RAY TRACING FOR UNDERGRADUATES

Christiaan Paul Gribble
Department of Computer Science
Grove City College

Introduction

The computer graphics research community has recently renewed its interest in ray tracing, an image synthesis algorithm that simulates the interaction of light with an environment to generate highly realistic images (Figure 1). Recent hardware trends and algorithmic developments make the technique competitive with raster-based algorithms, and some suggest that ray tracing will begin to dominate interactive rendering in coming years.

At Grove City College, we have mapped the contents of common graduate-level courses in ray tracing to an undergraduate audience. Students design and implement a full-featured ray tracing system in a semester-long course that focuses on:

- the essential physics and mathematics,
- software architecture and the impact of design decisions,
- writing efficient object-oriented code, and
- basic algorithm analysis.

The course also affords an opportunity to introduce students to the relevant computer science literature, both seminal works and recent innovations, throughout the semester.

In this paper, we provide a brief overview of the visibility problem and two competing algorithms that are commonly used to solve the problem, we detail the course topics and methodology we have used, and we describe our experience in a pilot course with a small group of undergraduate students.

Background

Despite the current trends in computer graphics research, a thorough study of ray tracing remains absent from most undergraduate computer science programs. Typical computer graphics courses focus on raster-based techniques and the corresponding application programming interfaces (APIs) such as OpenGL or DirectX. The availability of commodity graphics hardware makes raster-based techniques widely accessible, and these techniques serve as the basis of many interactive rendering applications, including computer and console gaming. These characteristics make the study of raster-based techniques and their APIs popular with undergraduate students.

Figure 1: Image synthesis using ray tracing. The ray tracing algorithm supports complex visual effects that are not easily implemented with raster-based techniques, including depth-of-field, glossy and specular reflections, refraction, soft shadows, and diffuse interreflection.

In contrast, ray tracing is considered by many to be an historic artifact—perhaps relevant as a research topic in the 1980s, but largely uninteresting today. This perception may stem from the traditional use of ray tracing as an offline or batch technique: until recently, generating a single image required either several minutes of computation on desktop systems or extremely expensive, highly parallel systems found only in large research laboratories. Others believe that the physical and mathematical underpinnings put the algorithm beyond the reach of undergraduate students, relegating the study of the algorithm to graduate-level courses.

However, we believe that ray tracing is an ideal vehicle through which to reinforce and apply fundamental concepts in computer science. To develop an understanding of the algorithm, students must integrate and extend knowledge in computer graphics, software engineering, mathematics, physics, and even human perception. These characteristics, together with hardware trends and recent research results, make ray tracing both relevant and accessible to undergraduate students.

Ray tracing versus rasterization

A fundamental problem in computer graphics is the *visibility problem*: Given a set of three-dimensional (3D) objects and a viewing specification, determine which lines or surfaces are visible from that view point. Many algorithms that solve the visibility problem are available in the literature, but currently the most prevalent are the z-buffer algorithm and ray tracing.

Current graphics hardware is based on the z-buffer algorithm[1] which consists of a loop over the objects in a scene:

for each object N **do**
  for each pixel P through which N might be visible **do**
    compute color c_{new} and depth z_{new}
    if $z_{new} < z_{pixel}$ **then**
      $c_{pixel} = c_{new}$
      $z_{pixel} = z_{new}$

The z-buffer algorithm projects an object toward the screen and writes to any pixels covered by that object both the distance from the object to the view point (the *depth* or *z* value) and the color information, but only if the new *z* value is less than the current *z* value associated with a given pixel (Figure 2).

