(4.1) IMPLEMENTATION GOAL

The goal of this implementation is to propose a real-time rendering technique that will render global illumination on the GPU through the use of OpenGL and GLSL. We will use the idea of virtual point lights from *Instant Radiosity* [Keller 1997] to render the indirect illumination. Shadow maps will be used as in the spirit of [Williams 1978] and [Reeves et al. 1987] to render shadows for direct illumination. The VPL's will be structured outward from a directional primary light source in a lattice. These VPL's will be structured in specific distances and angles around the primary light source in hemispheres. The goal is to simulate a multiple waves of VPL's flowing outward from the primary light source in order to simulate the light transport as wave-like in order to simulate indirect illumination using only the direct illumination of many VPL's while ignoring the bouncing of light among surfaces. This way, we will be simulating light as a particle with using VPL's and as a wave in the way the VPL's will be organized. This will allow the "wrapping" of light around objects in order to illuminate surfaces that may not be directly visible from the light source. An additional goal of this implementation is to have the resulting technique be scalable. The organization of the VPL's will allow us to use more or less VPL's depending on quality and performance desires. Specific implementation details will follow in section 4.2. Lastly, in support of this technique and its ignorance of indirect illumination for a direct illumination approximation of direct illumination, we discuss the importance of scientifically correct rendering versus visually appealing rendering.

SCIENTIFICALLY CORRECT VERSUS VISUALLY APPEALING

All of the previous work mention in section 3 used approximations when computing indirect illumination to varying degrees. This is because as mentioned in section 3.2, the rendering equation (equation 3.1) is calculated using a Neumann series and in order to get an exact calculation, an infinite number of iterations would need to be carried out to fully calculate the result. These iterations account for the infinite number of indirect light bounces that take place. Since this is unfeasible, approximations are made based off our demands for performance or realism. For real-time applications, we must err on the side of performance, but since the presence of indirect light has been shown to be perceptually important we can't just simply ignore it either [Stokes et al. 2004]. Also, according to [Stokes et al. 2004], diffuse indirect illumination is perceptually the most important component to consider. However, as the notion of "realism" is merely whether the rendering is visually pleasing to the human eye, major approximations can be made to increase performance provided the result is visually pleasing. As such, we ignore the bouncing of light and try to approximate it with just cheap direct lighting that will be able to reach surfaces that otherwise only indirect light would reach in order to simulate the “feeling” of indirect light. Proof of the idea that visually pleasing is sufficient is provided next.

In *Perceptual Influence of Approximate Visibility in Indirect Illumination* [Yu et al. 2009] it is proven that “accurate visibility is not required and that certain approximations may be introduced.” When a person sees lighting rendered in a scene or in real-life, the most revealing or obvious lighting is the direct illumination. When it is approximated or incorrect, it is apparent right away due to the high-frequency nature of direct lighting. Indirect lighting, however is usually low-frequency and therefore has smooth graduations in changes of intensity. This allows for more leeway for approximations and interpolations. The most expensive calculation being visibility determination for indirect illumination. To research the effects of this on the viewer's perception of the rendering, [Yu et al. 2009] conducted a psychophysical study involving two different experiments. The first experiment involved estimating the difference between the approximation rendering and the reference rendering based off of a number scale ranging from one to five corresponding to “not similar” and “extremely similar” respectively. This experiment had 14 participants. The second experiment involved ranking 10 approximated renderings and 1 reference rendering in order of least realistic to most realistic. This experiment had 18 participants. Over half of the participants in both experiments were computer scientists with a background in imaging. The approximated renderings consisted of four different categories of indirect visibility. These were imperfect visibility [Ritschel et al. 2008], ambient occlusion [Zhukov et al. 1998], directional ambient occlusion [Sloan et al. 2007][Ritschel et al. 2009], and no visibility ie. visibility for indirect illumination is completely neglected [Dachsbacher and Stamminger 2005] [Dachsbacher and Stamminger 2006]. All the renderings were prerendered 5-second video sequences and consisted of four different test scenes.

The results show that the imperfect visibility approximations were very much similar or moderately similar to the reference video in all scenes. The ambient occlusion approximations were also very much similar or moderately similar to the reference video in most scenes. Directional ambient occlusion was considered very much similar or moderately similar to the reference video in most scenes. Lastly, the no visibility approximations were considered moderately similar to the reference video. There was found to be a connection between the amount of indirect illumination and the perceived similarity to the reference. In terms of the perceived realism of the renderings, the reference video as well as the imperfect visibility, ambient occlusion, and directional ambient occlusion approximations were all considered equally realistic leaving just the no visibility approximations as less realistic than the rest. Overall, the imperfect visibility approximations as in [Ritschel et al. 2008] were considered the most realistic of the approximations.

