Analysis of the Path Tracing rendering method on CPU and GPU

by Jordi Gil González

April 2020

Content

- What is Path Tracing?
- Direct illumination vs. Global illumination
- Rendering Equation
- Anti-Aliasing
- Triangles
    - Möller-Trumbore
- Bounding Volume Hierarchy
    - Ray-slab intersection
    - LBVH - Tero Karras
- Geometry transforms
- Materials
- Image Filters
- Results

What is Path Tracing?

Path Tracing is rendering method that generates realistic images of three-dimensional scenes such that the global illumination is faithful to reality. The algorithm determines the colour of each pixel by tracing the path of a ray through the scene. These rays are launched randomly from camera (primary rays) and when hits a surface, this is scattered in a random direction generating new rays (secondary rays). In order to achieve this realism we need to approximate the Rendering Equation.

Direct illumination vs. Global illumination

As we said, we have realistic images if the global illumination is faithful to reality. The difference between direct illumination and indirect (global) illumination is that in global illumination the colour of a certain point is given by the primary rays plus the secondary rays.

Rendering Equation

The Rendering Equation is an integral equation the defines the radiance at each particular position and direction by the sum of the emitted light and the reflected light. Reflected light itself is the sum from all directions of the incoming light multiplied by the Bidirectional Reflectance Distribution Function. The Bidirectional Reflectance Distribution Function defines how the light interacts with the material that hits and defines the type of material that an object gets.

Anti-aliasing

As we said, the rays are launched randomly and one sample per pixel cannot represent how the light interacts in scene like reality. This randomness causes a lot of noise in the output images. In order to avoid this noise, more than one sample is needed to calculate the final pixel colour. For this, we choose randomly a fixed number of samples to average the final colour.

Triangles

We choose the triangle shape to represent the 3D models. Most of 3D models are defined by triangles instead of other shapes such as spheres, cones, points, etc. Once we have defined the shape, we need an algorithm to query if a ray hits this shape. For triangles, we choose the Moller-Trumbore algorithm because is the faster and the storage is minimum. (We only need the three vertexes that defines the triangle)

Möller-Trumbore

The use of barycentric coordinates is a very practical approach because with them we can interpolate vertex data such as texture coordinates, vertex colour, normal, etc. We can represent this barycentric coordinates as follows.

Möller-Trumbore

If we equate this barycentric formulation with formulation of the ray and rewrite it into matrix formulation, we get the following equation. This means the barycentric coordinates and the intersection distance t can be found by solving by a linear system the equation. In the first subfigure we have the original ray-triangle intersection and in the second one we have the ray-triangle intersection after the barycentric formulation. The last step can be achieved by taking the inverse to matrix M using the Cramer's rule.

Bounding Volume Hierarchy I

Another important component for a Path Tracing application are the acceleration structures. In our case we choose the Bounding Volume Hierarchy. A Bounding Volume Hierarchy is a tree structure that in each node we have the bounding volume of its children. Also, in the leaf node we have the final 3D object or the shape. In our case we construct the BVH at shape level.

Bounding Volume Hierarchy II

There are some types of bounding volume and each one have some characteristics. For example, the convex hull bounding volume is more accurate to the object than the sphere. But, the test ray/bounding volume is slower. For our purpose we choose the AABB.

Ray-slab intersection I

In the ray/AABB intersection test, we choose the Slabs algorithm. This method computes all t-values for the ray and all planes belonging to the faces of the AABB. This approach is implemented by describing the AABB by using slabs of space segments enclosed by planes. We store the minimum and maximum intersections between the ray and these planes. The ray and AABB will intersect when the ray enters 3 slabs before exiting one of them.

Ray-slab intersection II

LBVH - Tero Karras

Even though our project is not centred on the BVH construction, the construction of a high quality BVH its important. Is for this reason, that we choose the BVH presented by Tero Karras. For each object (in our case for each triangle) we assign a Morton Code. This Morton code provides an ordering along the space-filling curve while preserving data locality. We can construct this Morton code interleaving the binary representation of the three coordinates. Once all objects have their Morton code, we can order the set and proceed to construct the tree.

LBVH - Tero Karras

How each Morton code is a bit array we can see the tree as a radix tree where each node represents the longest common prefix and represents a range of his children. In order to construct the tree, we need to find the range that cover each internal node and the split position. We can obtain this values by performing a binary search.

Geometry Transforms

Another important factor is to be able of set the objects in the scene in the position that we want, to do this we need to specify a series of geometric transforms. These are translation, scaling and rotation.

