RAYTRACING AS A TOOL FOR LEARNING COMPUTER GRAPHICS

Ching-Kuang Shene

Abstract *— The commonly used programming approach in teaching computer graphics requires students to learn a lot before they can generate basic and not-so-realistic images. As a result, students may easily be lost in the jungle of programming primitives, and their high expectation fades away quickly. Moreover, the API based programming approach does not support global illumination models. To address these problems, a new approach that combines ray tracing and programming has been used in a junior level elective course* Intro. to Computing with Geometry *with great success. With ray tracing, we are able to cover the camera metaphor, basic shapes, geometric modeling, coefficients of an illumination model, light sources, textures, surface tessellation, smooth and non-smooth triangles, and algebraic surfaces. A student can learn all the basics and generate beautiful and realistic looking images easily and quickly. This paper details our approach and presents our course materials, exercises, student work and evaluation.*

Keywords —computer graphics, ray tracing, global illumination models

MOTIVATION

The *programming approach* is the most popular approach in teaching introduction to computer graphics courses. However, it does have some serious drawbacks [6]. **First**, students usually do not know if the created image is correct. C. A. R. Hoare once said: “You can’t teach beginning programmers top-down design because they don’t know which way is up.” Likewise, it is difficult to teach graphics programming to beginners because they do not know what the anticipated effect should be. As a result, we need an easy way for students to recognize the effect of each graphics parameter before they start to program. GraphicsMentor is a tool designed for this purpose [2]. **Second**, the programming approach depends on graphics APIs which are based on local illumination models. This means that student programs cannot do shadow, reflection and refraction. **Third**, the design and modeling component is unlikely to be touched upon because students may dedicate too much time on programming and because a typical graphics API has a limited modeling capability.

Five years ago, when we started teaching a junior elective course Introduction to Computing with Geometry [7], we found that 30% to 50% enrolled students do not have graphics background. Occasionally, students from non-CS departments also take this course to learn skills for their projects. Thus, helping students who do not have graphics background becomes part of the course. After an extensive study of various approaches, we believe that the best way of introducing graphics fundamentals to those who do not have graphics background is the use of ray tracing. In this course, we spend less than 5% of all lecture hours on ray tracing. However, we successfully cover all the fundamentals, design and modeling, and a number of advanced concepts. Furthermore, almost all students are eager to learn more and do more as demonstrated by their creative work. Consequently, we believe our approach is a very successful one. In the following, we will briefly discuss the course and course materials. At the end of this paper, we present evaluation, work in progress, and conclusions.

Ching-Kuang Shene, Department of Computer Science, Michigan Technological University, Houghton, MI 49931, shene@mtu.edu

COURSE OVERVIEW

Introduction to Computing with Geometry is an elective course for juniors. The design goal of this course is to provide students with important and useful skills in handling geometry related problems. Thus, this course covers commonly used topics such as parametric and implicit curves and surfaces, Bézier, B-spline and NURBS curves and surfaces, cross-sectional design, and curve and surface interpolation and approximation. Even though most of these topics require students to write programs using OpenGL in a scaled down environment of DesignMentor ([11], [12] and [13]), we use ray tracing through out the whole semester for students to do more design work and experiment various concepts that are only possible using a global illumination model (*e.g.*, show, reflection and refraction). Of course, the most important advantage of using ray tracing is for students to create flashy images fast, which, in turn, inspire them to do more in ray tracing.

There are many ray tracers available such as Radiance [5] and POV-Ray [10]. While the former offers more than the latter, we choose the latter mainly because its input is easier for students to learn. In the following, all student work and examples are traced with POV-ray.

CAMERA, LIGHTS, OBJECTS, AND COLOR

The first unit covers the most fundamental topics in ray tracing: camera, basic lighting, simple objects and color. They are discussed to the depth that is sufficient for students to do their first warm-up exercise, usually in the second week. In our experience, students are eager to do their first exercise to satisfy their curiosity, and usually they run much faster than the pace of the lecture.

