CS 3258/5258 Spring 2021 Assignment 5
Ray Tracing

April 6, 2021

1 Purpose

In this assignment you will construct a module to the CLI to implement a simple ray tracer. The ray tracer will be able to render images containing simple primitive elements and display the result on the screen. The result should be writable to a file using tiffwrite.

2 Due Date

The assignment is due by midnight on Tuesday, May 4, 2021. The last day that this assignment can be turned in, *regardless of late days*, is midnight on Thursday, May 6, 2021.

3 Ray Tracing

3.1 Command Format

Implement ten additional commands in your CLI, as follows. The format is:

Command	Parameters	Description
Screen	n_x, n_y	Set the screen dimension
Orthocamera	none	set an orthographic camera
Camera	$e_x, e_y, e_z, g_x, g_y, g_z, u_x, u_y, u_z, s, a_u, a_v, b_u, b_v$	position the camera and screen
Background	r, g, b	Set the background color
Sphere	$R, c_x, c_y, c_z, a_r, a_g, a_b, r_r, r_g, r_b, s_r, s_g, s_b$	create a sphere
Triangle	$u_x, u_y, u_z, v_x, v_y, v_z, w_x, w_y, w_z, a_r, a_g, a_b, r_r, r_g, r_b$	create a triangle
Box	$u_x, u_y, u_z, v_x, v_y, v_z, a_r, a_g, a_b, r_r, r_g, r_b$	create an axis-aligned box
Ilight	$l_r, l_g, l_b, d_x, d_y, d_z$	create a light at infinity
Clear	none	Clear the screen, object lists, and reset the background color.
Trace	none	render the image

All these commands take floating point arguments. NOTE: Any value of color in this assignment should be between 0 and 1, inclusive.

3.2 Implementation Details

3.2.1 Screen

Assume a screen and window of width n_x and height n_y.

3.2.2 Orthocamera

Assume an orthographic camera looking down the negative z axis. The origin is in the lower left corner. Rays will be shot out through each pixel parallel to the z-axis, i.e., for pixel (i, j), $\mathbf{p}(t) = (i, j, 0) + t\mathbf{d}$, where $\mathbf{d} = (0, 0, -1)$. Screen size can be set with the command:

```
glutReshapeWindow(ImageWidth,ImageLength);
```

3.2.3 Camera

Assume a pinhole camera implementing a perspective view as discussed in the lectures. The parameters e_x, e_y, e_z are the coordinates of the eyepoint, g_x, g_y, g_z are the gaze direction, u_x, u_y, u_z are the up vector, s is the distance to the screen, and a_u, a_v, b_u, b_v are the dimensions of the screen.

3.2.4 Background

Set the background color to (r, g, b). Note that $r, g, b \in [0, 1]$. The default should be black, $(0, 0, 0)$.

3.2.5 Sphere

Add a sphere of radius R with center (c_x, c_y, c_z) to the list of objects. The ambient color of the sphere should be (a_r, a_g, a_b), the reflected color should be (r_r, r_g, r_b), and the specular color should be (s_r, s_g, s_b). If the specular colors are omitted or zero, assume the material is diffuse.

3.2.6 Triangle

Add a triangle with vertices at (u_x, u_y, u_z), (v_x, v_y, v_z), and (w_x, w_y, w_z). The ambient color of the triangle should be (a_r, a_g, a_b) and the reflected color should be (r_r, r_g, r_b). In this assignment, we will make a simplifying assumption regarding triangle normals, since we will deal with single triangles and not triangle meshes: In generating the normal of the triangle, assume that triangles have two outward faces, so that a ray striking a triangle always has an outward facing normal for shading calculations, independent of the order of the vertices. If the ray-tracer were to be modified to deal with meshes, this assumption would need to be modified to be consistent with the discussion of triangle normals in the lectures.

3.2.7 Box

Add an axis-aligned box with lower-left bottom corner (u_x, u_y, u_z) and upper-right top corner (v_x, v_y, v_z). The ambient color of the box should be (a_r, a_g, a_b) and the reflected color should be (r_r, r_g, r_b).

3.2.8 Ilight

Add a light at infinity. The color of the light is (l_r, l_g, l_b) and the direction of the light is (d_x, d_y, d_z).

3.2.9 Clear

Clear the screen and the object lists; also reset the background color to the default. See the assignment 4 handout for a fast way to clear the screen.

3.2.10 Trace

Generate a ray-traced image. For each pixel on the screen, create the appropriate pixel ray and determine its color. The final image should be writable to a file using tiffwrite.

