3D Programming Report

Assignment 2

Francisco Sousa	86416
Francisco Nicolau	86419
João Martinho	86454

For this assignment, instead of Unity3D, we decided to delve into a Monte Carlo Path Tracing Algorithm capable of rendering 3D scenes featuring spheres, triangles, planes and object lights, employing the Blinn global illumination with BRDFs, reflections and refraction.

Besides that, adding some stochastic sampling techniques, like anti-aliasing (with the jitter or tent filter); natural soft shadows (an advantage of the path tracer); 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 built a Uniform Grid and a Bounding Volumes Hierarchy to work as acceleration structures.

To achieve this, we followed the smallpt renderer and their slides; implementing it on top of the ray tracer structure from the previous assignment. The course's slides along with Lund's university EDAN30 lecture slides were consulted for the BVH implementation.

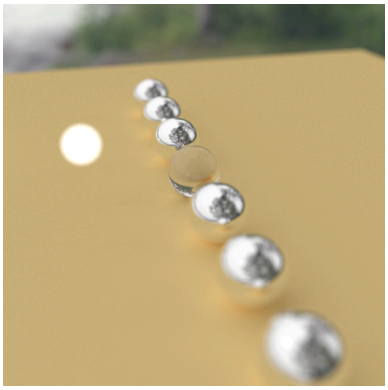
Previous Work

In this project, one of our main objectives was to build on top of the existing framework, in order to be able to obtain a fair comparison with the ray tracer we had done previously. As such, most of the code base which included **ray casting**, **geometry intersections**, **depth of field** effect and the **uniform grid** acceleration structure were already done and were used in the building of the path tracer.

Changes to Scene format

Because our path tracer is defined based in smallpt, we decided:

- Materials can be either diffuse; completely reflective; or refractive with some reflective properties;
- Lights are represented as spheres.



Img. 1: Lights are now spheres.

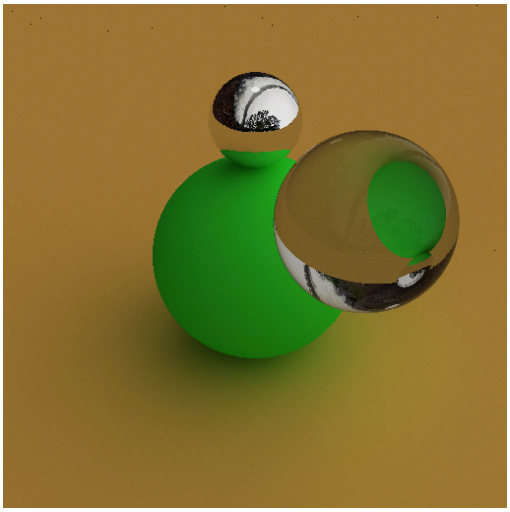
As such, the scene format needs some slight changes to be able to work with our path tracer algorithm:

- With path tracing, diffuse objects should have 1 for their diffuse component; mirror objects should only have their specular component set to 1; and refractive objects should have their diffuse and specular components set to 0. Lights should also have their diffuse component set to 1.
- Materials have three extra values at the end of the line, describing the emission of the object. Because only spheres can be lights, only spheres should have emission.

Note: with this change, the ray tracer scenes are compatible with the path tracer, if the last three (emission) values are added and the diffuse and specular components taken into account.

Path tracing

The path tracer algorithm revolves around the idea of global illumination. This is achieved by, when intersecting with a diffuse object, instead of simply returning its color, we cast rays in a random direction (in the normal hemisphere) in order to take into account other objects' contributions. For refractive objects, the angle also has some randomness, being possible for an incident ray to be only reflected, only refracted, or both. For intersections with mirror materials there is no angle, and the incidence angle is the same as the reflection angle.

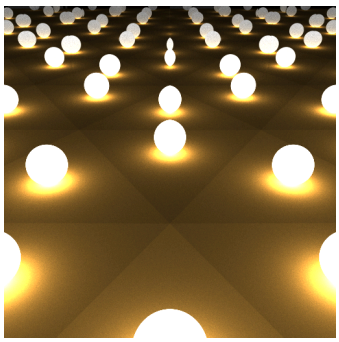


Img. 2: Diffuse, refractive and reflective objects.