We start with an overview of ray tracing. The camera is the first and the most important element. It is usually not very difficult to learn because almost everyone played with an actual camera. What makes the camera topic a little more difficult is that, unlike a real camera, a ray tracer requires a user to precisely specify position, orientation, angle of view and a look at point. These information are normally not explicitly required in real world. Some beginners frequently generate blank or clipped images because the characteristics of a camera are not specified properly. This unit also covers simple objects such as blocks, disks, planes, spheres, cylinders, cones, and tori. Light sources are always point lights, and a color always has four components: red, green, blue, and a transparency level. Textures are not covered. Figure 1(a) shows a student work using simple shapes and simple light sources, and (b) goes one step further to include textures and reflection.

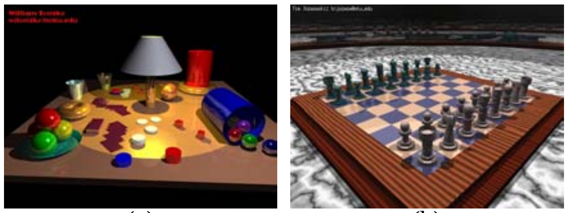

(a) (b)

FIGURE 1

More Lights

To prepare students for the next stage, we briefly cover other types of light sources such as *spotlight* and *area light*. In addition to the position and direction, a spotlight has three more parameters. The first is the angle of the light cone with respect to the direction vector. Within the light cone, the intensity of a spotlight does not vary. The second is the fall-off angle. The intensity of the spotlight diminishes to zero from the light cone to the fall-off angle. The third is tightness that describes how the intensity diminishes from the light cone to the fall-off cone. Figure 2(a) shows the effect of a number of combined cone angle, fall-off angle and tightness. With spotlights, we can demonstrate color mix of light sources. In Figure 2(b), three spotlights in color red, green and blue generate three colored circular areas and the color mixing effect is clearly seen.

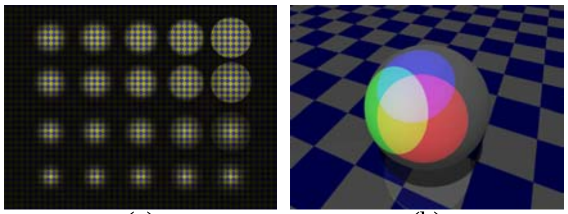

(a) (b)

FIGURE 2

Shadows in a ray traced image is very sharp (Figure 3(a)). Some ray tracers use *area light* to simulate soft shadows. An area light is simply a collection of $m \cdot n$ point light sources organized in a $m \times n$ array. In this way, each object will have multiple shadows, one from each light element in an area light. Figure 3(b) is the result of a 4×4 area light source. To make area lights even more attractive, ray tracers may use an *adaptive* way to more accurately determine how much light can reach a point. Figure 3(c) is the result of applying adaptive computation to the result in (b). However, even with an adaptive method, overlapping shadows are still there, although they are finer. To address this problem, each light ray may be randomly deviated a little from its original direction, and the effect of overlapping shadows is alleviated (Figure 3(d)).

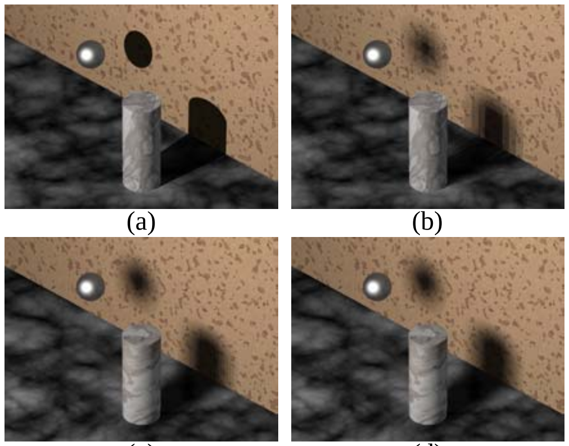

(c) (d)

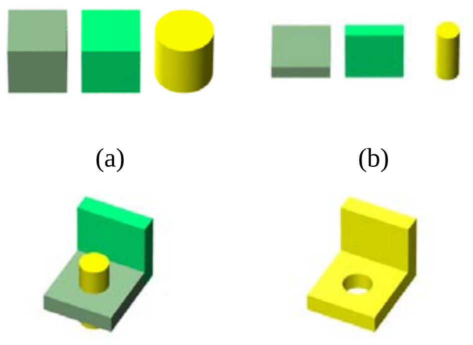
FIGURE 3

Constructive Solid Geometry

Constructive solid geometry, or CSG for short, is a commonly used technique in computer graphics and engineering design. CSG considers an object as a set that includes its interior and boundary (*i.e.*, the surface of the object). Commonly seen CSG primitives include blocks, spheres, cylinders, cones and tori. Boolean operators (*i.e.*, set union, intersection and difference) are used to construct new objects from CSG primitives. The construction is, naturally, not unique. Suppose we want to construct a latch-like object as shown in Figure 4(d). We first take two blocks and one cylinder (a), scale them to proper size (b), make the blocks perpendicular to each other, use a *set union* to obtain a L shape, move the scaled cylinder to the middle of the horizontal block (c), and uses a *set difference* to obtain the desired result (d).

