COSC 448: REAL-TIME INDIRECT ILLUMINATION

Written by Stephen Smithbower

Supervisor: Dr. Ramon Lawrence

January 2010 - April 2010

Table of Contents

Overview ii

- Purpose ii
- Overview ii
- Candidate Approaches ii
- The Chosen Approach iv

Implementation vi

- Overview vi
- Deferred Rendering vi
- Reflexive Shadowmap vi
- Instant Radiosity vii

Conclusion viii

Overview

Purpose

The aim of this project was to design and implement a system to allow for the real-time calculation of single-bounce indirect illumination in dynamic, interactive 3d-environments.

Overview

Indirect illumination is a natural phenomenon that results when light from a light source (known as direct lighting, or incident lighting) partially reflects off of a surface and continues to illuminate other surfaces until the majority of the light is absorbed. This effect is readily observed in every-day life, and is particularly noticeable in brightly-coloured environments. The primary effect of this indirect illumination is a general brightening of a given room, versus the ambient light level that would have resulted if lighting was a result of direct illumination only. A very pronounced consequence of this is an effect known as "light-bleeding", where coloured light is reflected off one surface onto another in a very noticeable fashion (such as when light bleeds off of a white couch onto a red wall, appearing to stain the wall red).

Indirect illumination, and in particular light bleeding, is a very powerful technique for softening the generally harsh lighting that direct lighting calculations provide in simulated 3d environments, and help to not only add a sense of realism to a scene, but also provide artists with a powerful tool to create mood and atmosphere.

Indirect illumination is not a new concept in 3d environments. In fact offline reproduction (pre-calculating lighting results in a pre-production phase) has been in use for several years - particularly in animated films. For example, Shrek 2 was the first film to incorporate the effect into its production. However until recently, commodity desktop graphics hardware has not been powerful enough to calculate the full lighting solution (both direct and indirect lighting) at speeds which would allow for seamless user interaction.

Over the last 10 years hardware dedicated to at first transforming 3d geometry and performing the direct illumination calculations, and recently to performing general purpose floating point calculates, has been developed at an incredible pace. This hardware is known as the graphics processing unit, or GPU, and works in tandem with the traditional CPU in order to provide ever-more elaborate lighting calculations at interactive frame rates. The GPU has now reached a performance threshold where it is now conceivable to move beyond direct illumination, which is the traditional approach to real-time lighting, and into performing both direct and indirect illumination in interactive scenes.

Candidate Approaches

A literature search in the area of real-time indirect illumination provided a list of possible candidate techniques that have been implemented on recent GPU architectures. The following were considered.

- **Spherical Harmonics**

During a pre-processing step, the geometry of the scene is considered, and a light transfer function is calculated for each vertex in the scene. This transfer function is then encoded as a third-order wavelet and stored along with the scene geometry. At runtime, the entire transfer function can be quickly evaluated by taking the dot product

of the wavelet and the incident light vector, for each light in the scene. This technique provides very fast run-time evaluation of the indirect lighting solution, and provides other benefits such as soft-shadows, ambient occlusion, and support for sub-surface scattering. However, drawbacks include a very intensive and long pre-processing procedure, the ability to represent a transfer function for only distant light sources (such as the sun) due to wavelet compression, and support only for static geometry (wavelets can be rotated easily, which allows for dynamic, moving light sources, but they cannot be displaced, rendering the lighting solution invalid if geometry within the scene were to move). Finally, because a wavelet must be stored per-vertex, the storage requirements for the lighting solution quickly become prohibitive for complex geometry. Unfortunately, the restrictions on scene geometry and light distance render this technique unfit for our purposes.

• Cube Map Array

During level design, a grid array of light probes (implemented as cube maps on graphics hardware) are placed throughout the scene. At runtime, the world is rendered using directed lighting into each cubemap-probe and then blurred, providing a low-frequency representation of the incoming light at each probe position. This technique is very similar to a popular photography technique used to measure light levels and propagation within a room. For each vertex in the scene, the nearest N probes are interpolated and sampled, providing the amount of indirect lighting available to that surface. This technique provides a rough approximation of indirect illumination that maps fairly well to commodity graphics hardware, utilizing a natural data structure, and supports completely dynamic geometry. However the amount of scene geometry that must be processed when updating the light probes can quickly become a bottleneck, and choosing the correct N probes per-vertex results in a very large under-utilization of the graphics hardware (due to cache misses, which hurts parallelism performance).

• Photon Mapping

During a pre-processing phase, photons are traced out from light sources into the scene geometry, and intersections are mapped to a spatially-aware data structure (such as a BSP tree, Octree, or K-D Tree). During runtime, light is traced from the incident light source and propagated using the accelerated data structure. The final values are then mapped into a texture for use on the GPU, and read in as part of the lighting equation. This technique has only been implemented with limited success, and has been largely abandoned as of late. The pre-processing step limits the solution to static geometry (unless highly elaborate schemes are employed to provide temporal updating of the accelerating data structure), and mapping the data structure to the GPU has proven to be a very large challenge.

