POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

# Ray Distribution Aware Heuristics for Bounding Volume Hierarchies Construction

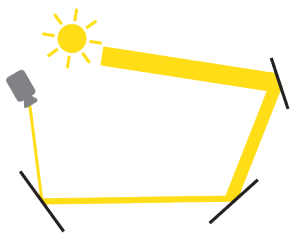
Laurea Magistrale in Computer Science and Engineering

Author: **Lapo Falcone** Advisor: **Prof. Marco Gribaudo** Academic year: **2023-24**

## 1. Background Theory

Ray tracing is a family of techniques capable to perform an approximated light transport simulation to render an image, starting from a methematical representation of a scene. In order to simulate light transport, the main algorithm of any ray tracer consists of the following steps:

1. Cast rays from the light sources present in the scene;
2. For each ray, find the first collision with an object of the scene;
3. Based on the material of the object, decrease the energy of the ray, and make it bounce to a different direction;
4. This process is recursively carried out until the ray loses all its energy, or it hits the camera;
5. If the ray hits the camera, color the hit pixel of its sensor based on the energy the ray is carrying.



Many of the rays cast from light sources end up not hitting the camera, therefore they are wasted. For this reason, in many ray tracing techniques the process is carried out in the opposite direction: the light rays are cast from the camera, and bounce around the scene until they hit a light source. These techniques fall under the name of backward ray tracing, and work thanks to the light reciprocity principle, which states that light transmit in the same way in both directions. From now on, when we refer to ray tracing, we implicitly reference backward ray tracing.

While ray tracing, at its core, is very simple, a lot of research effort has been put into making it faster. The main problem with light simulation is that, in the real-world, a huge amount of rays[1] are cast, in the form of photons, and are then captured by our eyes or a digital camera to produce an image. For this reason, one important family of optimizations has as main goal to reduce the number of rays needed to produce an accurate image.

### 1.1. Kajiya and Monte-Carlo

In order to understand better how these optimizations work, we first have to understand the Kajiya's rendering equation, which concisely describes how light transport work:

$$L_o(\bar{x},\bar{\omega}_o) = L_e(\bar{x},\bar{\omega}_o) + \\ + \int_\Omega b(\bar{x},\bar{\omega}_i,\bar{\omega}_o) \cdot \cos(\bar{n},\bar{\omega}_i) \cdot L_i(\bar{x},\bar{\omega}_i) d\bar{\omega}_i$$

In synthesis, this equation tells us that the amount of light exiting a poin $\bar{x}$ toward direction $\bar{\omega}_o$ is given by a constant emission term $L_e$,

[1] A $100W$ lightbulb produces around $10^{20}$ photons per second.

summed to a reflective term, decribed by an integral over an emisphere $\Omega$,
The integral term computes how much light the point $\bar{x}$ is receiving from all the directions of the hemisphere oriented toward the normal $\bar{n}$ to the surface the point is part of. Then, it uses the BRDF function $b$ to calculate how much of the light entering the point is reflected toward the direction $\bar{\omega}_o$.
The integral term is recursive, indeed, the light entering the point from a given direction $o\bar{m}ega_i$, corresponds to the light exiting another point.

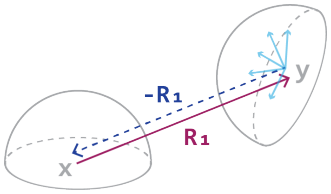



Figure 1: Recursiveness of the integral term.

The task of any ray tracer is to numerically approximate the Kajiya's equation for any point of the scene, in order to, eventually, find out how much light each point reflets toward the direction of the focal point of the camera.
Computing the emissive term knowing where the light sources are located $L_e$ is trivial, the most time consuming part is to resolve the integral. In order to do so, Monte-Carlo integration is used. In this summary we assume that Monte-Carlo integration is known by the reader, and we focus mostly on how to apply it to the ray tracing context. In particular, to estimate the light entering the point hit by a view ray[2], we can cast a certain amount of probe rays toward random directions. The probe rays will hit another point, where othe probe rays will be cast, recursively, until the probe rays lose all their energy[3], or they hit a light source.
The more probe rays we cast, the more the estimation of the light entering a point is accurate, and, consequently, the rendered image. The issue is that, by a fundamental Monte-Carlo estimator property, to cut the estimation error in half, we need 4 times more probe rays. For this reason, some variance reduction techniques have been developed to reduce error without increasing the number of rays.

[2]An initial ray originating at the camera.
[3]In practice, after a certain recursion threshold.

The variance reduction technique important for this thesis is called importance sampling. Instead of casting the probe rays toward uniformly random directions, we cast them by following a non-uniform Probability Density Function (PDF). The PDF should be proportional to the integrand function but, since it is unknown, we can approximate it by casting more rays directly toward light sources. This importance sampling method is called light sampling, or Next Event Estimation (NEE).
NEE makes it so that the ray distribution inside the scene is not uniform, but it is more dense in proximity of light sources. This is a core hypothesis of our work.