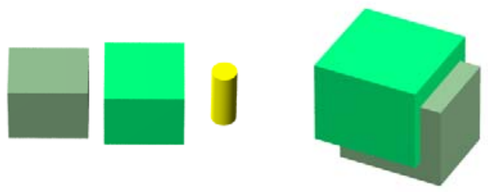
Figure 5 has a different construction. It starts with two differently scaled blocks (a), say B_1 and B_2, moves B_2 to a proper position (b), and uses a *set difference* to take out the B_2 component from B_1 to obtain the L shape. The remaining steps are identical to those in Figure 4.

The beauty of CSG construction is that the complete construction process can be written as an expression. For exam-



(a) (b) (c) (d)

FIGURE 4



(a) (b)

FIGURE 5

ple, the second construction's CSG expression is

$$\text{latch} = (\text{scale}(B) - \text{trans}(\text{scale}(B))) - \text{trans}(\text{scale}(C))$$

where B and C are the box and cylinder in some standard form. Figure 6 shows the corresponding expression tree, where S and T indicate scaling and translation.

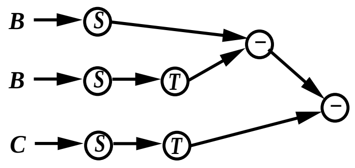
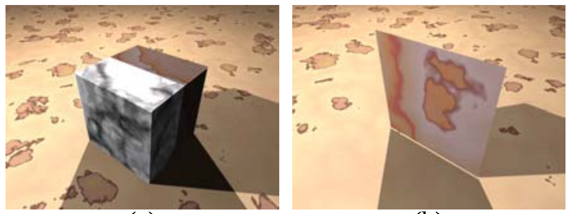


FIGURE 6

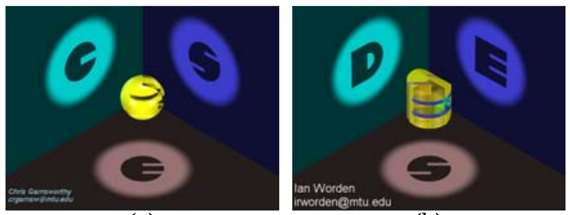
We must point out that Boolean operators may produce lower dimensional artifacts. Figure 7(a) shows two boxes that share a common face, and Figure 7(b) is the result of a set intersection, which could not happen in reality. Therefore, the *regularized* Boolean Operators are introduced to overcome this problem so that the result is always a three-dimensional object. Unfortunately, due to the complexity of regularized Boolean operators, they are rarely implemented. Students must be aware of this fact and employ a number of techniques to deal with it. For example, when specifying $A-B$ or $A \cap B$, object B should be larger than necessary (Figure 5(b)).



(a) (b)

FIGURE 7

The CSG exercise is a challenging one. We gave students a scene with one object, three walls, and three spotlights each of which projects a shadow of the given object on a wall. Students are required to use Boolean operators to sculpture the object so that its shadows on the walls are pre-defined letters. Figure 8 shows two student work. The left one shows CSE (computer science education or computational science and engineering) and the right one shows DES (data encryption standard).



(a) (b)

FIGURE 8

Surface Materials and Texture

An illumination model that expresses the factors determining a surface's color at a point includes at least three coefficients: *ambient*, *diffuse* and *specular*. A user can specify the value, in the range of 0 and 1, for each coefficient. Some ray tracers also support the Phong illumination model and include a *specular-reflection exponent*. It is easy to place a number of spheres in the scene with different combinations of ambient, diffuse and specular values, and vividly visualize the actual effect. For example, a high ambient value washes out shadows on a surface. In addition, student can also experiment with the Phong model using various specular-reflection exponents. In this way, they will be able to pick up the fundamentals quickly. It is faster than performing the same experiments with programming because the former is easier and more "photorealistic."

