Photon Mapping Made Easy *

Tin-Tin Yu, John Lowther and Ching-Kuang Shene
Department of Computer Science
Michigan Technological University
Houghton, MI 49931
{tiyu,john,shene}@mtu.edu

ABSTRACT

This paper presents the authors' introduction of photon mapping in an undergraduate computer graphics course, Software was designed as a pedagogical and demonstration tool which permitted students to practice and learn photon mapping. Classroom experience and examples that illustrate soft shadows, color bleeding, indirect illumination and caustic are also discussed.

Categories and Subject Descriptors

I.3.7 [**Three-Dimensional Graphics and Realism**]: Raytracing; K.3.2 [**Computers and Education**]: Computer science education

General Terms

Photon Mapping, Ray Tracing

Keywords

Photon Mapping, Ray Tracing

1. MOTIVATION

The *programming approach* is the most popular approach in teaching introductory computer graphics courses. However, it does have some serious drawbacks [4]. **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 [5, 7]. **Second**, the programming approach depends on local illumination-based graphics APIs which, in general, do not support shadow generation, 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.

To overcome this "local illumination" bias and maintain a well-balanced graphics syllabus, we teach students ray tracing. Our experience, as reported in [6], shows that students like ray tracing and learn global illumination models effectively and efficiently from exercises designed to explore the advantages of global illumination models and disadvantages of local illumination models. With this knowledge in hand, in our **Advanced Computer Graphics** course, we present ray tracing implementation details and discuss major shortcomings of ray tracing. We do cover radiosity in our course. However, radiosity theory is complex and, reflection and refraction are not available unless more theory is discussed. This is a very frustrating fact, and leads us to find a simple, easy to understand, and powerful enough global illumination based rendering method between ray tracing and radiosity. The answer is photon mapping.

In the following, Section 2 elaborates why photon mapping is a natural and reasonable choice, Section 3 briefly discusses the theory of photon mapping, Section 4 presents the details of our software tool designed to help students explore photon mapping and generate images, Section 5 demonstrates the quality of our system with a number of examples, Section 6 summarizes our classroom experience, and, finally, Section 7 has our conclusions.

2. WHY PHOTON MAPPING?

Traditional ray tracing systems trace rays from an eye (or a camera) rather than from light sources, and one light ray from a hit point to each light source is used to determine the shadow color of that point. As a result, traditional ray tracing techniques are not able to correctly generate shadows (and hence caustic) of refractive objects, produce indirect illumination from reflective objects, and implement diffuse reflection (*e.g.*, no color bleeding). Although soft shadows are possible, one must use area lights that can significantly increase processing time. On the other hand, radiosity provides soft shadows, color bleeding and indirect illumination for free; however, it does not handle specular reflection, has difficulty in processing transparency (*i.e.*, reflection and refraction), requires the scene to be subdivided into polygons, and is very time consuming. A second pass (*e.g.*, ray tracing) is needed to produce reflection and refraction.

Instead of complicating the radiosity implementation by

*This work is partially supported by the National Science Foundation under grant DUE-0127401. The third author is also supported by an IBM Eclipse Innovation Award 2003.

SIGCSE'05, February 23–27, 2005, St. Louis, Missouri, USA.

post ray tracing, an easier way would collect illumination information of the scene by a pre-trace from light sources. This is the basic idea of photon mapping. Since students have learned ray tracing, converting the process of tracing from the camera to tracing from each light source with some data structure manipulations is not very difficult. Furthermore, since photo mapping can easily generate area light sources, color bleeding, soft shadows, indirect illumination and caustic, we believe it would be easier for students to learn and program these effects via photon mapping rather than with radiosity. The major advantages of photon mapping are (1) using photons to simulate the transport of individual photon energy, (2) being able to calculate global illumination effects, (3) capable of handling arbitrary geometry rather than polygonal scenes, (4) low memory consumption, and (5) producing correct rendering results, even though *noise* could be introduced. Consequently, after the discussion of ray tracing and radiosity, our course adds a 5-hour presentation on photon mapping to its global illumination module.

