Volume xx (200y), Number z, pp. 1–5

Final Project: Real-Time Global Illumination with Radiance Regression Functions

Fu-Jun Luan

Abstract

This is a report for machine learning final project, which combines realistic rendering and machine learning. As is known, global illumination is very challenging for its expensive computing requirement. In general, Monte Carlo ray tracing or Photon Mapping techniques are used in rendering global illumination, which requires several hours or even days to generate a smooth and physically correct image. So it seems impossible to render global illumination in real time. Microsoft Research Asia solved this problem with a novel method combining rendering and neural networks. In my final project, I follow the MSRA 2013 SIGGRAPH paper "Global Illumination with Radiance Regressions" and make some extra extensions. The result turns out to be pretty good.

1. Introduction

Global illumination is a superset of radiosity and ray tracing. The goal is to compute all possible light interactions in a given scene, and thus obtain a truely photorealistic image. All combinations of diffuse and specular reflections and transmissions must be accounted for. Effects such as color bleeding and caustics must be included in a global illumination simulation.

In industry application, global illumination is difficult for its expensive computing requirement and hence is seldom used in movies and 3D games, instead traditional ray tracing is employed. Traditional ray tracing is not physically correct and thus doesn't need much computation, while it can't offer enough visual scene. The images in Figure1 is a comparison of industry ray tracing and global illumination.

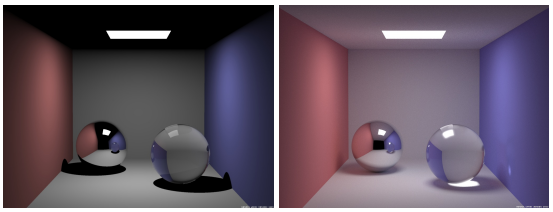

Figure 1: *Illustration of traditional ray tracing and global illumination: left is ray tracing and right is global illumination, notice the caustics, color bleeding and soft shadows in the right image.*

2. Related Works

In this section, we will illustrate what is global illumination, and then review some state of the art techniques in realistic rendering.

2.1. Global Illumination

Global illumination can be separated into direct illumination and indirect illumination.

Direct Illumination. This is the illumination accounts for only direct lighting, which means light that has traveled directly from a light source to the point being shaded, and ignores indirect illumination from objects that are not themselves emissive. The visual effect of direct illumination is shown in Figure2.

Indirect Illumination. Indirect light is all the inter-reflected light in a scene. Global illumination is an approximation of real-world indirect light transmission.

With indirect illumination, the contribution of bounced light from other surfaces in the scene is used to calculate the overall light contribution and the color values at points on objects that are not directly illuminated (that is, at points that do not receive light directly from a light source, such as a point light).

Global illumination occurs when light is reflected off of or transmitted through an opaque (reflection only), transparent or semi-transparent surface from a surface to bounce off or be absorbed by another surface. Visual effect of global illumination is shown in Figure2.

Figure 2: *Illustration of indirect illumination. The left image is direct illumination, which produces hard shadow and artificial visual effects. The right is direct illumination added by indirect illumination. Note the color bleeding (light transported from objects instead of light) in the shadow region.*

2.2. Techniques for Rendering Global Illumination

The state of the art techniques for rendering global illumination are Monte Carlo technique and Density Estimation. The representations of them are Path Tracing and Photon Mapping, respectively.

Path Tracing. Path Tracing is a computer graphics Monte Carlo method of rendering images of 3D scenes such that the global illumination is faithful to reality.

Path Tracing naturally simulates many effects that have to be specifically added to other methods (traditional ray tracing or scanline rendering), such as soft shadows, depth of field, motion blur, caustics, ambient occlusion, and indirect lighting.

Due to its accuracy and unbiased nature, path tracing is used to generate reference images when testing the quality of other rendering algorithm. In order to get high quality images from path tracing, a large number of rays must be traced to avoid visible noisy artifacts.

Although path tracing is robust to complex scenes and can handle variant difficult lighting effects, it suffers from the slow convergence rate, which is displayed in the image as, the noise. In general, a smooth and correct image generated by path tracing will cost very long time.

Figure3 is a realistic image rendered using Path Tracing technique.

Figure 3: *Scene rendered by Path Tracing, timeuse: 3 days.*

Photon Mapping. In computer graphics, Photon Mapping is a two-pass global illumination algorithm developed by Henrik Wann Jensen that approximately solves the rendering equation. Rays from the light source and rays from the camera are traced independently until some termination criterion is met, then they are connected in a second step to produce a radiance value. It is used to realistically simulate the interaction of light with different objects.

