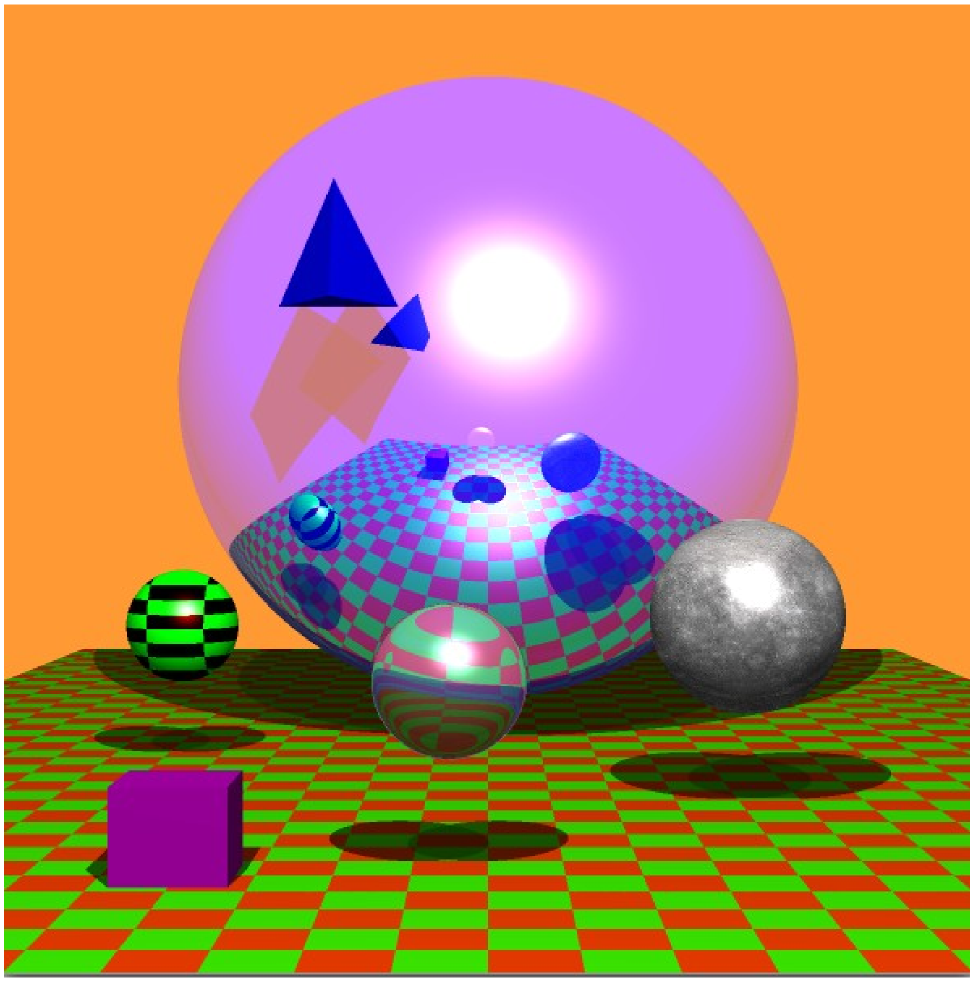
COSC 363 Computer Graphics Assignment 1

William Jelley
91765946
02/06/2016

Brief description of raytracer:
I have created a scene that includes a variety of objects, including spheres, a cube, a checkered plane and a tetrahedral. One of the spheres has been textured using an image of Mercury, one of them has been textured using a procedural pattern, and one of them is a large reflective sphere with a light purple colour. I have included a sphere that is not only transparent, but exhibits realistic refraction as rays pass through it. Shadows are cast from each object, which are determined by the positions of the two light sources. In regions of the shadows where neither light can reach, the shadow is noticeably darker.

Fig.1 Raytracer Scene



Implementation of extra features:

Tetrahedron:

I created a new subclass of the object class called Plane3 and used it to create triangles. This was similar to the Plane class, but I had to re-implement the "isInside" function to test if a point is within the triangle. I then created a "drawTetra" function which given a centre point, desired size and colour, created a tetrahedral using four triangular planes.

Multiple light sources:

Two light sources were created, at positions (10.0, 40.0, -5.0) and (-10.0, 40.0, -5.0). One issue I initially had was that the shadows created by each light merged together and looked very unrealistic. As shown below, I resolved this by using a darker grey when two shadows overlapped.

Fig.2 Merged Shadows

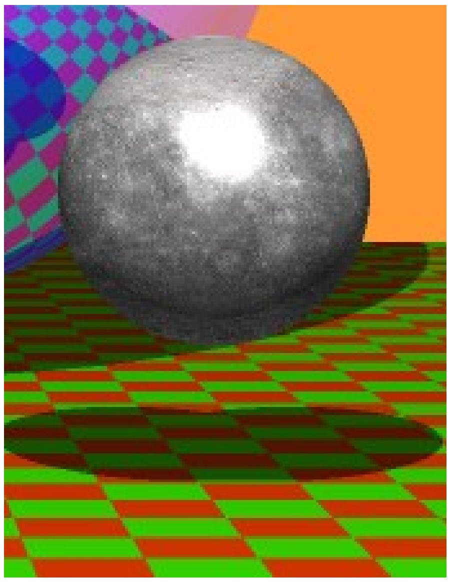
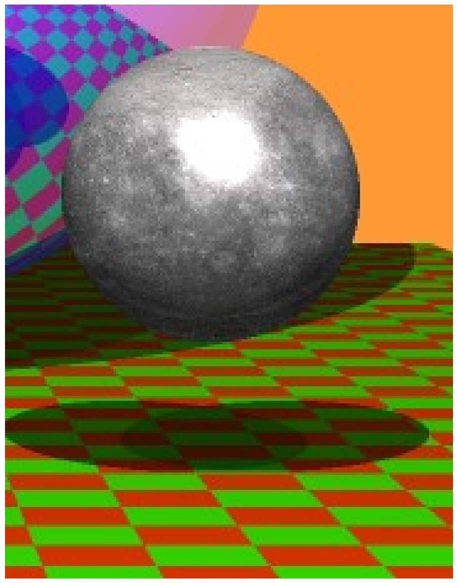