3. WHAT IS PHOTON MAPPING?

The basic idea of photon mapping is very simple [3]. It tries to decouple the representation of a scene from its geometry and stores illumination information in a global data structure, the *photon map*. Photon mapping is a two-pass method. The first pass builds the photon map by tracing photons from each light source (Section 3.1), and the second pass renders the scene using the information stored in the photon map (Section 3.2).

3.1 Pass 1: Light Emission and Photon Scattering

The first pass of photon mapping consists of two steps: *light emission* and *photon scattering*. In the first step, photons are generated and shot into the scene. A light source with higher intensity will produce more photons, and the direction of each photon is randomly selected based on the type of the light source (*e.g.*, spherical, rectangle, or directional). The processing of these photons is similar to ray tracing with one difference: photons propagate flux while rays gather radiance. When a photon hits an object, it can be reflected, transmitted, or absorbed. If a photon hits a specular object (*e.g.*, a mirror), it is reflected with its intensity scaled by the reflection index of that object. On the other hand, if a photon hits a diffuse surface, it is stored in the photon map and reflected. The direction of this "diffusely" reflected photon is a randomly chosen vector that is above the intersection point with a probability proportional to the cosine of the angle with the normal. This can be implemented by playing "Russian roulette."

The "Russian roulette" technique removes unimportant photons and ensures the same power. In the specular case, a random number $q \in [0,1]$ is generated, and if $q \in [0,f)$, where f is the reflection index, the photo is reflected. Otherwise, it is absorbed. In the diffuse case, a random number $q \in [0,1]$ is generated, and the photon is diffused (*resp.*, reflected) if $q \in [0,d]$ (*resp.*, $q \in (d,d+s]$), where d and s are the diffuse reflection and specular reflection indices, respectively. Otherwise, it is absorbed.

The photon hit information is stored in a photon map, which is usually a balanced *kd*-tree [1]. Each node of the *kd*-tree stores the information of a photon hit, which include the coordinates of the hit point (x,y,z) (used as the key for building the tree), color intensity (r,g,b), incident direction of the photon, and other important information. Since the number of photons and their hit points may be very large, some form of compression may be needed; however, in an undergraduate course, we skip compression concerns. While the concept of *kd*-tree is simple, the implementation of a balanced *kd*-tree is a non-trivial task. Fortunately, good balanced *kd*-tree code is available [3]. After all photons are shot and the *kd*-tree is built, the first pass completes.

3.2 Pass 2: Radiance Estimate and Rendering

The second pass renders the scene with the help of the photon map built in the first pass. A traditional ray tracing procedure is performed by shooting rays from the camera. When a ray hits a point $\mathbf{P}$ on a surface, the illumination information (*i.e.*, flux) of the neighboring photons collected from the first pass and stored in the photon map will be added to the radiance information collected from ray tracing at $\mathbf{P}$. Let the normal vector at $\mathbf{P}$ be $\mathbf{N}$ and $r > 0$ be a predefined small value. Consider all photons in the sphere $\mathcal{S}(\mathbf{P},r)$ of center $\mathbf{P}$ and radius r (Figure 1).

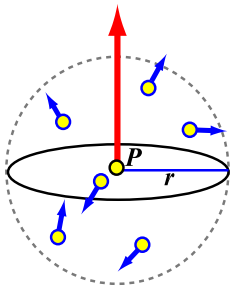


Figure 1: Radiance Estimate

Not every photon in $\mathcal{S}(\mathbf{P},r)$ would contribute to the radiance at $\mathbf{P}$. In fact, a photon with incident direction $\mathbf{d}$ can contribute only if $\mathbf{d}\cdot\mathbf{N} > 0$, because if $\mathbf{d}\cdot\mathbf{N} \le 0$, its direction goes inside of the surface. If a photon does not contribute, it is ignored in this radiance estimate for point $\mathbf{P}$. From the illumination equation, the radiance contribution of a photon with incidence direction is:

$$\text{intensity} \times (\mathbf{d}\cdot\mathbf{N}) \times \text{diffuse-factor}$$

