A5 Final Report
Title: Raytracing Project
Name: Eric Xinquan Pan
Student ID: 20576124
User ID: expan

Final Project:

1 Purpose

The purpose of this project is to design and implment a fully function distributed ray tracer. The final product will satisfy the 10 objectives below and any extra ones I attempted. These objectives will allow me to make a well-rounded and fairly complex ray tracer.

2 Statement

This final project is be an extension of the work completed in A4. In A4, a basic ray tracer program was developed, including primary and shadow rays, primitive geometry, hierarchical objects, point light, meshes, and etc. The extra feature for A4 is anti-aliasing, using super-sampling.

Now this improved ray tracer will use reasonably complex techniques, creating quality image renders

The ray tracer features selected offered a fair challenge to program; the implementation of gl reflection and refraction are well-documented, but aren't trivial to do. But, personally I am excited to try and do an animation scene and attempt something similar to the bump-mapping water sphere shown in class, even though it will most likely be time consuming.

In addition to my objectives, I added some extra features to the project: Fresnel Equations, multi-threading, depth of field, motion blur, and website to show off my project. I implemented Fresnel Equations for realistic reflection and refraction, using the handout given in class. I also added multi-threading, allowing scenes to render many times quicker. Depth of field and motion blur were implemented to create a distributed ray tracer, with more details in the technical outline section. I created the website to easily display my objective scenes and go in a bit of detail how I implemented each one.

The most challenging aspect of the ray tracer for me was the animation. I wanted to create a 3-second loopable animation. To do this and make the animation smooth, I decided to model the object's postion with the motion equations taught in class, based on the frame number. However, that took a lot of trial and error to determine if the objects were in the correct position.

By the end of the project, I can confidently say that I have learned a vast amount about ray tracing and techniques used for it. My ability to read long research papers and books has definitely improved and . I will also most likely be working on this project, after the term is over, to improve this project even further.

3 Manual

3.1 Usage

What the ray tracer can do, can be found in the objective list at the end of this document. After extracting the zip file, the ray tracer can be found in /A5

To Build:
```
premake4 gmake
make
```

To Build Clean:
```
make clean
make
```

To Run:
```
./A5 <lua file scene>
```
My program was tested on gl16.student.cs

3.2 Input

The program takes only one parameter - the LUA file used to describe the scene to render. If no input file is given, the default file "simple.lua" is used, like in A4. The LUA file will provide information about the objects and their specific orientation and position, light sources, materials, and view, eye, and camera directions. The following are the lua commands to create objects:

- gr.node(name)
  - Creates a SceneNode node, with name as the identifier
- gr.joint(name, {xmin, xinitial, xmax}, {ymin, yinitial, ymax})
  - Creates a JointNode node, with name as identifier and degrees of freedom for the node for x-axis and y-axis movement
  - Not too relevant for this program
- gr.nh_box(name, {xpos, ypos, zpos}, sideLen)
  - Creates a non-hierarchical box as a GeometryNode node with name as identifier.
  - xpos, ypos, and zpos as the bottom front corner coordinates with side length sideLen
- gr.cube(name)
  - Creates a unit cube GeometryNode node, with name as the identifier
  - Cube has the bottom front corner coorindates at (0,0,0) with side length 1
- gr.nh_sphere(name, {xpos, ypos, zpos},r)
  - Creates a non-hierarchical sphere as a GeometryNode node with name as identifier.
  - xpos, ypos, and zpos as the sphere's center with radius r
- gr.sphere(name)
  - Creates a unit sphere GeometryNode node, with name as the identifier
  - The sphere is centered at (0,0,0) with radius 1
- gr.mesh(name, fileObjPath)
  - Creates a mesh GeometryNode node, with name as the identifier
  - The fileObjPath is a file path to the .obj path. Note it should be either an absolute path or a relative path, starting from where the program is.
- gr.cylinder(name)
  - Creates a unit cylinder GeometryNode node, with name as the identifier
  - The cylinder is centered at (0,0,0) with radius 1 and height 2. The y-axis is the central axis
- gr.cone(name)
  - Creates a unit cone GeometryNode node, with name as the identifier
  - The cone is centered at (0,0,0) with radius 1 and height 1
- gr.light({xpos, ypos, zpos}, {r,g,b}, {f1, f2, f3})
  - Creates a point light object at {xpos,ypos,zpos}

  - {r,g,b} indicates the colour of the light
  - {f1,f2,f3} represents the falloff coefficents

- gr.arealight({xpos, ypos, zpos}, {r,g,b}, {f1, f2, f3},{corner1X,corner1Y,corner1Z},{corner2X,corner2Y,corner2Z},{

  - Creates a rectangular area light object at the 4 corner coordinates passed
  - {r,g,b} indicates the colour of the light
  - {f1,f2,f3} represents the falloff coefficents

- gr.material({d1,d2,d3},{s1,s2,s3},b,i,g,r)

  - Creates a material object
  - {d1,d2,d3} represents the material's diffuse coefficents
  - {s1,s2,s3} represents the material's specular coefficents
  - b represents the material's brightness factor
  - i represents the material's index of refraction. i=0.0 means the material does not reflect or refract
  - g represents the material's glossy factor, used in glossy calculations. g=0.0 means the material does not need any glossy reflection or refraction calculations
  - r represents the reflectance value. If r != 0.0, then the material's reflectance is set to r. Otherwise the reflectance is calculated using the Fresnel equations.