Unlike path tracing, photon mapping is a "biased" rendering algorithm, which means the averaging many renders using this method does not converge to a correct solution to the rendering equation. however, since it is consistent method, a correct solution can be achieved by increasing the number of photons.

Figure4 is a scene rendered using Photon Mapping.

Figure 4: *Scene rendered by Photon Mapping, timeuse: 4 hours.*

3. Backgrounds

3.1. The Rendering Equation

Kajiya [1986] first proposed the rendering equation, which faithfully describes the physical light transport and hence can be applied to graphics simulation. The rendering equation has the following formula:

$$L_o(x,\omega_o,\lambda,t) = L_e(x,\omega_o,\lambda,t) + \int_\Omega f_r(x,\omega_i,\omega_o,\lambda,t) L_i(x,\omega_i,\lambda,t)(\omega_i \cdot n) d\omega_i \tag{1}$$

where λ is a particular wavelength of light, t is time, x is the location in space, ω_o is the direction of the outgoing light, ω_i is the negative direction of incoming light, L_o is the total spectral radiance of wavelength λ directed along outward along direction ω_o at time t, from a particular location x, L_e is emitted spectral radiance, f_r is the bidirectional reflectance distribution function (BRDF), and $\omega_i \cdot n$ is the weakening factor.

3.2. Regression Methods

Regression methods have been widely used in graphics. For example, Grzeszczuk et al. [2003] used neural networks to emulate object dynamics and generate physically realistic animation without simulation. Neural networks also have been used for visibility computation. Dachsbacher et al. [2011] applied neural networks for classifying different visibility configurations. These visibility neural networks allow them to predict low-frequency self-shadowing when computing the direct shading of a dynamic scene. Meyer et al. [2007] used statistical methods to select a set of key points and form a linear subspace such that at run time, global illumination only need to be computed at key points.

3.3. Feed-Forward Neural Networks

A feed-forward neural network is an artificial neural network where connections between the units do not form a directed cycle. This is different from recurrent neural networks.

The feed-forward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the inputs nodes, through the hidden nodes (if any) and to the outputs nodes. There are no cycles or loops in the network.

Figure5 is an illustration of feed-forward neural network.

4. Main Experiment

4.1. Sampling: Sample Data using Off-line Renderer

First of all, a set of training data must be obtained. Given a scene with a predefined range of light source positions and viewpoints, the training set is generated by a Latin hypercube sampling scheme [2000] and compute BOTH the direct illumination and indirect illumination *separately* (Note

the Microsoft Research Asia 2013 SIGGRAPH paper only compute indirect illumination, the direct illumination part is calculated real-time, hence it is limited by a point light and can't support area light sources.)

I sample N_v viewpoint positions uniformly subdivide the scene, and then randomly shoot N_r rays with uniform sphere sampling method. N_v and N_r are 200 and 6000 separately in my implementation. Instead of sampling directly on surfaces, intersections on the surfaces of viewing rays are sampled in order to avoid visibility tests and ensure that the visible surfaces are well sampled.

In implementation, 1,500,000 data samples are generated using the above method. All sampled examples are computed using my off-line bidirectional path tracer (BPT), which is unbiased and physically correct. Each data sample is computed using 20,000 paths samples generated by BPT.

4.2. Training: Train Neural Networks

According to the SIGGRAPH paper, the input vector of the neural network is constructed by hit-point position x_p, view direction v, light position l, hit-point normal n, and BRDF parameters a.

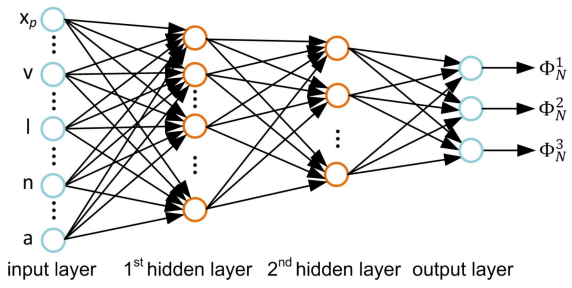

Figure 5: *Illustration of the neural network structure: two hidden layers with 20 nodes in the first hidden layer and 10 hidden nodes in the second layer.*

The structure of the neural network is shown in Figure5, which has 13 inputs nodes in the input layer, 20 hidden nodes in the first hidden layer, 10 hidden nodes in the second hidden layer, and 3 output nodes (RGB) in the output layer.

