3D Programming Report

Assignment 1

Francisco Sousa	86416
Francisco Nicolau	86419
João Martinho	86454

For this assignment, we were tasked with building a Turner Whitted Ray Tracing Algorithm capable of rendering 3D scenes featuring spheres, triangles, planes and multiple light sources by employing the Blinn Phong shading model and other ray tracing practices such as global illumination (for reflections and refraction) as well as hard shadows.

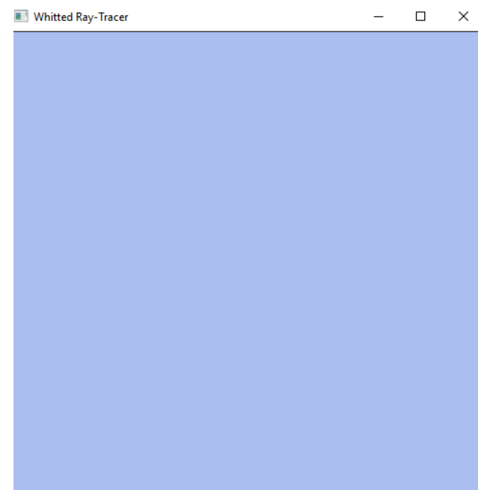
Besides that, adding some stochastic sampling techniques, like anti-aliasing (with the jittered method); soft shadows using an area of light with a set of N light source points (when without antialiasing) and the random method (when with antialiasing); and the depth of field effect, where the lens is simulated by a random distribution of N samples on unit squares and unit disks.

Finally, we had to build a uniform grid to work as an acceleration structure.

To achieve this, and some more **extras**, we tackled each of these features sequentially following the course's slides, as we will describe shortly.

Casting Primary Rays

Our first task was to implement the `primaryRay()` and `rayTracing()` functions. The first is responsible for creating a Ray object for each individual pixel in the viewport, the other is our ray tracer's main routine. This routine is called for all of the

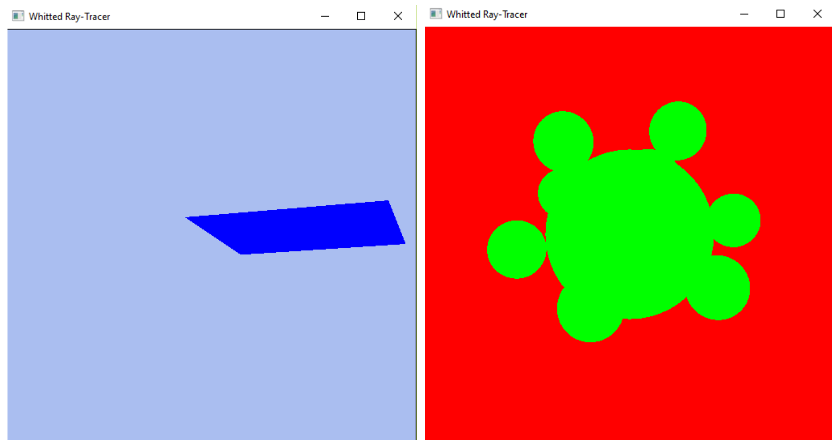


primary rays created and will then determine the color of the pixel by iterating through the scene's objects and finding the closest one in the ray's path.

Since no intersections were calculated, the initial result was just the background color.

Geometry Intersections

Now that we were able to cast rays and our ray tracer had its skeleton laid out, it was time to determine the intersections between rays and basic geometry, like triangles, spheres and pl

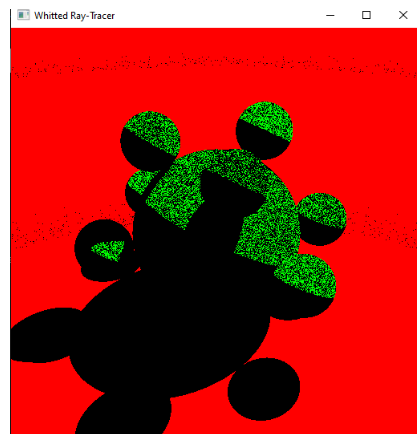


In these pictures we can clearly observe two triangles (on the left) and the *balls_low* scene being rendered with solid colors, blue for triangles, green for spheres and red for planes.

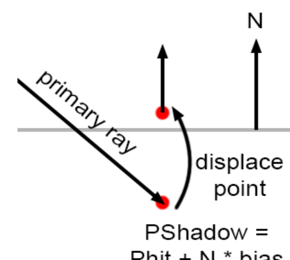
Shadow Feelers

To get hard shadows displayed in our rendered images, for every pixel and every object in our scene we cast a secondary ray from the intersection point in the direction of all light sources to determine if it collides (in shadow) or it doesn't (not in shadow).

