CS 371 Project 4:

Photon Mapping

Figure 1: *Sponza with indirect illumination rendered by photon mapping.*

1 Introduction

1.1 Overview

Photon mapping [1995; 1996] is a popular global illumination algorithm with attractive mathematical properties, especially for simulating caustics. It is popular for film and game production. For example, it was used for lighting in *Halo 3* [Chen and Liu 2008] and *Alice in Wonderland* [Martinez 2010]. Henrik Wann Jensen, the primary inventor of photon mapping, received an academy award ("Oscar") in 2004 for his algorithmic contribution to film rendering.

A common problem in global illumination is that most light transport paths traced backward from the eye never reach a light source, and most paths traced forward from the light sources never reach the eye. Photon mapping is a Monte Carlo algorithm for tracing both kinds of light paths half-way and then loosely connecting them to form full transport paths. It can capture all real-world illumination phenomena, and is mathematically **consistent**, meaning that its radiance estimate converges to the true solution as the number of path samples increase.

1.2 Schedule

This is a challenging, project that builds extends the previous ray tracer project. The program will only add about 150 lines to your existing ray tracer. However, they are mathematically sophisticated lines. You will take responsibility for the program design and experiment design yourself this week; they are not in the specification.

As a reference, my implementation contained 9 files, 400 statements, and 300 comment lines as reported by iCompile. The breakdown of my time on the project was about: 10% math and design, 5% implementation, 20% testing and debugging, 65% rigging scenes, running experiments, and writing the report.

Start:	Tuesday,	October 3,	2:00 pm
Checkpoint 1:	Thursday,	October 4,	2:00 pm
Checkpoint 2:	Thursday,	October 11,	2:00 pm
Due:	Monday,	October 15,	12:00 pm

1.3 Rules/Honor Code

I strongly encourage you to talk to other students and share strategies, programming techniques, documentation references, and test scenes. You should not look at any other group's *code* for this project or use *code* from other external sources except for: Jensen's publications, the pbrt library, RTR3 and FCG textbooks, materials presented in lecture, and the G3D library (as limited below). You may look at and use anyone's source code from *last week's* project, with their permission.

During this project, you may use any part of the G3D library and look at all of its source code, including sample programs. You may use any books, papers, websites, etc. Use common (academic) sense in selecting resources–just because something is written down and has the words that you searched for does not mean that it is correct or relevant to this project. Likewise, you're responsible for the end to end correctness of your program, regardless of who originally wrote individual pieces that you incorporated. Don't even assume that *my* code in G3D is correct–you might have to test, or even patch it.

1.4 Teams

You will choose your own teams this week. E-mail me a list of between three and six team members (and their Mac login names) and a team name. I will create a project group for you in SVN within 24 hours of receiving your e-mail.

Note that there is a small amount of math, and translating math into code, that everyone in your project will want to participate in and understand. There is also *a lot* of work rigging scenes and running experiments. That work is best distributed across the team. This balance is typical of commercial software development and of scientific research. So, while each of you could probably implement the algorithm on your own in a day or two, you'll require a large team to successfully validate your program and use it to produce the report and compelling images. Consider this when forming teams and writing schedules.

The command to check out your project this week is:

```
svn co svn://graphics-svn.cs.williams.edu/371/4-PhotonMap/photon-<group>
```

2 Specification

1. Implement a variant of the photon mapping [Jensen 1996] algorithm:
   (a) You may use any appropriate spatial data structure for the photon map; a k-d tree is not required.
   (b) Do not implement the separate caustic photon map.
   (c) Do not implement the shadow photon map.
   (d) Do not implement illumination maps (from Jensen's 1995 paper).
   (e) Do not use a projection map–emit photons within each spotlight's cone proportional to the emitted power of that light, but don't optimize the direction in which the photons are emitted.
   (f) Do not adjust the emitted photons to have power "near 1.0".
   (g) You are not required[1] to implement a general path tracer. Just write a Whitted ray tracer that replaces the "ambient" term with diffuse indirect illumination gathered from the photon map.
   (h) You are not required to grow the photon gather radius until it contains a constant number of photons. Just fix the radius to a scene-specific constant.

