Assignment 2 Report

Francisco Cecílio, 89444; Hélio Martins, 98692; Miguel Malheiro, 98846; Grupo 13

Intro

For the Assignment 1 we implemented ray Tracing techniques plus two acceleration structures, but as described in the report they are not ideal to render scenes with lots of randomly spread positioned objects. In this Assignment 2, the main goal was to try a GLSL implementation of a Progressive Ray Tracer that allows the rendering of “embarrassingly parallel” scenes in a short time.

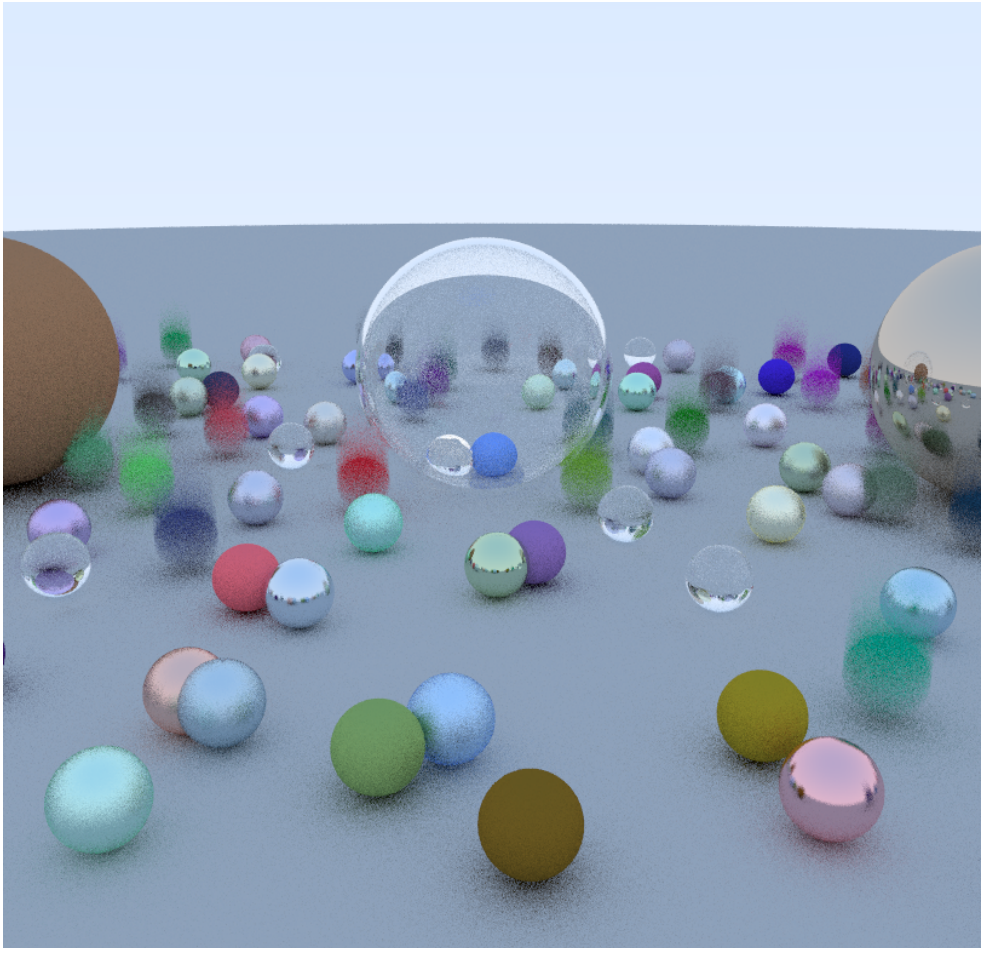

Figure 1: Goal output of the rendering scene.

Setup

We used *Visual Studio Code* and imported a “GLSL template” provided by the teacher. In order to see the output of our GLSL program we installed the *Shadertoy* plugin that generates a live WebGL preview of GLSL shaders within *VSCode*.

Process

1. Ray intersections with spheres and triangles

1.1 Create and Get Ray

As usual, we need the camera and a pixel sample to create a ray but in this assignment we also have a structure that has the *origin*, *direction* and *time* (used for motion blur, as we explain ahead) for each ray. GLSL doesn't allow recursion and because of that calculating the ray color is trickier.

