}$$

where, again, t is the frame number and $C(t)$ is the camera position.

When the modes at successive key-frames differ, the view vector must be interpolated from one mode to the other. This interpolation is accomplished by a blending function, α. The blending function must be defined on the interval [0, 1]. To interpolate from the vector function $V_1(t)$ at frame t_{start} to the vector function $V_2(t)$ at frame t_{end}, each frame number t must be scaled into the interval [0, 1], namely

$$t_{scale} = \frac{t - t_{start}}{t_{end} - t_{start}}. \tag{3}$$

Then the view vector at frame t, for $t_{start} \le t \le t_{end}$, is

$$V(t) = \alpha(t_{scale}) * V_2(t_{scale}) + [1 - \alpha(t_{scale})] * V_1(t_{scale}). \tag{4}$$

This equation assumes that $|V_1| = |V_2| = 1$.

A good blending function should range smoothly from $\alpha(0) = 0$ to $\alpha(1) = 1$, and should be flat at 0 and 1 to move smoothly from one camera orientation to the next. Good results were obtained by using

$$\alpha(t) = 0.5 + 0.5 \sin[\pi(t - 0.5)], \qquad 0 \le t \le 1. \tag{5}$$

There is one major drawback to the axis-angle representation, when the roll angle is defined so that the camera is always "upright" relative to an arbitrary global axis. There will be a discontinuity in the roll angle when the camera points straight up. If the camera swings near vertical, it will perform a disconcertingly rapid roll to maintain its uprightness. This problem is actually demonstrated after the third pass through the enzyme's active site in Part 1 of the film.

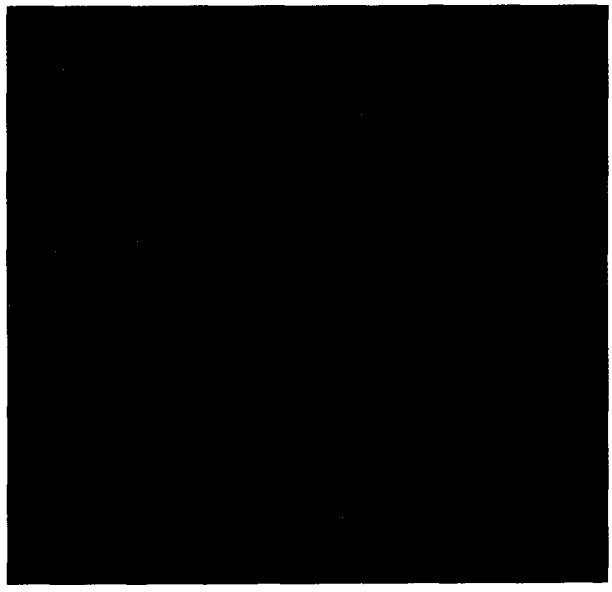

Figure 4

Still photograph of the complete molecule of *porcine pancreatic elastase*, taken from Part 2 of *A Molecular Voyage*.

Figure 5

A photograph showing visualization effects that can be achieved by using ray-tracing techniques.

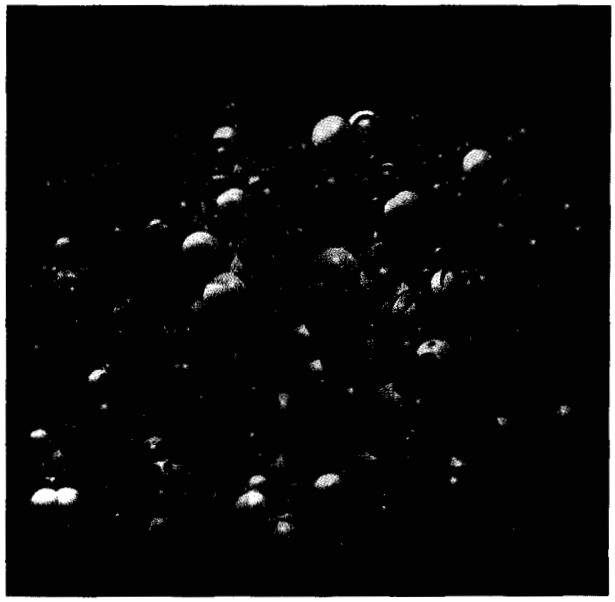

Figure 6

A photograph showing visualization effects that can be achieved by using ray-tracing techniques.