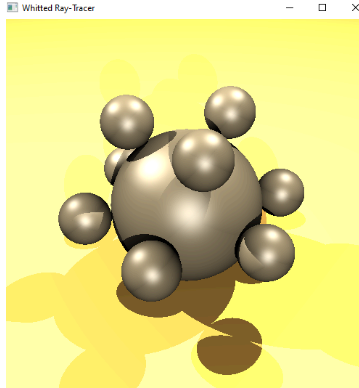
Without the shading we simply output a flat color (black) for the pixels in shadow, here we have the *balls_low* scene with a single light.



Avoiding self Intersection



In the image above we can see that our renders had some shadow-acne so we employed the method taught in class to fix this.

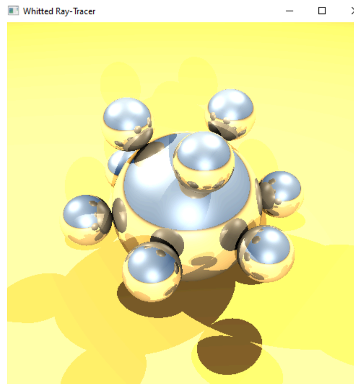


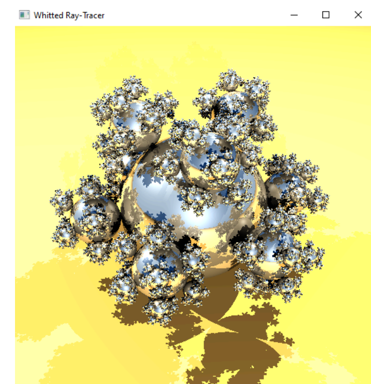
Blinn Phong Model

Finally it was time to implement the Blinn Phong Shading Model. We got some nice results for materials with diffuse and specular components.

Reflections

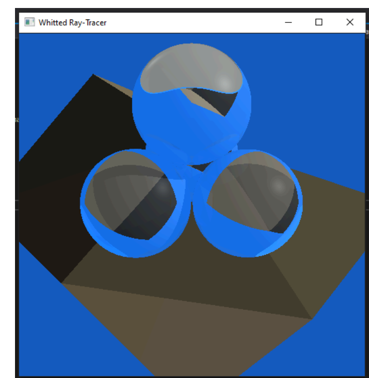
The next step was taking care of global illumination, beginning with reflections. Just like previously, we used the offset-corrected intersection point of the primary ray and the geometry to then cast a secondary ray.





We also provide a depth variable to make sure that rays won't bounce forever, and eventually stop.

Refractions

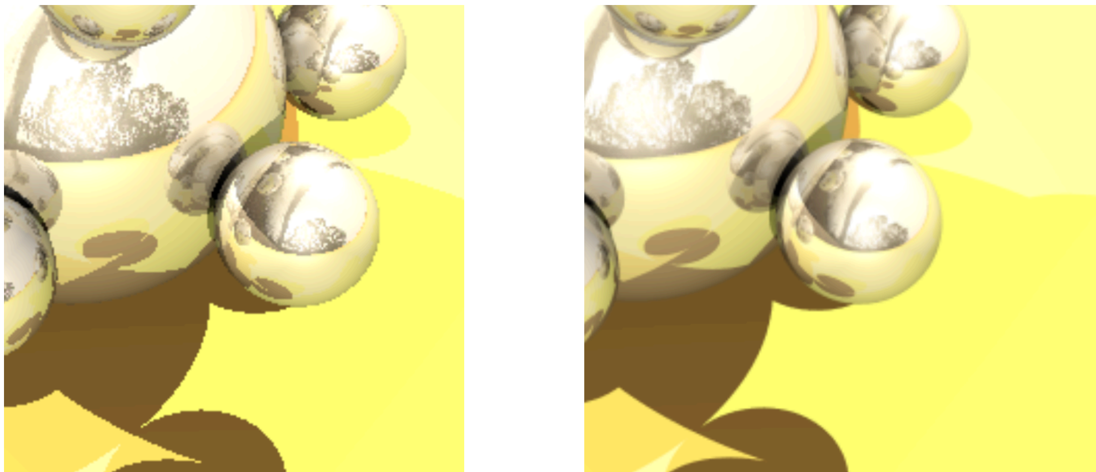


After that, and to finish the global color, we implemented the refractions where, with support of Snell's Law, the ray changes direction when changing between materials. To do this, we added a flag that marked if the ray was inside or outside an object, so that we knew what was the index of refraction. A bug

we had here was that we calculated shadow feelers when inside an object. This flag helped fix that.

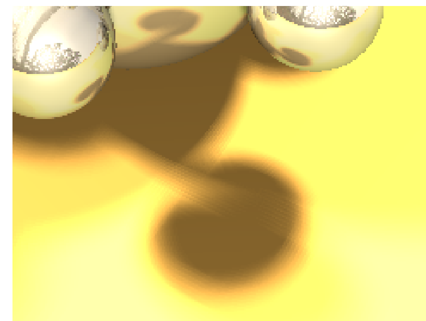
Anti-aliasing

Starting the stochastic sampling techniques and distributed ray tracing, we introduced anti-aliasing, which can be resumed as shooting multiple rays per pixel, collecting each ray's color contribution and averaging them. The variation we used was jittering, where the pixel is divided in a grid and, in each cell, the ray is shot at a random position in that cell. Below we have a side by side comparison of the same scene with and without antialiasing.

