Name: Richard Vong Student ID: 67436350

Cosc363 Assignment 2

Note: This assignment was done using a Linux Mint VM and Geany as the development environment

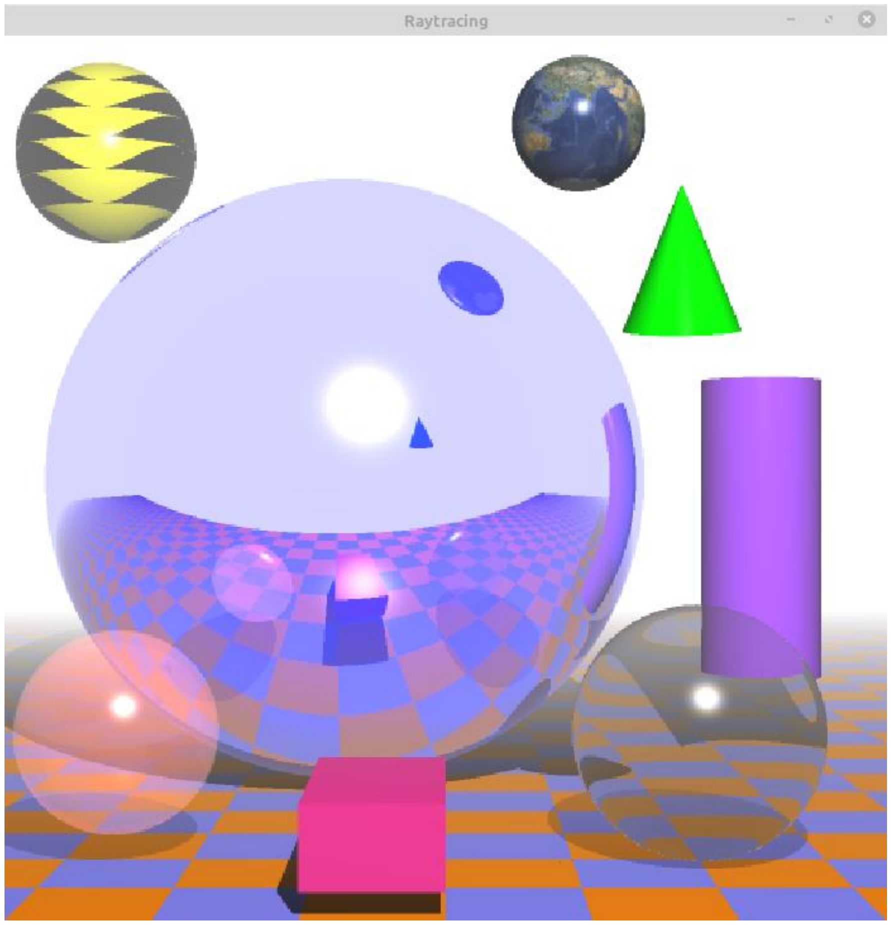


Figure 1: Ray tracing scene

Min requirements:

a) **Spatial arrangement**: Show in figure 1, all objects in the scene are spaced in a way to allow the viewer to clearly see each object's global illumination features and patterns.

b) **Transparent object:** Shown in figure 1, the scene has two transparent objects in the lower left and right corners of the scene.

c) **Shadows:** Shown in figure 1, all objects in the rendered scene casts shadows onto the floor plane and both transparent and refractive objects in the lower right and left of the scene both cast lighter shadows.

d) **Object constructed using planes:** Shown in figure 1, in the lower center of the scene there is a pink box constructed using a set of 6 planes.

e) **Checkered pattern on planar surface:** Shown in figure 1, the floor plane has a checkered purple and orange pattern generated on it.