• Screen-Space Indirect Illumination

This technique is relatively new, and is an extension of an earlier technique pioneered by Crytek in their game Crysis. Crytek developed a screen-space (working only on scene information encoded in a texture from the user's point of view) technique for faking ambient occlusion as a post-process, by comparing the depths and normals of small regions of pixels in order to accumulate a blocking factor for each given pixel in the scene. Screen Space Indirect Illumination works in a very similar fashion, however instead of accumulating a blocking factor, an incoming reflected light factor is accumulated from neighboring pixels. While not even close to being physically accurate, this technique provides fast light bleeding and works well with small details and low frequencies. It maps very well the GPU (being entirely texture-based) and makes no assumptions on the general makeup of the scene. However this technique suffers from extreme locality - the search radius for influencing neighboring pixels must be kept moderately small in order to maintain performance (a larger radius greatly increase the number of cache misses on the GPU and breaks parallelism), so many influences will be missed, and any influencers from outside of the user's field of

view will not affect the indirect illumination for the scene. As a result, light bleeding can "pop" in and out, depending on scene occlusion. This lack of continuity renders this technique suitable only for small, low-resolution details.

- **Reflective Shadow Maps**

Reflective shadow maps draws from the principle of deferred rendering in order to accelerate what is generally the most expensive step in the indirect lighting calculations - the light-ray intersection with the scene, required to location reflective surfaces. In this approach, the scene is rendered from each light's point of view, with geometry position, normal, and diffuse reflectance (the direct lighting) encoded as textures in GPU memory, forming what is referred to as a geometry buffer. Each pixel in this geometry buffer acts as a point light source. This geometry buffer is iterated over, and the lighting contribution from each pixel is propagated back into the scene. This technique maps very well to modern graphics hardware, requires no pre-processing of the scene, handles fully dynamic scenes, and is not view dependent. However, treating each pixel in the geometry buffer as a light source is extremely expensive - a very large number of calculations must be performed per-pixel, saturating both memory and FLOP bandwidth, to the point where the technique is not viable for more than a single light source.

- **Instant Radiosity**

Instant radiosity uses the fact that indirect lighting is low-frequency, and so attempts to minimize the number of light traces required to propagate light within a scene by interpolating lighting values between two points. This is mapped well to the GPU by treating each reflective point as a virtual 180 degree light dome (or spotlight) and utilizing the GPU's built in support for calculating direct lighting equations. At runtime, rays are traced from each light source until they intersect with the scene. At the point of intersection a virtual light source is placed, with its properties (radius of influence, colour) based on the reflective material properties of the given surface. The scene is then rendered with the accumulation of both the direct and indirect (via the virtual lights) light contributions. This technique is view-independent, makes no presumptions on the scene (and therefore handles dynamic geometry), and maps well to the GPU. However, much of the work is still done on the CPU (the scene ray-tracing) which is slow, and bottlenecks occur when attempting to transfer the virtual light information over to the GPU (the scene must be rendered once per virtual light). Several hundred virtual light sources are required in order to produce an aesthetically pleasing light-bleed effect, rendering this technique impractical for all but the simplest scenes.

The Chosen Approach

A combination of three approaches was chosen in order to attempt to implement a solution that would allow for a single bounce of indirect illumination in fully dynamic and interactive scenes. The overall goal was to take the strengths from each technique and combine them in order to move as much work onto the GPU as possible, and map scene and lighting information into data structures that maximize GPU utilization. To do this, a hybrid of Screen Space Indirect Illumination, Reflective Shadowmaps, and Instant Radiosity was designed, as well as changing rendering paradigms from traditional forward rendering to deferred rendering.

Instant radiosity's approach of using virtual light sources distributed throughout the scene to represent indirect light was chosen as it maps very well to the GPU architecture, taking advantage of modern techniques and advancements,

and is the least intrusive technique when used on arbitrary scene geometry. However, instead of tracing out light rays from each light source in order to place the virtual lights (a very slow process), a reflexive shadowmap will be used instead. The scene will be rendered from the light's point of view, with each pixel representing a possible virtual light that can then be placed into the scene without any intersection tests. In order to minimize the number of lights generated, only a subset of pixels will be sampled (for sake of simplicity and generality, a poisson disc sampling pattern was used, pre-computed and stored in a texture lookup table). In order to address the bandwidth problems associated with rendering the scene once for each virtual light source, deferred rendering is used to shift all light rendering onto the GPU, so that scene geometry has a fixed overhead. Screen-Space Indirect Illumination is then performed on the final scene with a very small search radius, so that it can quickly fill in only minor, highly local details (such as light bleeding at contact points).

