Introduction to the
Direct3D 11 Graphics Pipeline

Kevin Gee - XNA Developer Connection
Microsoft Corporation
Key Takeaways

• Direct3D 11 focuses on
  – Increasing scalability,
  – Improving the development experience,
  – Extending the reach of the GPU,
  – Improving Performance.

• Direct3D 11 is a strict superset of D3D 10 & 10.1
  – Adds support for new features
  – Start developing on Direct3D 10/10.1 today

• Available on Windows Vista & future Windows operating systems

• Supports 10 / 10.1 level hardware
Outline

• Drilldown
  • Tessellation
    – Compute Shader
    – Multithreading
    – Dynamic Shader Linkage
    – Improved Texture Compression
    – Quick Glance at Other Features

• Availability
Character Authoring Pipeline

Sub-D Modeling

Animation

Displacement Map

Polygon Mesh

Generate LODs

(Rocket Frog Taken From Loop & Schaefer, "Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches")
Character Authoring (Cont’d)

**Trends**
- Denser meshes, more detailed characters
  - ~5K triangles -> 30-100K triangles
- Complex animations
  - Animations on polygon mesh vertices more costly

**Result**
- Integration in authoring pipeline painful
- Larger memory footprints causing painful I/O issues

**Solution**
- Use the higher-level surface representation longer
  - Animate control cage (~5K vertices)
  - Generate displacement & normal maps
Direct3D 11 Pipeline

Direct3D 10 pipeline

Plus

Three new stages for Tessellation
Hull Shader (HS)

**HS input:**
*patch control pts*

**HS output:**
- Patch control pts after Basis conversion
- **TessFactors** (how much to tessellate)
- fixed tessellator mode declarations

One Hull Shader invocation per patch
Fixed-Function Tessellator (TS)

Note: Tessellator does not see control points

TS input:
- TessFactors (how much to tessellate)
- fixed tessellator mode declarations

TS output:
- U V \{W\} domain points
- topology (to primitive assembly)

Tessellator operates per patch
Domain Shader (DS)

**DS input:**
- control points
- TessFactors

**DS input:**
- U V \{W\} domain points

**One Domain Shader invocation per point from Tessellator**

**DS output:**
- one vertex
Direct3D 11 Pipeline

- D3D11 HW Feature
- D3D11 Only
- Fundamental primitive is patch (not triangle)
- Superset of Xbox 360 tessellation
Example Surface Processing Pipeline

Single-pass process!

**vertex shader**
- Animate/skin
- Control Points

**hull shader**
- Transform basis,
  Determine how
  much to tessellate

**tessellator**
- Tessellate!

**domain shader**
- Evaluate
  surface
  including
  displacement

**Patch**
- Patch control points

**Transformed control points**
- Transformed control points

**Control points in Bezier patch**
- Control points
  in Bezier patch

**U V \{W\} domain points**
- U V \{W\} domain points

**Displacement map**
- Displacement map

Sub-D Patch

Bezier Patch
New Authoring Pipeline

(Rocket Frog Taken From Loop & Schaefer, "Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches")

Sub-D Modeling  Animation  Displacement Map

GPU

Optimally Tessellated Mesh
Tessellation: Summary

• Provides
  – Smooth silhouettes
  – Richer animations for less

• Scale visual quality across hardware configurations

• Supports performance improvements
  – Coarse model = compression, faster I/O to GPU
  – Cheaper skinning and simulation
  – Improve pixel shader quad utilization
  – Scalable rendering for each end user’s hardware

• Render content as artists intend it!
Outline

• Drilldown
  – Tessellation
  – Compute Shader
    – Multithreading
    – Dynamic Shader Linkage
    – Improved Texture Compression
    – Quick Glance at Other Features

• Availability
GPGPU & Data Parallel Computing

- GPU performance continues to grow
- Many applications scale well to massive parallelism without tricky code changes
- Direct3D is the API for talking to GPU
- How do we expand Direct3D to GPGPU?
Direct3D 11 Pipeline

Direct3D 10 pipeline

Plus

Three new stages for Tessellation

Plus

Compute Shader
Integration with Direct3D

- Fully supports all Direct3D resources
- Targets graphics/media data types
- Evolution of DirectX HLSL
- Graphics pipeline updated to emit general data structures...
  - ...which can then be manipulated by compute shader...
- And then rendered by Direct3D again
Example Scenario

- Render scene
- Write out scene image
- Use Compute for image post-processing
- Output final image
Target Applications

• Image/Post processing:
  – Image Reduction
  – Image Histogram
  – Image Convolution
  – Image FFT

• A-Buffer/OIT

• Ray-tracing, radiosity, etc.

