Learning Light Transport the Reinforced Way Implementation Analysis

Callum Pearce

cp15571@my.bristol.ac.uk

***Abstract*—The purpose of this document is to discuss benefits the on-line reinforcement learning algorithm for learning light transport in a given scene, as well as provide some analysis on my current results. It is important to understand exactly what this paper has achieved and how it contributes to the field of computer graphics, specifically towards real-time ray tracing. This way, I will be able to present where my future modifications come and how they differ to the current algorithm. I will also provide a short description on my justification for using Deep Reinforcement learning to approximate the irradiance distribution for any point in the scene.**

I. Benefits of the On-line Reinforcement Learning Algorithm

Firstly, importance sampling is an area of interest for modern path tracing engines as it can dramatically reduce the number of zero contribution light path sampled in the algorithm. By using reinforcement learning, light paths which connect with a light are far more likely to be sampled then those which do not. Previous on-line importance sampling techniques struggle to deal with blockers in the light direction, instead the reinforcement learning approach learns where light comes from rather than just the location of light sources. There are two benefits from this scheme:

- Reduced path length for tracing a ray from the pixel to the light source (for unbiased path tracing). The reduction in the number of zero contribution light paths reduces the noise. The variance in the length in the path length is also reduced, reducing the noise present in rendering for the same number of samples.
- We learn importance from a smoothed approximation instead from higher variance path space samples

Having an on-line algorithm is also an interesting topic, this means the render gets progressively better up until convergence on the reinforcement learning within the scene (ensured via decaying learning rate α. We therefore avoid any pre-compute which would otherwise be used for importance sampling from the Irradiance Volume alone for example. This scheme in particular stands out to me fore games, a quick pass round new scenery in a ray-traced game would allow the render to progressively render higher quality frames at a faster rate as the path length reduces.

By Nvidia's experiments, the reinforcement learning approach was found to be 20% slower than standard path tracing per iteration for shooting a ray per pixel and tracing its path by one step, it converges far quicker which offsets this initial cost.

II. Drawbacks of the On-line Reinforcement Learning Algorithm

The main drawback that comes with the Reinforcement Learning scheme is the memory requirement for storing the sampled Q value approximations across the scene. While this variables is controllable, I found it to have a large impact the noise of my scene. I could not get the entire set of Q values and their related data to be stored in under 2MB for a good approximation of a small room. I generally need around 500MB of data to store my Q values and their associated data. Ken and Dahm mention this specifically in their paper as a potential issue. Implementation-wise I found this to be a colossal issue, as it meant I had to store my irradiance volumes in global memory on my GPU, adding a 100-200x performance penalty for memory access for nearest neighbour search compared to accessing on chip memory. This was the main reason my Reinforcement learning implementation is far slower than a default path tracing approach. Unless there is some way of getting round this nearest neighbour lookup on by ensuring search is done on memory within the chip, for larger scenes this algorithm may just not be an option. Memory access is so important it causes my reinforcement learning approach to run in 2000% of the time as the default path-tracer compared to 20% Nvidia seem to be observing. However as I previously mentioned, I don't believe they can sustain these times for a very large scene, even if a KD-Tree is used (which I did implement).

Also, having the parameter of memory usage to tweak is not ideal, as you must find the correct one by rendering many scenes. This may in practice take away the usage of this reinforcement approach. Guesses can be made by the number of polygons, but for each scene it ultimately changes.

III. Deep Reinforcement Learning

Deep reinforcement learning hopes to solve the major shortcoming of the Reinforcement Learning approach, which is a fixed memory usage. We instead represent the Q by a neural network, where we give it a state and it will give us the valuation for every angle to shoot a ray in. This means we have a fixed sized neural network which should be able to cope for a variety of complex scenes. The main worries I have with this approach are as follows:

- Will the neural network be able to accurately learn over time the best action to take given the supplied state s?

I think yes, if Deep Mind can train a single network to do this well across many Atari games which have an enormous amount of states, then surely the neural network can do it for single scene. In fact, a major selling point would be if a single network could learn this for multiple different scenes.

- How quickly will a rendering iteration be able to run? We will have to do a forward pass for every ray and a training step for every ray (may not be the case, could bundle all rays training together into a single batch?). I believe forward passes through a neural network are very quick for these kind of things but I am not totally sure on how the speed amount when there are hundreds of passes at once. Sure these forward passes can be parallelized on identical copies of the neural net, then we can potentially do all of our training between frames? Clearly there are a lot questions to be answered and this is no the document where I try to design my approach.