Raytracing

CIS 460/560 Homework 5

Due Monday, December 9, 2013 at 11:59 am

1 Goal

This is the big one. You'll be making a full, recursive raytracer to handle a more realistic rendering of your room. In the process you'll be implementing the Blinn-Phong light model and also adding materials to your configuration file.

2 Bigger Picture

You've almost done it all. You've made a volumetric renderer and even generated clouds using Perlin noise. You made a scene graph and got some exposure to OpenGL 3.2 along with GLSL 1.5.0. You made some procedural models to spice up your scene. Now, you know how to intersect some simple geometry. Time for the computer graphics rite of passage: the raytracer.

3 Software Environment

The same requirements of code working in Moore 100 are still in effect. Since raytracing is slow, we require that your code can be compiled in Release mode instead of just Debug mode. You'll be combining elements of homework 1, 3, and 4 as well as adding the necessary code. Thanks to GLM you can use the same matrices to render OpenGL as you can use to process the rays in your raytracer.

4 Supplied Code

None, but you will want to grab HW1's EasyBMP code and drop it in.

5 Requirements

5.1 The "P" Key (10 points)

Add to your keyboard input the p key. When pressed, this will initiate the raytrace using the parameters from the configuration file and the current state of the scene graph. With this key implemented you will use the program by loading in a configuration file at runtime, tweak the position of objects if desired, then push "P" to get the raytraced image output as a BMP file. You need to show some sort of progress bar during the run of the raytracer. The easiest way to do this is print the current horizontal line (y-value) when the raytracer moves down. You could also use a graphical progress bar but it's not as easy to get it right as it may seem.

5.2 Camera (10 points)

The camera's specifications will come from your configuration file, not the current view of the scene graph. The camera's position in world-space will be provided, as well as its forward direction and up-vector. Consult your notes on how these parameters are used. You will use a perspective projection with a specified FOV in the vertical axis. If this sounded familiar, it's because you did it already all the way back in September with the first assignment.

5.3 Materials and Reflection (50 points)

In a real raytracer like Radiance by Greg Ward, materials can be arbitrarily defined for any given piece of geometry. In this simplified raytracer, you will be responsible for 3 dynamic materials. They will be defined in the configuration file between the first line of numbers and the geometry locations. An example is available on Blackboard. You will need to make a static material for the floor.

A material consists of a diffuse/ambient color, a specular exponent (sometimes known as shininess), and a reflectivity coefficient. If the reflectivity coefficient is zero, the object is not reflective and the color at a particular point is determined entirely by the Blinn-Phong lighting model: the sum of diffuse, specular, and ambient contributions. If the reflectivity coefficient is 1, the Blinn-Phong-lit color is ignored, and the color is exactly the color visible in the reflection-ray direction. If the reflectivity coefficient is between 0 and 1, the color at a particular point is the weighted combination of the Blinn-Phong lighting result, and the color visible in the reflected direction. As usual with raytracing, reflection is performed recursively: if the reflection ray hits another reflected object, this causes another reflection ray. You should limit the maximum depth of the raytrace; we suggest a limit of 5.

5.4 Lighting and Shadows (30 points)

There's a single point light, with a provided color. You may add additional lights for extra credit. In a slight break from reality, there is no illumination falloff: a point light source will illuminate an object to an equal extent regardless of the object's distance from the light.

Objects may be shadowed by other objects; if another object is between a given object and the light point, there should not be any diffuse or specular light cast on the object. The object still has an ambient contribution, though, and shadow rays do not affect reflection. Even transparent objects (if you implement transparency for extra credit) should cast shadows. Shadows are a trademark of raytracing.

6 Configuration File

The following additions complete the input file format for this chain of assignments! At this stage, it has been vaguely inspired by Radiance. Even though the lines are now labeled they will appear in the same order seen below, it's just easier to keep track of them for editing/debugging purposes with the labels since there are so many numbers to keep track of. All numbers (except RESO/resolution) are floating point.

```
FILE outfile.bmp
RESO 640 480
EYEP 5 5 5
VDIR 1 0 0
UVEC 0 1 0
FOVY 45
LPOS 10 3 3
LCOL 1.0 1.0 1.0
ACOL 0.2 0.2 0.2
MAT1 0.5 0.5 0.5 0.25 0.0 0.0 0.0 0.0
MAT2 1.0 0.0 0.0 4.0 0.3 0.0 0.0 0.0
MAT3 0.0 1.0 0.3 4.0 0.3 0.0 0.0 0.0
```

