COSC 363 – Assignment 2

Zachary Sanson

58520526

Summary:

My graphics project "main.cpp" is a graphically generated scene of objects (ranging from spheres, cubes and to tori). The program graphically draws objects by the method of ray tracing. The objects have hundreds of rays projected onto them and the objects will calculate a reflected ray from the object to the eye.

Implemented Features (Basics):

This scene contains basic code for the following:

Lighting; diffuse & specular lighting. This is achieved by creating a light source, projecting rays onto the scene objects and calculating the ambient, specular and material colour at that ray intersection. I'm using the "Phong's" formula to calculate the specular colouring of objects.

Shadows; to create shadows, we create another ray (except instead of adding light it acts as a shadow mat) which iterates through the scene objects much like the main light and returning a shadow colour.

Reflections; reflections are achieved when a ray intercepts and object. A new ray will be created in the orthogonal direction to where the original ray intercepted the object. We then calculate the lighting effects on all the other objects in the scene with this new ray and return the accumulated colour values.

Parallelepiped (Cube); more information in the 'Object Definitions' section.

Textured Surface (Plane); there is a textured plane in my program which is done by loading a '.bmp' image into the program and mapping the correct colour value at the intersection of the plane and finding the corresponding colour in the image itself.

Extra Features:

Primitive Objects:

- Cone
- Cylinder
- Tetrahedron
- Torus

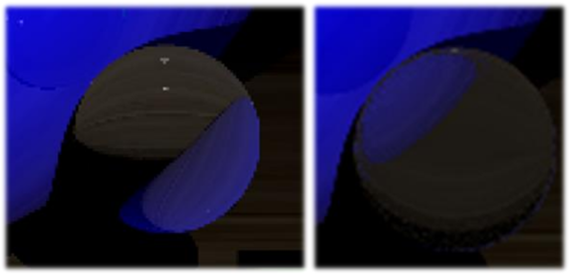
More information on the objects in found in the 'Object Definitions' section.

Multiple Light Sources; there is a second light which is lower down and acts as a 'softer' light in the scene. The shadows created by this light will appear lighter than the main light and will drape along the scene more due to its lower position. This requires there to be an extra function every time a light is used, so I have implemented a light array to loop over.

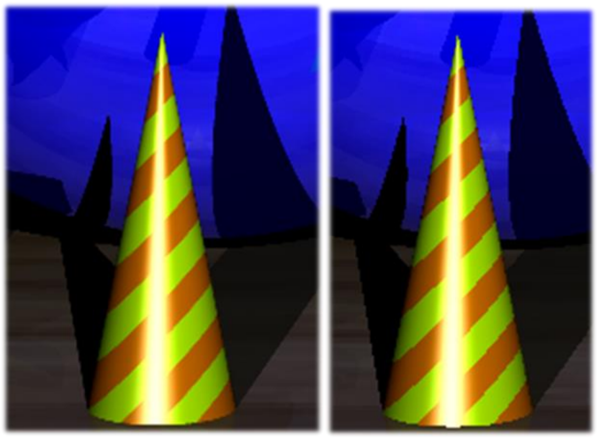
Spotlight; whilst trying to create a spotlight I could not get the correct lighting that a spotlight should show. What I received was a spotlight like effect although could not change the radius of the spot and therefore did not represent spotlight features. I have decided to scrap this for the moment until I have a greater understanding of how to shape a spotlight correctly.

Transparency; one object on the scene has transparency and that is the sphere in the bottom right. Transparency is done by setting an objects transparency to a decimal value (indicating the amount of transparency of the object). It then gets calculated after the refractions (but not as a refractions).

Refraction; there is one object that has refractence which is the sphere on the bottom right. It has support for differing refractence levels on separate objects and also works with transparency.



Anti-Aliasing; anti-aliasing can be toggled on and off from the start of the program by setting ANTI-ALIASING = true. This will split the pixels into quads and average the rays to provide a less jaggered edge to the objects by 'smoothing' the colour pixels between quadrents.



(Picture on the left is with anti-aliasing and on the right is without anti-aliasing.)

Object Transformation; one object on my scene requires matrix transformations to generate the shape. Of which being the torus, the transformation changes the generated shape from something like a crescent to a torus buy curving the cylindrical shape in a circle.

```
torusMagenta->transform = glm::translate(
        torusMagenta->transform, glm::vec3(15, 0, -80.0));
torusMagenta->transform = glm::rotate(
        torusMagenta->transform, -PI / 3, glm::vec3(0, 1, 0));
for (SceneObject *object : sceneObjects)
        object->transform = glm::inverse(object->transform);
```

Non-Planar Textures; multiple objects on the field include textures, these following objects have textures:

- Plane/floor with a woody (crate) texture, the texture mapping is done by the following equation:

  ```
  a1 = FLOOR.xMin;
  a2 = FLOOR.xMax;
  b1 = FLOOR.zMin;
  b2 = FLOOR.zMax;
  ```

```
s = (ray.oPoint.x - a1) / (a2 - a1);
t = (ray.oPoint.z - b1) / (b2 - b1);
```

- Sphere with a bubbly texture which is modelled by the following equation:
```
s = (0.5 - atan2(dir.z, dir.x) + PI) / (2 * PI);
t = 0.5 + asin(dir.y) / PI;
```

- Cubes with grass textures, currently mapping the texture to the cube uses the same function as the sphere until a proper function is implemented (since I'm not using six planes which would be easy).

