Graphics Programming with Direct X 9

Module II

(14 Week Lesson Plan)

Lesson 1: Meshes

Textbook: Chapter Eight (pgs. 2 – 77)

Goals:

The course begins by introducing some of the important mesh containers provided by the D3DX library. Discussion will center on performance issues, including attribute batching across mesh boundaries and subset rendering, as well as optimization techniques that speed up rendering on modern hardware. From there we will look at how to import X file geometry into our applications as well as how to construct and fill the mesh buffers manually. This will lead into a discussion of cloning (copying) mesh data and some of the features that can be exploited in the process. The next topic of discussion will be the management of geometric level of detail using view independent progressive meshes. We will look at how to construct and use progressive meshes and see how they work algorithmically. This will lead into an examination of one-off mesh simplification and how it can be done with D3DX support. We will conclude this lesson with a quick overview of a number of useful mesh utility functions.

Key Topics:

- ID3DXMesh Interface
  - Vertex/Index/Adjacency Buffers
  - Attribute Buffers and Subset Rendering
- Mesh Optimization
- ID3DXBuffer
- Mesh Loading
- Manual Mesh Creation
- Mesh Cloning
- ID3DXPMesh Interface
  - View Independent Progressive Meshes (VIPM)
  - Data Validation and Cleaning
  - Setting LOD
  - LOD Trimming
  - Vertex History
- ID3DXSPMesh Interface
- Global Mesh Utility Functions

Projects:

Lab Project 8.1: The CTriMesh Class (Mesh Viewer I)

Exams/Quizzes: NONE

Recommended Study Time (hours): 8 - 10

Lesson 2: Frame Hierarchies

Textbook: Chapter Nine (pgs. 2 – 87)

Goals:

In this lesson we will now look at how to import and manage more complex 3D models and scenes. We will introduce the concepts of frame of reference and parent-child hierarchical relationships and see how we can use these ideas to build elaborate scenes consisting of independent, animation-ready meshes. Early on in the process we will delve into the inner workings of X file templates to see how scene data is stored. This will set us up for a discussion of the very important D3DXLoadMeshHierarchyFromX function, which we will use many times in the coming lessons. Using this function properly will require an examination of the callback mechanisms and data structures used for application memory management. We will even talk about how to load custom data chunks. Finally, we will wrap up the lesson with a look at how to traverse, transform, and render a hierarchy of meshes. A very simple animation controller will be introduced during the process and in our lab project to setup our discussions in the next lesson.

Key Topics:

- Hierarchies
  - Frame of Reference
  - Parent/Child Relationships
- X File Templates
  - Open/Closed/Restricted Templates
  - Hierarchical X Files
- D3DXLoadMeshHierarchyFromX
- ID3DXAllocateHierarchy Interface
  - Allocating/De-allocating Frames
- ID3DXMeshContainer Interface
  - Allocating/De-allocating Mesh Containers
- Extending Hierarchy Data Types
- ID3DXLoadUserData Interface
  - Loading Custom Top-Level Data
  - Loading Customer Child Data
- Hierarchy Traversal and Rendering
- Simple Hierarchy Animation

Projects:

Lab Project 9.1: The CActor Class (Mesh Viewer II)

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 - 12

Lesson 3: Keyframe Animation I

Textbook: Chapter Ten (pgs. 2 – 64)

Goals:

In this lesson our goal will be to learn the fundamentals of animating game scenes. The primary focus will be on using keyframe data to animate the hierarchies introduced in the previous lesson. Our initial discussions will take us back into the inner workings of X file templates, where we will learn about the various ways that animation data can be represented and how it all translates into D3DX data structures. From there we will begin our exploration of the powerful animation system available in DirectX. This exploration will involve understanding how the animation controller interpolates keyframe data and how that process can be controlled using various subsystems in the controller. Along the way we will examine the construction of a custom animation set object that can be plugged into the D3DX animation system.