Extensions:

- **Cylinder object**: A cylinder has been added to the scene in the mid right-hand-side of the scene (figure 1). The cylinder's intersection is calculated by using the ray equations for the x, y and z axis and solving for the unknown value of t using the quadratic equation to find the roots t1 and t2. The height of the cylinder is then calculated by determining if the roots lie within the cylinder's height. (As shown in the lecture notes).
- **Cone object:** The points of intersection are calculated by substituting the ray equation in the conde's equation and solving for t, using the quadratic equation. The surface normal vector is then normalized and returned. (As shown in the lecture notes).
- **Refraction:** Refraction has been implemented in the scene on the sphere in the lower right corner of the scene (shown in figure 1). Its effect bends the light rays towards the normal, thus distorting the objects and illumination effects of the objects behind the refracted object. It is implemented by tracing a secondary ray along the direction of refraction, when this secondary ray meets a surface then the pixel color is calculated using its color + refraction coefficient * reflected color.
- **Anti-aliasing:** Anti-aliasing has been implemented in the scene to reduce jagged edges on the objects thus making them and their global illumination features (such as shadows) appear smoother. It is implemented using super sampling by diving a pixel quad into quarters, then generating individual rays through each quarter and then returning the computed average of the color values. Figures showing the rendered scene with and without anti-aliasing is shown below in figure 2 and figure 3 (you may have to zoom in to see the details more clearly).