Fig.3 Realistic Shadows



Refraction:
I found the trickiest part about making the sphere refractive was knowing when a ray of light was exiting the sphere. This was solved using the “closestPt” function, which finds the point on the surface of the next object a ray hits. The next point in the case of the refractive sphere, is the point where the light ray leaves the sphere. A derivation of Snell's Law shown below, was used to determine the direction of a ray entering a different medium. This step was repeated when the ray exited the sphere.

Fig.4 Vector representation of Snell's Law

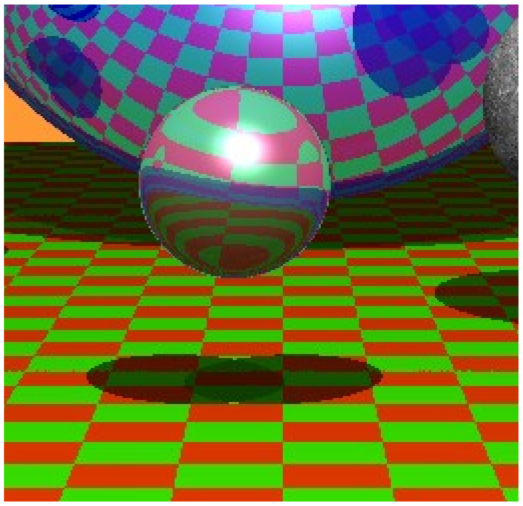
$$\boldsymbol{g} = \left(\frac{\eta_1}{\eta_2}\right)\boldsymbol{d} - \left(\frac{\eta_1}{\eta_2}(\boldsymbol{d}.\boldsymbol{n}) + \cos\theta_t\right)\boldsymbol{n}$$

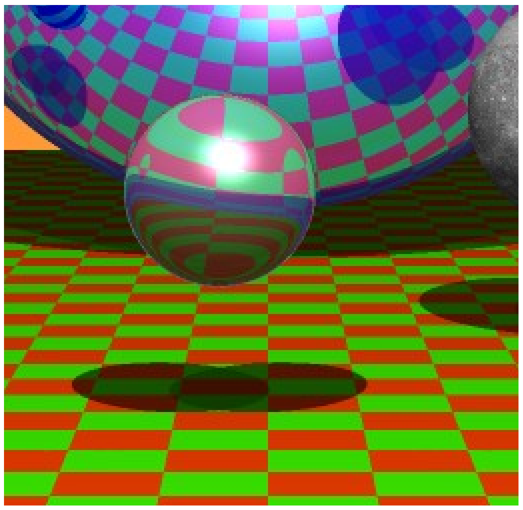
$$\cos\theta_t = \sqrt{\left(1 - \left(\frac{\eta_1}{\eta_2}\right)^2 \left(1 - (\boldsymbol{d}.\boldsymbol{n})^2\right)\right)}$$

Note: n1 and n2 are refractive indices of medium light is leaving and entering, d (a vector) is the direction of the original ray, n is the normal vector, theta is the angle with respect to the normal of the transmitted ray, and g (a vector) is the direction of the transmitted ray.

Anti-Aliasing:
Before anti-aliasing was introduced, each object had very jagged/pixelated edges. I used the method of super sampling and divided each pixel into four segments. The colour values computed from each segment were then averaged to give the new final colour of the original pixel. This had a noticeable effect in smoothing the edges of the objects in the scene.

Fig.5 Before anti-aliasing. **Fig.6** After anti-aliasing.





Non-Planar object texture using an image:
The (x,y,z) values were of the sphere were converted to two dimensional normalized coordinates (u,v) with values in the range [0,1]. Finding 'v' was relatively simple: The lowest y value in the sphere was mapped to 0, the highest mapped to 1, and the rest of the values scaled accordingly by dividing by the height of the sphere. The 'u' value depended on both the x and z values. If the z value was greater than z0 (the value at the center) then the minimum x was mapped to 'u = 0' and the maximum x was mapped to 'u = 0.5'. However if the z value was less than z0, the maximum x was mapped to 0.5 and the minimum x was mapped to 1 (values in between scaled linearly to be within [0.5,1].

Non-Planar object textured using a procedural pattern:
A simple procedure was used to texture the striped pattern onto the sphere, similar to the checkered floor. The colour of the sphere was set to green. The x and y values of each point were converted into integers and scaled by ½. If the scaled values of x and y were both even or both odd, the colour at that point was changed to black. If not, it remained green.

Resources:

The planet texture were sourced from
http://www.solarsystemscope.com/nexus/textures/planet_textures/
Small snippets of code were based on work done in the COSC363 labs.
The equations in Figure 4 are from Mukundan's notes on
http://learn.canterbury.ac.nz/mod/resource/view.php?id=130359