4 Clarification and Notes

The materials in the project are going to be *diffuse* except for the sphere, which may have reflections. Diffuse materials do not generate any reflection or refraction rays. They are be colored with intensity I using the formula (for each of R,G, and B):

$$I = I_{ambient} + \Sigma_m E_m R \cos(\theta_m)$$

where the sum is over all light sources, E_m is the color of light m, $R = (r_r, r_g, r_b)$ is the reflectance color of the surface and θ_m is the angle between the surface normal at the illuminated point and the direction to the light source. Values of I above 1 or below 0 should be clamped to those limits. Note that this will generate shadows.

Your ray routine will be a simplified ray routine in which each ray is tested against each object in the database of objects. The database is a list of objects, and each object should have the following fields: type, geometry, color attributes. The type is sphere, triangle or box; the geometry contains the relevant geometric information for that type; and the color attributes contain the relevant colors for that type.

The simplified ray routine steps through the list, and calls the appropriate hit routines for each primitive. It records the closest hit object. When the entire list has been scanned, determine whether a hit on any object has occurred. If not, return the background color. If there has been a hit, process the appropriate shadow rays and continue execution. Depending upon how you implement your ray routine, the next pixel can be processed only when the ray stack is exhausted (if using a stack) or the recursion terminates (if using recursion). If the shadow ray hits an object, the light color for that light will be 0.

For a sphere with a mirror-like surface, the lighting equation is

$$I = I_{ambient} + \Sigma_m E_m R \cos(\theta_m) + S I_{reflected}$$

where $I_{reflected}$ is the color returned by a reflection ray and $S = (s_r, s_g, s_b)$. Generate the reflection ray as discussed in class, and shoot it out. You can terminate the recursion after four (4) reflections.

4.1 Requirements

Turn in your code as usual with two additional things. A tiff image of a scene you have created containing at least 5 spheres (at least three reflective), two triangles, and three boxes, with three light sources. If you are a graduate student include a plane, and make a triangle and the box reflective (see below). Also include a script for the set of commands used to generate that image. In a README file tell me how long it took to create the image. The image should be at least 512 by 512 pixels.

4.2 Implementation Suggestions

Get a simple ray-tracer working that only does ambient light calculations with an orthographic camera first. This is equivalent to the simple example covered in class. Then add the reflection shading equation with one light, and finally start spawning shadow rays. Add reflections after diffuse lighting and shadows seem to be working. Chapters 4 and 13 of the book contains much helpful material.

4.3 Extra Credit

Several items of extra credit are available:

1. (5 pts.) Implement a "Plane" command as follows: Plane $n_x, n_y, n_z, p_x, p_y, p_z, a_r, a_g, a_g, r_r, r_g, r_b, s_r, s_g, s_b$. The command adds a plane with normal (n_x, n_y, n_z) and passing through point (p_x, p_y, p_z) with ambient color (a_r, a_g, a_b), reflected color (r_r, r_g, r_b), and specular color (s_r, s_g, s_b). [1]

2. (5 pts.) Add the specular components (s_r, s_g, s_b) and their color calculation to the Triangle and Box (and Plane) commands, to function identically to the sphere command.[2]

3. (5 pts.) Add an additional parameter F to the Sphere, Triangle, Box, and Plane Commands that approximates rough metal surfaces by implementing the Schlick approximation to the Fresnel equations for a metallic surface, indicated by $F = 1$. If $F = 0$ or is omitted, then calculate the lighting as normal. [3]

4. (5 pts.) Implement Phong lighting for diffuse objects as explained in Section 9.2 of the book. You will need to add additional Phong color parameters and a Phong coefficient to each of the sphere, triangle, and box commands above.

5. (10 pts.) Implement instancing and show several examples of it for spheres and boxes. Describe how you implemented it and the commands in your CLI you added.

6. (15 pts.) Implement dielectrics as discussed in class. You may have to add additional parameters to the object commands to allow this.

7. (10 pts.) Implement an isotropic point light source (isotropic means the same in all direction): Plight $l_r, l_g, l_b, i_r, i_g, i_b, x, y, z$. This is a light source of color l_r, l_g, l_b with intensity i_r, i_g, i_b located at the position in space (x, y, z). It is assumed to be infinitely small. It emits light uniformly in all directions. The intensity falls off as

$$\text{Intensity} = \frac{i \times l}{4\pi d^2}$$

   where d is the distance between the light and the surface where the lighting calculation is being done. Note that when calculating shadows for point light sources, since they have a definite location in space, a shadow ray should not intersect an object *beyond* the point light source.

For each of items 1-7, you must submit an image illustrating the effect you have added. Give a detailed description of what you have done in a README. Other extra credits are possible: if you have something you want to add to your ray-tracer, e.g., sub-linear object intersections, or Monte Carlo sampling techniques. Discuss them with me.

[1] This "Extra Credit" is mandatory for graduate students taking CS5258.
[2] This "Extra Credit" is mandatory for graduate students taking CS5258.
[3] This "Extra Credit" is mandatory for graduate students taking CS5258.