The authors developed software written in the C language for graphics workstations for key-framed computer animation. To assist the user in producing the key-frame data, there are programs for displaying and editing the interpolation curves and for previewing the animation using simplified rendering procedures. The animation can be previewed in real time by displaying only the connections between atoms, or more slowly using z-buffered polyhedral models. Though these programs form the core of a general-purpose animation package, many of the interfaces were designed for the particular requirements of *A Molecular Voyage.*

The animation of elastase in Part 2 was taken directly from the molecular simulation data. The simulation of the molecular dynamics of elastase resulted in sets of coordinates for the atoms of the molecule, taken at intervals of five picoseconds. Every third set of coordinates was used to generate images of the molecule. Shown in sequence at a rate of 12 frames per second, the images become a movie of the molecular dynamics, with one second of movie time corresponding to 180 picoseconds of real time.

Also in Part 2 of the animation sequence, the atoms of the active site were grouped separately from the rest so that they could be faded in first, emphasizing the location of these important atoms.

After a sequence of frames had been created, previewed, and modified until the animation was satisfactory, the data for each frame were passed to the renderer, developed in FORTRAN for the IBM 3090[1] processor. The renderer produced ray-traced images which were stored digitally and later recorded on film.

Image generation

Since the appearance of Whitted's [6] description of the basic procedure, ray-tracing has become exceedingly popular as a technique for image synthesis. Its many advantages over other image synthesis techniques include the ability to simultaneously model the effects of reflections, refractions, and both multiple and variably configured light sources. **Figures 5** and **6** illustrate some of the possible effects.

From the infinite set of rays emitted from a light source, only those rays arriving at the viewpoint are modeled. This is achieved by the "backward" tracing of the light rays from the viewpoint through an image plane to intersection points. At each intersection point, a reflectance ray, a refractance ray, and a shadow ray for each light source are generated. The reflectance and refractance rays are recursively treated identically as an incoming ray. The shadow rays are different in that they do not spawn reflection and refraction rays. **Figure 7** illustrates the basic physical phenomena being modeled.

Many enhancements to ray-tracing have been described [7–10]. Most of these modifications represent

[1] 3090 is a trademark of International Business Machines Corporation.

attempts to produce improved images, achieve additional special effects, or compensate for ray-tracing's few shortcomings (e.g., aliasing). However, one very fundamental problem still exists with all approaches to ray-tracing: cost.

Prohibitive costs often preclude the use of ray-tracing as a visualization tool. Mechanisms for reducing the cost of the technique have been the subject of extensive research [10, 11]. As part of this research, a ray-tracer was developed for the IBM 3090 processor with a goal of maximizing the use of the Vector Facility so that execution time could be minimized.

The ray-tracer for this project is quite basic. Molecular dynamics, the subject of this imagery, is modeled entirely with spheres. While this geometry is simple, the actual ray-tracing problem was nontrivial due to the number of spheres being modeled and the nature of the desired visual effects.

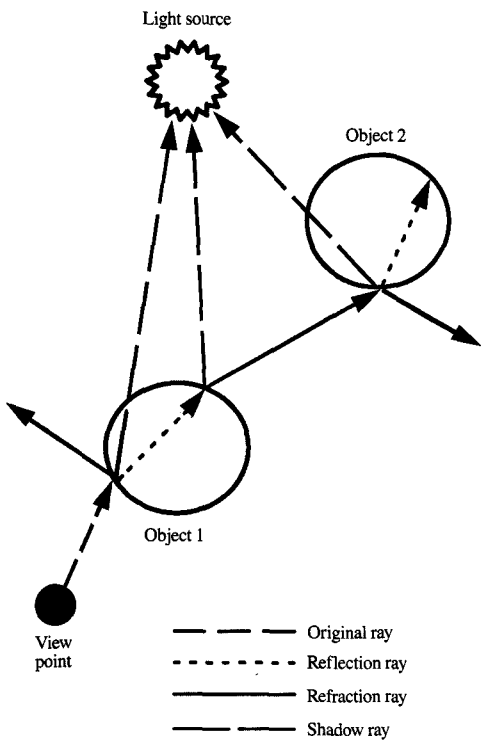

Figure 7

The physical phenomenon being modeled consists of tracing light rays backward from the view point. The ray arriving from any given direction is the sum of all rays that reflected off or refracted through an object. Shadow rays are cast from each intersection point to each light source so that the amount of light arriving at the view point can be determined.

