Introduction to the Hardware Graphics Pipeline

Cyril Zeller

Overview of the Tutorial: Morning

8:30  Introduction to the Hardware Graphics Pipeline
     Cyril Zeller
9:30  Controlling the GPU from the CPU: the 3D API
     Cyril Zeller
10:15 Break
10:45 Programming the GPU: High-level Shading Languages
     Randy Fernando
12:00 Lunch

Overview of the Tutorial: Afternoon

12:00 Lunch
14:00 Optimizing the Graphics Pipeline
     Matthias Wloka
14:45 Advanced Rendering Techniques
     Matthias Wloka
15:45 Break
16:15 General-Purpose Computation Using Graphics Hardware
     Mark Harris
17:30 End

Overview

Concepts:
- Real-time rendering
- Hardware graphics pipeline

Evolution of the PC hardware graphics pipeline:
- 1995-1998: Texture mapping and z-buffer
- 1998: Multitexturing
- 1999-2000: Transform and lighting
- 2001: Programmable vertex shader
- 2002-2003: Programmable pixel shader
- 2004: Shader model 3.0 and 64-bit color support

PC graphics software architecture
Performance numbers

Real-Time Rendering

- Graphics hardware enables real-time rendering
- Real-time means display rate at more than 10 images per second

3D Scene = Collection of 3D primitives (triangles, lines, points)
Image = Array of pixels

Hardware Graphics Pipeline
Tutorial 5: Programming Graphics Hardware

PC Architecture


Texture Mapping

Texture Mapping: Texture Coordinates Interpolation

Texture Mapping: Perspective-Correct Interpolation

Texture Mapping: Magnification
Texture Mapping: Minification

Screen Space

Texture Space

Texture Mapping: Mipmapping

Bilinear Filtering

Trilinear Filtering

or

Texture Mapping: Anisotropic Filtering

Bilinear Filtering

Trilinear Filtering

or

Texture Mapping: Addressing Modes

Wrap

Mirror

Clamp

Border

Texture Mapping: Addressing Modes

Raster Operations Unit (ROP)

Texture Unit

Fragments

Stencil Buffer

Z-Buffer

Color Buffer

Alpha Test

Stencil Test

Z Test

Fragments

Alpha Blending

Raster Buffer
**1998: Multitexturing**

- **CPU**
  - Application/Geometry Stage

- **GPU**
  - Rasterization Stage
  - Multitexture Unit
  - Raster Operations Unit

**Video Memory**

**System Memory**

- AGP: Accelerated Graphics Port
- NVIDIA's TNT, ATI's Rage

**1999-2000: Transform and Lighting**

- **CPU**
  - Application Stage
  - Transform and Lighting Unit

- **GPU**
  - "Fixed Function Pipeline"

**Video Memory**

**System Memory**

**Transform and Lighting Unit (TnL)**

- **Transform**
  - Material Properties
  - Light Properties

- **Lighting**
  - Material Properties
  - Light Properties

**Bump Mapping**

- Bump mapping is about fetching the normal from a texture (called a normal map) instead of using the interpolated normal to compute lighting at a given pixel.

**AGP**

- PCI uses a parallel connection
- AGP uses a serial connection
- AGP uses a shared-bus protocol
- AGP uses a point-to-point protocol
- Bandwidth is not shared among devices
- AGP uses a dedicated system memory called AGP memory or non-local video memory
- The GPU can lookup textures that resides in AGP memory
- Bandwidth: AGP = 2 x PCI (AGP2x = 2 x AGP, etc.)
Cube Texture Mapping

Cube Texture Mapping (covering the six faces of a cube)

Environment Mapping (the reflection vector is used to lookup the cubemap)

Cubemap lookup (with direction \((x, y, z)\))

Projective Texture Mapping

Projective Texture

Projective Texture lookup

Volume Texture Mapping

Volume Texture (3D Noise)

Volume Texture lookup (with position \((x, y, z)\))

Noise Perturbation

Volume Texture Mapping

2001: Programmable Vertex Shader

CPU

Application Stage

512 Triangles

System Memory

Video Memory

GPU

Geometry Stage

Vertex Shader (per vertex)

Rasterization Stage

Register Combiner

Raster Operations Unit

Rasterizer (with Z-Cull)

Texture Shader

Video Memory

Frame Buffer

System Memory

3D Triangles

Textures

CPU

Application Stage

A programming processor for any per-vertex computation

Vertex Shader

void VertexShader(

// Input per vertex

in float4 positionInModelSpace,

in float2 textureCoordinates,

in float3 normal,

// Input per batch of triangles

uniform float4x4 modelToProjection,

uniform float3 lightDirection,

// Output per vertex

out float4 positionInProjectionSpace,

out float2 textureCoordinatesOutput,

out float3 color

)