This approach supports dynamic scenes, is only partially view-dependent, is scalable, and maps very well to the GPU. Additionally, because no elaborate data structures are required, implementation is relatively simple, and can sit along side existing rendering infrastructures.

Deferred Rendering

Deferred rendering is an approach for rendering that attempts to separate geometry from lighting, decoupling the two so that an increase in complexity for one does not also increase the complexity for the other. In deferred rendering the scene is rendered from the user's point of view, and geometry information (such as depth and surface normal) are encoded into textures and stored in a geometry buffer. Each light is then drawn into the scene using proxy geometry (a sphere for point lights, a cone for spot lights), sampling any pixels from the geometry buffer that the proxy geometry overlaps. Lighting is then calculated for these pixels using values from the geometry buffer, and stored into a light accumulation buffer. The scene is then rendered again with more material properties, sampling lighting values from the light accumulation buffer, and the result is a fully illuminated scene. By encoding the scene geometry into textures which are stored on the GPU, a large number of light sources may be rendered in real time without have to re-download scene data to the GPU for each light source. This alleviates the bandwidth bottleneck that plagues traditional forward rendering with complex scenes.

Implementation

Overview

Original implementation was to be done using Java, on top of OpenGL (for cross-platform compatibility). However, after a couple of weeks worth of development on the java-based graphics engine, it was decided that too much time was being invested into infrastructure development. A switch was made to the TrueVision3D graphics engine, and DirectX. This allowed us to focus on implementation of the proposed lighting pipeline, and not have to worry about managing the graphics API, cross-platform compatibility, or resource management.

Light Pre-Pass, a specific technique for deferred rendering, was implemented with good success. A demo was built that showcased up to 1000 dynamic light sources (powered by the Newton physics engine) in view at the same time, running at over 60 frames per second.

Implementation of the reflexive shadowmap was also completed with success. Optimization techniques were used in both the geometry buffer required for the deferred rendering, and for the reflexive shadowmap.

Implementation of instant radiosity utilizing the reflexive shadowmap was met with mixed results. While the approach was technically implemented, performance and aesthetic results were not acceptable. More work is required in both the sampling pattern used for virtual light placement and scaling, and in optimizing the generation of the virtual lights.

Deferred Rendering

Fairly standard implementation of light pre-pass, utilizing a two-buffer system. One buffer stored the depth of the scene, from the camera, and another stored the world-space normals of the scene. Depth was used as it could be compressed from a floating point value to an integer using 4 channels of the buffer, cutting bandwidth requirements in half. Spherical coordinate compression was used for normals in order to reduce the number of required channels from 4 to 2, also to cut down on the bandwidth required for storage (specialized two-channel texture formats were used). Lights were rendered using instanced proxy geometry - light positions, ranges, colours and rotations were uploaded as a large block to the GPU, and the same light geometry was then modified solely on the GPU using these values. For 1000 light sources, this reduced the number of draw calls from 1000 to 4, for the light sources (a maximum of 250 light sources could be rendered in a single call). During rendering, depth information was decoded and used to reconstruct the world-space position of a given pixel, based on the view frustum. MRT (multiple render targets) was used in order to draw to multiple buffers with a single draw call. GPU utilization was fairly high with this approach, and performance exceeded expectations.

Reflexive Shadowmap

Implementation was very similar to the geometry buffer implemented for deferred rendering. The same compression techniques were implemented, and MRT was used. 3 buffers were required for the reflexive shadow map - depth, normals, and diffuse reflectance (often referred to as flux).

Instant Radiosity

Implementation at this stage was difficult and never reached satisfactory results. While the theory worked (some light bleeding was observed by redistributing virtual lights using the reflexive shadowmap), the sampling pattern employed (using a poisson disc distribution) yielded concentrations of light in some areas of the scene, and a lack of indirect in others. Additionally, performance was poor, mostly due to the unoptimized read-back of pixel values on the CPU. A further optimization that was not attempted due to lack of time would be to generate the instancing data for the virtual lights purely on the GPU with no read-back, by having each light source use its index to sample a lookup table that could provide correct texture coordinates in order to sample from the reflective shadow map. This would (potentially) remove the bandwidth issues bottlenecking the current pixel read-back. Further work is required in order to come up with an optimal sampling pattern for light distribution, and for properly modeling the light transfer function.

Conclusion

The feasibility of real-time indirect illumination in dynamic scenes is still an open question at this point in time. However, even with only the partial results observed in this study, there is indication that with further work and optimization, this combination of techniques could very well lead to a suitable implementation. The key work done in this study is adapting several existing techniques in order to better utilize the power of the GPU. Exceptionally positive results were observed with deferred rendering, and there is still a large amount of room available for further optimizations. At this point in time, instant radiosity still remains highly sensitive to "artist-based' parameters, such as sampling pattern, strength of virtual light sources, and scaling of the scene. Further work in this area is recommended in order to better produce a natural, self-adjusting system that can automatically adjust to several different scene types.