INSTITUTO SUPERIOR TÉCNICO

Assignment 1 - Group 3

Mónica Alves 90759
Miguel Mota 90964
André Santos 84699

June 30, 2022

1 MAIN IMPLEMENTED FUNCTIONS

1.1 File P3D_RT.glsl

1.1.1 Function ray_color()

This function receives a ray and determines the color of that ray. If the ray doesn't hit anything then the background colour is returned. If there is a hit, it calculates the color by calculating the direct lighting for each of the light sources(three in the case of our project) and after that calculating the secondary lighting using the scatter function.

1.1.2 Function directlightning()

This function calculates as its name implies, the direct lighting. In this function the specular color, diffuse color and shininess for each type of material are defined and shadows are calculated.

1.2 File common.glsl

1.2.1 Function get_ray()

This function is used for creating rays that will be used for the scene rendering.
As we know, the ray is defined by an origin and its direction.
In this assignment we also used a certain moment in time to define it, but it was only used to create the motion blur effect.

The origin (which we call *eye_offset* in our project), is calculated using the camera position (*cam.eye*) and the lens sample (used for the *DOF* effect), which is transformed into camera coordinates.
The ray direction (receives *ray_direction()* name in project), is calculated using different camera's components like dimensions (width and height), coordinates (u, v, n) as well as the plane distance and the focus distance. It is normalized before sending it to the ray's creation. (More detailed calculation can be seen in the function itself)

1.2.2 Function scatter()

This section is focused in one of the main components of our project. This function is responsible for deciding how "special effects can affect" the materials in our scene in order to achieve more realistic results.
This is why this function is divided in three parts: Diffuse materials, metal materials and dielectric materials.

If the object's material is diffuse, a random point is calculated for the diffuse scattered ray. This is done by adding to the object's position and normal, a randomized point within a unit sphere. That new calculated point is then used to compute the scattered ray's direction.
The diffuse materials are characterized not by emitting light but by modulating the color of their surroundings with their own intrinsic color, giving a matte final look. In order to achieve this, we calculate the attenuation using the material's property, *albedo*.

If the object's material is metallic, we simply used a built in function called reflect(), that calculates the reflection vector. This is used to create the scattered ray, giving it a metallic and reflective special effect that affects the objects.

If the object's material is dielectric, then we must take into consideration two parts: The refraction and the reflection.

Similarly to assignment 1, depending if the ray's origin started inside or outside the surface, different computations need to be done. In the case of the record being hit from the inside, it is needed to calculate the attenuation using the *Beer's Law*.
Beer's Law relates the attenuation of the light to the properties of the material through which the light is travelling, as can be seen in our code.

On another note, when the object's material is dielectric, we use the probabilistic math to decide if the scatter ray is reflective or refracted. The calculation of both is the same as in assignment 1.

1.3 Geometry related functions

We only added three types of geometric forms into our project: **Triangles**, **Spheres** (still and moving. The latter will be mentioned later in the *Visual Effects* section), and **cubes** (which will also be explained in detail in the *Extras* section).

Each of these geometric forms had its **structure** and **creation function**. Analogously to our previous project, each of them also have a function that calculates whether the object has been intercepted or not by the cast ray.
These functions (*hit_triangle*, *hit_sphere*, *hit_movingSphere* and *hit_box*) are used by the *hit_world()* function, which was already given to us at the beginning, in order to add the geometric forms to the scene with a particular color and material.

2 VISUAL EFFECTS

2.1 Depth Of Field

In order to add depth of field in our project, we had to use a lens sample. This was created equally to the assignment 1 we delivered.

By creating a lens sample with a unitary disk instead of a single point in the camera's eye, we simulated the camera's lens. We cast multiple rays distributed throughout the area and used the aperture to define said samples: The larger the aperture, the more out of focus near objects in the scene are.

2.2 Moving Spheres

We considered this section to be inside the *Visual Effects* one because it brings a sense of motion to the spheres in the scene. Moving spheres work almost exactly as spheres, the only big difference is that they have two variable additional variables as seen in the figure below:

```
struct MovingSphere
{
    vec3 center0, center1;
    float radius;
    float time0, time1;
};
```

Figure 2.1: Structure for Moving Spheres

Both these variables are used to calculate the center at a certain time t, as seen on the function below:

```
vec3 center(MovingSphere mvsphere, float time)
{
    return mvsphere.center0 + ((time - mvsphere.time0) / (mvsphere.time1 - mvsphere.time0)) * (mvsphere.center1 - mvsphere.center0);
}
```

Figure 2.2: Center Function

Whenever the moving sphere's center is required(for example in the hit_moving_sphere function) it is calculated using the function above.
The time used in the center function comes from the ray. To produce the motion blur effect, the camera generates rays at random times and intersects with the model one at time.

3 Extras

For extras in this project we added a **cube** (receives the name of *box* in our project) that can be seen in the scene. For this geometric form we basically used the bounding box functions we created for the previous project and adapted them for this new one.

As it has already been mentioned, the box is a geometric form and has the same kind of functions as the rest of the forms explained before (structure, creation function and hit_box that calculates interception between cast ray and object). The interception algorithm used is the same that was used in the previous project for the intersections with AABBs.

This box has a metal material and a pink color, as can be seen in figure 3.1.

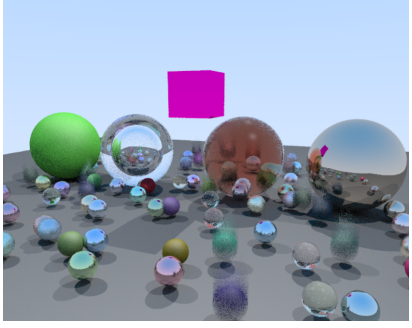

Figure 3.1: Cube for extras