The next set of coefficients includes *reflection*, *refraction* and the *index of refraction* or IOR for short. This is the first place where students can generate images that are not possible with a popular API. With reflection and refraction, they can build mirrors and lenses. In Figure 9(a), two lenses (constructed with CSG) are far away from the six spheres. Since the right lens having a larger IOR, we can see all six spheres. Because the distance between the lens and the spheres is large, the images seen through the lenses are reversed. If we move the lenses closer to the spheres, the effect of a magnifier is clearly seen in Figure 9(b).

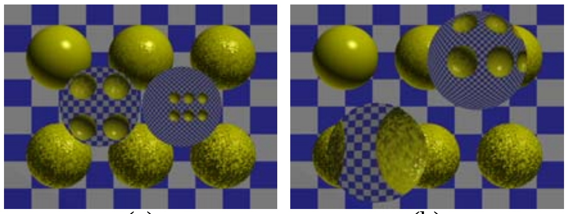

(a) (b)

FIGURE 9

We also discuss a drawback of ray tracing using reflection. Since rays are traced in a recursive way ([3] and [8]), completely tracing a scene with many reflective surfaces may cause the recursion going into an infinite depth. Hence, a ray tracer must stop at certain recursion level. In other word, reflection beyond this level will not be seen in the image. The scene in Figure 10 has three mirrors surrounding an object. Figure 10(a) is traced with recursion level 1. We only see one level of reflection, and the reflection from the mirror behind the object is not shown. Figure 10(b), (c) and (d) show levels 2, 5 and 20. As the recursion level increases, the traced image becomes more realistic. Unfortunately, tracing a scene with many reflective surfaces with high recursion level is very time consuming.

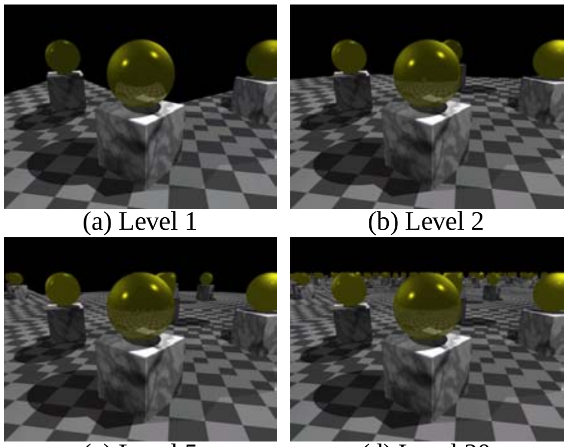

(a) Level 1 (b) Level 2
(c) Level 5 (d) Level 20

FIGURE 10

Then, we proceed to the discussion of textures. The bit-map type texture is easy to understand and use. Many ray tracers provide basic non-bit-map textures and permit a user to design a color pattern that can be modified by random transformations. Once a texture is created, it is mapped onto the surface. In this unit, we covered marble, granite, wood, and other interesting textures. We also mentioned how a user can "stir" a color map (*i.e.*, color stripes) into a very interesting texture. For example, Figure 11(a) shows a semi-transparent screen created with this technique, and Figure 11(b) is an image using the chrome metal and other marble type textures.

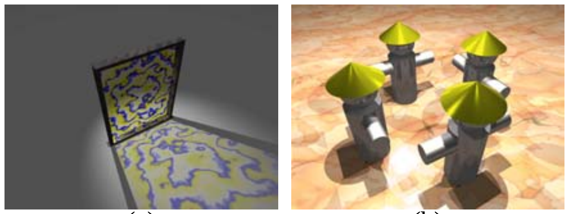

(a) (b)

FIGURE 11

The exercise for this unit consists of the use of *all* non-degenerate quadric surfaces and textures. Students must place all quadric surfaces, with textures, into a scene. Figure 12(a) shows the use of all five quadric surfaces and textures such as stone, sand, wave, sky and cloud, while Figure 12(b) shows another fine design.