The lines have the following meaning:

- FILE: The filename to give the raytraced image.
- RESO: Two integers specifying the width and height (in pixels) of the raytrace that should be performed.
- EYEP: The position of the eye/camera in world space.
- VDIR: The viewing direction of the center of the raytraced rendering, in world space.
- UVEC: The up-vector in world space.
- FOVY: The half-angle field of view in the Y-direction, in degrees.
- LPOS: The position of the point light in world space.
- LCOL: The color of the point light, in RGB order.

- ACOL: The color of the ambient light, in RGB order.
- MAT1, MAT2, MAT3: The description of the materials for the scene. The first three numbers give the diffuse color of the material, in RGB order. The fourth number is the specular exponent. The fifth number is the reflectivity parameter. The sixth, seventh, and eighth numbers are for additional material properties that may be ignored if not doing some extra credit options.
- When defining geometry in your configuration file, instead of the RGB values, there will be a single value; 1, 2, or 3. This indicates which material to use for that geometry.

7 Output

The raytraced image will be output to a BMP in the program's working directory. You don't have to write the BMP saving function, we'll give you one.

8 Extra Credit (50 points maximum)

8.1 Basic transparency (10 points max)

Make it possible for a material to be transparent, with a provided index of refraction. Transparency is accomplished by recursively casting a ray from the point of intersection, just like with reflection. The transparency will be given as the sixth number in the MAT parameter lines, as a 1.0 for transparency and 0.0 for no transparency. There will be no translucent objects; it's all or nothing. Transparent objects will not be lit with Blinn-Phong, but may still have a nonzero reflectivity. Just as with Blinn-Phong, if there is reflectivity you will want to use a weighted sum of the reflected component and the transparent component. If an object is transparent the RGB numbers of the MAT line will give the material color; that is to say light going through the object will be modulated by that color.

8.2 Physically correct refraction (10 points max)

The index of refraction will be given as the seventh number in the MAT parameter lines. Empty space has an index of refraction of 1. This is in addition to the 10 points you'll get implementing the basic transparency.

8.3 Fresnel reflection (10 points max)

Even more physically accurate, you will ignore the reflectivity parameter of transparent objects and use Fresnel equations instead. This is in addition to the 20 you'd have to earn from basic transparency and refraction.

8.4 Bounding spheres (10 points max)

For each object, define a bounding sphere. Test intersection with this sphere before checking intersection with the object's actual geometry. This will greatly speed up your rendering.

8.5 Multiple lights (10 points max)

Support 2 or 3 lights in your configuration file. Explain in your readme how your format has changed and supply some samples.

8.6 Something else! (50 points max)

Let Dr. Badler and the TAs know via email if you think you have a really cool idea and we'll discuss it with you. The points for this extra credit vary based on the idea, but could be as high as 50. You have to let us know far in advance of your idea; approximately when the unit tests are due.

9 Notes

- When you generate a recursive ray, make sure you don't immediately intersect the same object it's coming out of. Deltas and epsilons will come in handy here.
- Start early! Part of the legendary difficulty of this assignment is directly traceable (pun not intended) to students that generate their first image 1-2 days before the deadline only to see a complete mess rather than what they expected.
- When debugging the raytracer, have paper and pencil ready to do the math yourself to help see where the code is wrong. Dropped negatives and faulty dot-products (using the w-component) are some errors that can only be discovered by doing the math by hand and seeing your program do it wrong as you step through. Get your paper-pencil equations from the notes, NOT your code! The processor's mathematical capabilities are fine!
- A great raytracer is an excellent addition to a demo reel, so go all-out on this assignment. It's an assignment that will keep giving back!
- Some of the most impressive raytraced images rely heavily on spheres with different material properties all playing together. We did not make you define a simple sphere geometry, but you may consider doing that on your own sometime (it's very simple).

10 Deliverables

You will submit a zipped folder via Canvas containing your Visual Studio project and a readme file with anything you think the graders need to know. Submit some of your BMP renderings too! Please make note of the extra credit you implemented, or you may not get credit for it. As always, please test your code in Moore 100 before submitting!