Key Topics:

- Animation Blending
  - The Animation Mixer
    - Setting track weight, speed, priority
    - Enable/Disable Tracks
  - Priority Blending
- Animation Controller Cloning
- The Animation Sequencer
  - Registering Events
  - Event Handles
- The Animation Callback System
  - Callback keys and animation sets
  - Executing callback functions
  - ID3DXAnimationCallbackHandler Interface

Projects:

Lab Project 10.1: Animated CActor (Mesh Viewer III)
Lab Project 10.2: The Animation Splitter

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 – 12

Lesson 4: Keyframe Animation II

Textbook: Chapter Ten (pgs. 64 – 114)

Goals:

In this lesson our goal will be to continue our discussion of animation fundamentals by examining some different animation controller subsystems. The first major controller subsystem encountered will be the animation mixer, where we will learn about the important topic of blending multiple simultaneous animations. After learning how to use the mixer and configure its tracks for blending, we will conclude our discussions by looking at how to setup various user-defined special events using both the animation sequencer and the animation callback system. These features will allow us to sync together our animation timeline with events like playing sound effects or triggering specific pieces of function code.

Key Topics:

- Animation Blending
  - The Animation Mixer
    - Setting track weight, speed, priority
    - Enable/Disable Tracks
  - Priority Blending
- Animation Controller Cloning
- The Animation Sequencer
  - Registering Events
  - Event Handles
- The Animation Callback System
  - Callback keys and animation sets
  - Executing callback functions
  - ID3DXAnimationCallbackHandler Interface

Projects:

Lab Project 10.1: Animated CActor (Mesh Viewer III) cont.
Lab Project 10.2: The Animation Splitter cont.

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 – 12

Lesson 5: Skinning I

Textbook: Chapter Eleven (pgs. 2 – 115)

Goals:

In this lesson we will finally integrate animated game characters into our framework. This will build on all of the topics covered in the prior lessons including meshes, hierarchies, and the animation system. We will begin our examination by looking at some of the methods used for animating game characters in older games. What we learn will lead us straight into the idea of skinning and skeletal animation as a means for providing more realistic visual results. We will learn all about what skins and skeletons are, how they are constructed, and how they can be animated and rendered. As before we will look at the X file data templates and see how these translate into our game data structures. Then we will examine the various skinning options available via D3D. This will include detailed examinations of software skinning and hardware skinning; both non-indexed and palette-driven indexed skinning techniques.

Key Topics:

- Vertex Tweening
- Segmented Models and Animation
- Bone Hierarchies/Skeletons
- Vertex Blending
- Skinning
- X File Templates for Skinning
- The Bone Offset Matrix
- Software Skinning
- ID3DXSkinInfo Interface
- Non-Indexed Skinning
    - Setting multiple world matrices
    - Enabling/disabling vertex blending
    - ConvertToBlendedMesh
- Indexed Skinning
    - Determining Support
    - Matrix Palette Indices
    - ConvertToIndexedBlendedMesh
- Transforming and Rendering Skinned Characters

Projects:

Lab Project 11.1: Skinned CActor (Mesh Viewer IV)
Lab Project 11.2: The Animation Splitter II

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 - 12

Lesson 6: Skinning II

Textbook: Chapter Twelve (pgs. 2 – 160)

Goals:

In this lesson we will conclude our exploration of skinning and animation by taking a different angle from the prior lesson. This time, rather than load skinned characters from an X file, we are going to construct an entire skeleton and skin model programmatically. The end result will be a robust tree class that we can use to generate realistic looking animated trees for populating our outdoor landscape scenes. Since this is the halfway point in the course, we are also going to make an effort to bring together much of what we have learned to date into a single demonstration lab project. One important focus in this second lab project will be the extension of our middle-tier to include data driven support between our application and the D3DX animation system. This upgraded system will handle animation switching, blending, and the other key areas that are necessary to simplify the communication pipeline between the application and the low level animation code. This will allow students to more easily integrate animation support into their game projects and have their AI or user-input systems interact and control the process.

Key Topics:

- Trees
  - Procedural Skins and Skeletons
  - Procedural Keyframe Animation
- The Animation Middle Layer
  - Data Driven File Support
  - Animation Set Blending
  - Controller Configuration
  - Playing Back Complex Animations

Projects:

Lab Project 12.1: The CTreeActor Class (Mesh Viewer V)
Lab Project 12.2: Summary Lab Project

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 – 12

Lesson 7: Midterm Exam Preparation and Review

Textbook: Chapters 8 - 12

Goals:

The midterm examination in this course will consist of 50 multiple-choice and true/false questions pulled from the first five textbook chapters. Students are encouraged to use the lecture presentation slides as a means for reviewing the key material prior to the examination. The exam should take no more than two hours to complete. It is worth 30% of the final grade.