Let the sum of all radiance contribution be s. The radiance estimate at $\mathbf{P}$ is $s/(\pi r^2)$, where πr^2 is the area of a great circle of sphere $\mathcal{S}$. Therefore, the color at $\mathbf{P}$ is the sum of this radiance contribution and the radiance calculated from ray tracing. This method is theoretically sound; however, due to its required mathematics, we do not offer any proof. The sum of the radiance estimate and the radiance collected from ray tracing may be larger than one, and normalization is needed. Additionally, if the number of photons that can contribute to radiance estimate is too small, they are all ignored because the computed radiance estimate from very few photons may produce blurred images.

4. SOFTWARE DESIGN

This software that supports ray tracing and photon mapping is the cumulative work of many undergraduate students. It grew out from various undergraduate student research projects on ray tracing and photon mapping. The

current design consists of an API written in C++ classes with a structure as shown in Figure 2. Under `GICore`, `GIType` defines various data types, `GIObjIntersection` provides the ray-object intersection mechanism, and `GILightSrc` handles light sources. `GIType` defines object types and light source types via `GIObjType` and `GILightSrcType`. A user may inherit these classes to extend their functionality.

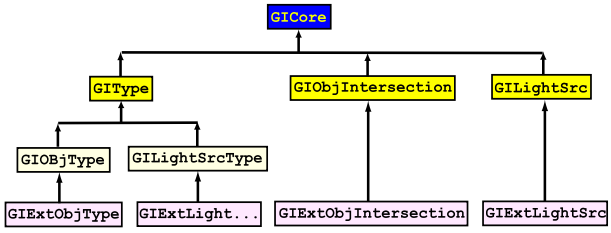


Figure 2: System Programming API

To write a program that performs photon mapping, a user must include the header file `GICore.h`, define a scene class derived from the base class `GIScene`, and provide a constructor. Figure 3 shows that the scene `MyScene` has a rectangular area light, a red sphere and a white floor, Figure 4 has the main program that actually renders the scene, and Figure 5(a) is the result of running this program. Therefore, it is very easy for a user to render a scene using photon mapping. If the call to method `EnablePhotonMapping()` is removed, the system will only ray trace the scene. Figure 5(b) is a ray traced example.

```
class MyScene : public GIScene
{
   public:
      void ConstructScene();
};

void MyScene::ConstructScene()
{
  GI_FLOAT l=-1.0, r=1.0, t=5.5, b=7.5, h=5.25;

  this->backgroundColor = GI_RGB(0.0,0.0,0.0);
  AddRectAreaLight(
    Vector3(0,-1,0),    // normal
    Vector3(l,h,t), Vector3(r,h,t), // vertices:
    Vector3(r,h,b), Vector3(l,h,b),
    1.00,               // brightness,
    GI_RGB(0.95, 0.85, 0.90) ); // light color

  AddSphere(            // red sphere
    Vector3(1.65,1.000,5.5), 1.995, // center & radius
    red_plastic);       // material (pre-defined)

  AddPlane(             // white floor
    Vector3(0,1,0), Vector3(0,-1.0,0), // normal & vertex
    white_plastic_dull);// material (pre-defined)
}
```