Non-Planar Patterns (Procedurally Generated); there are several objects that have procedurally generated textures, two objects (cone & cylinder) have a spiral texture in opposing directions and one has a checked pattern on a cube (depreciated → replaced with a texture) they are produced by the following formulas:

- *Checker*
```
((int(ray.oPoint.x) - int(ray.oPoint.z)) % 2) ?
  checkerColour = glm::vec3(0) :
  checkerColour = glm::vec3(1);
```

- *Spiral*
```
if (int(ray.oPoint.x + ray.oPoint.y) % 2 == 0)
  materialColour = selectedObject->textureColourMod;
```

Fog; fog is currently implemented based on the main light ray. The amount of fog increases as the distance from the rays' origin and the intersecting object increases using the following formula:
```
fogColour = glm::vec3(-0.001f * ray.xDist);
```
Note: the fog colour is black and therefore acts more like a FOV, getting darker the further you move away from the light.

Object Definitions:
[Cone, cube, cylinder, plane, sphere, tetrahedron, torus]
All objects also come with a colour parameter.

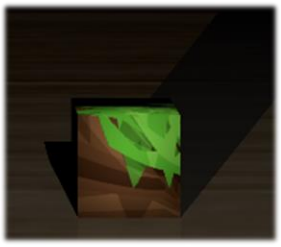
Cone:

This object will create a cone of $radius(r)$, $height(h)$ and with a centre point at a given vector ($center(x, y, z)$). The cone uses an intersection function which finds the roots of a quadratic function which represent an equation of a cone. The intersection function will find and return the t-values that intersect the cone; and if there is only one point of contact, it will return the nearest t-value to create the cap of the cone.

Parallelepiped (Cube):

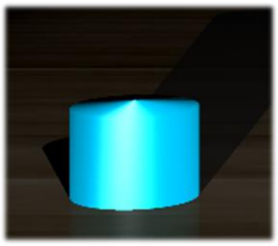
In my program I have decided not to use six planes to create my cube but instead calculate the ray intersections by finding how the ray hit the object relative to the eight vertices of the cube.

In doing so, I have made the cube reliably generate by itself (without the usage of other object files) although am currently running into issues finding how to texture map properly onto the cube (details in bug report).



<u>Cylinder</u>:

Much like the cone we have a similarly designed object file. This object will create a cylinder of $radius(r)$, $height(h)$ and with a centre point at a given vector ($center(x, y, z)$). The cylinder uses an intersection function which finds the roots of a quadratic function which represent an equation of a cylinder. The intersection function will find and return the t-values that intersect the cylinder; and if there is only one point of contact, it will return the nearest t-value to create the cap of the cylinder.

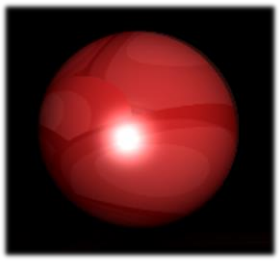


<u>Plane</u>:

This object takes four parameters in the form of vectors $ptA(x, y, z)$, $ptB(x, y, z)$, $ptC(x, y, z)$, $ptD(x, y, z)$ and creates a plane which spans those vertices.

<u>Sphere</u>:

This object will create a sphere of $radius(r)$ and with a centre point at a given vector ($center(x, y, z)$). The cylinder uses an intersection function finds the roots of the equation of a sphere, there is nothing much to it (see lab files if you want info). The main modification I have made to the sphere is allowing for a semi-sphere or a dome. It will cut the sphere from its horizon downwards, leaving a dome behind.



<u>Tetrahedron</u>*:

This is a work in progress object, I was initially going to try generate a pyramid with pure mathematics much like the cube. Although has proven a larger challenge so I have instead decided to try create the pyramid using the four triangular planes. (Shadows also don't appear to be working with my 'make-do' triangles.)

<u>Torus</u>:

To create the torus it takes the parameters $radius(r)$ and a center point in vector form $center(x, y, z)$. The torus assumes that the inner radius of the torus is equal to three. The torus is an incredibly complicated object requiring quartic equation solving for real roots. So finding the intersection of a ray onto this object requires quartic equation solvers and polynomials that take form of $ax^4 + bx^3 + cx^2 + dx + e$. I will not go through the intersection equation due to its complexity although can provide information if needed.
Note: to actually shape the torus I needed to apply object transformations and rotations on the object so it would take form of a torus and not an object like a crescent.