Office hours will be held for student questions and answers.

Key Topics:

Projects: NONE

Exams/Quizzes: Midterm Examination (50 questions)

Recommended Study Time (hours): 12 - 15

Lesson 8: Collision Systems I

Textbook: Chapter Thirteen

Goals:

In the second half of the course students will begin to explore important generic topics in the area of game engine design. While we will not conclude our game engine design studies until Module III, we will begin to lay the foundation for most of the core systems. In this lesson and the next we will undertake the development of a robust collision detection and response system. We begin with an overview of collision detection and look at the difference between broad and narrow phase algorithms. From there we will explore a sliding response system that is a staple of many first and third person games. After we have tackled the overall system architecture, including the management of geometry, we will introduce the concept of ellipsoid space and see how it will be used to facilitate the entire process. Then we will start our examination of the intersection algorithms that are going to be used in the narrow phase of our collision detection engine. We will talk about rays, what they are and how they can be tested against common game primitives. Then we will begin to look at how spheres can be tested against triangle interiors. This will lead into the additional testing algorithms covered in the next lesson.

Key Topics:

- Collision Systems Overview
- Broad Phase vs. Narrow Phase Collision Detection
- Collision Response
    - Sliding
- Ray Intersection Testing
    - Ray vs. Plane
    - Ray vs. Polygon
- Ellipsoids, Unit Spheres, and Ellipsoid Space
- Swept sphere vs. Triangle

Projects:

Lab Project 13.1: Collision System

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 - 12

Lesson 9: Collision Systems II

Textbook: Chapter Thirteen

Goals:

In this lesson we will complete the development of our collision detection and response system. Since intersection testing is fundamentally about solving equations, we will begin by reviewing the concept of quadratic equations and some of the fundamental mathematics used during the detection phase. Quadratic equations are used in a number of our system's most important routines, so this overview should prove helpful to students who have forgotten some of this basic math. This will ultimately lead into an examination of intersection testing between swept spheres and the edges and vertices of our triangles. This is where we will wrap up our core routines for the narrow phase tests against static objects and environments. Once done, we will move on to discuss the means for colliding against dynamic objects that are part of the game environment. Dynamic object support will require a number of upgrades to both the detection and response stages in our code.

Key Topics:

- Quadratic Equations
- Swept Sphere Intersection Testing
    - Swept Sphere vs. Edge
    - Swept Sphere vs. Vertex
- Animation and the Collision Geometry Database
- Dynamic Object Collision Support

Projects:

Lab Project 13.1: Collision System

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 - 12

Lesson 10: Spatial Partitioning I

Textbook: Chapter Fourteen

Goals:

In this lesson we will introduce some of the most important data structures and algorithms that are used to improve game engine performance. We will begin with simple axis-aligned hierarchical spatial partitioning data structures like quadtrees, octrees, and kD-trees. These systems will allow us to introduce broad phase collision detection into the system we developed in the prior lessons. Once done, we will introduce the very popular non axis-aligned spatial subdivision technique called binary space partitioning (BSP). BSP trees will actually be used in the next few lessons of the course to accomplish some very important ends, but for now only basic theory will be covered. The goal in our lab project will be to create a single base class with a core set of polygon querying functionality (intersection testing) and a set of derived classes for all tree types. This will allow students to mix and match the tree types as it suits the needs of their own projects.

