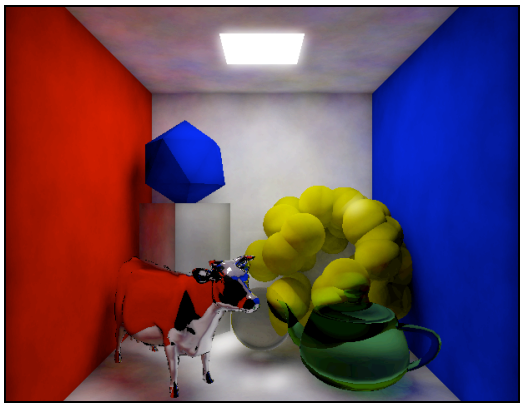
CS371 Spring 2007
Prof. Morgan McGuire

Assigned: Mon. 2/26/07
Due: Mon. 3/5/07 8:00 pm

Project 0x03: Recurse




For this project you will extend your bidirectional ray tracer with advanced features. You are only required to implement two new features (of your choosing) because a large part of the challenge of this assignment is in figuring out how to implement those features.

Please create many images and scenes to show off your features both technically (e.g., demonstrate specific illumination effects in simple scenes) and artistically. For example, if you implement refraction, show an object distorted by a glass sphere, a caustic formed under the sphere, and an artistically interesting scene like a vase full of colored glass marbles.

Previous assignments have required you to design the ray tracing system on your own to help you really learn how the internal details function. Now that you have that experience and appreciate various design choices, this handout recommends a particular structure for a recursive bidirectional ray tracer and photon map. Certain features may be hard to implement unless your structure roughly corresponds to the one described here.

This is the last photorealistic assignment using the ray tracer. Next week you'll start with new code.

Recurse Specification

Implement a bidirectional ray tracer with photon mapping that simulates light at three wavelengths through multiple bounces (i.e., the Scatter project from last week).

Next, add two of the following numbered features:

1. Reflection and Refraction (i.e., specular BSDFs)
2. Transformation objects:
    a. Translate
    b. Rotate
    c. Non-uniform scale (be careful with surface normals)
3. Metaball implicit surfaces
4. Constructive solid geometry (i.e., "Booleans") nodes:
    a. Union
    b. Intersection
    c. Subtraction
5. Final gathering
6. Fresnel BSDF (requires refraction)
7. Compound BSDFs for, e.g., reflect + diffuse (requires other BSDFs...)
8. Explicit direct illumination and shadow rays
9. Blinn-Phong, Torrence-Sparrow, anisotropic, or other another classic BSDF
10. Adaptive super sampling
11. Texture-mapped triangle mesh with support for
    a. Quake 3 BSP files (See G3D::Q3Map) or
    b. Quake MD2 models (See G3D::MD2Model)
12. Participating media like fog
13. Depth of field (i.e., physically correct lens simulation)
14. Full tone mapping (i.e., bloom, global adaptation, local adaptation, and gamma correction) and anti-aliasing
15. Distributed ray tracing across multiple computers (check out the GD ReliableConduit and Discovery APIs for easy networking)
16. HDR light probe illumination

You may choose to use any language, platform, and libraries you wish, although only C++ and G3D on the department's FreeBSD systems are supported. If you use C++ as I recommend, please use Java (yes, Java) coding conventions for your C++ code.

Implementation Tips

This handout intentionally does not describe most of the features that it asks you to implement. That is because you can find excellent descriptions of them in Real Time Rendering and on the internet. In fact, the primary skill that you will learn this week is translating an out-of-the-book description of an illumination phenomenon into code in the context of your own ray tracer.

Some basic concepts will help you organize your thoughts about the new features and help you understand what you read in other resources.

Path Classification

"LSDE" notation is a common shorthand for discussing the various multiple scatter paths that light takes from the source to the eye. The idea is that any light path through the scene can be described as a string indicating the bounces. That string contains the characters:

L – Light source (this is not the radiance variable—different notation)

S – Specular (mirror or refraction) scatter

D – Diffuse scatter (not necessarily perfectly Lambertian)

E – Hit the eye (camera)

For rendering an image, all of the paths we care about begin with L and end with E. In between they may bounce multiple times. For example, direct illumination paths have the form LDE or LSE. The light that bounces off a diffuse red wall, hits a white floor, and then is seen by the camera follows the path LDDE.

To compactly classify large categories of paths, BNF grammar notation is used:

[A | B]	Interaction of type A or interaction of type B
A^+	One or more interactions of type A
A*	Zero or more interactions of type A

Using this notation, we can define different kinds of light paths:

L [S|D] E	Direct (one bounce) illumination
L [S|D]* D	Indirect (multiple bounce) illumination
L[S|D]S^+E	Specular: reflections and refractions
LE	Emitted illumination (looking straight at the light)

This notation appears in books and online. It is also a good way to organize your thinking. It is important that different methods of estimating radiance do not take the same paths into account—if they did, you would double-count some illumination. Which paths are represented in the photon map? What do the different estimation functions in your backward tracer measure?