Translation

Translation is move one object from one point to another. This can be done by adding the second point to the first one. Instead of this, we choose to represent as a matrix form because maybe we need to concatenate more than one transformation, and we can represent this set of transformations as single matrix.

Scaling

Scaling is very similar to translation, but instead of sum we multiply each coordinate by the scaling factor. This scaling factor can be uniform (all coordinates are equals) or non-uniform. With this matrix we can represent the scaling.

Rotation

We can define the rotation matrix as the multiplication of the rotation matrix of each axis with his Euler rotation angle.

Materials I - Diffuse & Metals

Materials are one of the most important components of Path Tracing, because defines how the light interacts with the surface of an object. In our project we have diffuse (opaque), metals, dielectrics (like glass or plastic), and also we have textures.

Diffuse materials are those which the light is scattered in all directions, so when a ray hits a diffuse surface, another ray is created in a random direction.

For Metals, we choose a perfect reflection model. In this case, when a ray hits a metal, the ray is reflected perfectly, but perturbed in the destination point. This perturbation allows playing with the roughness of the material. A metal object with roughness 0 is a perfect mirror.

Materials II - Dielectrics & Textures

Dielectric materials like glass can reflect the light and refract the light. Each material has her own refractive index that describes how light propagates through them. This can be described by the Snell's law. When a light ray hits the interface between two mediums (for example air to glass) both reflection and refraction may occur. The Fresnel equations describe the ratios of this reflection and refraction. When the incident angle is close to critical angle of the medium, the total internal reflection occurs and the ray is reflected instead of refracted. We use Schlicks Approximation to approximate the Fresnel Equations because is less expensive.

Textures are used to add more realism and details to objects. Using the barycentric coordinates calculated by the Moller-Trumbore algorithm we can map a 2D image in a 3D object.

Materials II - Skybox

Finally, we have the skyboxes. Skyboxes are used to add more details to the scene. Instead of using a plane colour for background we can use a texture. This texture is mapped in a cuboid that wraps the scene.

Image Filters

The last component are the image filters. The image filters implemented have the purpose of denoise the output image. Instead of use a great number of samples per pixel we can take advance of some filter to denoise the image.
The mean filter is one of the easiest filters. In this filter, we define a window (kernel) of size S and for each pixel we calculate their colour by the average of all pixels in the kernel.
The median filter is quite similar, but instead of set the average, we order all the values of the pixels inside the window and chose the median value.
The last filter implemented is Bilateral Filter. The bilateral filter replaces the intensity of each pixel with a weighted average of intensity values from nearby pixel. This weight can be based on a Gaussian distribution. The spatial kernel depends on Euclidean distance of pixel and the range kernel depends on the radiometric differences (colour intensity).

Results

The first scene is very simple. The amount of polygons is relatively small. Here we can see how the metallic object reflects the white cube and the red and green walls. Also, we can see how the red wall affects the final colour of white cube on the left side. In this case all the rays, except those that are absorbed by the source light, are bouncing until reach the cut-off limit because we only have one source light to be absorbed.
The second scene is quite similar to the first one, but using other material, in this case a texture, for the wall and the floor in this case is mirror-like. Also, we have a two low poly 3D models instead of two cubes. In this case, the amount of polygons is greater than the first scene.
Finally, the last scene is very different. In this case we have a high resolution 3D model, so the amount of polygons is huge (one hundred and forty-four triangles) and we have a skybox instead of a small room. In this case the majority of rays will be absorbed by the skybox before reaching the cut-off limit. This will directly affect the runtime.

Scene one

If we look the blue cells, we can see how the recursive traversal of BVH is very expensive and is a bad approach for the GPUs. For one GPU from BOADA, we have 21s in the recursive traversal versus four and half seconds for the same GPU with iterative traversal. In red cells we can see how the BVH introduces an overhead for scenes with a small amount of polygons.

Scene two

However, when the number of polygons increases, we can see the advantage of the BVH. If we look the coloured cells, for the execution with 10 SPP the BVH version is 10 times faster than the version without it.

Scene three

In the last scene we can see the impact that the skybox have into the scene. In this case, even having one hundred times more polygons than the last scene, the execution time is more than 10 times faster. The reason of that is the skybox. Here, as I said before, the rays are absorbed sooner, however, in the last scene the probability of reach the cut-off limit by a ray is greater.

Results

Finally, we can see the filtered images choose. The bilateral filter is the filter that acts better, is able to reduce the noise without blurring the image so much.

Thank you for your attention!