COSC363 REPORT

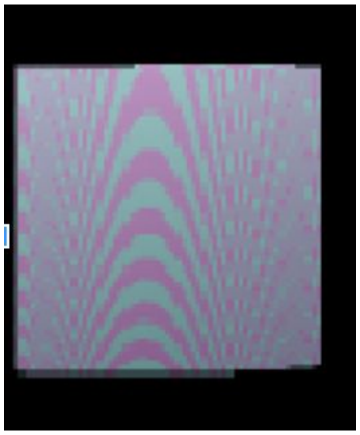
<u>ASSIGNMENT 2</u>

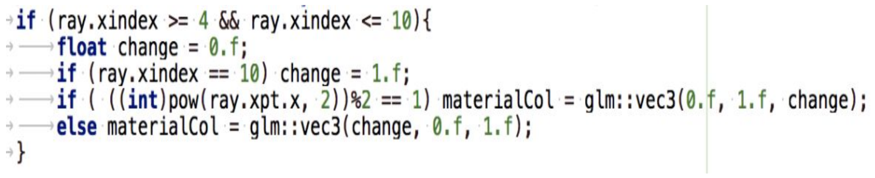
THAVY THACH | ID: 45868143

Build Command: g++ -Wall -o "%e" "%f" Sphere.cpp SceneObject.cpp Ray.cpp Plane.cpp TextureBMP.cpp Cylinder.cpp -framework OpenGL -framework GLUT

[Extra Features (Marks Attempted: 20+)]

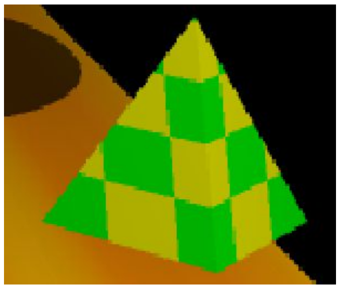
- **Non-Planar Object Textured using a procedural pattern #1 (Cylinder: 1 Mark)**

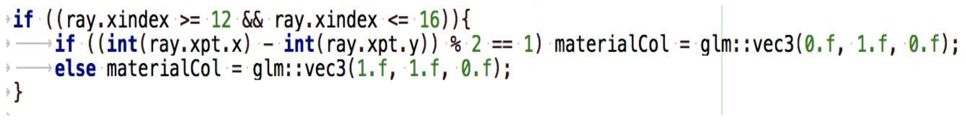




```
if (ray.xindex >= 4 && ray.xindex <= 10){
    float change = 0.f;
    if (ray.xindex == 10) change = 1.f;
    if ( ((int)pow(ray.xpt.x, 2))%2 == 1) materialCol = glm::vec3(0.f, 1.f, change);
    else materialCol = glm::vec3(change, 0.f, 1.f);
}
```

- **Non-Planar Object Textured using a procedural pattern #2 (Tetrahedron: 1 Mark)**

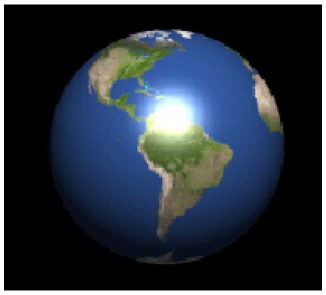


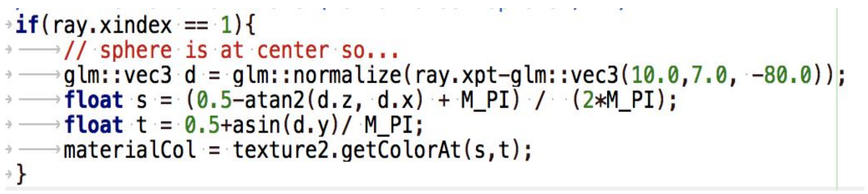


```
if ((ray.xindex >= 12 && ray.xindex <= 16)){
    if ((int(ray.xpt.x) - int(ray.xpt.y)) % 2 == 1) materialCol = glm::vec3(0.f, 1.f, 0.f);
    else materialCol = glm::vec3(1.f, 1.f, 0.f);
}
```

- **Primitive Object(s) (Tetrahedron: 1 Mark):** By utilizing the plane class, the tetrahedron.c and its corresponding header file were constructed by setting the final point on one of the edges of that particular plane: `glm::vec3( (C.x + D.x) / 2, (C.y + D.y) / 2, (C.z + D.z) / 2);`). After that we compute the triangle (plane) as `Plane *tri3 = new Plane(B,D,CD,C, color);`