Current Bugs:
Cube textures are currently based on spherical coordinates. Whilst this method s working although is not the optimal way of dealing with texture mapping cubes.

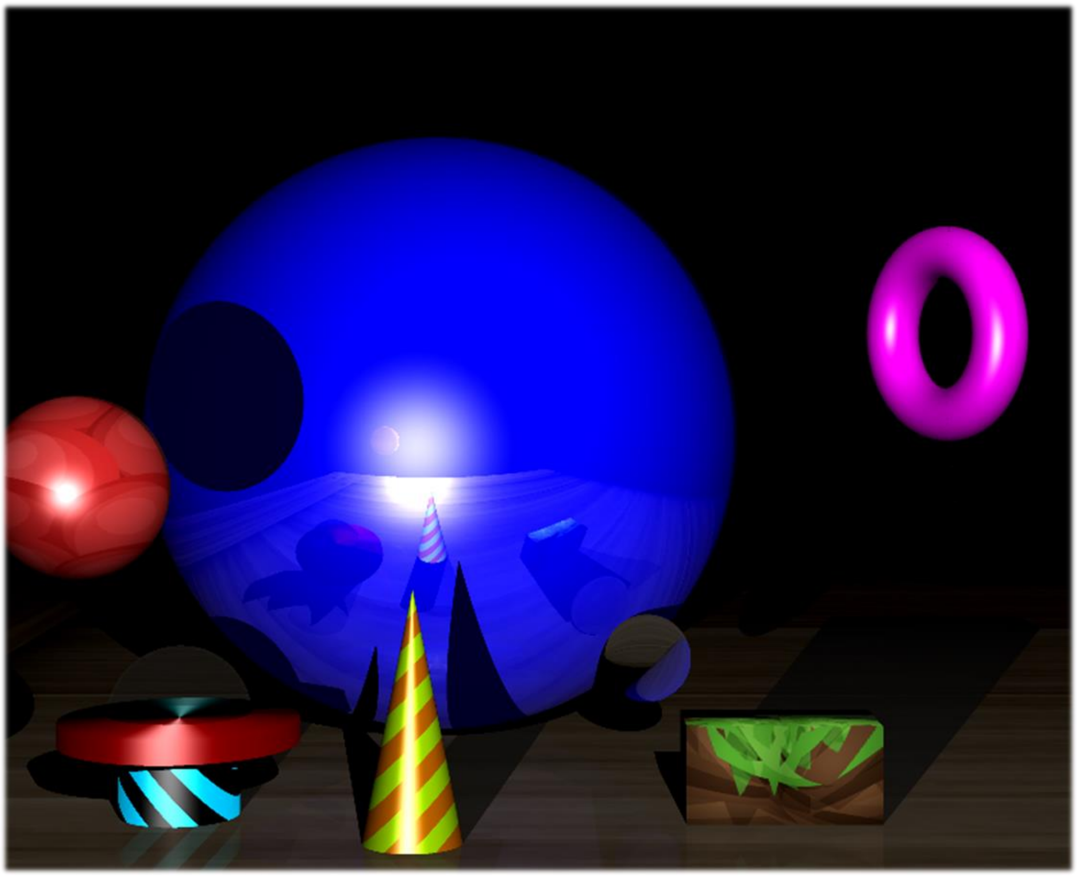
Tetrahedron is not currently implemented and I'm unable to find any reliable material on how to generate tetrahedrons with properly working shading and light interactions.

Currently shadows in transparent objects don't appear correct and sometimes ignore the shadows behind them.

Torus (On Lab Machines); for some reason that I have not figured out yet, the torus will not generate properly on the lab machines. Although provides a perfectly smooth torus on my desktop. There is some weird Linux compatibility issue here and I'm unsure how to fix this (might actually be glm library versions?) Important note: the quartic solver does require experimental glm features.

Render times seem to increase drastically when adding the torus which is understandable since the program needs to solve quartics every time a ray intersects the torus.

Examples:
This first image was rendered with 800 divisions with anti-aliasing set to 2.
Rendering this image did take a while but provides a magically smooth render of my scene.



You can see all the objects being generated:

- The refracted sphere on the bottom right (with quite a large refractive index).
- Reflective sphere (blue sphere in the centre) and the plane is also slightly reflected giving it a 'glossy' look.
- Transparent sphere on the bottom left.
- Textured sphere on the top right, cubes & planes.
- Patterned cylinders & cones.
- Transformed/sheared object forming a torus.

External Resources:
Used textures provided from these sites under an open source license:
https://opengameart.org/content/3-crate-textures-w-bump-normal
https://bdcraft.net/downloads/purebdcraft-minecraft/

- Allowed usage of their work