Ray Tracing Assignment Report

Haruka Ichinose / 35943870

Success

From the basic requirement, this ray tracer has the function of:
> Three spheres and three planes, aligned symmetric.
> Transparent sphere, with refraction and without refraction
> Shadows casted depending on the transparency, reflectivity, and refractivity of objects.
> Tetrahedron and open-end triangle prism as objects constructed using planes.
> Chequered pattern on the bottom plane.

From the extension requirement, this ray tracer has the function of:
> Cone and cylinder
> Refractive sphere.
> Anti-aliasing with depth one
> Texture on non-planer object.
> Tilted stripe pattern as a procedural pattern.
> Fog.

Failure

This ray tracer does not have:
> Multiple light source
> Spotlight

Trace function

In the trace function, the flower texture*1 was mapped on a top-left sphere (Please see image 1). It computes theta to get the s coordinate, and phi to get the t coordinate. s = 0 corresponds to the minimum x value, and s = 1 corresponds to the maximum x. Hence the sphere is only textured on the z direction.

The bottom plane has a chequered pattern. The algorithm used to generate this pattern is application of stripe pattern from the lab. Now the ray tracer has a variable ix to compute the stripe pattern across the x direction and taking modulus to determine which colour to use.

The two planes on the sides are making a large triangular prisms with the bottom plane. They have tilted stripe pattern which is generated using the similar algorithm with stripe patter. The planes were coloured according to the modulus of the sum of y coordinate and z coordinate. However, since the planes are leaning for 30 degrees from the

vertical direction, the y coordinate used in the sum was factored by $\frac{2}{\sqrt{3}}$, which is the ratio of (side length)/(height) of equilateral triangle.
The fog was implemented just the same as outlined on the lecture slides with $z_1 = -40.0$ and $z_2 = -200.0$

The shadows have different thickness depending on the property of the objects with the cases of (1) refractive and reflective object, (2) reflective object, (3) refractive object, (4) transparent object, (5) non-transparent object. It takes in account of reflection coefficient, refraction coefficient, and transparency coefficient. However, it does not consider the material's colour of obstacle.

The final part of trace function is to add the colour from secondary rays. Their algorithms are the same as shown in the lecture slides.

Display function

The display function has been modified to do anti-aliasing. The rays of four subcells were computed for each original resolution cells. The ray1 goes through the bottom-left subcells, ray2 for bottom-right, ray3 for top-left, and ray4 for top-right. The difference in RGB values for the 1st cell and 2nd cell, 1st cell and 3rd cell, and 1st cell and 4th cell was compared. The comparison for 2nd and 3rd, 2nd and 4th, 3rd and 4th were abbreviated in order to save the running time. The anti-aliasing was done only when the sum of the absolute difference of (r, g, b) are greater than 0.2. Other wise, the colour obtained form the ray through the centre of original cells were used.

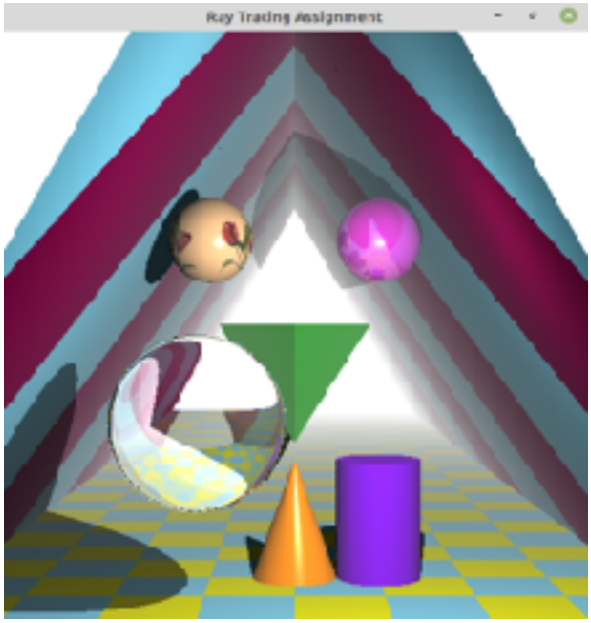
The purpose of anti-aliasing is to reduce the jaggedness along edges of polygons and shadows due to using discretised image space. Anti-aliasing in this program gives four times resolutions coming from the four subcells. A hybrid of supersampling and adaptive sampling was used in this program since the anti-aliasing is activated only when there is a significant difference in the colour within a cell which can save the running time. However, this tracer does compute the colours of further subcells. It only divides the original cell into 4 cells once. This program takes about 19 seconds on Linux Mint of my mac computer. Please see image 2 and image 3 for anti-aliasing output and without anti-aliasing output.

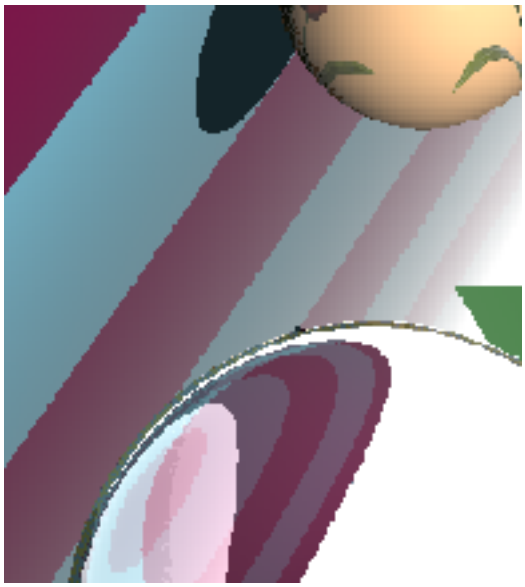
Objects

Corn.cpp, Corn.h, Cylinder.cpp, and Cylinder.h are provided. The Cone is closed and the intersection point and normal vector for the side surface and top are computed using the algorithm learned from lecture. The Cylinder is rendered for only the bottom side by omitting the case of (hit.y > center.y + height). The algorithm for the cylinder is also

from the lecture slides, however, a website *2 about ray tracing was referred to check the t equation.

Images





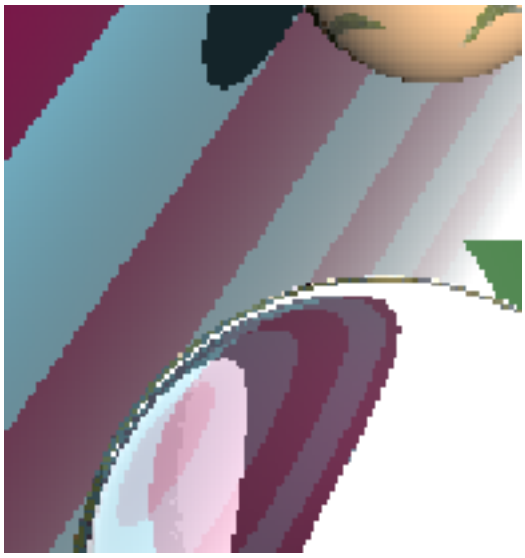


Image 1(top) shows the whole scene.
Image 2(bottom-left) shows the result with anti-aliasing.
Image 3(bottom-right) shows the result without anti-aliasing.

Reference

[1]… “VaseTexture.bmp” from University of Canterbury, COSC363
[2]… “Ray Tracing Primitives” https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html