(a) (b)

FIGURE 12

Parametric Surfaces and Tessellation

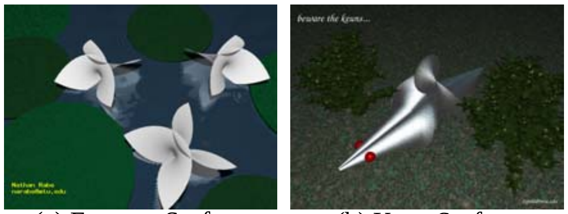
Parametric surfaces are the most commonly used type of surfaces. A graphics API such as OpenGL may be a better tool for handling parametric surfaces because these local illumination model based APIs render triangles efficiently. We include this topic because we want our students to compare the results obtained from local illumination models. Parametric, Bézier, B-spline and NURBS curves and surfaces are covered in class using a tool DesignMentor ([11], [12] and [13]). To learn how an API displays a parametric surface, we must cover surface tessellation. Since surface tessellation is such an extensive and complex topic and may be too difficult for juniors to digest, instead of providing a complete theory, we discuss a naive algorithm and indicate the existence of other powerful algorithms.

This algorithm is very simple. The u and v directions of the domain $(u,v) \in [0,1] \times [0,1]$ is divided by $u_0 = 0$, u_1, ..., $u_m = 1$ and $v_0 = 0$, v_1, ..., $v_n = 1$. Then, the rectangle defined by vertices (u_i, v_j), (u_{i+1}, v_j), (u_{i+1}, v_{j+1}) and (u_i, v_{j+1}) is further divided into two triangles, say triangle (u_i, v_j), (u_{i+1}, v_j) and (u_{i+1}, v_{j+1}) and triangle (u_i, v_j), (u_{i+1}, v_{j+1}) and (u_i, v_{j+1}). If the surface equation is $\mathbf{S}(u,v)$, the first approximation triangle is defined by $\mathbf{S}(u_i, v_j)$, $\mathbf{S}(u_{i+1}, v_j)$ and $\mathbf{S}(u_{i+1}, v_{j+1})$, and the second is defined by $\mathbf{S}(u_i, v_j)$, $\mathbf{S}(u_{i+1}, v_{j+1})$ and $\mathbf{S}(u_i, v_{j+1})$. These triangles can be rendered with an API's Gouraud shading algorithm. If the API has Phong shading, the normal vector at each vertex can also be computed.

While this algorithm is simple, it is good enough for many surfaces. Moreover, students can vary the values of m and n to strike a balance between details and efficiency. Normally, we gave students the parametric equations of a well-known surface in differential geometry and/or in application, and ask them to perform the following:

- Write a program using the available API to display the surface and study its geometry. In this stage, students must determine the values of m and n, divide the domain, compute the triangles, and render them.
- The triangles are saved to a file in a proper format.
- Use these triangles along with other objects to design a scene. The grading criteria are not only the correctness of the surface, but also the scene design.

This exercise turns out to be quite challenging due to the unknown geometry of the surface. However, students always manage to get the surface right and use it in their scene. Figure 13 shows two examples: Enneper and Kuen surfaces. These images demonstrate that students are capable of generating very creative scenes with dull mathematical surfaces. This is exactly what we anticipate from ray tracing exercises.



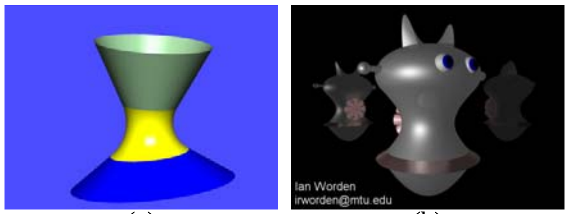
(a) Enneper Surface (b) Kuen Surface

FIGURE 13

ALGEBRAIC SURFACES

Algebraic surfaces is the last topic of our ray tracing component. Unlike parametric surfaces, no programming APIs can handle algebraic surfaces well. Because programming APIs render triangles and/or quadrilaterals, an algebraic surface, a surface defined by a polynomial of form $p(x,y,z) = 0$, must be tessellated into triangles. While there are algorithms available, none of them can handle singularities well. Singularities of an algebraic surface are the self-intersection points, lines and curves where the partial derivatives are zero. Known algorithms perform well if a surface is singularity free; otherwise, they may generate a polyhedron that is topologically different from the given surface. Currently, ray tracing is the only viable and easy way of coping with singularities, although it is not very efficient. Algebraic surfaces have one more advantage: they can be used with constructive solid geometry, because an algebraic surface has a natural way to define its interior (*i.e.*, $p(x,y,z) < 0$), boundary (*i.e.*, $p(x,y,z) = 0$) and exterior (*i.e.*, $p(x,y,z) > 0$).