The neural network is trained using Matlab NN Tool Box, with *trainlm* train function, which is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization.

Unlike trained by kd-tree separated partitioning small neural networks, I use different neural networks for different objects in the scene, and each object has two neural networks for predicting direct illumination and indirect illumination.

4.3. Rendering: Predict RGBs using Neural Networks

After training the neural network, the direct illumination and indirect illumination of each scene point can be evaluated by the neural network, given the input vector which can be easily obtained in the rendering pipeline. Hence, images are generated by prediction using neural networks.

4.4. Make it Real-Time: GPU Shader

Although we have obtained the images rendered using neural networks, this is far from enough. Our goal is a real-time global illumination renderer, in order to obtain the high rendering speed, GPU version code has to be implemented.

My PC is a consumer level PC with an Intel Core i7-3770 3.4GHz CPU and 8GB memory, and AMD Radeon HD 7400 Series GPU (which supports OpenGL 2.0 and DirectX). The GPU shader is employed combined with OpenGL, which means the neural networks should be computed and predicted on GPU, thus achieve real time.

4.5. Extension of the Paper Work: Area Light Source Supported

Note that we trained BOTH direct illumination neural network and indirect illumination neural network, *separately*. With the help of the neural network for direct illumination, we can achieve real-time rendering scenes with area light sources (because we don't need to render direct illumination, which requires expensive computation if light source is not a point, since we need to sample the area light).

5. Implementation

5.1. Off-line Renderer

The off-line renderer is wrote by me this year, and is still going on. It currently supports both surfaces (diffuse, specular and glossy) and volumes (subsurface and participating media, both homogeneous and heterogeneous), with different integrators (Path Tracing, Photon Mapping, Bidirectional Path Tracing, Vertex Connection and Merging).

The renderer is used as a data samples generator in this machine learning project.

5.2. Training

Training turns out to be easy with the help of Matlab NN Tool Box. In my implementation, networks are trained using *trainlm* function.

Overfit is an important issue in training. I also test my neural networks with variant training data size and AdaBoost. While little further improvement could be found in my test. As shown in the MSRA SIGGRAPH 2013 paper, this is another sign illustrating that overfitting is under control.