This study shows that visibility approximations can be made when rendering indirect illumination while remaining perceptually similar or as realistic as a reference rendering. This provides validity to the approximations already done as well as support further approximations to come. Having the imperfect visibility methods declared as the most realistic leads to the assumption that randomly corrupted visibility is more pleasing than incorrect visibility. Lastly, it is shown that although all the approximations had noticeable differences in the renderings, many of them were still thought of as being realistic.

(4.2) IMPLEMENTATION DETAILS

As stated in section 4.1, the goal of this implementation is to propose a real-time rendering technique that will render global illumination on the GPU through the use of OpenGL and GLSL. The primary focus is to render indirect illumination with the findings on scientifically accurate versus perceptually pleasing in mind. As such, we will ignore the indirect bouncing of light (the infinite series portion of the rendering equation), but try to simulate using the VPL technique as introduced in [Keller 1997]. We will accomplish this simulation through the method of VPL placement and handling. The VPL's will be structured in hemispheres around the primary light source. The original implementation will include VPL's at every 5 degrees around the z-axis of the light source resulting in 360degrees/5degrees per ray = 72 rays of VPL's forming a circle around the light source.

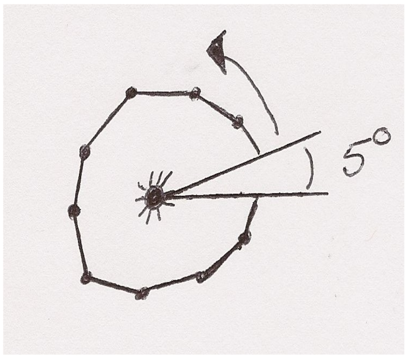

Figure 4.1: The light source is facing the reader (ie. direction/normal pointing out of the paper. Angles not drawn to scale.

Next, the VPL's will be structured at every 5 degrees around the y-axis of the light source resulting in 90degrees/5degrees per ray = 18 rays of VPL's for every one of the 72 rays around the z-axis. This totals in 1296 rays plus the vertical ray along the z-axis resulting in 1297 rays to form a hemisphere around the light.

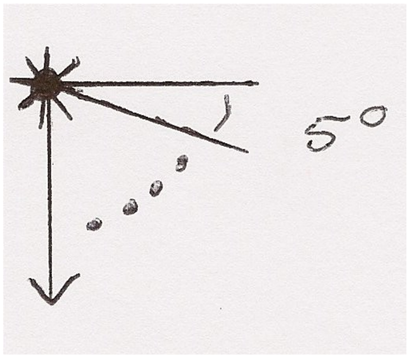

Figure 4.2: The light source is facing downwards. The arrow going down signifies the vertical ray added to the total of 1296 VPL's. Angles not drawn to scale.

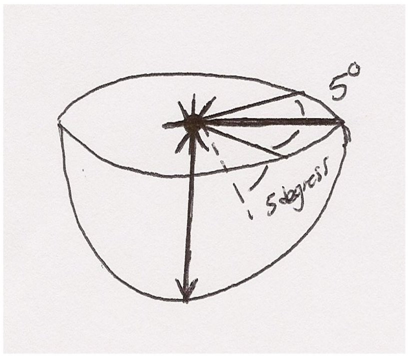

Figure 4.3: Figures 4.1 and 4.2 together form a hemisphere. The arrow signifies the primary light source's direction and is the additional vertical ray added to the total of 1296 VPL's.

Next, we will have multiple VPL's on each of these rays. This implementation will have 5 VPL's per ray. This results in having 5 stacked hemispheres of increasing radii and a total of 6485 VPL's.

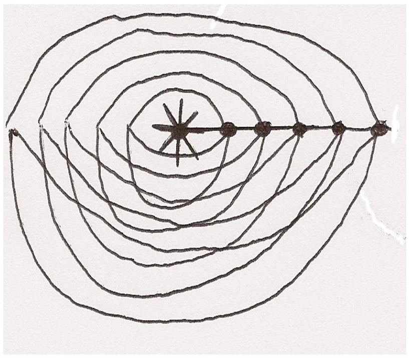

Figure 4.4: 5 stacked hemispheres formed by the VPL's around the primary light source. Distances between hemispheres are not drawn to scale.

The distance from the primary light to each of the VPL's on each ray will be calculated based off of the depth of the scene. This information is made available in the shadow maps in order to render direct shadows. The distance between VPL's on each ray will be logarithmic as shown below and the attenuation will be exponential.