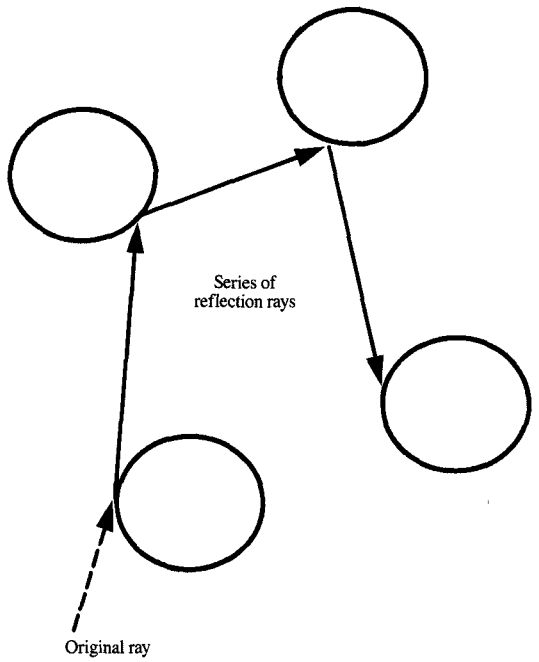

Figure 8

Reflections are calculated as described in Whitted [6]. The number of levels of reflection is arbitrary; however, the existence of more than two or three levels is rare in nature.

The most important special effect desired was transparency so that the viewer would not lose his perspective when viewing the molecule at close range; i.e., landmark atoms and structures would not be obscured by intervening atoms. Also, atom radii were kept proportional to the van der Waals standard so that bonded atoms would intersect. Thus, the bonds could still be visualized, and cylinders, which are generally used to represent bonds, would not be needed.

The ray-tracer follows that of Whitted [6], and comprises two phases. In the first phase, ray–sphere intersections are computed by recursively generating reflective and refractive rays to a specified depth. When refractive rays are generated, the next intersection point is the other side of the object that it just entered. At this point, new reflective and refractive rays are again generated—but it is the refractive ray that exits the object. The reflective ray begins the generation of a series of internal reflection rays that continues to the depth of recursion. **Figures 8**, **9**, and **10** illustrate this section of the ray-tracer.

The second phase consists of casting a shadow ray from each intersection point to each light source. The shading model follows [6, 12, 13]. If a shadow ray

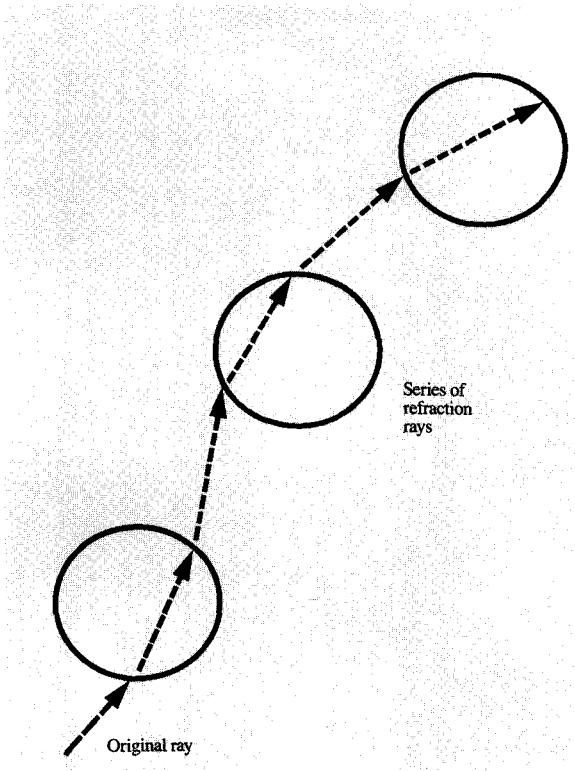

Figure 9

Refractions are calculated as described in Whitted [6]. Each object in the world may have a unique index of refraction. As with reflections, the number of levels of refraction is arbitrary.

