Path tracing with Reinforcement Learning

Palash Bansal, IIIT Delhi

Introduction

Path tracing is a computer graphics Monte Carlo method of rendering images of three-dimensional scenes such that the global illumination is faithful to reality. Fundamentally, the algorithm is integrating over all the illuminance arriving to a single point on the surface of an object. This illuminance is then reduced by a surface reflectance function (BRDF) to determine how much of it will go towards the viewpoint camera. This integration procedure is repeated for every pixel in the output image. Since this process is executed independently for every pixel, it is embarrassingly parallel in nature, and can be implemented in CUDA to get great improvement in performance.
Furthermore, on combining path tracing with reinforcement learning, we will be able to memoize more data about the path of lights and will be able to simulate the light paths much faster than traditional monte-carlo path tracing. This technique is discussed in [1].

Path tracing with RL

Ken Dahm and Alexander Keller in the paper Learning Light Transport the Reinforced way pointed out the structural similarities between the Light transport and the Q learning equation, and hence formalized a way to learn the important light paths using hit and trial and reinforcement learning by modelling a MDP and using Monte-Carlo path tracing to feed the process.
This is basically a replacement/improvement over importance sampling and related methods, where we feed the light position into the algorithms and sometimes lose the details and quality of the scene. Since Path Tracing is already a parallel task, this could be accomplished on GPU using CUDA without much overhead.
has shown that the equations of reinforcement learning and light transport simulation are related integral equations. Based on this correspondence, a scheme to learn importance while sampling path space has been derived. This approach is helpful in a consistent light transport simulation algorithm that uses reinforcement learning to progressively learn where light comes from. As using this information for importance sampling includes information about visibility, too, the number of light transport paths with zero contribution is dramatically reduced, resulting in much less noisy images within a fixed time budget.

```
ALGORITHM 1: Augmenting path tracing with RL
Input: The scene, and camera.
ray = rayForCurrentPixel(); bounces = 0;
repeat
    r2 = intersect(ray, scene);
    updateQTable(r2, color);
    if isLightOrEnvironment(r2) then
        break and return color;
    else
        direction = calculateDirectionProportionalToQ();
        color = calculateColor(r2, direction);
        ray = generateRay(r2, direction);
    end
until bounces > maxBounces;
```

Figure 1: The basic algorithm.

Implementation details

In the main function, the code uses a renderengine to render one frame of the image, this data after being copied to the CPU, gets send to an OpenGL context as a texture, which is then rendered on the screen/window.

GPU implementation details

- Each pixel is subsampled 64 times. For each ray, a GPU thread is assigned. So, each pixel is spread across 2 warps.
- At every frame, the current bitmap is calculated by using the previous bitmap as the seed, which is then copied back to the host to display.
- Each block is divided into 3D threads, where xDim=8, yDim=8 and zDim is variable depening on the number of threads per block. The X and Y space is used directly to subsample a pixel, and the threads in the Z dimension are different pixels.
- In the kernel, first the ray direction is calculated. Stochastic jittering is used to vary the starting position of the ray while sub sampling. After calculating the ray direction, the light path is simulated upto 64 bounces.
- At each intersection with an object the color is multiplied.
- The final colors in all threads are reduced to one value(using shuffle) per-pixel which is then written to the image-bitmap in the global memory bitmap.

RL implementation details

- **State space** This is defined differently from the paper. The state space is the voxel representation of the entire scene. Where each sphere/object may lie in one or more voxels.
- **Action space** This is also defined differently from the paper. The actions from each state is a set of 8 directions represented by the 8 octants in the 3d space.
- **Updation** The QTable is updated everytime a ray hits a new object. The value is very large in case of a light and slightly less than the illumination in case of any other.

BRDFs Supported The implementation supports **Diffuse, Specular, Emmisive, Glossy and Dielectric** for monte-carlo path tracing in both GPU and CPU implementation.

Speedup

For a basic scene designed to test the performance and the working of the algorithm, the speedup is great.
For all the experiments, the MAX_LEVELS = 8, i.e the rays are simulated upto 8 bounces.
Configuration for the experiment: Image size: 360x240, Samples per pixel: 64, Area lights: 1, Objects: 6

- 1 frame on GPU(GTX 970): 51ms
- 1 frame on CPU(i7 4.4GHz): 144308ms
- **Speedup: 2880x**

Configuration for the 2nd experiment: Image size: 640x480, Samples per pixel: 64, Area lights: 1, Objects: 9

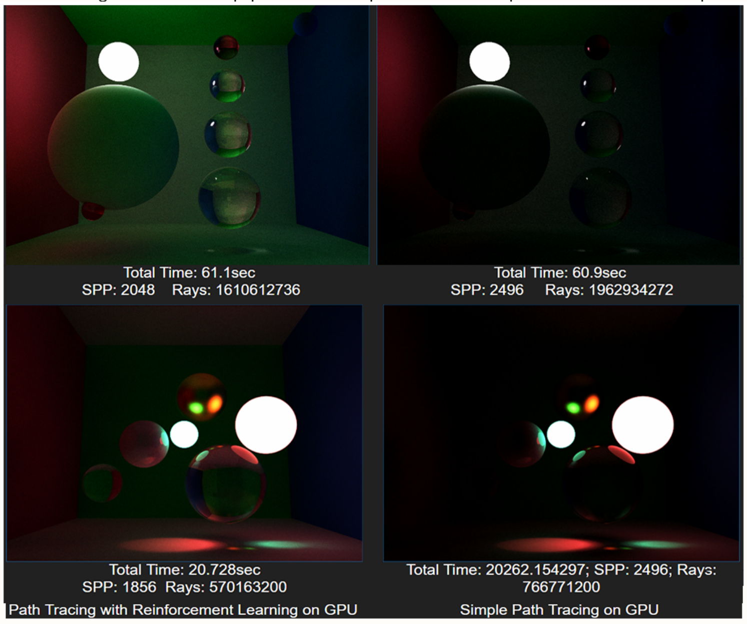
- 1 frame on GPU(GTX 970): 251ms
- 1 frame on CPU(i7 4.4GHz): 819200ms
- **Speedup: 3264x**

Configuration for the 3nd experiment: Image size: 640x480, Samples per pixel: 64, Area lights: 2, Objects: 13

- 1 frame on GPU(GTX 970): 428ms
- 1 frame on CPU(i7 4.4GHz): 1041920ms
- **Speedup: 2435x**

References

[1] K. Dahm and A. Keller. Learning Light Transport the Reinforced Way.ArXiv e-prints, January 2017.
Complete Code: Github


Total Time: 61.1sec
SPP: 2048 Rays: 1610612736
Total Time: 60.9sec
SPP: 2496 Rays: 1962934272
Total Time: 20.728sec
SPP: 1856 Rays: 570163200
Total Time: 20262.154297; SPP: 2496; Rays: 766771200
Path Tracing with Reinforcement Learning on GPU
Simple Path Tracing on GPU