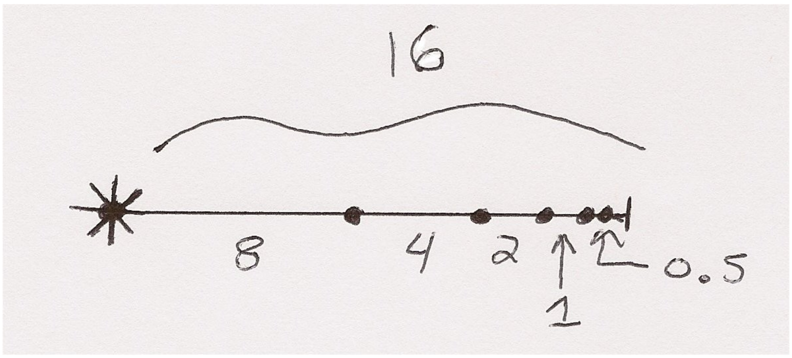

Figure 4.5: Showing the logarithmic structure of the VPL's. Attenuation levels would be 5%, 10%, 20%, 40%, and 80% from the left to right.

The goal of this structure is to simulate multiple waves of VPL's flowing outward from the primary light source in order to simulate the light transport as wave-like in order to simulate indirect illumination using only the direct illumination of these VPL's while ignoring the bouncing of light among surfaces. An additional goal of this implementation is to have the resulting technique be scalable. This organization of the VPL's will allow us to use more or less VPL's depending on quality and performance desires. As such, we can increase or decrease the corresponding angles to reduce or increase the number of VPL's used as well as increase or decrease the number of hemisphere shells used. Also, we can use stochastic sampling in order to turn off or on certain VPL's in order to introduce "noise" in the implementation as well as decrease the number of VPL's required. As shown in the previous section, noise can be perceptually pleasing provided it is in smooth graduations.

The VPL's contributions will computed as shown below. The VPL's will act similar to directional lights in that the normal of the VPL will be used in computing the radiance that it provides. If the surface point is in front of the VPL, it will receive the full radiance amount accounting for distance to the VPL as well as its attenuation. If the surface point is behind the VPL, it will receive half of the radiance it would otherwise receive.

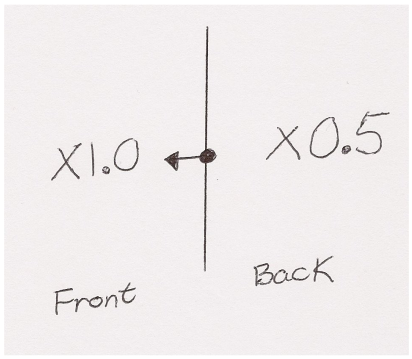

Figure 4.6: Showing the radiance contribution of a VPL on a surface point based off whether it is in front or behind the light. The arrow signifies the VPL's direction/normal.

Occlusion will be handled with a shadow map as in [Williams 1978] and [Reeves et al. 1987] to render shadows for the direct illumination based solely off of the primary light source. This method is very cheap and provides sufficient shadows for the purposes of this implementation. The indirect occlusion will handled one of two ways. One way is to use the idea of imperfect visibility as in [Ritschel et al. 2008] for all VPL's or through the use of regular shadow maps on stochastically sampled VPL's

(4.2.1) GPU AND CPU FUNCTIONS

This implementation will be designed to use both the CPU and the GPU. The CPU will handled all preliminary tasks such as setting up the window, initializing variables and the shaders, setting up the scene including position for the light source and objects, handling interactive actions including keyboard callbacks, initializing and filling textures, and initializing the VPL's position, direction, and attenuation. The GPU will calculate shading and illumination based off vertices inputted along with the VPL texture and calculate shadows based off the provided shadow maps passed in from the CPU.

(4.2.2) OPENGL

OpenGL or Open Graphics Library is a multi-platform API for graphics that will be used for this implementation.

(4.2.3) GLSL

GLSL or OpenGL Shading Language is a high-level C-syntax shading language to be used with OpenGL in order to give the developer control over what instructions are executed in the vertex and fragment shaders of the graphics pipeline. In this implementation, all illumination or shading calculations are computed in the vertex and fragment shaders.

(4.2.4) WORKING ENVIRONMENT

The implementation is being designed and tested on the following hardware and software using the following libraries. Any and all results such as rendered images and fps results will be gathered while running the method on the following.

HARDWARE

CPU: AMD Athlon 62 X2 Dual Core 5200+ 2.61Ghz
GPU: Nvidia GeForce GTX 465 (1GB memory)
Memory: 6.00GB

All other hardware specifications are irrelevant.

SOFTWARE

OS: Windows 7 64-Bit
Languages: C++, OpenGL, OpenGL Shading Language
Developer Tools: Microsoft Visual Studio 2008

LIBRARIES

The following libraries are used in this implementation and provided in the repository.

OpenGL Easy Extension Library (GLEE)
OpenGL Extension Wrangler Library (GLEW)
OpenGL Utility Toolkit (GLUT)