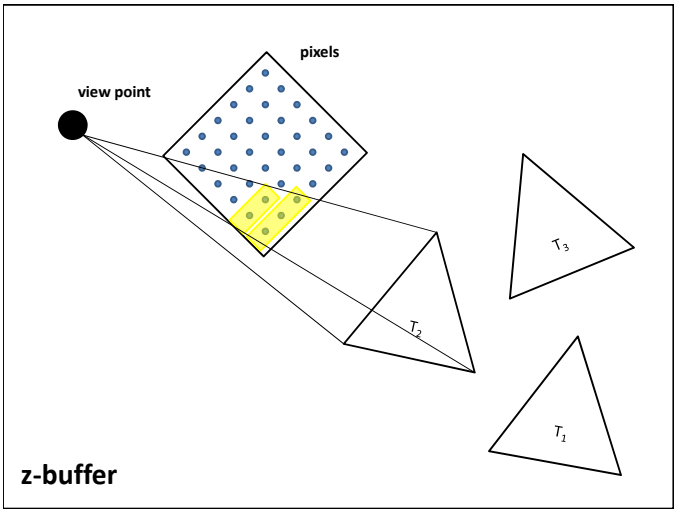

Figure 2: The z-buffer algorithm. Scene geometry is projected toward the screen, and the z-buffer is used to resolve visible surfaces based on the distance between an object and the view point. In this case, triangle T_2 becomes the object visible through the highlighted pixels.

In contrast, the basic ray tracing algorithm[2] consists of a loop over all of the pixels in an image:

for each pixel P **do**
  find nearest object visible through P

A 3D line query is used to find the nearest object in the parametric space of the ray (Figure 3).

Ray tracing boasts several key advantages over the z-buffer algorithm. First, for preprocessed models, ray tracing is sub-linear in the number of objects, *N*; thus, for some sufficiently large value of *N*, ray tracing will always be faster than the z-buffer algorithm, which is linear in *N*.[3] Second, the computational kernel of the algorithm performs a 3D line query, and that same operation can be reused to generate global illumination effects that also depend on a 3D line query; these effects include shadows, reflection, and refraction.[2] Third, ray tracing is highly parallel and has been demonstrated to have over 91% efficiency on 512 processors.[4] This characteristic, combined with the advent of multicore microprocessors and algorithmic developments over the past five years, has made ray tracing an attractive alternative for interactive rendering. Interactive ray tracing algorithms typically trade software complexity for performance, but impose only modest hardware requirements to obtain interactive frame rates for complex scenes.[5]

With the introduction of interactive ray tracing, computer graphics courses must evolve to reflect the current trends in computer graphics research. Shirley et al.[6] have identified several areas in the design of computer graphics courses on which interactive ray tracing will likely have an impact:

- **Illumination models.** Recursive ray tracing integrates visibility and illumination computations based on a single computational kernel, the 3D line query. Many common physical effects that are implemented with specialized tricks or hacks in raster-based pipelines will evolve into their natural illumination computations.

- **Perspective transformation and homogeneous coordinates.** Ray tracing simulates perspective naturally, so the mathematic models necessary to simulate foreshortening will become less important. This process simplifies the traditional transformation pipeline and eliminates the need for perspective transformation.

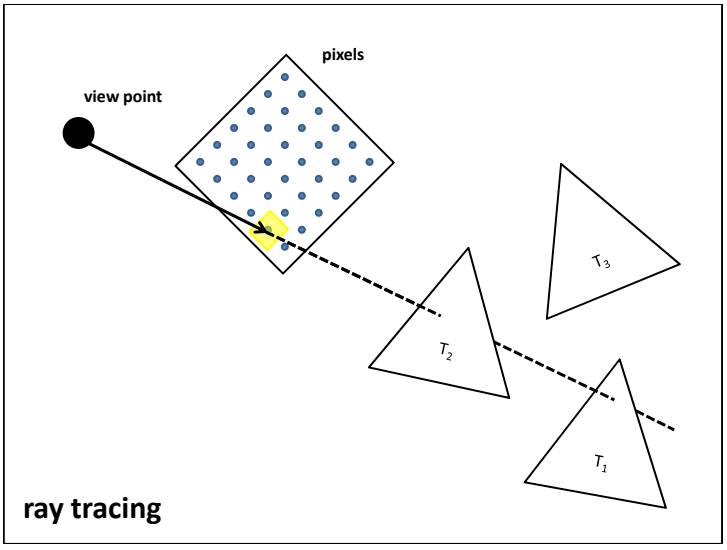

Figure 3: The ray tracing algorithm. Rays (3D half-lines) originating at the view point are traced into the scene, and the object closest to the view point is found. Here, the 3D line query determines that triangle T_2 is visible through the highlighted pixel.