Key Topics:

- Spatial Partitioning Data Structures/Algorithms
  - Quadtrees
  - Octrees
  - kD Trees
  - BSP Trees
- Polygon Clipping
- Polygon Database Intersection Querying
  - Ray/AABB/Sphere testing
- Broad Phase Implementation for Collision System

Projects:

Lab Project 14.1: Broad Phase Collision Detection

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 - 12

Lesson 11: Spatial Partitioning II

Textbook: Chapter Fourteen

Goals:

In the last lesson we introduced some of the core partitioning structures that are used to improve game engine performance. Our prior focus was on the means for assembling the tree data structures and for traversing them to do polygon queries. This allowed us to add the very crucial broad phase to our collision detection system. In this lesson we will examine another aspect of using these trees – rendering. Hardware friendly rendering is an important idea that students need to think about when designing their partitioning systems. This can be a challenging thing to accomplish and there are many considerations, so this is something we will examine in this lesson. We will also introduce some additional concepts to speed up scene rendering by exploiting the hierarchical nature of our data during frustum culling and integrating the idea of frame coherence to add some additional optimization. Finally, we will look at using a BSP tree to perform accurate front to back sorting for rendering transparent polygons.

Key Topics:

- Accurate Alpha Polygon Sorting
- Frame Coherence
- Hardware Friendly Rendering
    - Static vs. Dynamic Solutions
    - Polygon Caches (Pros/Cons)
- Hierarchical Frustum Culling/Rendering

Projects:

Lab Project 14.2: Hierarchical Scene Rendering w/ BSP Alpha Sorting

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 - 12

Lesson 12: Spatial Partitioning III

Textbook: Chapter Fifteen

Goals:

This lesson will conclude our studies of spatial partitioning techniques and begin the transition into the development of potential visibility sets. We will begin by learning how to add solid and empty space information to our BSP tree representation. This will provide the foundation for a number of important tasks that follow in this lesson and the next. With that code in place, we will look at how to build BSP trees out of scene brushes in order to perform constructive solid geometry (CSG). The CSG techniques we study will allow for merging together of geometric objects, carving shapes out of other shapes, and other interesting dynamic tasks. We will conclude the lesson by introducing the concept of portals, the first step towards development of PVS. We will look at how to generate them, split them, and remove any duplicates.

Key Topics:

- Solid Leaf BSP Trees
  - Rendering
  - Line of Sight
- Constructive Solid Geometry (CSG)
  - Union/Intersection/Difference Operations
- Portal Generation
  - Portal Splitting

Projects:

Lab Project 15.1: Solid Leaf Tree Compiler/Renderer
Lab Project 15.2: CSG Operations

Exams/Quizzes: NONE

Recommended Study Time (hours): 10 – 12

Lesson 13: Spatial Partitioning IV

Textbook: Chapter Fifteen

Goals:

Our final lesson will complete the development of our first performance oriented game engine design. We will begin by looking at the process of calculating potential visibility sets. The first step is the discussion of penumbras and anti-penumbras and how volumetric lighting models can be used to determine visibility. Using the portals created in the last lesson we will model the flow of light through the scene to mark off areas of shadow and light. This will provide enough information to be able to draw conclusions about visible areas in the level. After we have calculated our PVS, we will look at how to compress the information and then use it to efficiently render our scenes. We will conclude the lesson with a look at a different approach to BSP tree compilation. Since illegal geometry can corrupt the BSP building process, we will look at methods that allow us to avoid these problems and at the same time generate BSP trees and PVS for just about any type of scene our artists can create. These BSP trees will not use the level polygons as the means for compilation, but will instead use artist-generated simplified primitives that bound the actual level data. With fast rendering technology, efficient collision detection, spatial subdivision, and geometry and animation support all in place, students will be fully ready to wrap up their game engine development studies in Module III.

Key Topics:

- Potential Visibility Sets
  - Zero Run Length Encoding
  - Scene Rendering
- Anti-Penumbras
  - Generator Portal Visibility
  - Portal Flow
- Polygon-less BSP Trees
  - Illegal Geometry

Projects:

Lab Project 16.1: The BSP/PVS Compiler
Lab Project 16.2: Final Project

Exams/Quizzes: NONE

Recommended Study Time (hours): 12 - 15

Lesson 14: Final Exam Preparation and Review

Textbook: NONE

Goals:

The final examination in this course will consist of 75 multiple-choice and true/false questions pulled from all textbook chapters. Students are encouraged to use the lecture presentation slides as a means for reviewing the key material prior to the examination. The exam should take no more than three hours to complete. It is worth 70% of the final grade.

Office hours will be held for student questions and answers.

Key Topics:

Projects: NONE

Exams/Quizzes: NONE

Recommended Study Time (hours): 12 - 15