Because this algorithm has some randomness when considering diffuse and refractive materials, a lot of noise is introduced, as the pixel colors are averages of the collected colors. To simply mitigate this, we can collect a lot of samples per pixel, which increases the convergence of the colors to the real pixel color.

Russian Roulette

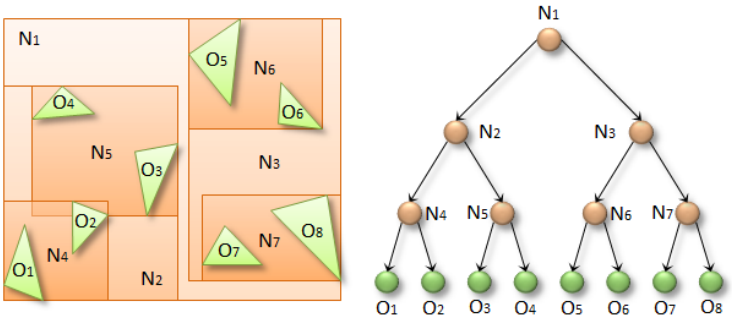
To simulate the loss of light strength over multiple light bounces, a roussian roulette is introduced. This dictates that, after a set number of jumps, there is a probability that a ray won't bounce again. This is also chosen randomly.



Img. 3: Lights bouces.

Bounding Volume Hierarchy (BVH)

To work as an acceleration structure, we also built a Bounding Volume Hierarchy. Here, objects are divided in a tree (as the name suggests), that describes the hierarchy of the objects' bounding volumes. This tree is composed of several nodes, each split in left and right child nodes of their own, which both can have more sub-nodes or be the objects in the scene, if they are a leaf node.



Img. 4: Object separation in root nodes, child nodes and leafs.

The construction of such a hierarchy follows a top-down approach and, according to this algorithm, it starts with every object's bounding volume (AABB) contributing to a root node's bounding volume.
After that, we recursively obtain the largest component of the current node bounding volume, and then calculate a "center" by which to separate the objects. To explain, let us assume, without loss of generality and because this is recursive, that the largest component in this iteration is X:

1. We calculate the current node centroid X component (mid-point);
2. If no object in the current node has its centroid X component smaller or larger than this mid-point, we calculate the object's centroid average over all the centroids X components (avg-point);

3. If, after this calculation, there is no clear division of objects from “smaller” and “larger” centroids, we set the left node to host the maximum number of objects a leaf node can host, and the remaining objects go to the right child node.
4. Else, if there is a clear division, objects are split into the left and right nodes, if they have X smaller and larger (respectively) than the mid-point or avg-point. We then recursively apply the same, but to the left and right nodes individually.

To check if a ray intersects any object, using this BVH, we start by intercept it with the root bounding volume; if it hits, we intersect with both children’s bounding volumes. We then consider recursively the closest child (if any hit occurs) until we reach a leaf, and test intersections with objects in that leaf. If a hit occurs with the object, and this is indeed the closest object to the ray’s origin, this prevents testing objects farther away, possibly decreasing the number of intersection tests.

Tracers comparison

A ray tracer and a path tracer differ but neither is better than the other. While the first is good at simulating all types of materials, the second can't really cope with semi-diffuse semi-transparent objects. On the other hand, the path tracer represents color much better with a natural ambient light that ‘paints’ an object with the colors of other objects (as they do in real life). In this matter, the ray tracer is really soft and only does this for reflective materials. Because of this, the ray tracer bounces rays much less and is faster, while the Monte Carlo algorithm also requires a lot of rays being cast in the beginning. This makes our second tracer have anti-aliasing and soft-shadows given, while we have to add code for it in the first. However, the ray tracer can generate us a raw acceptable image fast and with few ray casted, the path tracer requires a lot of rays for an image without acne. But the results are better.
Both the tracers have pros and cons, and must be chosen according to the situation.

Acceleration structures comparison

The Internet says that the uniform grid and the BVH also produce very similar results, varying on the scene. As we implemented both, we decided to

run the path tracer in two scenes, using 15x15 rays/pixel, and measure their times with each of the acceleration structures.

	path_mirror	path_balls
Uniform Grid	361.0	526.6
BVH	327.78	512.43

Table 1: Time in secs to render scenes with uniform grid vs BVH.

As expected, both the uniform grid and the BVH take roughly the same time to render each scene, but the tree still can do better. We justify these results with the scenes chosen and the low graphics card used.