Catarina Sousa 93695 Maria Ribeiro 93735 Patrícia Vilão 93604
3D Programming - Assignment 2

# Group 1: Assignment 2

## Introduction:

In this report, we explain the steps we had to implement in order to develop a Raytracer in GLSL.

## Intersections:

The first thing to develop in order to create a Raytracer is the intersections. Having this in mind, we are going to clarify how we deal with the intersections of different polygons.

**Sphere:**

To calculate if there is an intersection between a ray and a sphere we used the same method as in the first assignment. First, we start by checking if the ray's origin is outside the sphere. If it is, then we check the sign of the dot product between the ray's direction and the vector between the center of the sphere and the origin of the ray to understand if the sphere is "behind" the ray (dot product with a negative sign).

- If the sphere is behind the origin of the ray we can discard it right away and we avoid unnecessary calculations.
- Otherwise, we calculate the discriminant, which indicates if there's a hit or not.
    - If the discriminant is less or equal to zero there is no hit.
    - Otherwise, there is an intersection between a ray and a sphere. If the ray's origin is outside the sphere, we calculate the smallest root and if it is inside we calculate the positive one, which corresponds to the hit points.

**Moving Sphere:**

In order to implement the intersection with the moving sphere object, we used the same approach as with the sphere. Since the moving sphere center is between two points (center 0 and center 1) we also created a function that calculates the position of the center, considering the time.

**Triangle:**

For the triangle intersection, we implemented the Moller Trumbore Intersection Algorithm because it is an efficient algorithm since it doesn't require the precomputation of the plane equation of the plane containing the triangle. In order to complement the slides, we also viewed this video which explains the algorithm https://www.youtube.com/watch?v=fK1RPmF_zjQ.

## Global Color:

**Diffuse objects:**

For diffuse objects, we implemented diffuse reflections for color bleeding. To achieve this color we consider a unit radius sphere tangent to the hit point and calculate a point on its surface by adding the hit point, the hit normal, and a random unit sphere. To obtain the ray's direction, all we need to do is to compute the new direction that goes from the hit point to the new point. After that, we can create a new ray that travels from the hit point in the new direction.

Catarina Sousa 93695 Maria Ribeiro 93735 Patrícia Vilão 93604

3D Programming - Assignment 2

**Metallic objects:**

For metallic objects, we compute the ray's direction with the formula that was taught in classes and we obtain the fuzzy reflection effect by deviating the computed direction into a random new one using a sphere with a radius equal to the material's roughness value. After obtaining the new direction, we can create a new ray that travels from the hit point in the new direction.

**Dielectric objects:**

For dielectric objects, firstly we verify if the ray hits from the inside or the outside of the object and compute the values (for example, the normal) to do the necessary calculations accordingly. The next step is to use probabilistic math in order to decide if it is to scatter a reflection or reflection ray, for that, we calculate the discriminant (as described on this website: https://blog.demofox.org/2020/06/14/casual-shadertoy-path-tracing-3-fresnel-rough-refraction-absorption-orbit-camera/). Then if it is not a total reflection, we calculate the Schlick approximation of Fresnel equations to get the contribution of the Fresnel factor in the reflection of light, else the reflected probability is 1. Then taking into account the probabilities calculated in the previous step, we either create a reflection ray using the formulas provided in class and the link previously mentioned, or a refraction ray, once again with the formulas provided in class and with the information available on page 30 of the book "Raytracing in One Weekend" by Peter Shirley.
Regarding this type of objects, we also applied the Beer's law in order to simulate the light absorption inside a transparent dialetric object ($e^{-refractionColor * distance}$).