• Physics

• AI
Compute Shader: Summary

• Enables much more general algorithms
• Transparent parallel processing model
• Full cross-vendor support
  – Broadest possible installed base
Outline

• Overview

• Drilldown
  – Tessellation
  – Compute Shader

  Multithreading
  – Dynamic Shader Linkage
  – Improved Texture Compression
  – Quick Glance at Other Features

• Availability
D3D11 Multithreading Goals

• Asynchronous resource loading
  – Upload resources, create shaders, create state objects in parallel
  – Concurrent with rendering

• Multithreaded draw & state submission
  – Spread out render work across many threads

• Limited support for per-object display lists
Devices and Contexts

• D3D device functionality now split into three separate interfaces
• Device, Immediate Context, Deferred Context
  – Device has free threaded resource creation
  – Immediate Context is your single primary device for state, draws, and queries
  – Deferred Contexts are your per-thread devices for state & draws
D3D11 Interfaces

Render Thread
- Immediate Context
  - DrawPrim
  - DrawPrim
  - DrawPrim
  - DrawPrim

Load Thread 1
- Device
  - CreateTexture
  - CreateVB
  - CreateShader
  - CreateIB

Load Thread 2
- Device
  - CreateShader
  - CreateTexture
  - CreateShader
  - CreateVB
Async Resources

• Use the Device interface for resource creation
• All functions are free threaded
  – Uses fine-grained sync primitives
• Resource upload and shader compilation can happen concurrently
State & Draw Submission

• First priority: multithreaded submission
  – Single-use display lists
• Lower priority: per-object display lists
  – Multiple reuse
• D3D11 display lists are immutable
D3D11 Interfaces

Immediate Context
- DrawPrim
- DrawPrim
- DrawPrim
- Execute
- Execute

Deferred Context
- Display List
  - DrawPrim
  - DrawPrim

Deferred Context
- Display List
  - DrawPrim
  - DrawPrim
  - DrawPrim
Deferred Contexts

• Can create many deferred contexts
  – Each one is single threaded (thread unsafe)
• Deferred context generates a Display List
  – Display List is consumed by Immediate or Deferred contexts
• No read-backs or downloads from the GPU
  – Queries
  – Resource locking
• Lock with DISCARD is supported on deferred contexts
D3D11 on D3D10 H/W

• Deferred contexts are implemented at an API-capture level

• Async resource creation uses coarse sync primitives
  – No longer free threaded; thread safe though

• D3D10 drivers can be updated to better support D3D11 features

• Will work on Windows Vista as well as future Windows releases
Outline

• Overview
• Drilldown
  – Tessellation
  – Compute Shader
  – Multithreading
  – Dynamic Shader Linkage
  – Improved Texture Compression
  – Quick Glance at Other Features
• Availability
Shader Issues Today

• Shaders getting bigger, more complex
• Shaders need to target wide range of hardware
• Optimization of different shader configurations drives shader specialization
Options: Über-shader

Über-shader

foo (...) {
    if (m == 1) {
        // do material 1
    } else if (m == 2) {
        // do material 2
    }
    if (l == 1) {
        // do light model 1
    } else if (l == 2) {
        // do light model 2
    }
}
Options: Über-shader

“One Shader to Rule them All”

• Good:
  – All functionality in one place
  – Reduces state changes at runtime
  – One compile step
  – Seems to be most popular coding method

• Bad:
  – Complex, unorganized shaders
  – Register usage is always worst case path
Options: Specialization

Multiple specialized shaders for each combination of settings

• Good:
  – Always optimal register usage
  – Easier to target optimizations

• Bad:
  – Huge number of resulting shaders
  – Pain to manage at runtime
Combinatorial Explosion