Tip: G3D::PointHashGrid, G3D::PointKDTree, G3D::KDTree

2. Create a graphical user interface and functionality for:
   (a) Setting the number of impulse scattering events traced backwards from the eye ("maxBackwardBounces")
   (b) Setting the maximum number of photon scattering events from the light ("maxForwardBounces")
   (c) Enabling final gathering. When this is disabled, gather diffuse indirect illumination directly from the photon map.
   (d) Number of final gather rays.
   (e) Enabling explicit direct illumination (vs. handling direct illumination via the photon map)
   (f) Enabling shadow rays when direct illumination is enabled
   (g) Enabling real-time visualization of the stored photons over the wireframe (see Section 4.3).
   (h) Setting the photon gather radius r, `RenderSettings::photonRadius`
   (i) Setting the number of photons emitted
   (j) Displaying the photon trace time
   (k) Displaying the number of photon rays traced
   (l) Displaying the number of photons stored (which may be either larger or smaller than the number of photons emitted).
   (m) All features from the Recursive Rays tracer project

Tip: Debug with final gathering disabled because it is really slow.

[1] "not required" implies, "but may, if you wish to explore this idea"

3. Document the interfaces of your source code using Doxygen formatting. Overview documentation is especially important this week because I am not giving you a design for the program. Be sure to highlight all key classes in the overview.

4. Devise, create, and render the following scenes:

   (a) The required three classic scenes described in the report.

   (b) As many custom scenes as needed to demonstrate correctness or explore errors and performance. I recommend sharing these between groups, with appropriate credits.

   (c) A custom scene that mimics an actual photograph as closely as possible. Your goal is for the viewer to consider the images indistinguishable when viewed quickly.

   (d) A visually compelling scene of your own creation, as described in section 2.2. Render many variation that explore significantly different lighting (including skybox) and camera viewpoints. Variations are inexpensive to create compared to creating the scene and the program in the first place. For each shot, show a reference image from which you have mimicked the composition, lighting, and camera angle. The content of the reference image need not be exactly similar. For example, George Lucas directed all of the X-wing combat shots in *Star Wars* to match classic WWII films of Japanese Zero fighters, but obviously added color and changed the setting.

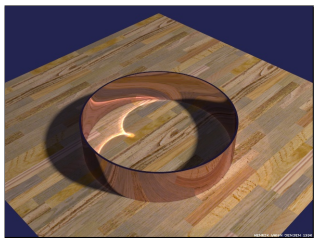
5. Produce the report described in section 2.2, and maintain a development journal throughout the process. The images in your report should be a subset of those used in the journal.

2.1 Checkpoints

Present your project to me at each of the checkpoints. Your presentation should take five minutes and be rehearsed. I want to understand your design, the current status of your program, and problems that you are currently encountering. Your presentation does not need to be "fancy"–you aren't giving a public talk, you're explaining the status of a project to a peer who understands the basic idea. However, it should be professional and concise. This is the kind of presentation that you'll give regularly to your manager, director, or graduate advisor after you graduate.

You can use presentation software, or simply scroll through your report, journal, and documentation, since they should contain exactly the information that you'd use to describe the program. I can comprehend pictures, diagrams, short lists, and method overview documentation in this context. I can't comprehend prose or code during a short presentation.

You will create your own schedule and team management structure and decide what are reasonable milestones for each checkpoint. Keep in mind that the more of the specification that you have covered (even without refinement), the more likely that you'll have encountered the details on which you'd like my advice.



(a) Cardiod caustic from a metal ring rendered by Henrik Wann Jensen using photon mapping.

(b) Cornell box photographic reference image by Francois Sillion

(c) Sponza atrium rendered by Matt Pharr and Greg Humphreys using photon mapping.

Figure 2: *Classic global illumination test scenes.*

2.2 Report

In addition to overview documentation, specifically address the following in your report. As always, I recommend sketching out the report before implementing code, and then updating it as you progress. (Observe that the format of the reports has been migrating towards that of a computer graphics research paper...)

1. **Algorithm**: Give pseudo-code for the entire photon mapping algorithm, similar to the way that you did last week.

   - This should be about 30 lines long and include LaTeX equations.
   - I recommend using either nested HTML lists (`<ol><li>...</ol>`) inside a Doxygen `\htmlonly ... \endhtmlonly` block.
   - The details you need are in this document, Jensen's notes [2007], and McGuire and Luebke's appendix [2009].
   - Use actual (and anticipated actual) names of `RayTracer` methods and members in the code so that they will be linked correctly when you really implement them. With careful use of `\copydoc` and `\copybrief` you can avoid typing the documentation twice.
   - Be sure to give *all* important algorithmic details for the implementation of photon emission and gathering.