Figure 3: Scene Constructor

5. EXAMPLES

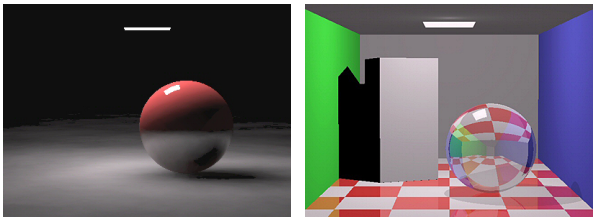
Figure 6 demonstrates the quality of our photon mapping API. The scene consists of a box and a glass sphere, with a rectangular light source (*i.e.*, area light) on the ceiling (Figure 5(b)). Thus, we expect to have soft shadows, color bleeding from the green wall to the left side of the box, and

```
int main(int argc, char **argv)
{
  MyScene *scene = new MyScene(); // create MyScene
  GIImage img;                    // image area
  int width = 400, height = 400*3.0/4.0;

  GICamera cam = CreateCamera(
     Vector3( 0,  3.20, -24),     // position
     Vector3( 0, -0.05,   1),     // look-at direction
     Vector3( 0,     1,   0),     // up-vector
     45.0,                        // view angle
     width/height);               // aspect ratio
  scene->EnablePhotonMapping();   // use photon mapping
  img = CreateImage(width, height); // create image

  scene->MaxRayPerLightSrc = 10 * 1000;
  scene->MaxRayTraceDepth     = 3;
  scene->MaxPhotonTraceDepth  = 6;
  scene->photonMap.MaxPhotonSearchRadius = 2.0;
  scene->ConstructScene();        // scene construction
  scene->Render(cam, img);        // render the scene
  SaveImgPPM("test.ppm", img);    // Save to file
  free(img.data);                 // free image memory
  return 0;
}
```

Figure 4: The Main Program

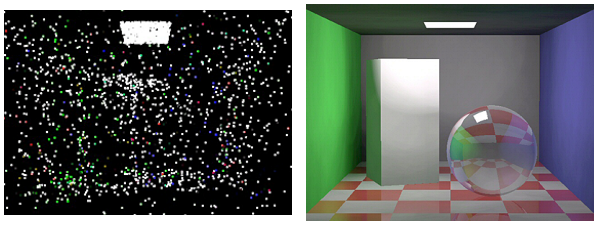


(a) (b)

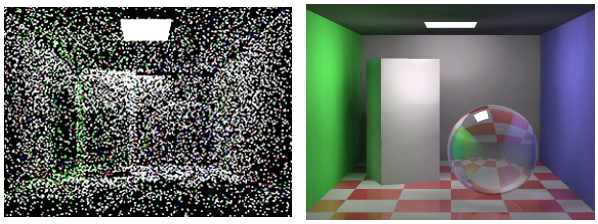
Figure 5: Sample Photon Mapping and Ray Tracing Images

some form of caustic produced by the glass ball. In Figure 6, the first image shows the photon hit positions and the second has the rendered result. We choose to use 1,000 photons with $r = 4.0$, 10,000 photons with $r = 2.0$, 100,000 photons with $r = 1.0$ and 200,000 photons with $r = 0.4$. Note that the radius for radiance estimate is inversely proportional to the number of photons shot. The color bleeding effect is weak when the number of photons is small; however, as the number of photons increases, the effect of color bleeding is more evident as shown by the color on the left side of the box. The ceiling above the box also shows a touch of white. Caustic of the glass ball also becomes clearer as the number of photons shot increases. In fact, in the case of 200,000 photons, the glass sphere generates a very pleasant and soft caustic. The effect of soft shadows also becomes more realistic as the number of photons shot increases. Figure 5(b) is the same scene generated by ray tracing. Compared this result with Figure 6(d), it shows a huge difference.

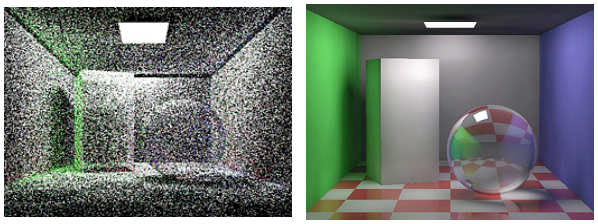
Figure 7 is an example that illustrates the effect of indirect illumination. This scene has an object between the light source and the left wall, and a reflective floor. If this scene is ray traced, the left portion of the ceiling would become dark. With photon mapping, some photons will be bounced to the ceiling, left wall, and even the back side of the blocking object (Figure 7(a)). As a result, these areas are illuminated indirectly by the light reflected from the floor (Figure 7(b)).