Basic Program Structure for Bidirectional Raytracing

Build Scene:

1. create all BSDF instances, store in a Table or Array
2. add all objects to the scene
3. balance the scene tree

Forward Trace:

REPEAT numPhoton times:

1. Let P = randomly emitted photon
2. REPEAT:
    2.1. advance P slightly (0.0001) along its propagation direction so that it doesn't immediately intersect its own source.
    2.2. Let hit = first intersection between P and the scene
    2.3. move P to hit position
    2.4. store P in photon map
    2.5. scatter P according to the BSDF at hit, or kill it
3. UNTIL killed
4. balance the photon map

Backward Trace:

FOR each pixel (x, y):

1. Let R = ray through (x, y)
2. Let $\eta = 1.0001$
3. set image (x, y) = estimate radiance L(R.origin, R.direction, η)

Estimate Radiance(R, η**)** -- during Backward Trace:

1. Let hit = first intersection between R and the scene
2. $L_{direct} + L_{indirect}$ = radiance estimate from photon map at hit, η
3. $L_{emitted}$ = radiance estimate from emitter at hit
4. Let (R', $\rho_{reflect}$, η') = mirror scatter using BSDF at hit
5. $L_{reflected}$ = estimate radiance L(R', η') * $\rho_{reflect}$
6. Let (R', $\rho_{refract}$, η') = refraction using BSDF at hit
7. $L_{refracted}$ = estimate radiance L(R', η') * $\rho_{refract}$
8. RETURN $L_{direct} + L_{indirect} + L_{emitted} + L_{reflected} + L_{refracted}$

Estimate Photon Map Radiance(bsdf, x**,** ω_o**,** η**)** – during Radiance Estimation

Estimate the amount of radiance from photons that bounced at x *in direction* ω_o

1. Let photons = all nearby photon map photons within a sphere
2. Let L = 0
3. FOR p in photons:

   3.1. IF (p is not too far away along the surface normal direction):
   (*We skip distant photon that would cause light seepage at sharp corners*)

   3.1.1. Let ω_o = photon direction

   3.1.2. Let Φ = photon power

   3.1.3. L += bsdf(x**,** ω_o, ω_i η) * Φ
4. RETURN L / (numPhotons * sphere area)

BSDF Class

Your BSDF class should support the following functionality through methods. The η variables and specular refraction method are only needed for refraction and can be omitted if you aren't implementing translucent materials.

- $(\omega_o, \eta_o, \Phi_o)$ = diffuseForwardScatter(n, η_i, ω_i, Φ_i)

    Select a direction randomly from the scattering distribution. I.e., use "Vector3::cosRandom" for Lambertian surface and compute the flux as was demonstrated in the lecture notes. You need some way of representing absorption in the return value. Good ideas are returning a boolean or setting the outgoing flux to zero.

- f = evaluate(n, ω_i, ω_o, η_i)

    Evaluate the BSDF; this is constant for a Lambertian surface

- $(\omega_o, \eta_o, \rho_o)$ = specularRefractionBackwardScatter(n, η_i, ω_i)

    For a transmissive material, compute the transmissivity and angle of refraction. Use "Vector3::refractionDirection"

- $(\omega_o, \eta_o, \rho_o)$ = specularReflectionBackwardScatter(n, η_i, ω_i)

    For a mirror reflective material, compute the reflectivity and angle of reflection. Use "Vector3::reflectionDirection"

I recommend creating all of the BSDFs for your scene once at the beginning of the program and storing them in Array<BSDF*> bsdfArray. Then you can just pass BSDF pointers around (which is efficient), and call bsdfArray.deleteAll() at the end of your program to reclaim memory.

Cosine, uniform, and specular distributions are easy to sample over because G3D provides routines for them. It is hard to sample from other distributions. What you can do is sample from a cosine distribution and them modulate the photon power by the PDF for that distribution. For example, in a BSDF instance that looks like f = (ω dot n), the sample function should return cosRandom() for the direction and flux = (ω dot n) * reflectivity.

Refraction

Refraction will require more code than reflection, and that code will spread throughout your system because you'll have to track the η variables.

Make sure that your "intersect" method for the whole scene flips the surface normal it found if ray.direction dot normal > 0. Otherwise you'll refract the wrong way when you hit the back of a surface. I found it easier to fix these in the

Raytracer's intersect method than correcting the surface normal for every Entity in the scene.

To implement refraction, you need to keep track of the refractive index of the medium that a photon is currently in, and pass this to the BSDF's scatter methods.

G3D's Vector3 reflectionDirection and refractionDirection assume that the incident vector is the direction of photon propagation; that is, it is $(-\omega_i)$.

Innovate!

Why stop at two new features?

You will receive extra credit for interesting scenes as well as additional features. A teapot and a cow in the Cornell box is not interesting. A palm tree on a sun-drenched white-sand beach with waves rolling in is very interesting.

Submitting Your Solution

1. Put your name, e-mail address, and the name of the file in a doxygen comment at the top of **each file**.

2. Create a "doc-files" directory that contains a **readme.html** file with your name, e-mail address, partner's name (if you worked in a pair) and anything you'd like to point out to me when I'm reviewing your program.
    - This HTML file must display at least **two** JPG screenshots (stored in the same doc-files directory) of your program in operation that prove that the features you have added are working correctly.
    - CLEARLY IDENTIFY the primary location of the code for your specific features by file and line number.
    - If there are known bugs, extra credit features, or design points of note, list them here.
    - Credit any code that you used from someone else's Spheres project or found on the internet.
    - If you are using your 2-day grace period or a prearranged deadline extension, explain that in the readme file.

3. Delete all generated files using "icompile --clean". Do not hand in a build this week.

4. Change to the **parent** directory of your project and run the FreeBSD command:

```
submit371 recurse
```