Tip: Parts of the photon mapping algorithm will be presented in lecture the Wednesday *after* the project starts, however you can begin now because the handout and papers contain all of that information and more.

2. **Theoretical Analysis**:

   (a) Prove that the emitted photons collectively represent the total emitted power of the lights (i.e., the sum of `G3D::Light::emittedPower` over all lights),

   $$\sum_{P \in photons} \Phi_P = \sum_{E \in lights} \Phi_E \tag{1}$$

   (b) Explain why depositing photons at multiple points along a path doesn't "double count" the same energy and result in an image that is too bright.

   (c) Prove that L_o in equation 8 has radiance units.

Tip: CS371 proofs should be mathematically rigorous, in the same way as in CS361 or a math course.

(d) Prove that your rendering algorithm is mathematically consistent (i.e., that it converges to the true solution as the number of rays and photons approach infinity).

(e) **BSDF Diagrams.** Create a set of BSDF diagrams as 2D schematics of the 3D shape for a fixed angle of incidence, as we commonly draw in lecture. Show the BSDF for red, green, and blue wavelengths (in the appropriate colors) either side-by-side or overlaid on the same image. To save time, I recommend that you draw them by hand on paper or a black/whiteboard and just include the images. Show BSDFs for the following materials. You may wish to include a photograph of the material to support your diagrams.

i. Aluminum foil
ii. Opaque purple plastic
iii. Green cloth
iv. Green glass

3. **Experimental Analysis** ("Results"):

(a) Show a screenshot of the specified user interface.

(b) Render the following classic global illumination test scenes, **attempting to match the reference images as closely as possible**:

Tip: `ifs/ring.ifs`

i. Metal ring from by Jensen [2001] shown in Figure 2(a), `http://graphics.ucsd.edu/~henrik/images/caustics.html`
ii. Cornell box photographic reference by Francois Sillion shown in Figure 2(b), `http://www.graphics.cornell.edu/online/box/compare.html`
iii. Sponza atrium from Pharr's and Humphreys's book shown in Figure 2(c), `http://www.pbrt.org/scenes_images/sponza-phomap.jpg`

In doing so, you are attempting to replicate the results of a experiment, which is one of the cornerstones of scientific research.

(c) Show one image comparing a photograph to a rendered scene (this is one of your custom scenes). In this case you're performing a new experiment instead of replicating an existing one. You can choose an existing photograph–for example, of the real Sponza atrium–or take a photograph yourself. It is a good idea to choose a scene for which you already have a 3D model or for which it would be easy to create a model. Your choice of scene should also be one that takes advantage of photon mapping, and the viewpoint and lighting should be such that a human observer immediately understands that your result is correct (e.g., rendering the inside of a black box does not satisfy this part of the report). If you use a pre-existing photograph, ensure that it is a real, unretouched photograph.

(d) Show other images of custom scenes to demonstrate correctness in specific cases and to support your discussion, as needed. You may collaborate with other groups on this step.

(e) Show one "teaser" image of a custom scene intended to impress the viewer at the top of your report. Plan to spend at least four hours just creating the scene for this image. This is the scene for which you should also experiment with many variations.

4. Feedback

   (a) How many hours per-person did you spend, on average, on required elements of this project? This should include time spent in lab sessions. It should not include time spent on assigned reading or lecture.

   (b) How many hours did you spend making your scene interesting, adding features, or exploring the algorithm beyond the minimum required to implement the specification?

Tip: Your analysis is the most important part of your report. Render a lot of different images, create plots and tables of data, and take the time to explore the behavior of the photon mapping algorithm.

Sample questions: Below are the kinds of questions that you should be asking yourself and discussing in your report. Don't answer these questions specifically–instead, pose and answer your own questions, which may overlap with these. Begin each part of the discussion with either an explicit question in boldface or an appropriate section title, such as "performance." Aspire to the kinds of result analysis presented in scientific papers. See McGuire and Luebke [McGuire and Luebke 2009] for examples of exploring parameter space, comparing, and presenting dense data.

- How do your results compare, quantitatively and qualitatively with the input and output of previously published images in the papers that we've read?
- What illumination effects and types of paths are visible in each result? Conclude that the images confirm correctness or specific errors.
- Are errors due to approximations in the algorithm, the data, or errors in your implementation?
- How do the input parameters affect the quality and performance of results?
- How much time is spent in forward and backward trace steps?
- How expensive is the radiance estimate compared to the trace time?
- What set of parameters gives the best quality for limited execution time. E.g., if photons are less expensive to trace than primary rays, maybe it makes sense to trace more photons and fewer primary rays.