We also cover applications of algebraic surfaces. We pick two important topics: *blending* and *offsetting*. A surface B blends surfaces X and Y along curves x on X and y on Y if surface B is tangent to X and Y along curves x and y. This is very useful in *smoothing* sharp corners of an object. Figure 14(a) uses a Dupin cyclide to blend two cones, and (b), another student work, shows the use of a hyperboloid of one sheet as a blending surface for two ellipsoids. Surface X is an *offset surface* of a surface Y if the signed distance from Y to X along any normal vector of Y is a constant. Blending and offset surfaces are usually algebraic even though the given surface(s) may be parametric, and are frequently used in geometric design.



(a) (b)

FIGURE 14

The exercise given to students is similar to that of the parametric surface case. Students are provided with an implicit equation $p(x,y,z) = 0$ and some clipping information. They are asked to study the shape of the surface. Once the geometry of the surface is understood, use the surface to design a scene. Normally, the surface equation also contains an extra parameter. By varying this parameter, a family of algebraic surfaces is generated. Thus, students are required to explore the shapes in the family and use various shapes in their scenes. Figure 15(a) shows a number of Barth's surface, and (b) shows the use of a number of Kummer's surface from a family to make a rose. Both are very creative and imaginative.

EVALUATION

Our course is evaluated with a pre-test, a post-test, and an attitudinal survey. Since ray tracing is *not* a major topic in this course, and is used for introducing the basics of graphics and to do design/modeling problems, the anonymous attitudinal

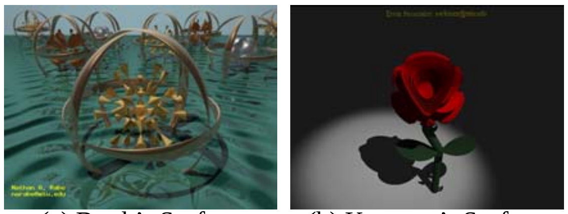

(a) Barth's Surface (b) Kummer's Surface

FIGURE 15

survey only asks one question: in the scale from 1 (strongly disliked) to 5 (really liked), do you like the ray tracing software (*i.e.*, POV-Ray)? The following table summarizes the survey results. More than 80% of our students do like the ray tracing software. There are always one or two "outliers" who dislike it very much: "[I] hated [POV-Ray] with a passion." For those who like the software, they rated it highly. One student even checked a "6" and indicated: "[I] really really like it."

Responses	1	2	3	4	5	*Avg*	*Var*
2000	1	2	0	6	6	3.9	1.3
	7%	13%	0%	40%	40%		
2001	0	1	0	8	7	4.3	0.6
	0%	6%	0%	50%	44%		

Responses to this question only answer one part of the equation. In order to ensure ray tracing can help students understand graphics and modeling/design, we select the top 20% work of each exercise and post them on a Raytracing Hall of Fame page. In this way, students will be able to learn more from what other students did in previous years. We have shown some of the best student work in this paper. Others can be found on the Introduction to Computing with Geometry course page at `http://www.csl.mtu.edu/cs3621/www/Home.html`. We found that the quality of student work increases every year. This shows that they do learn a lot and use what they learn in the image creation process. Here are some typical comments from attitudinal surveys: "POV-Ray assignments let us have a lot creativity," and "[I like] the use of POV-Ray to generate realistic objects." As mentioned earlier, ray tracing only consumes less than 5% of all lecture hours, and is used for teaching the fundamentals. The rest of the course is for curves, surfaces and their applications. Thus, intensive programming is required. For those who like ray tracing more than programming, here is a typical "suggestion": "Easier programs. More POV-Ray!"

WORK IN PROGRESS

We believe that an introduction to computer graphics course should be a balanced one that includes both local and global illumination models. The programming approach covers local illumination models and animation efficiently. GraphicsMentor [2] is a tool that can help an instructor to go through almost all fundamentals related to local illumination models easily and quickly before asking students to program. However, we should not abandon global illumination models. While ray tracing provides an easy and nice way for students to learn global illumination, it only tells one side of the story. We need to do more in order to create a well-balanced course. Currently, GraphicsMentor is able to output a scene to POV-Ray so that students can compare the differences between the result obtained with a programming API and the result generated with ray tracing. Another important global illumination theory is radiosity ([1] and [9]), which is usually skipped in an introduction course. We believe that this is a topic that should be discussed to certain depth. Therefore, we plan to add a radiosity subsystem that will accept the output of GraphicsMentor and render the scene with various radiosity algorithms. Note that radiosity is available in some ray tracers such as Radiance and POV-Ray. However, we would like to have an independent pedagogical system that can help students visualize the computation activities and understand the algorithms.