1.2 Intersections with spheres

We calculated the spheres intersections by checking if the ray intersects the sphere in at least one point, being the closest hit point the intersection used.

1.3 Intersections with triangles

The triangles' intersections were calculated using the Möller-Trumbore algorithm with the Cramer Rule.

1.4 Intersections with moving spheres

Calculate the moving sphere intersections by checking if the ray intersects with the sphere around the moving center with the moving center at time *t*.

2. Local color, multiple source lights, hard Shadows

2.1 Local color

To get the local color of a pixel, after the ray intersects an object, we will loop MAX_BOUNCES times to "build" the color over each frame. To build the color, we begin to sum the scene lights' influence on that pixel, which takes into account the object material (using the *directLighting* method). Then we want to calculate the secondary rays and update the *throughput*, which is the accumulation of the colors of the rays that originated the current ray. In explanation, for the first ray the *throughput* is 1, for the secondary rays after the first, the throughput is multiplied with a factor called attenuation (if *atten* = 0.5, then *throughput* = 1 * 0.5 = 0.5). This formula is to be followed in the subsequent rays.

2.2 Source lights

Direct Lighting was implemented by adding to the color of the point the color of the light times the diffuse color of the object material plus the specular color. The intensity of this color depends on the maximum between zero and the dot product between the halfway vector and the normal of the hit object to the power of the object shininess. For the materials we defined the following values of shininess:

1. diffuse: 10
2. metallic: 500
3. dialetric: 200

2.3 Hard shadows

For the hard shadows we create shadow rays (rays from the hit point to the light sources), to check if they hit any geometry. If they do, that point will not get the color from that light added onto its final color.

3. Global Color

3.1 Diffuse reflections for color bleeding

To achieve color bleeding, the colors of a material are calculated from the surrounding colors of the other objects. We implemented the code but when we turn it on the colors of the diffuse objects become the same color of the floor (bin too strong).

3.2 Mirror and fuzzy specular reflections for metallic objects

We did a normal reflection, therefore rough reflections are not implemented.

3.3 Mirror reflection and refraction for dieletric materials

For dialetric material reflections, since we don't have multiple child rays we need to choose whether to reflect or refract at a given point. This is done by applying the S*chlick* approximation, where we calculate the probability of a ray being refracted or reflected on a given point. After this, we just randomly choose between the two using a random number and this value. After this, we proceed to either reflect or refract the ray and return that as the scattered ray for that point.

4. Motion blur with spheres

The motion blur with spheres was accomplished by calculating the moving center of the sphere in function of the time passed in the frame and then drawing the sphere with lower opacity. This way, after a couple frames you see the motion blurred sphere with the linear motion.

5. Depth-of-Field

For the depth of field, we implemented … but it doesn't work because ...

6. Extras (nothing implemented)

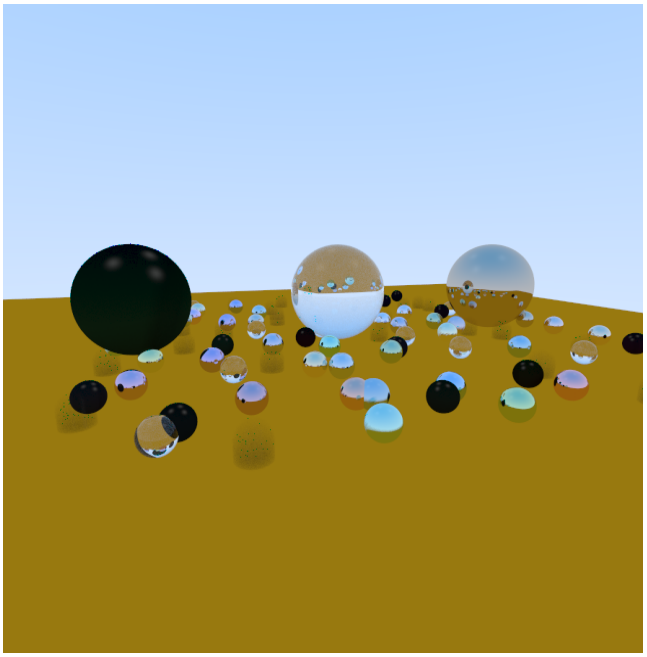

Figure 2: In our output we can see the dialetric material in the middle sphere, the black sphere is the diffuse material (not working) and the right sphere is the metallic .