3 Photon Mapping

The algorithm consists of two phases: a **forward photon trace** outward from the light sources and a **backward ray trace** from the eye. These are linked by a **radiance estimate** performed where the paths nearly meet. The points on paths discovered by the forward trace are stored in a **photon map**. The core data structure within the photon map and throughout the photon trace is the **photon**. Each photon P has a position Y_P in meters, an incident direction $\hat{\omega}_P$ (i.e., it propagates along $-\hat{\omega}_P$), and incident power (a.k.a. radiant flux) Φ_P in Watts.

In the following, I denote the average value of a quantity c over all frequencies as $\bar{c}$. This corresponds to `Color3::average`.

3.1 Forward Trace

The forward photon trace computes the photon map, scattering ("bouncing") each photon a limited number of times. Each emitted photon may produce multiple stored photons. Repeat the following $numEmitted$ times:

1. Select an emitter $E \in emitters$ with scalar emission probability ρ_{e} proportional the emitter's relative power, averaged over wavelengths:

$$\rho_{\mathrm{e}}(E) = \frac{\bar{\Phi}_E}{\sum_{F \in emitters} \bar{\Phi}_F}. \tag{2}$$

2. Let E be the selected emitter. Create a new photon P with power[2]

$$\Phi_P \leftarrow \frac{\Phi_E}{numEmitted \cdot \rho_{\mathrm{e}}(E)}, \tag{3}$$

and initial position

$$Y_P \leftarrow Y_E \tag{4}$$

at the emitter. For an omnidirectional point emitter, choose direction $\hat{\omega}_P$ uniformly at random on the sphere. For a spot light, use rejection sampling against the cone of the spot light to ensure that the emitted direction is within the cone.

[2]Note that the power of an individual photon is small when either $numEmitted$ or $|emitters|$ is large. Jensen chooses to scale power such that the average photon has power 1.0 at each wavelength, claiming that this avoids underflow and improves floating point accuracy [Jensen 2001]. I believe this is a misunderstanding of the IEEE floating point format and have not experimentally observed any loss of precision even when using millions of photons.

3. Repeat at most $maxForwardBounces$ times:

   (a) Let X be the first intersection of ray $Y_P - \hat{\omega}_P t$ with the scene. If there is no such intersection, abort processing of this photon by immediately exiting this loop.

   (b) Update the photon by

$$Y_P \leftarrow X. \tag{5}$$

   (c) Store a *copy* of photon P in the photon map.

   (d) Test if the photon scatters. At surface whose reflectivity is ρ_s (which varies with frequency, i.e., is a `G3D::Color3`), let the scalar scattering probability be $\bar{\rho}_s$. If photon P is absorbed instead of scattering, abort processing it by immediately exiting this loop (this is **Russian roulette** sampling of the scattering function).

   (e) Scatter the photon, i.e., choose a new direction $\hat{\omega}_P \leftarrow \hat{\omega}_o$ by importance sampling, ideally with respect to function $\hat{\omega}_P \rightarrow f_{X,\hat{n}}(\hat{\omega}_P, \hat{\omega}_o)|\hat{\omega}_i \cdot \hat{n}|$. Let W be the weight if the distribution sampled is not ideal. Update the power of the photon by equation 6. Note that under ideal importance sampling in a scene with no spectral frequency variation, the power of the photon is unchanged by this step–we're only correcting for bias due to computational limitations of sampling techniques.

$$\Phi_P \leftarrow \Phi_P \, W \, \rho_s / \bar{\rho}_s. \tag{6}$$

If explicit direct illumination is enabled in the GUI, do not *store* photons on their first bounce, but do still scatter them. Always store them on subsequent bounces.

Some advice for the photon forward trace and the photon radiance estimate:

- `G3D::Light::bulbPower` is the power that a spot light would have if it were a point light (which is what you use for direct illumination). The actual emitted power of the light, taking the cone into account, is `G3D::GLight::emittedPower`, which is what you should use during the photon trace. This interface makes it so that lights don't appear to get "darker" at points already within the cone when you adjust the cone angle.