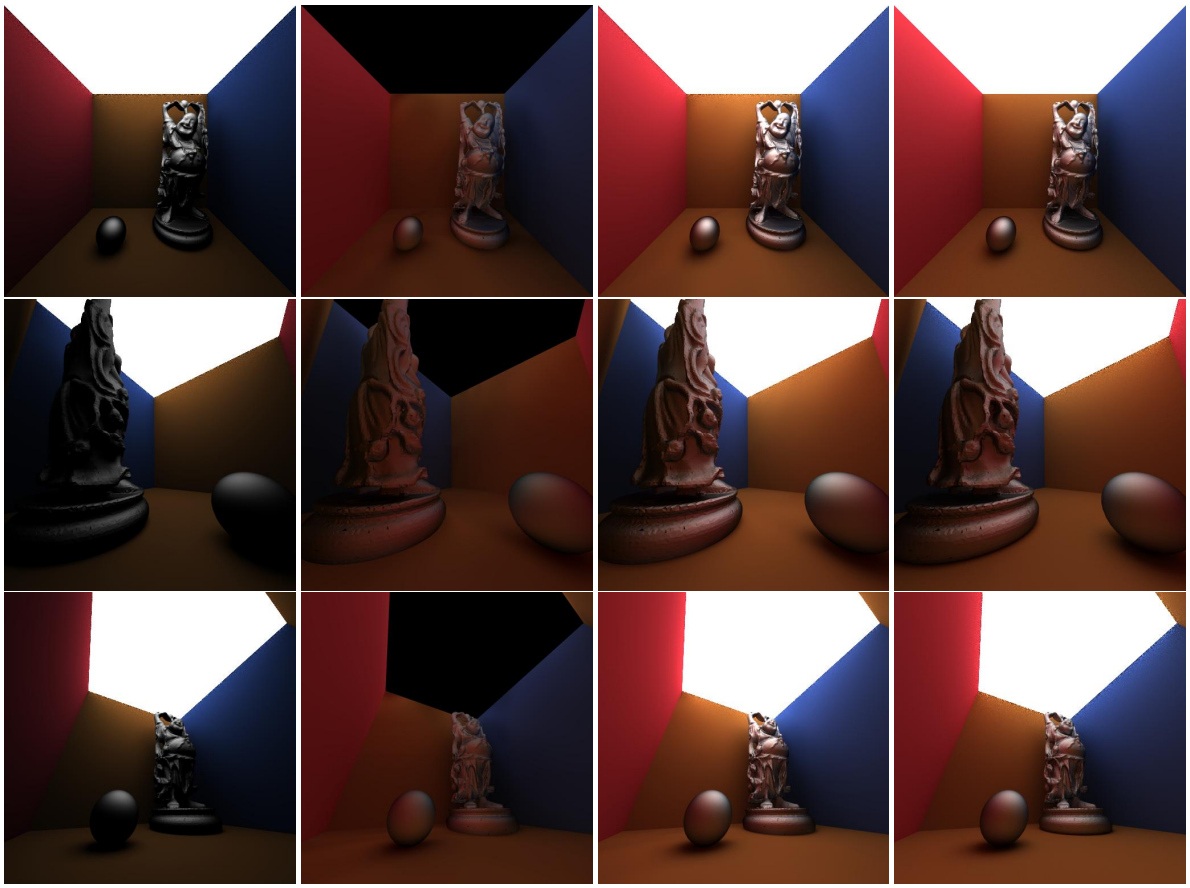

Figure 6: *Comparison. From left to right (columns): direct illumination, indirect illumination and global illumination predicted by neural networks, and the reference rendered by off-line bidirectional path tracer.*

5.3. Rendering

In rendering phase, we first push meshs to OpenGL, then load neural network weights and load them onto GPU. With a simple GUI, the renderer supports interaction such as key board and mouse motion. The RGBs of pixels are computed on GPU in real time.

6. Comparison and Results

I make some demos of the real-time roaming in my test scene, and this section we will make some comparisons of results produced by neural networks and off-line renderer.

As shown in Figure6, three different view positions screen shot are compared, the first three columns are predicted using neural networks (direct, indirect and global illumination), and the last column images are references rendered using my off-line BPT.

Figure7 shows some visual difference between global illumination predicted by neural networks and off-line rendered, we demonstrate that the different is almost negligible.

The real-time roaming demos are in the folder "/Realtime-GI/demo/".

7. Conclusion

In this final project, I implement the SIGGRAPH 2013 paper "Global Illumination with Radiance Regression Functions" and make a small extension (area light support). Through the project, I get familiar with neural networks (especially feed-forward one), and obtain some skills about shaders on GPU.

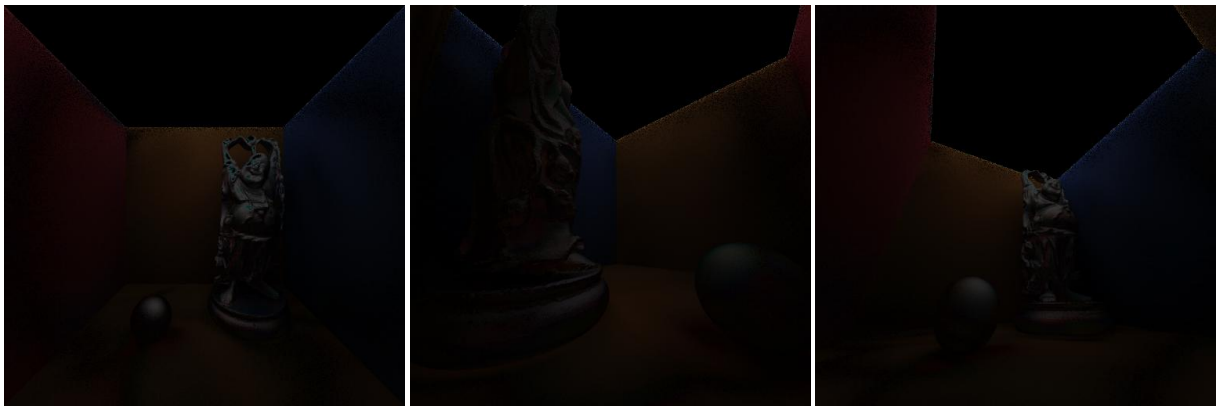

Figure 7: *Difference of global illumination images by neural networks and off-line renderer.*