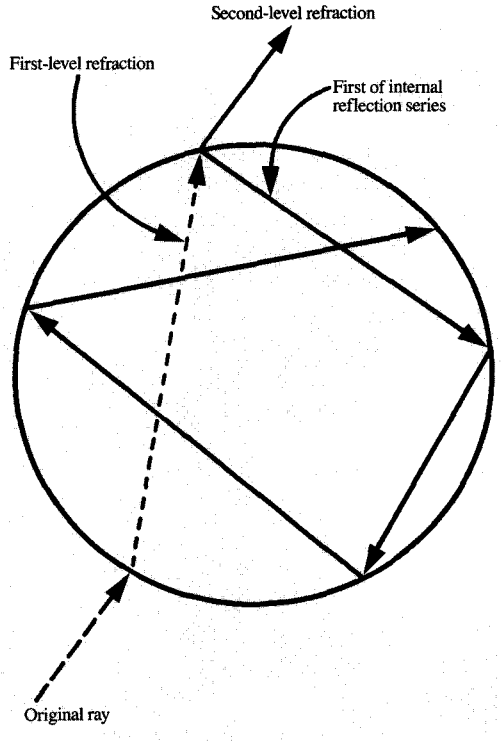

Figure 10

Once a refraction ray is produced, a series of internal reflection rays is begun. The series ends at the maximum level of reflection.

intersects a transparent sphere, the shadow ray is cast again toward the light source, with the intermediate intersection point as a new origin for the shadow ray. The total contribution by the given light source is the product of all transparency factors from all objects intersected by the shadow ray. This recasting of shadow rays continues until either the light source is reached or an intersection with an opaque object occurs. **Figure 11** illustrates this ray-tracing phase.

Many refinements can be made to the basic algorithm stated above, which in themselves would make no visual difference but would reduce cost. For example, the recasting of shadow rays through multiple objects may seem pointless, since the final contribution is likely to be very small. Similarly, the continued generation of internal reflection rays down to the maximum level of recursion could be wasteful. However, with the intent of comparing different approaches to minimize the cost of the ray-tracer, the basic problem, i.e., the generation and management of rays, still exists. All of the refinements that are commonly used and that do significantly reduce computer time can be incorporated in all of the approaches and will reduce cost equally, but do not aid in illustrating the benefits of one approach over another.

Three unique approaches were taken to implement the ray-tracer. The first is designed to execute on a scalar processor and contains no vectorized loops. It employs parallelepiped extents stored in an octree structure. Execution cost is reduced by first traversing the octree structure until an octant contains a single object. At this point, the intersection between the ray and the object is computed. This method delays the expensive ray–object intersection calculations by eliminating regions of space where intersections will not occur.

The two other techniques are intended to execute on a vector processor. The first computes the vector intersections of a single ray with all objects in the scene. The second takes a single object and computes its vector intersections with V rays, where V has been set to a multiple of the hardware vector length; for the IBM 3090

```
   generate a ray from vp to Pixel (x,y)
c
c—>  Generate intersection tree first

   for level i of intersection tree
        for node j of level i
           search octree for closest octant with intersection
           calculate intersection points
           generate reflective and refractive rays
c
c—>  Compute shading components for each intersection point

   for each level i of intersection tree
        for each node j of level i that contains an intersection
           for light source k
           generate ray from intersection node(i,j) to light source k
              while ray intersects objects
                 search octree for closest octant with intersection
                    calculate shading component
              endwhile
```

Algorithm 1

Scalar octree.

```
      generate a ray from vp to Pixel (x,y)
c
c—>  Generate intersection tree first

      for level i of intersection tree
      for node j of level i
         for each object
            calculate intersection points
         for each object that intersects
            find closest
         generate reflective and refractive rays
c
c—>  Compute shading components for each intersection point

      for each level i of intersection tree
         for each node j of level i that contains an intersection
            for light source k
               generate ray from intersection node(i,j) to light
               source k
                  while ray intersects objects
                     for each object
                        calculate shading component
                  endwhile
```

Algorithm 2

Vector on objects.

Table 1 The factors that contribute to the cost of computing and imaging via ray-tracing.

Factor	*Description*
α	Number of objects
β	Number of light sources
δ	Image resolution
γ	Depth of reflection/refraction
Φ	Object surface characteristics, e.g., reflectivity, transparency, and index of refraction
Θ	Scene geometry
ρ	Number of rays generated
λ	Number of pixels composing the image

```
while pixels are left to ray trace
    generate MAX_Q_SIZE - Q_SIZE rays from vp to pixels

    while V rays are available for intersection
        for object i
            intersect N rays with object i
            keep closest intersection points

        for each shadow ray
            if ray intersects then
                generate new origin
            else
                compute shading component

        for each non-shadow ray that intersects an object
            generate reflective, refractive, and
            shadow rays
    endwhile
endwhile
```

Algorithm 3

Vector on rays.

processor this is 64 for double precision and 128 for single precision.

