

Graphics Programming with DirectX 9

- Module III -

Workshop I: Lightmapping and Radiosity

by

Gary Simmons/Adam Hault

(Lesson Plan)

Lesson 1: Geometry-Based Lighting

Textbook: Chapter 18 (pgs. 1-276)

In this lesson students will review the use of pre-compiled lighting in games and implement the first of three lighting compiler tools that we will be building over the next three lessons (a fourth compiler will follow in Workshop II after pixel shaders have been introduced). Pre-compiled lighting remains a vital technique in commercial titles for illumination of static environments. Our first compiler will generate lighting at the vertex level for later use in the runtime component. This will be a command line tool that can be spawned by the GILES™ 2.0 process pipeline, thus allowing us to integrate our illumination model into the art development pipeline.

Our vertex lighting compiler will use the Blinn-Phong illumination model to record vertex colours, much like the Direct3D fixed-function lighting pipeline does. Because we are calculating the vertex colours offline, we will also be able to factor static geometry occlusion into the end result. The input to the compiler will be an unlit level that contains one or more standard and/or emulated area light sources. The output will be a scene that exhibits smooth shadowing and filtered lighting and will be saved to disk using our proprietary IWF format. Because colours have been stored at the vertices by the compiler, all the runtime component needs to do is disable the Direct3D lighting pipeline and render the scene as pre-lit vertices.

Since the quality of any vertex lighting system relies on the level of geometric detail, students will build a geometry tessellation unit into the compiler. The tessellator will carve the level up into smaller polygons to evenly distribute the vertices (the light sample points) throughout world space. This will ensure smooth and consistent illumination across the scene.

Because specular lighting is view-dependant and cannot be compiled offline, students will examine how to integrate it into their rendering pipeline at runtime. We will also look at how to generate a specular mask in our compiler so that specular highlights do not show up in the shadowed areas of our level.

Many other topics will be covered along the way as we delve into the code for this lighting tool, including the implementation of line of sight systems, an examination of barycentric coordinates and how they can be used to perform filtered texture lookups, supporting transparent surfaces so that light rays can pass through them and be tinted by their colour, and more.

Key Topics:

- Pre-compiled Lighting Theory
- The Blinn-Phong Illumination Model
 - Ambient, Diffuse, Specular Lighting
 - Light Sources & Materials
- Surface Casters – Area Light Emulation
- Geometry Tessellation
 - Carving the world into evenly sized areas
 - Avoiding T-junctions during tessellation
 - Generating arbitrary plane-aligned bounding boxes
- Line of Sight Testing
- Using Transparent Polygons with Light Sources
- World Space Lumel Filtering
- Specular Masks and Dynamic Specular Contributions
- Barycentric Coordinates

Projects:

- Lab Project 18.1: Lighting Compiler I – Blinn-Phong (Geometry)

Recommended Study Time: 2 weeks

Lesson 2: Texture-Based Lighting - Lightmaps

Textbook: Chapter 19 (pgs. 277-446)

In this lesson we will implement a second lighting compiler, once again based on the Blinn-Phong illumination model. Unlike the previous chapter, this compiler will store its results in texture surfaces so that detailed lighting can be efficiently integrated at runtime without the need for high levels of geometry tessellation. Such textures are referred to as 'lightmaps' and are used extensively by the latest commercial games to provide illumination for the static environment. Because textures will be used to store the lighting results, very detailed lighting becomes available at runtime simply by blending our lightmap and base textures during rendering.

Students will learn many new techniques to make their lighting compiler more powerful, such as the ability to create and clip patches to more accurately account for lumels that may be partially occluded in world space. We will also talk about how to implement safeguards so that runtime texture filtering does not cause issues at the borders of our lightmaps; all common problems that must be overcome to avoid visual artefacts.

The compiler we implement in this chapter will also support any terrains that may exist in the level. The end result will be a terrain lighting texture that will contain the correct illumination for the entire terrain (i.e., buildings or hills casting shadows onto the terrain, etc.). Students will learn how to use heightmaps for efficient line of sight testing so that terrains can also be factored into full scene occlusion tests.

As with our previous compiler, only diffuse lighting can be captured as an offline process and thus a system must be invoked at runtime to generate specular contributions. In this lesson we will look at a way to do this.

Before concluding, students will learn how to perform efficient texture consolidation using binary trees. The goal will be to take the n individual lightmaps generated by the compiler (one per polygon) and pack them onto as few physical texture surfaces as possible. This will minimize texture swapping and maximize batch rendering.

Key Topics:

- Texture Based Lighting Theory
- Texture Coordinate Generation
- Generating Patches and Performing Patch Clipping
- Terrain Lighting
- Standard & Terrain Line of Sight Tests
- Specular Rendering using Environment Maps
- Texture Image Consolidation
- Compiler Optimizations
- Memory Overhead Considerations
- Generating Random Points on a Polygon Surface

Projects:

- Lab Project 19.1: Lighting Compiler II – Blinn-Phong (Texture)

Recommended Study Time: 2 weeks

Lesson 3: Texture-Based Lighting II: The Radiosity Model

Textbook: Chapter 20 (pgs. 447-624)

In this chapter we expand our texture-based lighting solution to include a global illumination model. Our third compiler tool in this course will implement the radiosity method. Unlike the Blinn-Phong model, radiosity allows for light transfer between diffuse surfaces in the scene. Thus our tool will support direct lighting from nearby light sources as well as reflected light from other surfaces. This global interaction will replace the crude ambient approximation introduced previously and produce more realistic results. Students will examine phenomena like colour bleeding, where a green box situated by a white wall might tint the wall slightly green due to its reflected light energy. They will also be able to finally model true area light sources and proper emissive surfaces.

Traditional light types (spot, directional and point) do not technically exist in a radiosity environment, since the model calls for surface-based light sources. This can be a bit of a cumbersome limitation, so students will create a hybrid compiler that implements a 'charge pass' using the Blinn-Phong model with traditional light source types. In a second pass we will use the radiosity method to bounce surface energy amongst all other surfaces in the scene until the final colour of each surface is determined. This will allow students to compile radiosity solutions for level data that may not have been constructed with radiosity in mind and provides total flexibility and ease of use.

Like the compiler we implemented in the previous chapter, the radiosity results will be stored in textures (lightmaps) and saved to disk in a consolidated format. All the rendering component has to do is load the lightmaps and apply them to the polygons to reproduce the results of the radiosity solution at runtime.

Key Topics:

- Global Light Interaction between Surfaces – Theory
- Classic Radiosity
- The Radiosity Equation Explained
 - Integration Over the Environment
 - Matrix Radiosity
 - Form Factor Calculation
 - The Hemicube Approach
 - The Ray Tracing Approach
 - Form Factor Error Correction
- Progressive Refinement Radiosity
- True Emissive Surfaces
- Two-Pass Hybrid Approach
 - Blinn-Phong Charge Pass using Traditional Light Sources
 - Radiosity Pass using Progressive Refinement
 - Exporting the Two-Pass results as Combined Lightmaps or Separate Sets

Projects:

- Lab Project 20.1: Lighting Compiler III – Radiosity (Texture)

Recommended Study Time: 2 weeks