## 1.2. BVHs

Until now, we have assumed we know where a ray intersect an object of the scene. In the real world, this is not the case, and finding the ray-scene intersection is a task common to any ray tracing algorithm. For this reason, the second big family of optimization is to find a way to accelerate this process. In the vast majority of graphics applications, the scene is described by a huge amount of triangles, approximating the surfaces of the objects. An object formed by triangles is called a mesh.
Therefore, the problem of finding the ray-scene intersection can be reduced to the problem of ray-triangle intersection, which has a known and very optimized solution. The first, naive, way of finding the intersection between a ray and the scene is a brute-force loop over all the triangles present in the scene. However, this method complexity increases linearly with the number of triangles or the number of rays in the scene.
A better technique, which is used in state-of-the-art ray tracers, is to organize the triangles hierarchically, in a spatial binary tree called Bounding Volume Hierarchy (BVH). The idea is to divide the scene into two parts, and enclose the triangles inside each one into two parallelepiped aligned with the cartesian axes, called Axis-Aligned Bounding Boxes (AABB). Now, if a ray doesn't intersect an AABB, it won't intersect any of the triangles inside it, therefore AABBs can be used as rejection test. In a BVH this simple idea is applied recursively: the scene is divided into two AABBs, then each one is further divided into two, and so on, until a stopping

criterion is met.
With a balanced BVH, finding the ray-scene intersection becomes logarithmic with reference to the number of triangles, and is therefore a big improvement over the brute-force approach.
Building the BVH optimally is an NP complete problem, therefore, in real-world applications, a greedy algorithm and heuristics are used to make the problem tractable. In particular, the heuristics are used to decide along which axis to cut the AABB, and to decide how to split the triangles into the two children nodes.
The first heuristic used in state-of-the-art applications is the Longsest Spliting Plane Heuristic (LSPH), where the AABB is cut with a plane perpendicular to its longest dimension. It is worth noting that, in all the state-of-the-art builders, only the three cartesian planes orientations are took into considerations. In other words, it is not possible to cut the AABB with an oblique plane, as it wouldn't make sense when the AABBs are, indeed, axis aligned.
The second heuristic is Surface Area Heuristic (SAH). With SAH, when the builder has to split the triangles by using the plane orientation selected by LSPH, it tries many different options[4], and assigns to each one a cost, which can be compute with this formula:

$$Cost(Aabb) = \frac{SurfaceArea_{Aabb}}{SurfaceArea_{root}} \cdot \#triangles_{Aabb} \cdot K$$

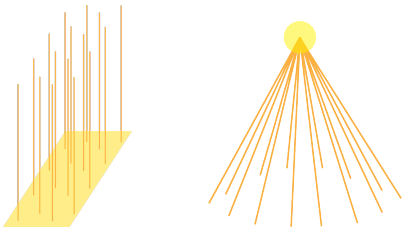
In summary, the formula tries to approximate the probability that the AABB is hit by a random ray, and weights it with the number of triangles in the node and a constant. The split with the lowest combined cost (i.e. hit probability) among the two AABBs gets greedly chosen.
The fact that the hit probability can be computed as $\frac{SurfaceArea_{Aabb}}{SurfaceArea_{root}}$ has as hypothesis that the rays are uniformly randomly distributed in the scene. However, as we noted while talking about importance sampling, we know this is not the case.

## 2. Projected Area Heuristic

The first novelty in our work is the Projected Area Heuristic (PAH). The core principle behind PAH is to try to estimate better the hit probability of an AABB, by leveraging the knowledge about the ray distribution in the scene, caused by light importance sampling.
In particular, we take into account two possible ray distributions:

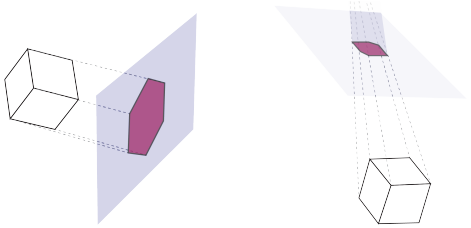
**Parallel ray distributions** arise in proximity of planar area light sources;

**Radial ray distributions** arise in proximity of point light sources, such as spotlights.



(a) Parallel (b) Radial

If one of these ray distributions is present in a given region of the scene, then we can better estimate the probability a ray hits an AABB present in this region, by projecting the AABB: in case of parallel ray, the projection is orthographic, whereas, for radial rays, the projection is perspective.



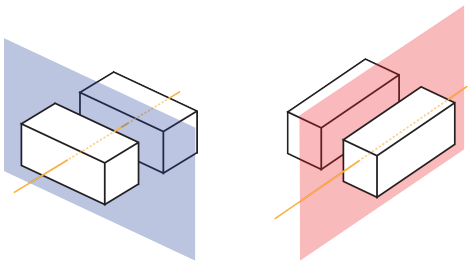
(c) Orthographic (d) Perspective

## 3. Splitting Plane Facing Heuristic

The second novel heuristic we propose, is a possible alternative to LSPH, and is called Splitting Plane Facing Heuristic (SPFH). It is based on the same considerations we made for PAH, but it is applicable to how to decide the orientation of the splitting plane, and takes into consideration if the AABB is in a region with a parallel or radial ray distribution.
As described by [1], a big 3D overlap between two children nodes, leads to a low quality BVH. With SPFH, we try to minimize the overlap between children nodes in the 2D space of their projections. The idea is based on the observation that, if the splitting plane is perpendicu-

[4] Binned approach.

lar to a ray, there is a large probability that the ray will intersect both the resulting children AABBs. Conversely, if the plane is parallel to a ray, it is not probable that it will hit both, because the overlap of their projections will be a lot smaller in many scenarios.



(e) Perpendicular (f) Parallel

In our implementation, better described in the relative section, for each one of the possible three splitting planes orientations, we assign a value which tells the builder how much parallel a plane is to the rays of the ray distribution the AABB is placed into. The builder will then try the splitting plane with the highest value, and use PAH to find the actual best cut.

## 4. Top Level Structure

In the previous sections we summarized the two novel heuristics we propose. In this section, we will introduce two structures that can be used to enable the adoption of our heuristics in a concrete case.

## References

[1] Timo Aila, Tero Karras, and Samuli Laine. On quality metrics of bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 101–107, 2013.