The algorithms for each approach are given below. All versions of the software run on an IBM 3090, under MVS/XA,[2] using Version 2.3 of the vectorizing compiler.

Scalar/vector statistics for ray-tracing algorithms

In comparing the run-time characteristics of each approach, it is necessary to minimize the individual variances of each factor that contributes to the total cost of execution.

The calculation of a single metric as a function of the factors described in **Table 1** is very difficult. For example, by varying the relative positions of the objects and keeping all other factors constant, run-time execution can vary by orders of magnitude. This is explained by the following. First, the number of rays generated can change, since the number of pixels having rays which

[2] MVS/XA is a trademark of International Business Machines Corporation.

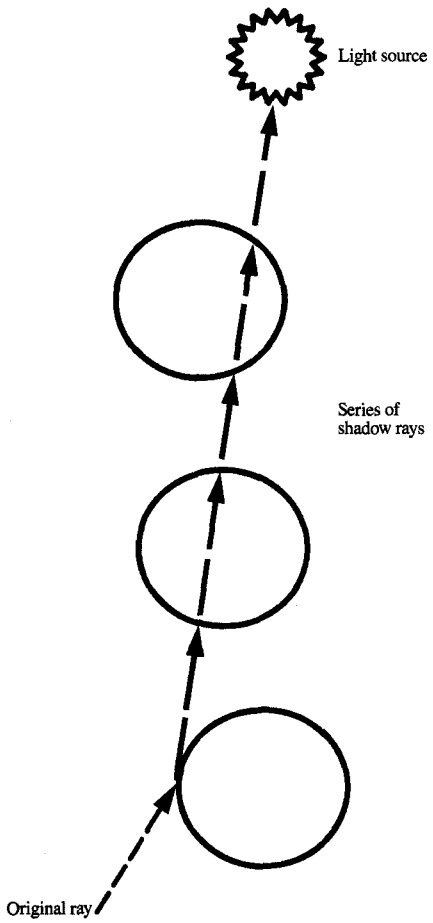

Figure 11

At each intersection point, a ray is cast to each light source. The transparency value for each object lying between the light source and intersection point is multiplied to compute the magnitude of light arriving at the intersection point.

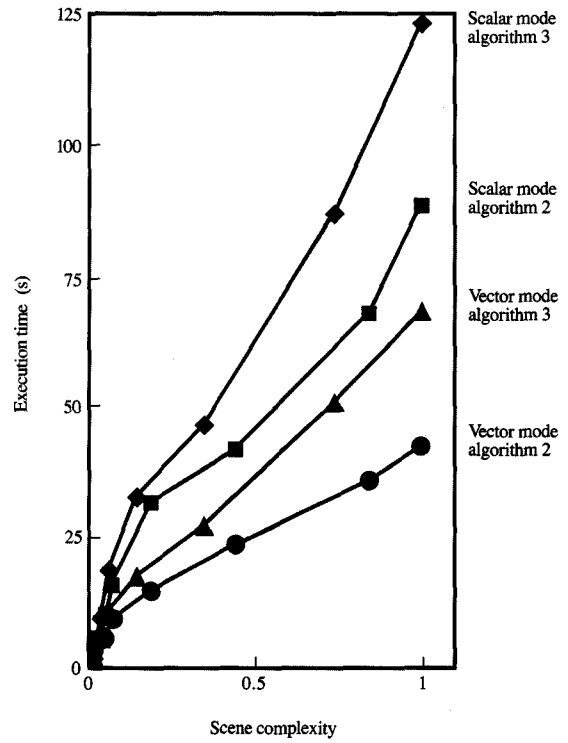

Figure 12

Scalar and vector execution times as a function of scene complexity for the vector on objects algorithm (2) and the vector on rays algorithm (3).

intersect objects can vary. Second, the relative position of the objects also affects the quantity of reflective and refractive rays that are generated. Thus, there is no simple direct method—short of ray-tracing—that will compute the effect of the placement of objects on run-time execution.