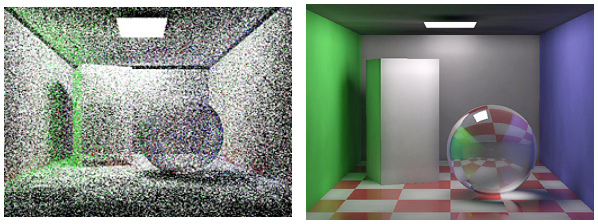
(a) 1,000 photons $r = 4.0$



(b) 10,000 photons $r = 2.0$



(c) 100,000 photons $r = 1.0$



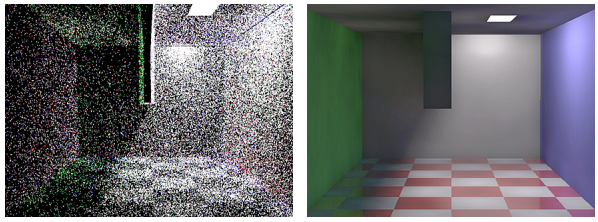
(d) 200,000 photons $r = 0.4$

Figure 6: A Few Photon Mapping Examples

If a glass sphere is bombarded with concentrated photons from a spot light, we will be able to generate very realistic caustic effects. In this case, two photon maps may be needed, one for general rendering while the other for caustic generation. Figure 8, created by undergraduate Josh Anderson with his own caustic code, shows a beautiful and realistic caustic. However, if we use only one photon map to record all photon hits without concentrated photons, caustic may be soft as shown in Figure 9, which is actually Figure 6(d) enlarged to reveal the effect.

6. CLASSROOM EXPERIENCE

Photon mapping was taught twice in our **Advanced Computer Graphics** course, a course designed for those who have completed an introductory course and are interested in more advanced topics beyond graphics programming. Class size was small with around 10 students, most of them were



(a) (b)

Figure 7: Indirect Illumination

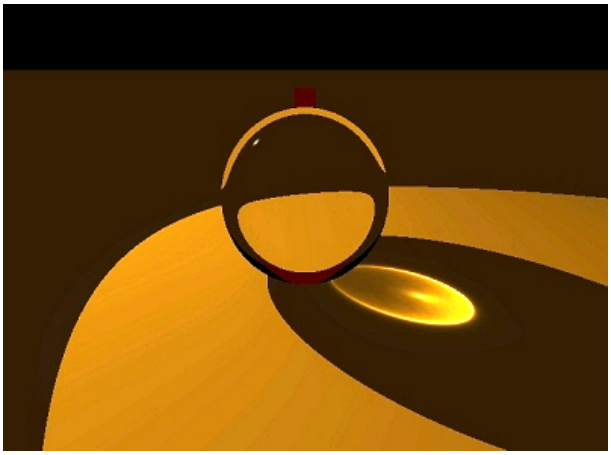


Figure 8: Strong Caustic

undergraduates with a few graduate students. The organization of this course is topic-oriented and project-based. We present about ten topics and each student is required to perform a term project, which includes a report, an implementation, and a presentation with demonstration. In the past, students did ray tracing, photon mapping, radiosity, advanced texture mapping, mesh simplification, geometry compression, multiresolution modeling, morphing and non-photorealistic rendering. We have a long history in ray tracing and photon mapping development by undergraduate students in directed study and other courses. As a result, students who are interested in photon mapping have sample source code to use. Interestingly enough, every student preferred to write their own version of ray tracer and photon mapping system with a difference emphasis (*e.g.*, real-time photon mapping and caustic).