Number of Lights

Number of Materials

Environmental Effects
Solution: Dynamic Shader
Linkage & OOP

- Introducing new OOP features to HLSL
  - Interfaces
  - Classes
- Can be used for static code
- Also used as the mechanism for linking specific functionality at runtime
Interfaces

```c
interface Light
{
    float3 GetDirection(float3 eye);

    float3 GetColor();
};
```
class DirectionalLight : Light
{
    float3 GetDirection(float3 eye)
    {
        return m_direction;
    }

    float3 GetColor()
    {
        return m_color;
    }

    float3 m_direction;
    float3 m_color;
};
Dynamic Shader Linkage

**Über-shader**

```cpp
foo (...) {
    if (m == 1) {
        // do material 1
    } else if (m == 2) {
        // do material 2
    }
    if (l == 1) {
        // do light model 1
    } else if (l == 2) {
        // do light model 2
    }
}
```

**Dynamic Subroutine**

```cpp
Material1(...) { ... }
Material2(...) { ... }
Light1(...) { ... }
Light2(...) { ... }

foo(...) {
    myMaterial.Evaluate(...);
    myLight.Evaluate(...);
}
```
In the Runtime

- Select specific class instances you want
- Runtime will inline class methods
  - Equivalent register usage to a specialized shader
- Inlining is done in the native assembly
  - Fast operation
- Applies to all subsequent Draw(...) calls
Outline

• Overview
• Drilldown
  – Tessellation
  – Compute Shader
  – Multithreading
  – Dynamic Shader Linkage
  Improved Texture Compression
  – Quick Glance at Other Features
• Availability
Why New Texture Formats?

• Existing block palette interpolations too simple
• Results often rife with blocking artifacts
• No high dynamic range (HDR) support
• NB: All are issues we heard from developers
Two New BC’s for Direct3D11

• BC6 (aka BC6H)
  – High dynamic range
  – 6:1 compression (16 bpc RGB)
  – Targeting high (not lossless) visual quality

• BC7
  – LDR with alpha
  – 3:1 compression for RGB or 4:1 for RGBA
  – High visual quality
New BC’s: Compression

• Block compression (unchanged)
  – Each block independent
  – Fixed compression ratios

• Multiple block types (new)
  – Tailored to different types of content
  – Smooth gradients vs. noisy normal maps
  – Varied alpha vs. constant alpha

Also new: decompression results must be bit-accurate with spec
Multiple Block Types

• Different numbers of color interpolation lines
  – Less variance in one block means:
    • 1 color line
    • Higher-precision endpoints
  – More variance in one block means:
    • 2 (BC6 & 7) or 3 (BC7 only) color lines
    • Lower-precision endpoints and interpolation bits

• Different numbers of index bits
  – 2 or 3 bits to express position on color line

• Alpha
  – Some blocks have implied 1.0 alpha
  – Others encode alpha
Partitions

• When using multiple color lines, each pixel needs to be associated with a color line
  – Individual bits to choose is expensive
• For a 4x4 block with 2 color lines
  – $2^{16}$ possible partition patterns
  – 16 to 64 well-chosen partition patterns give a good approximation of the full set
  – BC6H: 32 partitions
  – BC7: 64 partitions, shares first 32 with BC6H
Example Partition Table

A 32-partition table for 2 color lines

Shapes with corresponding indices

Region 0
Region 1
Comparisons

Orig BC3

Abs Error

Orig BC7
Comparisons

Orig | BC3
--- | ---

Orig | BC7
--- | ---

Abs Error
Comparisons

HDR Original at given exposure  Abs Error  BC6 at given exposure
Outline

• Overview

• Drilldown
  – Tessellation
  – Compute Shader
  – Multithreading
  – Dynamic Shader Linkage
  – Improved Texture Compression

Quick Glance at Other Features

• Availability
Lots of Other Features

- Addressable Stream Out
- Draw Indirect
- Pull-model attribute eval
- Improved Gather4
- Min-LOD texture clamps
- 16K texture limits
- Required 8-bit subtexel, submip filtering precision
- Conservative oDepth
- 2 GB Resources
- Geometry shader instance programming model
- Optional double support
- Read-only depth or stencil views
Outline

• Overview

• Drilldown
  – Tessellation
  – Compute Shader
  – Multithreading
  – Dynamic Shader Linkage
  – Improved Texture Compression
  – Quick Glance at Other Features

Availability
Questions?