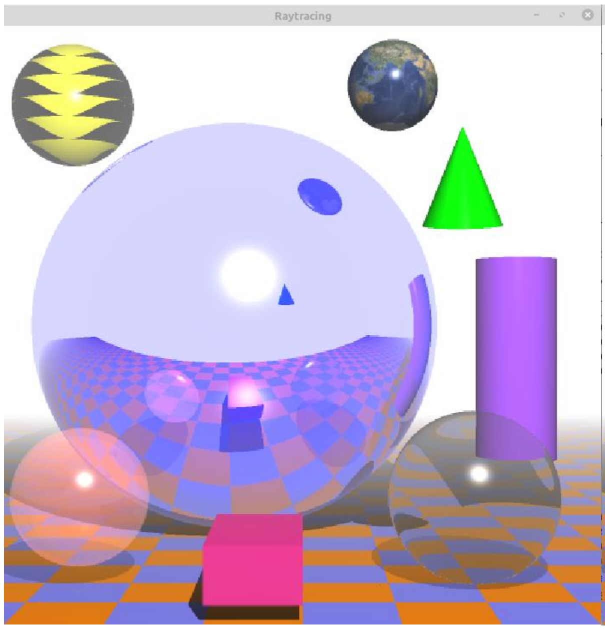


Figure 2: Ray Tracing scene with anti-aliasing

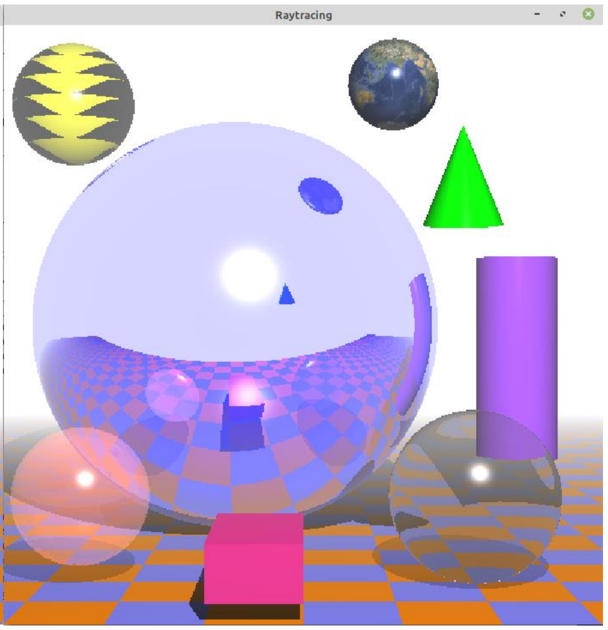


Figure 3: Ray tracing scene without anti-aliasing

- **Non-planar object textured using an image:**
  An earth texture has been mapped onto a sphere shown in the upper right-hand-side of the scene (Figure 1). The texture mapping was implemented by calculating the X coordinate of the normal (u) and the Y coordinate of the normal (v) and mapping the color of the texture co-ordinate to each u and v value.
- **Procedural pattern generated on a surface:**
  A procedural pattern has been generated on the sphere in the upper left of the scene by using the mathematical formulae of a sine curve repeated by testing whether the current y value is above or below the wave. The equation for the sine curve is:
  $$y = amplitude * \sin(((ray.hit.x) * pi)\ /b) + amplitude)$$
- **Fog:** Fog is generated on the scene to add atmospheric depth. This is implemented in the rendering by setting the background color to white, defining a fog range [z1, z2], calculating the interpolation parameter t, and modifying the color as calculation of t and the background color. Figures showing the rendered scene with and without fog is shown below in figure 4 and figure 5 (you may have to zoom in to see the details more clearly).

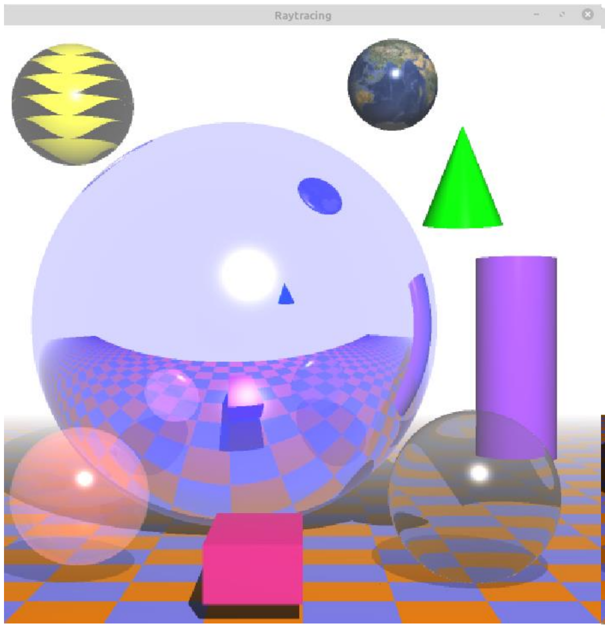


Figure 4: Ray tracing scene with fog

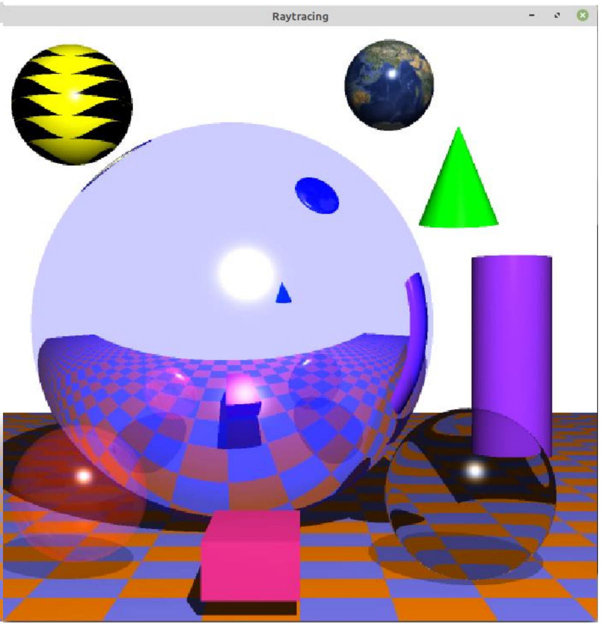


Figure 5: Ray tracing scene without fog

Estimated time to compile, build and run program:
45 seconds.

Build Process:
1. Download and install VirtualBox
2. Download Linux Mint
3. Once inside your Linux Mint virtual machine, run the following commands one at a time from the terminal to install OpenGL, CMake and the build tools:

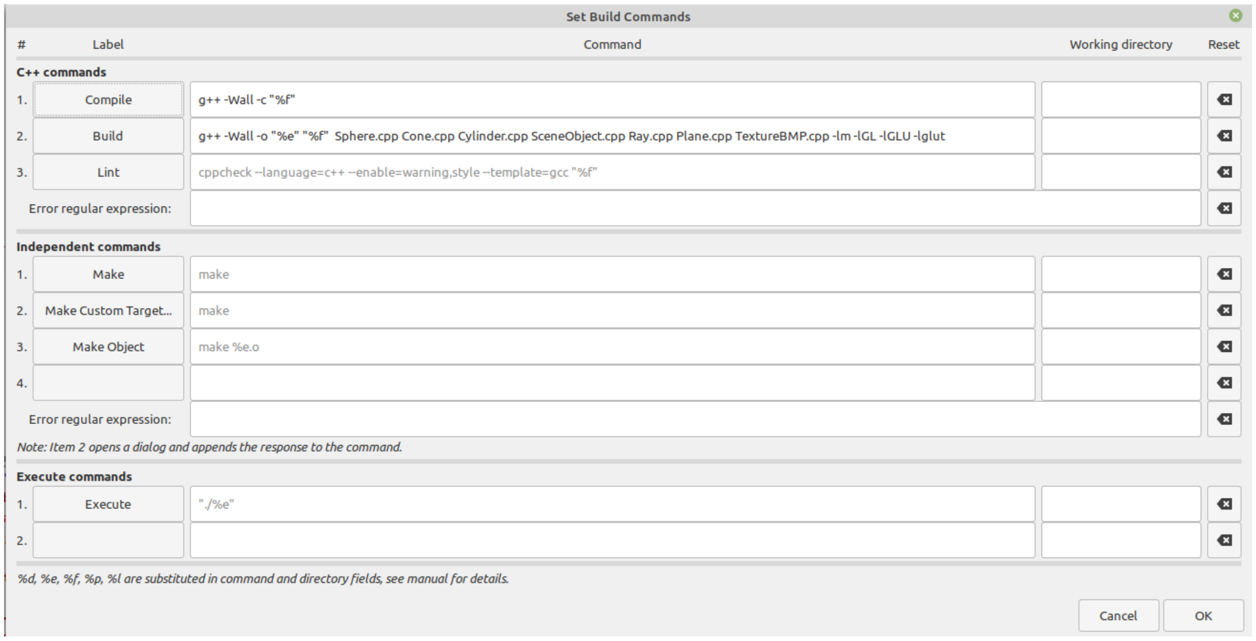
```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install freegult3-dev
sudo apt-get install cmake
sudo apt-get install libglm-dev
```

4. Install the open source version of Geany using the following command from terminal:

```
sudo apt-get install geany
```

5. Open Geany and access the build settings using the "Set Build Commands" window under the "Build tab.

6. Set up your Geany build commands so the fields "Compile", "Build" and "Lint" match the image below:



Set Build Commands

#	Label	Command	Working directory	Reset
C++ commands				
1.	Compile	`g++ -Wall -c "%f"`		
2.	Build	`g++ -Wall -o "%e" "%f"  Sphere.cpp Cone.cpp Cylinder.cpp SceneObject.cpp Ray.cpp Plane.cpp TextureBMP.cpp -lm -lGL -lGLU -lglut`		
3.	Lint	`cppcheck --language=c++ --enable=warning,style --template=gcc "%f"`		
	Error regular expression:			
Independent commands				
1.	Make	make		
2.	Make Custom Target...	make		
3.	Make Object	make %e.o		
4.				
	Error regular expression:			
Execute commands				
1.	Execute	"./%e"		
2.				

Note: Item 2 opens a dialog and appends the response to the command.

%d, %e, %f, %p, %l are substituted in command and directory fields, see manual for details.

Cancel OK

7. In geany, open "RayTracer.cpp", compile, build and run the program. *The scene takes an estimated 45 seconds to compile, build and run*.

References and resources:

- Cosc363 Lec08_RayTracing
- Cosc363 GLM
- Cosc363 Lecture slides (19th May)
- Earth Texture: https://www.deviantart.com/peteridish/art/Earth-texture-284132938
- BMP converter: https://image.online-convert.com/convert-to-bmp
- Spherical Mapping with Normals: https://www.mvps.org/directx/articles/spheremap.htm?fbclid=IwAR1oqMgX16640RlyBSDDj-yDNFtCR5jq2vfoqU1BY0-G2ObbwMrkyeQ6OCA