Most students in our class are very competent in ray tracing. Figure 10 shows two student ray traced images using POV-Ray [9]. Hence, we only present the way of building a recursive ray tracer and refer those who are interested to two standard references [2, 8], followed by the discussion of several problems of recursive ray tracing. This usually takes three lecture hours. Then, we spend about six hours on radiosity to address the shortcomings of ray tracing. Topics include the basic heat transfer theory, the radiosity equation, gathering and shooting, progressive refinement, form factor computation, substructuring and adaptive subdivision. After the radiosity discussion, students have a thorough understanding of the basics as well as pros and cons of both global illumination based methods. Photon mapping is introduced as a compromise between ray tracing and

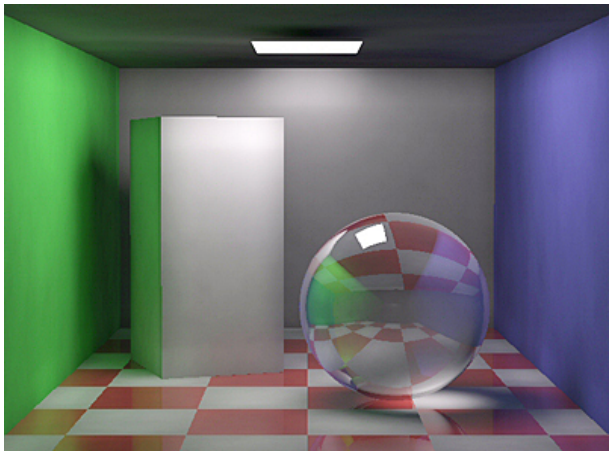


Figure 9: Soft Caustic

radiosity. Since it is easy to understand, students have no difficulty in following the lectures and greatly enjoy photon mapping. This is demonstrated by the success of students' work on photon mapping. For example, Figure 8 is the first successful implementation of caustic using photon mapping. An early version of the software presented here was implemented by the first author of this paper when he was an undergraduate student.

Figure 10: Two Student Ray Tracing Images

Since the size of the class was small and since students follow different research/study paths for the second half of this course with many one-on-one discussions, we believe course evaluation would not help draw any significant conclusion in such a non-traditional course setting.

7. CONCLUSIONS

We have presented our approach of discussing photon mapping in our **Advanced Computer Graphics** course. The major contribution of our work is evidence showing that it is possible to teach global illumination rendering with ray tracing and photon mapping to undergraduate students with very good and convincing results. Our ultimate goal is to provide a good and easy-to-use software tool for students in an introduction computer graphics course to explore, practice and perform photon mapping rendering. Although our software is an API, we plan to add a command processor for users to specify the scene without programming. A long-term goal would be integrating GraphicsMentor and our ray tracing and photon mapping system into a single unit so that students can design a scene using GraphicsMentor and render it with local illumination, ray tracing and photon mapping. In this way, instructors may use this new system to discuss and demonstrate both local and global illumination rendering methods, and students will have a system to practice ray tracing and photon mapping. A radiosity component is also planned. We do hope our effort to making global illumination rendering method available in an introductory course can soon be realized.

8. REFERENCES

[1] J. L. Bentley, Multidimensional Binary Search Trees in Database Applications, *IEEE Transactions on Software Engineering*, Vol. 5 (1979), No. 4 (July), pp. 333–340.

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

[3] Henrik Wann Jensen, *Realistic Image Synthesis Using Photon Mapping*, A K Peters, 2001.

[4] John L. Lowther and Ching-Kuang Shene, 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.

[5] Dejan Nikolic and Ching-Kuang Shene, GraphicsMentor: A Tool for Learning Graphics Fundamentals, *ACM 33rd Annual SIGCSE Technical Symposium*, 2002, pp. 242-246.

[6] Ching-Kuang Shene, Raytracing as a Tool for Learning Computer Graphics, *ASEE/IEEE 32nd Frontiers in Education*, 2002, Volume III, pp. (S4G-7)-(S4G-13).

[7] Ching-Kuang Shene, Teaching and Learning Computer Graphics Made Easy with GraphicsMentor, *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, October, 2002 (online journal).

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

[9] Chris Young and Drew Wells, *Ray Tracing Creations*, Second Edition, Waite Group Press, 1994.

Acknowledgments

The third author appreciates the hospitality of Dr. Horng-jinh Chang, the former President of Tamkang University, Taipei, Taiwan, and Dr. Chuan-Jen Chyan, Chair of Department of Mathematics, for a short-term visit in the summer of 2004 during which the writing of this paper and other research activities were carried out.