The following are node modifiers:

- ¡parent¿:add_child(¡child¿)

  - Adds ¡child¿ as a child of ¡parent¿

- ¡node¿:set_material(¡material¿)

  - Adds ¡material¿ to ¡node¿

- ¡node¿:set_texture(¡textureFilePath¿)

  - Sets the texture of the node
  - Note this overwrites the material of the node

- ¡node¿:set_material(¡material¿)

  - Adds ¡material¿ to ¡node¿

- ¡node¿:translate(x,y,z)

  - Translates the node by (x,y,z) units

- ¡node¿:rotate(axis, ang)

  - Rotates the node about the axis by the ang in degrees

- ¡node¿:scale(sx,sy,sz)

  - Scales the node by (sx,sy,sz) units

Finally there's also the render command, to render the scene

- gr.render(node, fileOutput, w, h, eye, view, up, fov, ambient, {listLights})

- node is the root node. It and all of its children will be rendered
- fileOutput is the name of the PNG file rendered
- w is the image width in pixels
- h is the image height in pixels
- eye is the eye position in {x,y,z} format
- view is the direction the eye is looking at in {x,y,z} format
- up is the up direction of the eye in {x,y,z} format
- fov is the field of view for the camera in degrees
- ambient is the ambient lighting in {x,y,z} format
- listLights is a list of light sources defined by gr.light or gr.arealight, separated by commas.

3.3 Interaction

The program doesn't require any user interaction other than the initial input command. The program outputs the scene's environment information (lights, view, and etc) other than the node objects.

3.4 Output

Once the program finishes rendering, the final image will be found as a PNG file in the same directory that the program is ran from. It will be named the given file name specified in the gr.render command

4 Implementation

At least two new primitives (cone and cylinder) will be added, and function properly, using equations given in course notes.

Soft shadows will be implemented by using area lights (instead of the basic point lights) and cast multiple rays per light source. I plan to implement my area light as a rectangle and divide it into 64 parts (uniform distribution of rays). Then I'll shoot a ray from each part to the point and cast a shadow. Area light sources will be implemented as circles defined by a center and a radius [1].

Reflection and refraction will work with all primitives and are well documented in our graphics textbook and in the textbook by Hughes [1], which I plan to follow closely.

Glossy reflection and transmission will be implemented by taking the intersection point and get a normal distribution of valid rays to shoot from the intersection point. Then I'll set the pixel to appear as the average colour of the valid rays.

Adaptive anti-aliasing will be done through a simple edge detection algorithm that will analyze the image pixel by pixel. It will then try to detect any sharp changes between the pixel and it's neighbours. If so, anti-aliasing will be done.

Bump mapping [2] will be done for spherical objects, by keeping track of a displacement map (bump map) to be combined with the surface to display bumps. I'm looking to implement bump mapping for a water sphere, like shown in class, and for a world globe that will have texture and bumps for land.

Lastly, a final scene will be presented, with a 3 second animation video with various objects moving. The following lua command will be added:

- gr.cone
- gr.cylinder
- gr.texture
- gr.arealight

- gr.union, gr.intersection, gr.difference if CSG is implemented

Also existing lua commands for materials will be modified for reflection, refaction and glossiness.

5 Bibliography

[1] John F. Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley, 2014.
[2] M. Tarini. *Real Time, Accurate, Multi-Featured Rendering of Bump Mapped Surfaces.* 2000.

Objectives:

Full UserID: expan **Student ID: 20576124**

___ 1: Extra primitives (Cone, Cylinder)

___ 2: Reflection

___ 3: Refraction

___ 4: Soft shadow

___ 5: Glossy reflection

___ 6: Glossy transmission

___ 7: Bump mapping

___ 8: Anti-aliasing using adaptive

___ 9: Animation (Generate 90 ray tracing images with objects to create 3 second animation)

___ 10: Final Scene

A4 extra objectives:

- Add a nice web page to easily display my objectives. (https://ericsraytracer.wordpress.com/)
- Implement a multi-threaded ray tracer
- Motion blur
- Depth of Field