// Vertex transformation

positionInProjectionSpace = mul(modelToProjection, positionInModelSpace);

// Texture coordinates copy

textureCoordinatesOutput = textureCoordinates;

// Vertex color computation

color = dot(lightDirection, normal);

Hardware Shadow Mapping

Shadow Map Computation

The shadow map contains the depth \(z/w\) of the 3D points visible from the light's point of view:

\[ z/w < \text{value of shadow map at } (x/w, y/w) \]

A hardware shadow map lookup returns the value of this comparison between 0 and 1

Shadow Rendering

A 3D point \((x, y, z, w)\) is in shadow if:

\[ z/w < \text{value of shadow map at } (x/w, y/w) \]

A hardware shadow map lookup returns the value of this comparison between 0 and 1
Antialiasing: Definition

- **Aliasing**: Undesirable visual artifacts due to insufficient sampling of:
  - Primitives (triangles, lines, etc.) → jagged edges
  - Textures or shaders → pixelation, moiré patterns

Those artifacts are even more noticeable on animated images.

- **Antialiasing**: Method to reduce aliasing
  - Texture antialiasing is largely handled by proper mipmapping and anisotropic filtering
  - Shader antialiasing can be tricky (especially with conditionals)

Antialiasing: Supersampling and Multisampling

- **Supersampling**: Compute color and Z at higher resolution and display averaged color to smooth out the visual artifacts
- **Multisampling**: Same thing except only Z is computed at higher resolution

2002-2003: Programmable Pixel Shader

- **MRT**: Multiple Render Target
- **NVIDIA’s GeForce FX, ATI’s Radeon 9600 to 9800**

Pixel Shader

- **A programming processor for any per-pixel computation**

Shader: Static vs. Dynamic Flow Control

- **Static Flow Control**: (condition varies per batch of triangles)
- **Dynamic Flow Control**: (condition varies per vertex or pixel)

2004: Shader Model 3.0 and 64-Bit Color Support

- **PCIe**: Peripheral Component Interconnect Express
- **NVIDIA’s GeForce 6800**
### PCIe

- **Like AGP:**
  - Uses a *serial* connection → Cheap, scalable
  - Uses a *point-to-point* protocol → No shared bandwidth

- **Unlike AGP:**
  - General-purpose (not only for graphics)
  - Dual-channels: Bandwidth is available in both direction

- Bandwidth: PCIe = 2 x AGPx8

### Shader Model 3.0

- **Shader Model 3.0 means:**
  - Longer shaders → More complex shading
  - Pixel shader:
    - Dynamic flow control → Better performance
    - Derivative instructions → Shader antialiasing
    - Support for 32-bit floating-point precision → Fewer artifacts
    - Face register → Faster two-sided lighting
  - Vertex shader:
    - Texture access → Simulation on GPU, displacement mapping

### 64-Bit Color Support

- **64-bit color** means one 16-bit floating-point value per channel (R, G, B, A)

- Alpha blending works with 64-bit color buffer
  (as opposed to 32-bit fixed-point color buffer only)

- Texture filtering works with 64-bit textures
  (as opposed to 32-bit fixed-point textures only)

- **Applications:**
  - High-precision image compositing
  - High dynamic range imagery

### High Dynamic Range Imagery

- **The dynamic range** of a scene is the ratio of the highest to the lowest luminance

- Real-life scenes can have high dynamic ranges of several millions

- Display and print devices have a low dynamic range of around 100

- **Tone mapping** is the process of displaying high dynamic range images on those low dynamic range devices

- High dynamic range images use *floating-point colors*

- **OpenEXR** is a high dynamic range image format that is compatible with NVIDIA’s 64-bit color format

### Real-Time Tone Mapping

- The image is entirely computed in 64-bit color and tone-mapped for display

From low to high exposure image of the same scene

### PC Graphics Software Architecture

- The application, 3D API and driver are written in C or C++
- The vertex and pixel programs are written in a high-level shading language (Cg, DirectX HLSL, OpenGL Shading Language)
- Pushbuffer: Contains the commands to be executed on the GPU
The Future

- Unified general programming model at primitive, vertex and pixel levels
- Scary amounts of:
  - Floating point horsepower
  - Video memory
  - Bandwidth between system and video memory
- Lower chip costs and power requirements to make 3D graphics hardware ubiquitous:
  - Automotive (gaming, navigation, heads-up displays)
  - Home (remotes, media center, automation)
  - Mobile (PDAs, cell phones)

More Information

- www.realtimerendering.com
- developer.nvidia.com
- Questions: czeller@nvidia.com