However, in demonstrating the differences among the three approaches, an empirical study was made to determine a measure of the complexity of the scene definition input. The effect on cost by variances in scene geometry (Θ) is approximated by Equation (6). The metric *scene complexity* (C) is computed using Equation (7):

$$\prod(\delta, \gamma, \rho, \lambda) \approx \Theta, \tag{6}$$

$$\prod(\Theta, \alpha, \beta) \rightarrow C. \tag{7}$$

The *scene complexity* of an input file comprises both user-defined characteristics and run-time statistics. This metric was computed for 12 separate cases for all three algorithms. In this analysis, β, δ, γ were held constant; surface characteristics and scene geometry were uniformly distributed; and the number of objects was varied from 8 to 256. **Figure 12** illustrates the relationship between execution time and scene complexity.

As shown in **Figure 13**, vectorization provides little performance enhancement for very simple scenes. However, for more complex scenes, the use of the IBM 3090 Vector Facility cuts the execution time by 50 percent.

Conclusion

Visualization methodologies such as computer animation and ray-tracing have significantly contributed to the

understanding of the structure and change of scientific processes.

A software system using computer animation and ray-tracing visualization methodologies has been developed. Working with biochemists at Texas A&M University, we used this system to animate the structure and dynamics of the enzyme *porcine pancreatic elastase.*

Prohibitive costs often preclude the use of ray-tracing as a visualization tool by researchers. Three approaches to ray-tracing were implemented to investigate vector processing performance. The results indicate that vectorizing on objects is a more cost-effective approach to ray-tracing than either vectorizing on rays or using an optimized scalar approach.

Acknowledgments

The authors wish to express their gratitude to the IBM Corporation, which sponsored this research project as a part of the Research Support Program at the IBM Palo Alto Scientific Center. The Program Coordinator was Barbara Straka. The authors also wish to thank Franz Krager, Music Director and Conductor of the Brazos Valley Symphony, for his choice of Gustav Holst's "The Planets" as background music for the video.

References

1. B. H. McCormick, T. De Fanti, and M. D. Brown, "Visualization in Scientific Computing," *ACM Computer Graph.* **21,** 3 (November 1987).
2. John G. Herriot, "Procedures for Natural Spline Interpolation," *Commun. ACM* **16,** 763–768 (December 1973).
3. Scott N. Steketee and Norman I. Badler, "Parametric Key-Frame Interpolation Incorporating Kinetic Adjustment and Phrasing Control," *ACM Computer Graph.* **19,** 255–262 (July 1985).
4. Ken Shoemake, "Animating Rotation with Quaternion Curves," *ACM Computer Graph.* **19,** 245–254 (July 1985).
5. Ken Shoemake, "Quaternion Calculus and Fast Animation," SIGGRAPH 1987 Tutorial *Computer Animation: 3-D Motion Specification and Control,* ACM Press, New York, 1987.
6. Turner Whitted, "An Improved Illumination Model for Shaded Display," *Commun. ACM* **23,** 343–349 (June 1980).
7. Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear, "A Ray Tracing Solution for Diffuse Interreflection," *ACM Computer Graph.* **22,** 85–92 (August 1988).
8. John Amanatides, "Ray Tracing with Cones," *ACM Computer Graph.* **18,** 129–135 (July 1984).
9. Robert L. Cook, Thomas Porter, and Loren Christopher, "Distributed Ray Tracing," *ACM Computer Graph.* **18,** 137–145 (July 1984).
10. Andrew S. Glassner, *An Introduction to Ray Tracing,* Academic Press Ltd., London, 1989.
11. A. S. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Comput. Graph. & Appl.* **4,** 15–22 (October 1984).
12. Bui Tuong Phong, "Illumination for Computer Generated Pictures," *Commun. ACM* **18,** 311–317 (June 1975).
13. Henri Gouraud, "Continuous Shading of Curved Surfaces," *IEEE Trans. Computers* **C-20,** 623–629 (June 1971).

