Samuel Yamoah
SID: 63643621

COSC363 Computer Graphics Assignment 2: Raytracer

Cylinder:

Intersection Equation:

$$t^2(d_x^2 + d_z^2) + 2t\{d_x(x_o - x_c) + d_z(z_o - z_c)\} + \{(x_o - x_c)^2 + (z_o - z_c)^2 - R^2\} = 0$$

Vector Normal:

Vector n = Vector(p.x - center.x, 0.0, p.z - center.z)
This vector was then normalised with the normalise method.

The cylinder class calculates the point of intersection and also the normal of the traced ray. The intersection method solves the two points of intersection based on a quadratic equation where the a is the t^2 term, b is the t term and c is the rest of the equation. The min and max was calculated to cap the height of the cylinder otherwise it formed a cylinder with infinite height.

Cone:

Intersection Equation:

t^2(dir.x * dir.x) + (dir.z * dir.z) - (k * dir.y * dir.y) + 2t{((pos.x * dir.x - center.x * dir.x) + (pos.z * dir.z - center.z * dir.z) + ((k * height * dir.y) - (k * pos.y * dir.y) + (k * center.y * dir.y)))} + pow((pos.x - center.x), 2) + pow((pos.z - center.z), 2) - k * ((center.y - pos.y + height) * (center.y - pos.y + height))

Where k is (radius/ height)^2

Vector Normal:

alpha = atan((p.x - center.x) / (p.z - center.z))
theta = atan(radius/height);
Vector n = Vector(sin(alpha) *cos(theta), sin(theta), cos(alpha) * cos(theta))

The cone class performs the same calculations as the Cylinder class but with normal calculations specific to a cone.

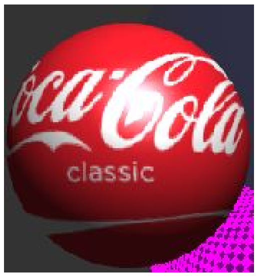
Image Textured sphere:

To texture this sphere, I followed the instructions given on the class forum page and mapped the u and v using the equations below to my image:

```
u = asin(n.x)/M_PI + 0.5
v = asin(n.y)/M_PI + 0.5
```

The image below was downloaded of the internet and converted into a 24 Bit A8 R8 G8 B8 bmp image using GIMP and then the trace function sets the colour of the image at the corresponding mapped point using *col = texture1.getColorAt(u, v)* giving the resulting image with phong lighting also added.

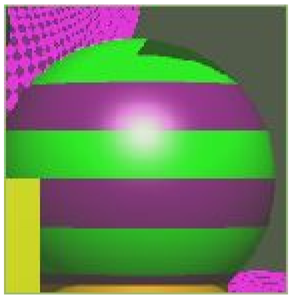
Coca-Cola
classic



Procedural Pattern Textured sphere:

The pattern that was created on this sphere uses a similar method to generate the checkerboard pattern for the floor plane. The sphere and floor size were mapped out so that at every even or odd position would be a different colour.

```
if((int)(q.point.y - 7) % 2)
    col = ballColTwo
```



The floor pattern was created with the following equation:

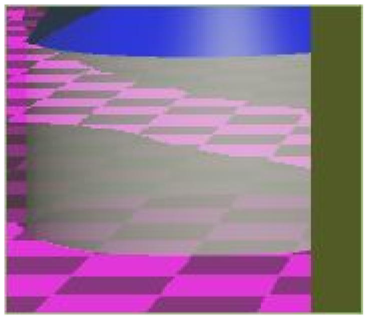
```
//generate the checkerboard floor
if (q.index == 3)
int modx = (int)((q.point.x - 200) /2) % 2
int modz = (int)((q.point.z - 200) /2) % 2

if((modx && modz) || (!modx && !modz))
colorSum = col.phongLight(groundCheck, 0.0, 0.0)
```

Transparency:

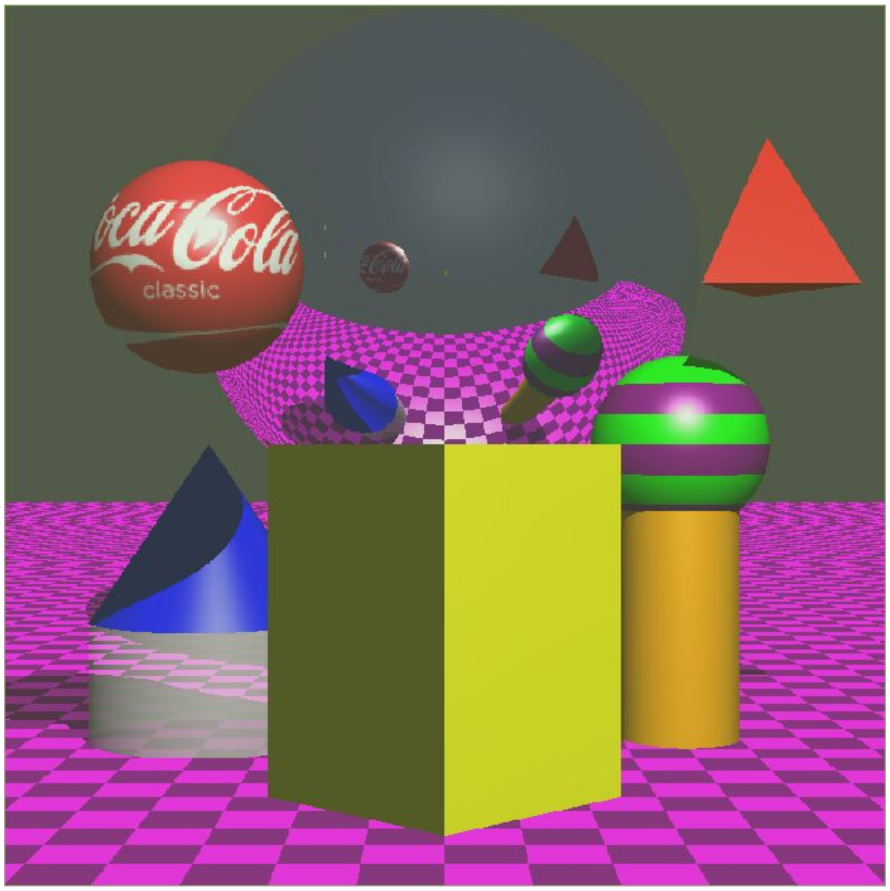
The cylinder on the left has transparent calculations applied to it allowing light to pass through the object so that other objects behind can be distinctly seen. The transparent colour of the object is calculated by tracing the colour of the object back and multiplying it by a transparent Coefficient. This is done recursively adding the current calculated value to the previous one but only up to the maximum number of recursions as computation becomes time and resource expensive with too many recursions.

```
//generate the transparent ray
if (q.index == 4 && step < MAX_STEPS)
{
Color transparentCol = trace(q.point, dir, step+1);
colorSum.combineColor (transparentCol, transparentCoeff);
}
```