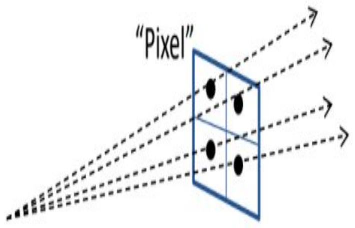
- **Non-Planar Object Textured using an Image (Earth: 1 Mark)**





```
if(ray.xindex == 1){
    // sphere is at center so...
    glm::vec3 d = glm::normalize(ray.xpt-glm::vec3(10.0,7.0, -80.0));
    float s = (0.5-atan2(d.z, d.x) + M_PI) /  (2*M_PI);
    float t = 0.5+asin(d.y)/ M_PI;
    materialCol = texture2.getColorAt(s,t);
}
```

❑ **Anti-Aliasing (Super-Sampling: 1 Mark):** Supersampling visualization is shown on the right. For each arrow pointer, there is a ray being traced four times to be computed and returned to be averaged. This method is to minimize distortion artefacts such as jaggedness along the edges of polygons and shadows that may occur during generation in a discrete image space. Usually, without anti-aliasing only allows for one tracing computation. The subsequent photos below will illustrate objects with no anti-aliasing and objects with anti-aliasing.



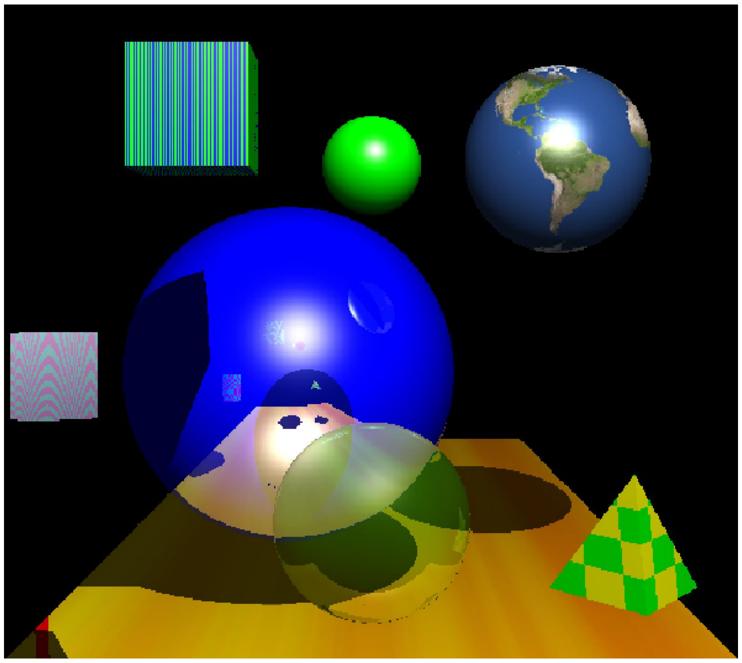


Figure 1.0: Without Anti-Aliasing

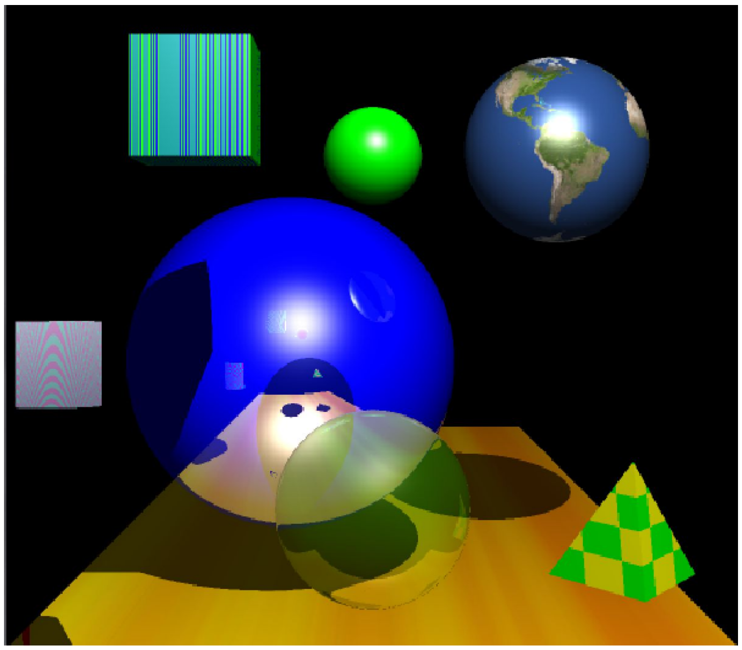
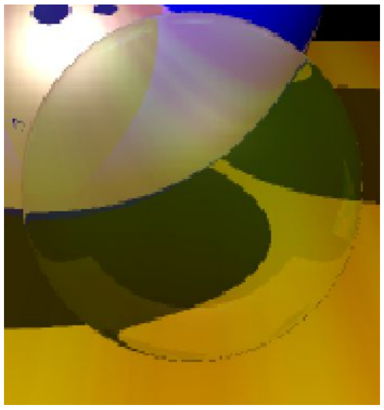
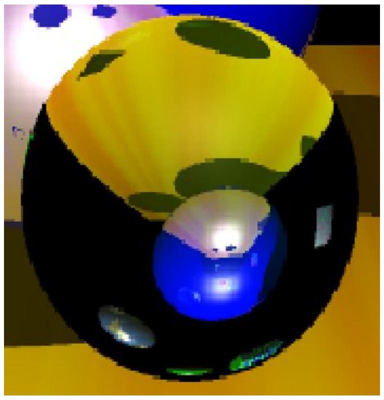