- **Higher-order surfaces.** Ray tracing computes visibility by mathematically intersecting a line and an object, and it is often straightforward to ray trace mathematically defined, higher-order surfaces without tessellation. Current emphasis on object models composed of triangle meshes will begin to shift to models based on the native representation of higher-order surfaces.

- **Volumetric effects.** Ray tracing supports participating media such as aerial fog or smoke using semitransparent primitives with dedicated illumination models. Here again, the need for specialized tricks or other hacks will be greatly reduced.

While we agree that the advent of interactive ray tracing will likely have a significant impact on the structure of traditional computer graphics courses, we believe that the study of the ray tracing algorithm is an ideal vehicle through which to reinforce and apply fundamental concepts in undergraduate computer science programs. We have thus mapped the contents of common graduate-level courses in ray tracing to an undergraduate audience.

Course topics and pedagogy

Our undergraduate ray tracing course, currently entitled "Image Synthesis using Ray Tracing", is intended to provide the student with an understanding of the details of the algorithm through the design and implementation of a full-featured ray tracing program. Extensive programming in C++ facilitates knowledge development in the core areas related to image synthesis using ray tracing: (1) the physical and mathematical underpinnings of the algorithm, and (2) efficient system design and implementation. A description from the course catalog is given in Figure 4.

Course outcomes

Upon completion of the course, it is expected that students will be able to:

- Design and implement the basic components of the ray tracing algorithm using object-oriented programming and a variety of development tools.

- Identify, describe, and analyze the basic components of the ray tracing algorithm, including ray/surface intersection methods, lighting and shading models, and advanced rendering effects.

- Describe and derive the physical and mathematical basis for the ray tracing algorithm, including ray/surface intersection methods and the geometric model of light transport.

> **Image Synthesis using Ray Tracing.** An introductory course in the design and implementation of the ray tracing algorithm for computer graphics. Course topics include: intersection methods for basic and advanced three-dimensional objects; local and global shading models; acceleration structures for ray tracing; and introduction to surface, solid, and volume texturing. Prerequisites: *Introduction to Computer Graphics*.

Figure 4: Details of the ray tracing course from the course catalog. Our current ray tracing course is intended for advanced (junior- and senior-level) computer science students.

- Locate, analyze, and present articles describing the theoretical and practical research results concerning the ray tracing algorithm.

By the start of their junior year, our students have completed two semesters of physics (statics, dynamics, and kinematics), three courses in calculus, a course in discrete mathematics for computer science, and a semester of mathematical methods for engineers (probability/statistics and linear algebra). These courses provide the background required to understand the physical and mathematical underpinnings of the ray tracing algorithm.

Moreover, our students have completed a three-semester programming core, which provides in-depth experience with object-oriented programming in C++. The students have also completed a course in data structures and algorithms, and are comfortable with designing, analyzing, and coding complex data structures and high-performance algorithms. Finally, the students have completed courses in computer architecture/organization and operating systems, and are familiar with process/thread management, memory management, and file input/output. These courses provide the background required for the design and implementation of an efficient ray tracing system.

Teaching and assessment methodology

The course consists of traditional classroom lectures and discussion sections that focus on questions arising from the programming assignments. During the discussion sections, implementation details and code examples are provided, as is debugging assistance for common problems. Chapters in the text, research articles, and student presentations also facilitate the discussion.

Most computer graphics textbooks include at least one chapter on the ray tracing algorithm. However, these texts are not sufficient to support a semester-long study of the algorithm and its applications. We currently are aware of two textbooks dedicated entirely to ray tracing that are suitable for an undergraduate audience:

- Peter Shirley and R. Keith Morely. Realistic Ray Tracing, 2nd edition. AK Peters, ISBN 1-56881-198-5, 2003.

This text takes a nuts-and-bolts approach to implementing a ray tracing system. Each chapter tackles a relatively self-contained piece of the overall system, and code examples are provided at the end of each.

- Kevin Suffren. Ray Tracing from the Ground Up. AK Peters, ISBN 978-1-56881-272-4, 2007.

This text offers a more comprehensive view of ray tracing theory and practice. Topics range from algorithm basics to advanced techniques that would be explored in the context of a second course on ray tracing. A wide range of code examples are also included throughout the text.