The newly developed photon mapping technique [4] may help bridge the gap between ray tracing and radiosity. Photon mapping algorithms shoot photons into the scene from a light source. The photons carry color and other information from objects to objects as they hit until their energy diminishes to zero. These information are stored in a photon map, which is then combined with a ray traced result. With photon mapping, we can generate color bleeding and caustics. Color bleeding is a trademark of radiosity and caustics cannot be easily and efficiently produced by both ray tracing and radiosity. We plan to add photon mapping to GraphicsMentor. Once these capabilities become available, we will have a very powerful tool for teaching all important and modern topics easily.

CONCLUSIONS

We have presented our approach of using ray tracing to teach many fundamental concepts and design/modeling techniques in a junior level elective course Introduction to Computing with Geometry. The major contribution of our work is the evidence showing that the combination of ray tracing and programming can take the advantage of both approaches to produce a very productive and interesting course and to provide students with a robust environment for creativity and artistic expression development. The interested readers may find more about our work, software availability, and future announcement at the following site:

`http://www.cs.mtu.edu/~shene/NSF-2`

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation under grant numbers DUE-9653244, DUE-9952621 and DUE-0127401.

REFERENCES

[1] Cohen, M. F. and Wallace, J. R., *Radiosity and Realistic Image Synthesis*, Academic Press, 1993.

[2] Nikolic, D. and Shene, C.-K., GraphicsMentor: A Tool for Learning Graphics Fundamentals, *ACM 33rd Annual SIGCSE Technical Symposium*, February 27 - March 3, 2002, pp. 242–246.

[3] Glassner, A. S. (editor), *An Introduction to Ray Tracing*, Academic Press, 1989.

[4] Jensen, H. W., *Realistic Image Synthesis Using Photon Mapping*, A K Peters, 2001.

[5] Larson, G. W. and Shakespeare, R., *Rendering with Radiance*, Morgan Kaufmann, 1997.

[6] Lowther, J. L. and Shene, C.-K., Rendering + Modeling + Animation + Postprocessing = Computer Graphics, *The Journal of Computing in Small Colleges*, Vol. 16 (2000), No. 1 (November), pp. 20–28. Reprinted in *Computer Graphics*, Vol. 34 (2000), No. 4 (November), pp. 15–18. Reprinted in *Computer Graphics*, Vol. 24 (2000), No. 4 (November).

[7] Lowther, J. L. and Shene, C.-K., Computing with Geometry as an Undergraduate Course: A Three-Year Experience, *ACM 32nd SIGCSE Technical Symposium*, February 21–25, 2001, pp. 119–123.

[8] Shirley, P., *Realistic Ray Tracing*, A K Peters, 2000.

[9] Sillion, F. X. and Puech, C., *Radiosity and Global Illumination*, Morgan Kaufmann, 1994.

[10] Young, C. and Wells, D., *Ray Tracing Creations*, Second Edition, Waite Group Press, 1994. POV-Ray 3.x is available free from `http://www.povray.org`.

[11] Zhao, Y., Lowther, J. L. and Shene, C.-K., A Tool for Teaching Curve Design, *ACM 29th Annual SIGCSE Technical Symposium*, February 26 - March 1, 1998, pp. 97–101.

[12] Zhao, Y., Zhou, Y., Lowther, J. L. and Shene, C.-K., Cross-Sectional Design: A Tool for Computer Graphics and Computer-Aided Design Courses, *29th ASEE/IEEE Frontiers in Education*, November 10-13, Vol. II (1999), pp. (12b3-1)–(12b3-6).

[13] Zhou, Y., Zhao, Y., Lowther, J. L. and Shene, C.-K., Teaching Surface Design Made Easy, *ACM 30th Annual SIGCSE Technical Symposium*, March 24 - March 28, 1999, pp. 222–226.