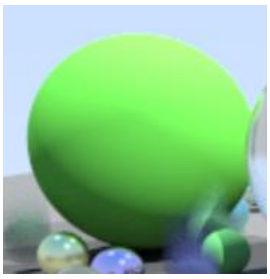

*Image 1 - Diffuse Sphere*

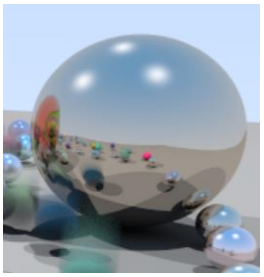

*Image 2 - Metallic Sphere*

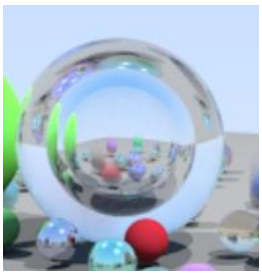

*Image 3 - Two Dialetric Spheres*

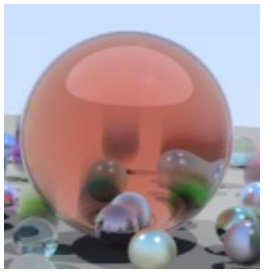

*Image 4 – Transparent Spheres w/ Beer's Law*

## Direct Lighting:

This method is called for each point light created. First, we need to check if the intersection previously computed in the hit_world is in shadow or not. This is done by calling hit_world again but with the new ray that goes from the intersection point to the light point. In case it isn't in shadow, we sum to our current color variable the color computed.

## Depth of field:

To implement the depth of field effect, we created a flag in the document *P3D_RT.glsl* called *DEPTH_OF_FIELD* that, when enabled, activates the depth of field effect. When this flag is enabled we increase the camera's aperture since this opening controls the amount of light coming through the lens. The larger the aperture, the more light is recorded and the shallower the depth of field. With smaller apertures, less light is recorded and the depth of field is greater.

*Image 5 - Overview of the work implemented with Depth of Field*

## Ray Color:

The RayColor method assigns a color to each pixel iterating over a for cycle.

The way it works is by shooting the ray from the camera and, when that ray hits an object it adds its color to the throughput.

First, we calculate the direct lighting with 3 white point lights, calling the *directLighting* function, and then we compute the secondary ray. When we calculate this ray, we need to exchange the current ray with this one in order for it to be updated when the next cycle begins, and we also need to update the throughput with the attenuation.

## Motion Blur:

To create the motion blur effect in this project, it was only necessary to successfully implement the moving sphere object, since the camera object already had the time0 and time1 attributes to simulate the shutter speed.

## Extra - Fuzzy Refraction:

To implement the fuzzy refraction in the project, the approach was similar to the fuzzy reflection, we added a new attribute to the material object, named *refractionRoughness*, and like in the fuzzy reflections, the idea is to change slightly the direction of the refracted ray, taking into account the value of the refraction roughness attribute.

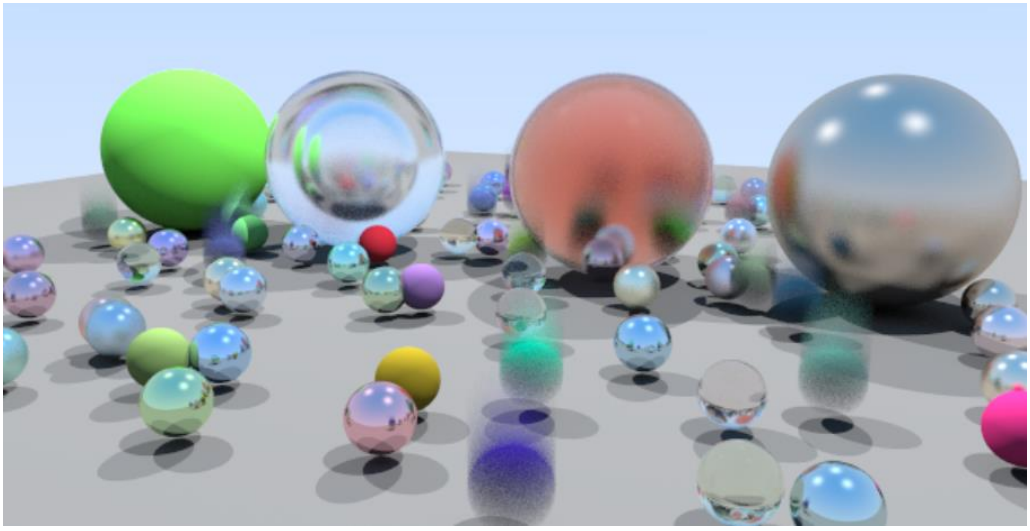

*Image 6 - Overview of the work implemented with fuzzy reflections and refractions, motion blur and without depth of field.*