The Suffern text was not yet available for our initial offering, so we opted for the Shirley text. However, in future versions of the course, we plan to adopt the Suffern book as the main text and suggest the Shirley text as a highly recommended (but optional) resource.

Topics from the text are supplemented by readings from the ray tracing research literature. A partial bibliography of key ray tracing papers that are read and discussed throughout the semester includes:

- T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343-349, 1980.
- J. Kajiya. The rendering equation. In *Proceedings of SIGGRAPH '86*, pp. 143-150, 1986.
- S. Parker, W. Martin, P.-P. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. In *Symposium on Interactive 3D Graphics*, pp. 119-126, 1999.

- I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153-164, 2001.

These articles detail many of the theoretical and practical aspects of ray tracing, and are appropriate for advanced undergraduate students with some guidance. Students are expected to read each article and actively participate in the corresponding discussions. Moreover, students are periodically required to submit highlighted versions of some articles, as well as a short (one to two pages) summary of the authors' contributions. In these cases, guiding questions are provided to help focus the students' reading comprehension.

Graded work products for the course consist of weekly programming assignments in which each student builds his or her own ray tracer from scratch. Students create web pages to display their results, and the contents of these pages are submitted for grading. Programming assignments are typically composed of the following elements:

- **Required images:** Students must duplicate one or more required images that demonstrate the required functionality. Scene descriptions in a common format are provided for the required images.
- **Code listing:** Students are required to provide a link to the project source code on their web pages. Students are not graded on the quality of the code or comments, only on its presence.
- **Creative images:** Students are typically required to render one or more images that demonstrate the features of their ray tracer. Again, students are not graded on the creativity displayed in these images, only on their presence.
- **Design decisions:** Students are required to describe any design decisions that were made while implementing the assignment. Students are not graded on their choices, only on the completeness of the description and justifications.

The required images represent the vast majority of the credit for any particular assignment (typically 70-80%), whereas the remaining elements are intended to ensure academic integrity and reasonable thought processes.

The majority of student programming effort occurs early in the semester, as they bootstrap the necessary infrastructure. For example, the first three programming assignments include:

1. **Basic infrastructure.** Students construct the basic foundation for their ray tracer. C++ classes for `Point`, `Vector`, `Ray`, `Color`, and `Image` objects are designed and implemented. Students must provide all of the legal operators for a given class, including arithmetic operators, dot product, cross product, length, and normalization routines.

   In this assignment, students also implement a simple `Sphere` class for testing their infrastructure. The sphere stores its center and radius, and has a single member function `intersects` that indicates whether or not a ray hits the sphere. Throw-away code to generate rays and test the various classes is provided.

2. **Ray tracing architecture.** Students continue to develop their ray tracing infrastructure throughout the course of the semester. Students must implement C++ base classes for the `Object`, `Camera`, `Light`, `Material`, and `Background` objects, as well as an intermediate `Primitive` class (optional).

   In this assignment, students also implement derived classes for sphere and plane primitives, a pinhole camera, point light sources, simple Lambertian surfaces, and a constant-colored background.

3. **Triangle meshes.** Students add simple triangle meshes to their ray tracers by

completing the member function definitions for a `Mesh` class that is provided.

Later assignments typically consist of minor modifications to the existing infrastructure or the addition of new functionality that requires very few changes to the code developed in earlier assignments. Examples include the introduction of acceleration structures such as bounding volume hierarchies or kd-trees, as well as additional material models. Students are encouraged to follow the basic software architecture presented in class, but they are free to experiment with alternative designs.

We have developed a self-contained reference implementation that provides efficient solutions to each of the programming assignments in the course. The reference implementation provides students with a target for ray tracing performance, and is available to instructors at other institutions interested in developing their own ray tracing courses (upon request).

All students at Grove City College receive the most current Hewlett-Packard tablet computers when they enter as freshmen. The current freshmen model, the HP Compaq Tablet PC tc4400 mobile computer, boasts a 2.16Ghz Intel Core2 Duo Processor and 2GB of RAM. These systems are more than sufficient for generating the required images within a reasonable period of time, typically just tens of seconds. Computer laboratory facilities with reasonably up-to-date equipment would also suffice.