- Be really careful with the photon direction convention. It is confusing for two reasons. First, the propagation direction is different than the incident direction that you eventually want to store. Second, the photon travels between two points during each iteration, and the vectors describing the direction between them point in opposite directions at each end. So your convention will be backwards for half of the iteration no matter what you do. Use assertions heavily and run targeted experiments with small numbers of photon and small numbers of bounces to ensure that you have this correct. There are several places that you will have to negate the photon direction.

- Just as with backwards tracing, you have to bump photons a small amount along the geometric normal of the last surface hit to avoid getting "stuck" on surfaces. If you see strange diagonal patterns of photons or the photons generally have the same color as the surface that they are on, you aren't bumping (or are bumping in the wrong direction!)
- Photons store *incident* illumination–before a bounce.

3.2 Backward Trace

Setting aside the final gathering process, the backward trace is exactly the same as for your previous ray tracing project, except shading now contains a new **indirect illumination** component in addition to the previous **direct illumination** and **specular illumination** components. If explicit direct illumination is disabled in the GUI, do not compute the direct contribution from light sources (but do still compute the specular component).

The indirect component is also called a **radiance estimate**. Consider a point X with normal $\hat{n}$ and direction $\hat{\omega}_\mathrm{o}$ to the eye (or previous intersection), that we reached by backwards tracing. If there were many photons stored at X in the photon map, that would tell us the incident radiance. We could apply the BSDF and know the outgoing radiance to the eye.

However, it is extremely unlikely that any previously-traced forward path will terminate *exactly at* Y. So we estimate the incident flux from photons *near* X, which are discovered by gathering all photons within the neighborhood of X from the photon map. As we make our definition of "near" more liberal, the indirect illumination will become blurrier. As we make it more conservative, the indirect illumination will be sharper–but also noisier.

The reflected (outgoing) radiance estimate is given by:

1. Let $L_\mathrm{o} \leftarrow 0\ \mathrm{W}/(\mathrm{m}^2\mathrm{sr})$ be the initial estimate of radiance reflected towards the viewer.
2. Gather the photons nearest X from the photon map. Two methods for doing this are growing the gather radius until a constant number of photons have been sampled, and simply gathering from a constant radius. Regardless of the method chosen, let r be the gather radius in meters.
3. For each photon P within radius r of X,

$$\text{Let } \beta = \frac{\Phi_P}{\displaystyle\int_0^r\int_0^{2\pi} \kappa\left(s\right) \cdot s \; d\theta \; ds} \cdot \kappa\left(||Y_P - X||\right) \tag{7}$$

$$L_\mathrm{o} \leftarrow L_\mathrm{o} + \beta \cdot f_{X,\hat{n}}(\hat{\omega}_P, \hat{\omega}_\mathrm{o}) \tag{8}$$

Compute the double integral in Equation 7 by hand; the denominator should be a constant over the loop. Note that determining the units of this equation is key to one of the proofs that you are required to write in your report.

Function $\kappa()$ is the 2D falloff[3] for which Jensen recommends a cone [Jensen 1996] filter, $\kappa(x) = 1 - x/r$. When debugging, it is easier to first choose to have no falloff within the gather sphere, i.e., $\kappa(x) = 1$. In that case, the double integral is equal to the area of the largest cross-section of the sphere, πr^2. To help build some intuition for this, recall one derivation of the area of a disk of radius r. The area of a small rectangular patch on a disk at radius s is $(s \cdot d\theta) \cdot ds$; the length of this patch along the radial axis is clearly ds and, in the limit as $d\theta \to 0$, the length perpendicular to the radial axis is the length of the arc at radius s: $sd\theta$. The integral of the patch area expression over the full circle and disk radius is

$$\int_0^r \int_0^{2\pi} s \; d\theta \; ds = \frac{1}{2} s^2 \cdot 2\pi \Big]_0^r = \pi r^2. \tag{9}$$

The more general expression given in Equation 7 is the "area" of a disk of variable density, where the density at distance s is $\kappa(s)$.

Tip: When you disable the explicit direct illumination, the noise and blurriness should increase, but the intensity of all surfaces should be the same. Test this with 1-bounce photons so there is no "indirect" light to confuse the issue.

[3] This κ has nothing to do with the attenuation constant that is similarly notated when discussing index of refraction and transmission.

4 Advice

4.1 Scheduling

A large part of this class, and this project, is about managing development of large projects. It would take six months to implement this project fully and perfectly. You have five business days to complete it. Obviously, you can't afford to perform every possible experiment and polish every aspect of the project.

