Assignment 4: Ray

Comp175: Introduction to Computer Graphics – Fall 2022

Algorithm due: **Tuesday November 15** at 1:30pm
Project due: **Monday November 21** midnight at 11:59pm

1 Introduction

In `Intersect`, you saw a glimpse of what you could do with a rendering algorithm that stressed quality over speed. Truly curved surfaces are possible, and everything has a sort of "perfect" feel to it. As you may have noticed, there are a few things that our renderer does not yet handle. For example, what happens if you have a shiny surface? What about texture mapping? Shadows? These issues and more are addressed in this assignment. In `Ray`, you will be filling out the renderer you wrote for `Intersect` to support reflections, texture mapping, shadows, more advanced lighting, and perhaps even transparency, motion blur, spacial subdivision, or bump mapping.

The demo for this project is similar to `Intersect`. Try some of the new XML scene files, and try to implement some of the more advanced options in ray tracing.

There is no additional support code for `Ray`; in this assignment, you are building entirely on your `Intersect` code.

2 Requirements

For this project you are required to extend your `Intersect` code to implement additional features in your ray tracer. Your ray tracer must be able to handle:

- Reflection
- Texture mapping for, at least, the cube, cylinder, cone, and sphere[1]
- Specular highlights
- Shadows
- Point vs. directional lighting
- User-specified depth of recursion

In addition, you need to create a new scene file (i.e. a .xml file) that showcases your recursive ray tracer. This scene file can be based on the one you created in Assignment 3, or a completely new one that you think is fun. Note that you're welcome to include the use of texture maps, but when you do that, remember to include the texture map as part of your provide (also, please keep the size of your texture map to something reasonable).

To calculate the intensity of the reflection value, you need to determine the reflection vector based on an object's normal and the look vector. You then need to recursively calculate the intensity at the intersection point where the reflection vector hits. With each successive recursive iteration, the contribution of the reflection to the overall intensity drops off. For this reason, you need to set a limit for the amount of recursion with which you calculate your reflection. You must make it possible to change the recursion limit easily[2] because I may want to change it during grading.

To review, the lighting model you will be implementing is:

$$I_\lambda = k_a O_{a\lambda} + \sum_{i=1}^{m} \left[f_{att_i} I_{i\lambda} * (k_d O_{d\lambda}(\hat{N} \cdot \hat{L}_i) + k_s O_{s\lambda}(\hat{R}_i \cdot \hat{V})^f) \right] + k_s O_{r\lambda} I_{r\lambda}$$

[1] you will likely need to modify your SceneParser to parse texture files. HINT: you are welcome to use the ppm parser from a previous lab.

[2] By "easily," we mean that you should have this as an interface option.

Here, the subscripts a, d, s, and r stand for ambient, diffuse, specular, and reflected, respectively.

I_λ	$=$	final intensity for wavelength λ; in our case the final R, G, or B value of the pixel we want to color
k	$=$	a constant coefficient. For example, k_a is the global intensity of ambient light; `SceneGlobalData::ka` in the support code
O	$=$	the object being hit by the ray. For example, $O_{d\lambda}$ is the diffuse color at the point of ray intersection on the object
m	$=$	the number of lights in the scene
f_{att_i}	$=$	the attenuation for light i (which you are not required to do for this assignment)
$I_{i\lambda}$	$=$	intensity of light i for wavelength λ
$\hat{N}$	$=$	the unit length surface normal at the point of intersection
$\hat{L}_i$	$=$	the unit length light vector to light i
$\hat{R}_i$	$=$	the unit length reflected light from light i
$\hat{V}$	$=$	the normalized line of sight
f	$=$	the specular component
$I_{r\lambda}$	$=$	the intensity of the reflected light

3 Testing

Your ray tracer's output should look like the demo (for a given scene file and render settings).

4 FAQ

4.1 Cannot Load PPM

You will need to modify the paths to the texture maps in the scene files that came with Assignment 3. This is for two reasons: (1) the current path is something I hard coded to work on my computer, and (2) because of Mac / Windows (and depending on where you launch your application), the relative path / absolute path issue will likely arise. You probably had to deal with this already in Lab 4 (Paint), so check your solution there if you're stuck.

4.2 Texture Mapping SNAFUs

When texture mapping planes you need to be careful. If you're texture mapping the negative z face of the cube, you'll be mapping the intersection point's x position to the u in (u, v) space. The problem is when you go left-to-right on that face, your x values are actually going from positive to negative. This isn't the only cube face that something like this will happen on, so check each face to make sure.

To texture map the cone, just do it the same way as a cylinder (except there's only one cap, of course).

4.3 My Secondary Rays Keep Failing

Recall from the lecture that you need to move your intersection point a little bit away from the surface. Be careful though, one tendency is to move it by EPSILON. However, if EPSILON is also used to check if a point is close enough to the surface, then the two EPSILON will "cancel" each other out. As such, the amount to move the intersection by should be slightly larger than EPSILON.

5 Extra Credit

`Ray` is one of the coolest projects you'll ever write at Tufts. You, yes, you, can make it even cooler by doing some sweet extra credit. Here are some ideas (book sections are included if there's significant discussion of the topic).

- **Antialiasing** Brute force supersampling isn't hard to do and antialiased images look really sexy. If you're feeling brave, try your hand at adaptive supersampling or stochastic supersampling
- **Transparency and Refraction**
- **Motion blur**
- **Depth-of-field**
- **Fewer intersection tests** Bounding volumes, hierarchical bounding volumes, octrees, and kd-trees are all things to try that will get big speedups on complex scenes since most of the clock cycles go to intersection tests. Mucho bonus points if you do one of these. Really fast intersection tests might get you a few points too, but only if they're really good. It is highly encouraged that you do some sort of spacial subdivision, but certainly not required at all.
- **Bump mapping** Like texture mapping, except each texel contains information about the normals instead of color values. It's a great way to add geometry to an object without having to actually render the geometry.
- **Texture mapping and/or intersecting other shapes**, like the torus
- **Optimizations** Be careful here. "Premature optimization is the root of all evil"[3]. You'll learn that he's right at some point in your career, but let's not learn that lesson on `Ray`. Get the basic functionality done then go for the gusto. This isn't as important as it once was now that we have blazing fast machines, admittedly, but it's still awful fun. Multithreading Raytracing is embarrassingly parallel because a ray does not depend on the outcome of any of the other rays. Each ray cast per scanline can be made into its own thread.
- **Texture filtering** (bilinear, trilinear, what have you)
- Whatever else you can think of!

6 How to Submit

Complete the algorithm portion of this assignment with your teammate. You may use a calculator or computer algebra system. All your answers should be given in simplest form. When a numerical answer is required, provide a reduced fraction (i.e. 1/3) or at least three decimal places (i.e. 0.333). Show all work.

For the project portion of this assignment, you are encouraged to discuss your strategies with your classmates. However, all team must turn in ORIGINAL WORK, and any collaboration or references outside of your team must be cited appropriately in the header of your submission.

Submit your assignment using Canvas as usual.

[3]Knuth