Experience

We have offered the ray tracing course as a special topics course, which is one way new courses are piloted at Grove City College. Four senior students enrolled in the course for credit, and three sophomore students attended the lectures and discussion sections but did not receive academic credit.

Although special topics courses are not evaluated in the same manner as standard semester-length courses, feedback directly from the students indicates that the initial offering was well-received. At the conclusion of the course, one student writes:

> Thanks for this study Dr. Gribble, it was very interesting and now I have a cool program to show off. It's the biggest system I've made since being at Grove City College so I now feel like all my time coding and in math classes has paid off, I think that's been the coolest thing about this ray tracing stuff.

Additional feedback from the senior students indicates that:

- Students gained an appreciation of the extensive mathematics background required by our computer science program. Ray tracing puts many concepts in calculus, linear algebra, and numerical analysis to practical use in a context that has been described as “cool” by the students.
- Students gained an appreciation of the natural science courses required by our computer science program, particularly physics. Often, students reported that the physical concepts related to a particular topic (for example, ideal Lambertian surfaces) made much more sense after implementing the concept in the context of the ray tracer.
- Students gained an appreciation for the difficulties involved in developing and debugging a complex software system. The ray tracer was, for many students, the first experience with a non-trivial code base that had to be designed and written from scratch.
- Students spent a significant amount of time on the programming assignments (presumably relative to their other course work), but the results were satisfying. We did not receive any complaints about the level of effort required by, nor the time spent on, the programming assignments.

Student performance in the initial offering was exceptionally high. This result is due, at least in part, to a biased sampling: the four seniors taking the course for credit initiated the effort,

and the sophomores that attended regularly were invited by the instructor. We hope to see similarly excellent performance in future versions of the course, but a wider range of students will undoubtedly bring mixed performance.

Summary

We believe that a detailed study of the ray tracing algorithm provides an opportunity to reinforce and apply fundamental concepts in computer science with undergraduate students. We have mapped the contents of common graduate-level courses in ray tracing to an undergraduate audience. Students design and implement a full-featured ray tracing system in a semester-long course that focuses on the essential physics and mathematics, software architecture and the impact of design decisions, writing efficient object-oriented code, and basic algorithm analysis. The course also affords an opportunity to introduce students to the relevant computer science literature throughout the semester.

Ray tracing provides opportunities for more advanced study and undergraduate research. We are currently planning a second course in ray tracing focused on topics such as sampling theory and the aliasing problem; advanced global illumination effects such as soft shadows, depth-of-field, and motion blur; and Monte Carlo methods for simulating the physical transport of light throughout an environment.

As a small comprehensive college, we hope to demonstrate that ray tracing is accessible to undergraduate students at a broad range of colleges and universities, both large and small. We also hope that our experiences are both insightful and useful to other instructors interested in developing their own ray tracing courses.

References

1. E. Catmull. A subdivision algorithm for computer display of curved surfaces. PhD dissertation, University of Utah, 1974.

2. T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343-349, 1980.

3. J. Cleary, B. Wyvill, G. Birtwistle, and R. Vatti. A Parallel Ray Tracing Computer. In *Proceedings of the Association of Simulat Users Conference*, pp. 77-80, 1983.

4. S. Parker, W. Martin, P.-P. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. In *Symposium on Interactive 3D Graphics*, pp. 119-126, 1999.

5. I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153-164, 2001.

6. P. Shirley, K. Sung, E. Brunvand, A. Davis, S. Parker, and S. Boulos. Rethinking Graphics and Gaming Courses Because of Fast Ray Tracing. In *ACM SIGGRAPH 2007 Educators Program*, article number 15, 2007.

Biographical Information

Christiaan P. Gribble is an Assistant Professor in the Department of Computer Science at Grove City College. His research focuses on global illumination algorithms, interactive and realistic rendering, scientific visualization, and high-performance computing. Gribble has served as a post-doctoral research fellow and research assistant for the Scientific Computing and Imaging (SCI) Institute at the University of Utah, and as a research assistant at the Pittsburgh Supercomputing Center. In 2005, he received the Graduate Research Fellowship from the University of Utah. Gribble received the BS degree in mathematics from Grove City College in 2000, the MS degree in information networking from Carnegie Mellon University in 2002, and the PhD degree in computer science from the University of Utah in 2006.