Decide what level of quality you want for each element and then schedule to that. All projects make tradeoffs. Choose yours proactively and intelligently, instead of letting the clock run out and then choosing them them retroactively and randomly. Success doesn't depend on completing everything perfectly, it depends on choosing where to spend your time appropriately, and then excelling at only the important parts.

Keep in mind that you don't have to work right up to the deadlines. For example, if you are planning an aggressive midterm project, you might want to finish off this project in the first week and then start the midterm early.

A 2pm checkpoint doesn't mean that you have between 1pm and 2pm to accomplish it. It just means that it is due at 2pm. You could complete it days ahead of time if that fits your schedule better.

4.2 Algorithm Variations and Resources

Photon mapping algorithm was introduced by Jensen and Christensen [1995] and refined by Jensen [1996]. Use those primary sources as your guide to the algorithm. Additional information about the algorithm and implementation is available in Jensen's book [2001] and course notes [2007]. The notes are almost identical to the book and are available online. Section 3 of this project document summarizes mathematical aspects covered in class that are ambiguous in the original papers. Note that Ma and McCool [2002] observed that a hash grid is superior to a k-d tree for photon gathering.

Ignore the 1995 implementation of the photon class. Instead use a variant on the 1996 implementation: store position, incident direction, and power for each photon. Both papers recommend compressing photons. Don't compress your photons–it makes the implementation hard and likely will give no performance advantage. Use a natural floating point representation instead of a compressed one. Also do not store surface normals in the photons.

Use a constant-radius gather sphere (as described on page 7 of the 1996 paper). When you have that working, you may optionally implement the ellipsoid gathering method or the k-nearest-neighbor method and compare performance and image quality. Note that gathering from a large sphere (or box) from a data structure and then choosing only the k-nearest neighbors may be more efficient than repeatedly querying the data structure.

Beware that the description of the trace from the eye sounds more complicated in Jensen's papers than it actually is. You're just going to extend your existing ray tracer by adding indirect light to the direct illumination at every point.

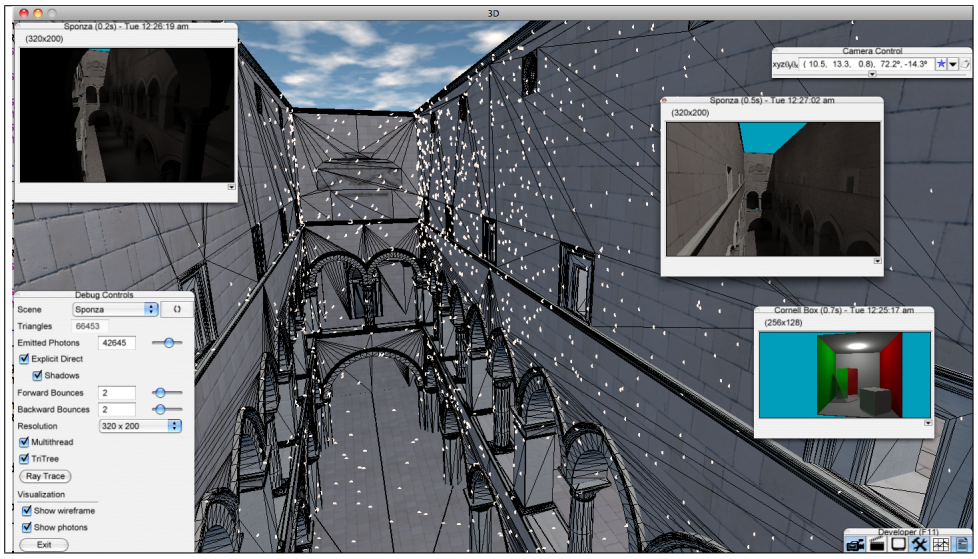


Figure 3: *User interface and debugging visualization for large numbers of photons.*

4.3 Visualization

Implement the GUI and photon visualization first because they are essential tools for debugging. When implementing the forward (photon) trace, I recommend making a button to trace the photons but not actually produce an image. That will reduce your debugging time. Likewise, some of you considered an optimization to only compute the TriTree on load, rather than every time that the scene is rendered. That optimization could save you a lot of time in this project.

Visualize small (< 1000) numbers of photons as arrows pointing along their $\hat{\omega}_i$ directions. For large numbers of photons, visualize them as points. In each case, set the color based on the photon power. Because the power of each photon is fairly small, normalize the power so the largest component is 1.0.