Received November 16, 1989; accepted for publication July 18, 1990

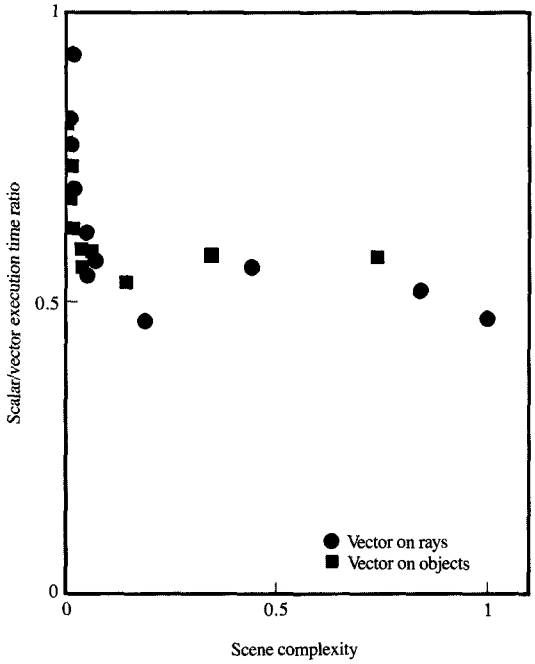

Figure 13

Use of the IBM 3090 Vector Facility halves the execution time for both of the vectorizable algorithms.

Glen N. Williams *Department of Computer Science. Texas A&M University, College Station, Texas 77843.* Dr. Williams, Professor of Computer Science, holds a Ph.D., M.E., and B.S. in civil engineering from Texas A&M University (1965, 1961, and 1960). His primary research interests are in the areas of computer graphics, numerical methods, and scientific/engineering computer applications. He has served as advisor and graduate advisory committee chairman for numerous doctoral and Master's students. Dr. Williams has been the principal investigator for more than ten research grants, including a grant by the IBM Palo Alto Scientific Center to support the production of the computer-animated film *A Molecular Voyage,* which depicts the molecular dynamics of an enzyme. He is currently the project director for the Intelligent Control Systems for Autonomous Underwater Vehicles Project, a multi-year hardware/software development effort funded by the U.S. Department of the Navy. In addition to his extensive research and consulting activities, Dr. Williams has chaired numerous sessions for the Oceans Conference and the IEEE Ocean Engineering Society (OES), including the technical progam for the IEEE/OES 1990 Symposium on Autonomous Underwater Vehicle Technology. He has also served on the Board of Directors for the Offshore Technology Conference since 1985.

Eric L. Nelson *Department of Computer Science, Texas A&M University, College Station, Texas 77843.* Mr. Nelson is a Research Associate for the Intelligent Control Systems for Autonomous Underwater Vehicles Project in the Computer Science Department

at Texas A&M University. He received B.S. and M.C.S. degrees in computer science from Texas A&M University in 1981 and 1985, respectively, and is currently pursuing a Ph.D in computer science. He was a Research Assistant in the Department of Plant Pathology at Texas A&M University from 1984 to 1986. Mr Nelson was a Research Assistant from 1986 to 1987 and a Lecturer from 1987 to 1988 in the Department of Computer Science at Texas A&M University. He has been involved in several research projects, including Scientific Visualization of Molecular Dynamics (1987–1989), sponsored by the IBM Research Support Program, and Parallel Methods for Ray-tracing Bi-cubic Patches on a Cray (1988–1989), sponsored by General Dynamics Corporation. Mr. Nelson is a member of the Association for Computing Machinery and the Institute of Electrical and Electronics Engineers.

David M. Barnett *Department of Computer Science, Texas A&M University, College Station, Texas 77843.* Mr. Barnett is a Research Associate for the Intelligent Control Systems for Autonomous Underwater Vehicles Project in the Computer Science Department at Texas A&M University. He received a B.S. in computer science from Texas A&M University in 1983 and is currently pursuing a Master of Computer Science degree. He was a Research Assistant in the Department of Computer Science at Texas A&M University from 1986 to 1989. Mr. Barnett was a computer programmer for the Academic Programs Office in the College of Engineering at Texas A&M University from 1987 to 1989. He was involved in the research project "Scientific Visualization of Molecular Dynamics" (1987–1989), sponsored by the IBM Research Support Program.

Kelly Parmley *Department of Computer Science, Texas A&M University, College Station, Texas 77843.* Ms. Parmley is a Graduate Assistant (Teaching) in the Computer Science Department at Texas A&M University. She was a Texas Engineering Experiment Station Fellow in 1989 and received a Bachelor's degree in computer science from Texas A&M University in 1988. Ms. Parmley is currently pursuing a Master of Computer Science degree. From 1985 to 1987, she was Systems Manager at Sterling E. Evans Library, Texas A&M University. She was a student technician in the Department of Computer Science at Texas A&M University from 1987 to 1988 and a Research Assistant in the same department in 1989. Ms. Parmley is a member of the Association for Computing Machinery.