Figure 2.0: With Anti-Aliasing

❑ **Refraction & Transparency (Sphere: 2 Marks):** The green sphere on the right displays 90% transparent object. This implies a refractive object. In code “transparency = 0.1”. Refraction is computed using secondary rays to find shadows and mirror reflections leading to two normal vectors and refraction rays being generated when transmitting out and of the sphere. It wouldn’t be a computationally expensive without recursion which is what we do when we are raytracing. As for the transparent object, our accomplishment is done by multiplying the sphere’s current color by the transparency (colorSum_variable * transparency). With that value, we may add this color to a refracted color to get the ideal dream of both refraction and transparency. One ETA on the top right shows 1.002. Another ETA on the bottom right shows 1.43.





❑ **Primitive Object(s) (Cylinder: 1 Mark):** Cylinder.c and its corresponding header file affords us the opportunity to calculate the point of intersection and surface normal of the traced ray. Formulas are illustrated below.

Ray equation:

$$x = x_o + d_x t\,; \quad y = y_o + d_y t\,; \quad z = z_o + d_z t$$

where o denotes the ray’s origin, d denotes the direction of the ray, c denotes the center of the cylinder, and the value of t denotes the distance from the ray’s origin to the point on the ray.

Surface normal vector for a cylinder:

$$\text{(un-normalized)} \quad \mathbf{n} = (x - x_c,\ 0,\ z - z_c)$$

$$\text{(normalized)}\ \mathbf{n} = (\,(x - x_c)/R,\ \ 0,\ \ (z - z_c)/R\,)$$

```
glm::vec3 Cylinder::normal(glm::vec3 p){
    glm::vec3 d = p - center;
    glm::vec3 n = glm::vec3(d.x, 0, d.z);
    return glm::normalize(n);
}
```

Intersect Equation: $t^2(d_x^{\,2} + d_z^{\,2}) + 2t\{d_x(x_o - x_c) + d_z(z_o - z_c)\} + \{(x_o - x_c)^2 + (z_o - z_c)^2 - R^2\} = 0$ The intersection point is computed by solving the quadratic equation for t where R is the radius of the cylinder.

```
// using Ray-Cylinder Intersection Equations (split) w/ t factored out and used later.
float a = pow(dir.x, 2) + pow(dir.z, 2);
float b = 2 * ( (dir.x * d.x) + (dir.z * d.z) );
float c =  pow(d.x, 2) + pow(d.z, 2) - pow(radius, 2);

float delta = pow(b,2) - 4*a*c; // comes from general formula for ax^2+bx+c=0
```

Failures and Successes:

- ❏ **Failure:** Texturing the floor as a sun correctly on a plane. My compromise was to just make it procedural texture.
- ❏ **Success + Failure:** Super-Sampling was interesting conceptually and easy to grasp. I almost went and did adaptive sampling, but it would’ve slowed down the whole process. I scrapped the idea entirely.
- ❏ **Failure:** Multi-threaded raytracer? I have no proof i’ve attempted it via code, but the parallelism i implemented was not working as intended. It kept spawning an extra thread when I didn’t need it. I encountered a couple of memory errors.

References:

- ❏ COSC363-S1 Computer Graphics Lectures by Professor R. Mukundan