Use `G3D::Draw::arrow` to render arrows in `App::onGraphics3D` after the visible parts of the scene are rendered. To render points, use something like[4]:

```
rd->setPointSize(5);
rd->beginPrimitive(PrimitiveType::POINTS);
for (PhotonMap::Iterator it = m_photonMap.begin(); it.hasMore(); ++it) {
    rd->setColor(it->power / it->power.max());
    rd->sendVertex(it->location);
}
rd->endPrimitive();
```

4.4 Program Trace

You will probably also want to instrument your forward (photon) trace to print information after each scattering event and then trace a small number of photons in order to verify that the importance sampling is working correctly. Use `G3D::debugPrintf` to output to the OS X console. Beware that if your pro-

[4] This is one of those cases where the difference between `++it` and `it++` can be significant

gram crashes, `G3D::debugPrintf` may not actually print the last output before the crash. In that case, use `G3D::logPrintf` to write to `log.txt`, which is guaranteed to complete before the function returns. Of course, gdb is often the best tool for debugging a program that crashes.

4.5 GuiTextureBox

The `G3D::GuiTextureBox` interactive inspector allows you to see the radiance values that your program wrote to the image. Remember that you can zoom in and out and change the exposure of this box. If you see unexpected black or white areas, hold the mouse over them to see their floating-point values. A value of `nan` or `inf` means that somewhere in your code you divided by zero or performed another undefined mathematical operation. You can create a `GuiTextureBox` explicitly or use the one that `GApp::show` produces.

4.6 G3D::PointHashGrid

You may want to use the `G3D::PointHashGrid` class on this project. It uses the C++ **trait** design pattern to support arbitrary keys. This is useful for cases where a data structure must work with a class that may not have been designed to work with it, and therefore does not have the right interface. The idea of this design pattern is that adapter classes can describe the *traits* of other classes.

`G3D::PointHashGrid<T>` requires a helper class that tells it how to get the position of an instance of the `T` class; in this case, `T` is `Photon`. The helper class must have a static method called `getPosition`. `PointHashGrid` also requires either an `operator==` method on the `Photon` class or another helper that can test for equality. See the documentation, which offers examples on this point.

`G3D::PointHashGrid` is not threadsafe, so you'll have to protect it with a mutex if you plan to write from multiple threads. But photon tracing is usually the fastest part of the program. So keep the forward trace single-threaded and avoid the complexity and cost of locks.

References

ASHIKHMIN, M., AND SHIRLEY, P. 2000. An anisotropic phong brdf model. *J. Graph. Tools 5*, 2, 25–32.

CHEN, H., AND LIU, X. 2008. Lighting and material of halo 3. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, 1–22. 1

JENSEN, H. W., AND CHRISTENSEN, N. J. 1995. Photon maps in bidirectional Monte Carlo ray tracing of complex objects. *Computers & Graphics 19*, 2, 215–224. 1, 12

JENSEN, H. W., AND CHRISTENSEN, P. 2007. High quality rendering using ray tracing and photon mapping. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, New York, NY, USA, 1. 5, 12

JENSEN, H. W. 1996. Global illumination using photon maps. In *Rendering Techniques*, 21–30. 1, 3, 11, 12

JENSEN, H. W. 2001. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA. 6, 8, 12

MA, V. C. H., AND MCCOOL, M. D. 2002. Low latency photon mapping using block hashing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 89–99. 12

MARTINEZ, A., 2010. Faster photorealism in wonderland: Physically based shading and lighting at sony pictures imageworks, August. in Physically Based Shading Models in Film and Game Production SIGGRAPH 2010 Course Notes. 1

MCGUIRE, M., AND LUEBKE, D. 2009. Hardware-accelerated global illumination by image space photon mapping. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, ACM, New York, NY, USA, 77–89. 5, 7

Index

backward ray trace, 8
backward trace, 10

consistent, 1

direct illumination, 10

forward photon trace, 8

G3D::debugPrintf, 13
G3D::Draw::arrow, 13
G3D::GApp, 14
G3D::GuiTextureBox, 14
G3D::PointHashGrid, 14
G3D::PosFunc, 14

indirect illumination, 10

Photon, 14
photon, 8
photon map, 8
Photon mapping, 1

radiance estimate, 8, 10
Russian roulette, 9

specular illumination, 10

trait, 14