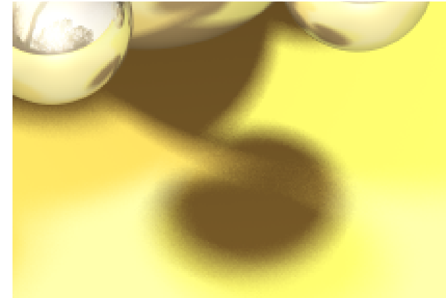


Soft Shadows

Having hard shadows, soft shadows were the next step. The basic idea behind this is to emulate that our light sources are now areas, instead of single points. Now, and because can be used with or without anti-aliasing, we have these two cases:



When without, we create multiple light sources around the original ones, give them a lower potency and then use shadow feelers as usual.



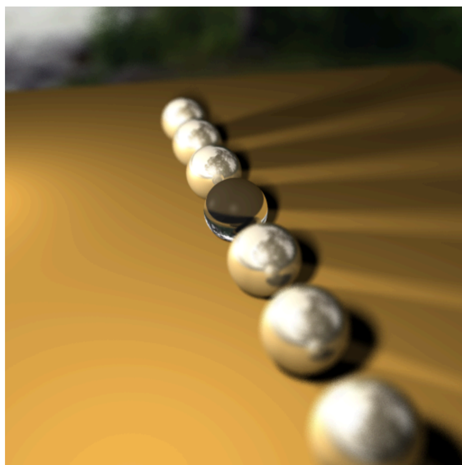
With the anti-aliasing on, we shoot the multiple rays per pixel at positions around the light source in a seemingly random fashion, which results in some noise for each pixel, giving the impression of softness in the shadows. As some **extra work**, since we already use jittering for the antialiasing, we also used this technique for the casting of the shadow feelers. We do this by casting the feelers in the same relative pixel cells as the primary ray, but in the context of a grid around the light source instead.

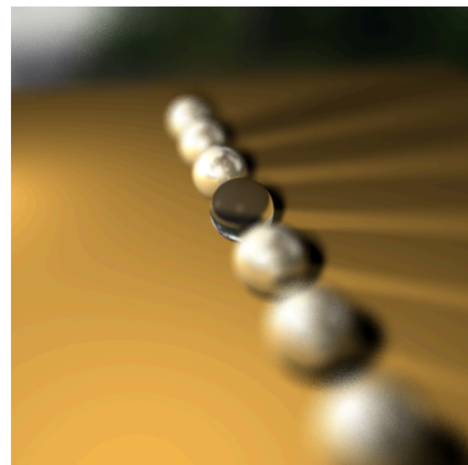
Depth of Field

At last, for the distributed ray tracing, we implemented the depth of field notion. This simulates how a real camera, by having its lens more or less opened, as well as focusing on different planes of the scenery, is able to focus only specific objects, while others outside of the plane of focus appear blurry.

For this to happen, we cast rays starting from samples around the eye of the camera (which emulates the lens) and then shoot them in the direction of where a normal ray would hit the plane which we want to focus. Because the rays start at different points in the camera, when we average their contributions for each pixel's color, noise is created and objects outside of the focal plane appear to be blurry.

The images below show how different apertures affect the notion of blurriness as objects are further away from the focal plane. On the left we have an aperture of 15, and on the right it's 30.





Uniform Grid Acceleration Structure

Since ray-tracing is a very computation heavy process, we need ways to improve its performance.

As such, we implemented a structure called Uniform Grid, where the environment is split into equally sized cells and objects are stored in the cells where their bounding boxes are present. Then, when we cast a ray, we see which cells it traverses through and only test intersections with objects in these cells. This way, we're able to reduce by a lot the number of test intersections we do, because otherwise the ray's intersections would be tested with every object in the scene.

Because object bounding boxes on objects are not the object's actual boundary, the ray may intercept an object that belongs to a cell but isn't actually in the cell's space. This causes a problem of multiple intersection tests, because in the next cell the ray may test again it's intersection point with the object, which is redundant and unnecessarily expensive. As such, and as **extra work**, we implemented the notion of mailboxes, which store the id of the latest ray they were tested against and only calculate intersections for rays they haven't tested intersections with.

Extra work

We mentioned before that we did as extra work the jittering soft shadows as well as the notion of mailboxes.

Other than this, two other extra steps we implemented in the project were intersections with axis aligned boxes (picture on the left) and also square lenses, instead of round ones, for the depth of field effect (picture on the right).