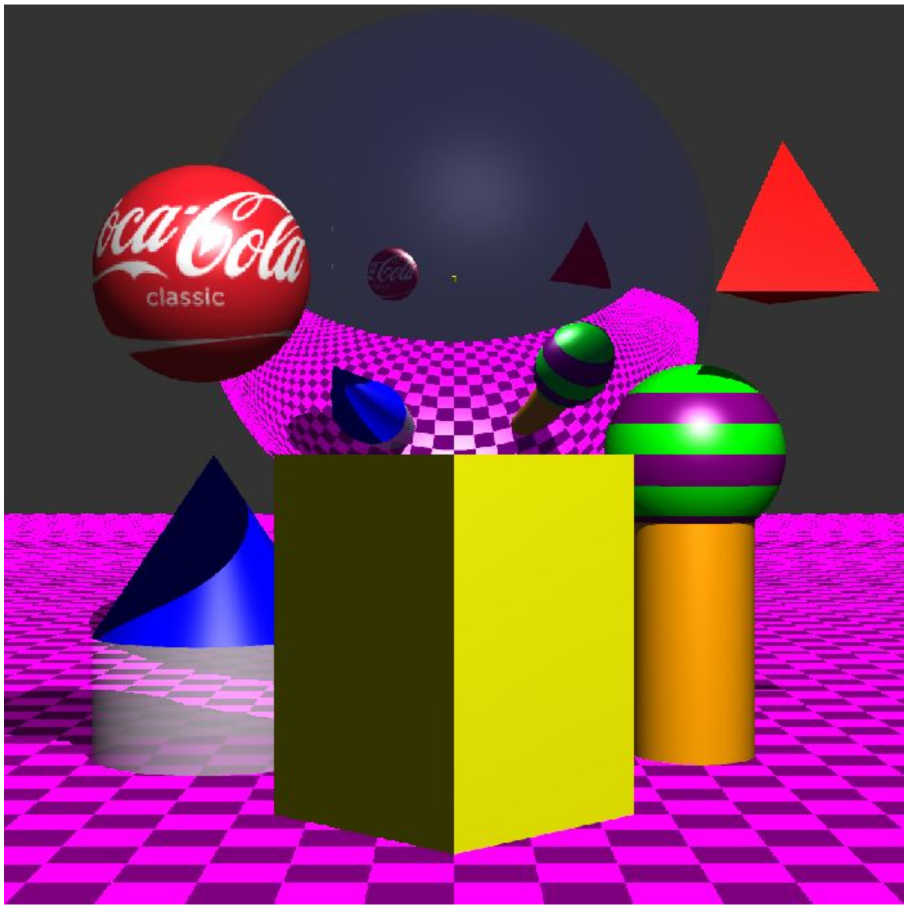


Anti-Aliasing:

The ray tracing algorithm samples the light field using a finite set of rays generated through a discretized image space. This results in distortion artefacts such as jaggedness along edges of polygons and shadows. I used Supersampling to generate several rays through each square pixel (eg. divide the pixel into four equal segments) and compute the average of the colour values.
Three more additional vectors were created subdividing the pixel into quarters then normalised.
Then the average RGB colours from the four traced rays were found and then passed into *glColor3f*.



This image above shows anti-aliasing off (Note the yellow tint is from the screen capture software)



This image above shows anti-aliasing on

Tetrahedron:

This Was created by using the plan class and for the fourth vector was the same as the first vector.

References:

- Raytracing Lecture Notes by R. Mukundan
- https://www.cs.cmu.edu/~fp/courses/graphics/pdf-color/10-texture.pdf
- http://www.mvps.org/directx/articles/spheremap.htm
- http://web.eecs.utk.edu/~huangj/cs456/notes/456_texturemap1.pdf
- http://www.eng.utah.edu/~cs5610/lectures/two-part%20texture%20mapping%202010.pdf
- http://1.bp.blogspot.com/-a_nqLtzoQ6Y/UQ-3fUUs09I/AAAAAAAAD4E/KHtuBOQ3DQ8/s1600/coke+logo.jpg