Modern Graphics Hardware

MIT EECS 6.837
Jonathan Ragan-Kelley, Frédo Durand
Slides and demos from
Hanrahan & Akeley, Gary McTaggart, NVIDIA, ATI

MIT EECS 6.837, Durand
Modern graphics hardware

Hardware implementation of the rendering pipeline

Programmability ("shaders") within specific stages of the pipeline

- Recent, last eight years
- At the vertex and pixel level

// bowling pin, based on RenderMan example
(CIRCLE over (BRUNS over BASE)) * MARKS * Cd + Cs
First Generation - Wireframe

**Vertex:** transform, clip, and project

**Rasterization:** color interpolation (points, lines)

**Fragment:** overwrite

**Dates:** prior to 1987
Second Generation - Shaded Solids

Vertex: lighting calculation

**Rasterization:** depth interpolation (triangles)

**Fragment:** depth buffer, color blending

**Dates:** 1987-1992
Third Generation - Texture Mapping

**Vertex:**
texture coordinate transformation

**Rasterization:**
texture coordinate coordinate interpolation

**Fragment:**
texture evaluation, antialiasing

**Dates:**
1992-2000
Fourth Generation - Programmable

**Vertex:** programmable transformation

**Rasterization:** user attribute interpolation

**Fragment:** programmable color computation

**Dates:** 2000-2009
Fifth Generation - Programmable

Vertex, Rasterization, Fragment, ???:
  arbitrary parallel programs

Dates:  2009+
Questions?
Geometry and Pixel (fragment) stage become programmable

– Elaborate appearance
– More and more general-purpose computation (GPGPU)
Vertex Shaders

Vertex Shaders are both Flexible and Quick

vertex shaders can be used to move/animate vertices

Linear Interpretation of vertex lighting values

Slide from NVidia
Vertex Shader: Blendshapes

- 50 face geometries
  - angry, happy, sad, move eyebrow,…
- Each target stored as difference vector
  - For each vertex: average position + 50 differences
- **Result is a weighted sum of all targets**
  - We only transmit the weights, the targets remain in graphics memory
  - Big multiply-add
    - Per *active* blend target
    - Per attribute
Job #2 for vertex shaders

• Prepare data for pixel shaders (e.g. texture coordinates)
  – Stored/computed at vertices
  – Interpolated per pixel

• Modern graphics hardware provides tons of interpolants (32x4 floats)
Pixel Shaders

Each pixel is calculated individually

Pixel shaders have limited or no knowledge of neighbouring pixels
Brushed Metal

- Procedural texture
- Anisotropic lighting
Melting Ice

- Procedural, animating texture
- Bumped environment map
Toon & Fur

Toon shading

Volumetric fur
Vegetation & Thin Film

Translucence
Backlighting

Custom lighting
Simulates iridescence
Allows for amazing quality
Earth, 4.5x10⁹ B.C.E.  Crysis, 2007
Rich scene appearance

• Vertex shader
  – Geometry (skinning, displacement)
  – Setup interpolants for pixel shaders

• Pixel shader
  – Visual appearance
  – Also used for image processing, GPGPU abuses

• Multipass
  – Render the scene or part of the geometry multiple times
  – Different viewpoints (e.g. shadow map, shadow volume)
  – More complex shading (e.g. pass per-light)
  – Image processing (e.g. HDR tone mapping, bloom)
Multipass: Shadow Mapping

- Texture mapping with depth information
- Requires 2 passes through the pipeline:
  - Compute shadow map (depth from light source)
  - Render final image, check shadow map to see if points are in shadow

Shadow Map Look Up

- We have a 3D point \((x, y, z)_{WS}\)
- How do we look up the depth from the shadow map?
- Use the 4x4 perspective projection matrix from the light source to get \((x', y', z')_{LS}\)
- \(\text{ShadowMap}(x', y') < z'\)

Programming

• Pass 1:
  – Setup GL state, setup viewpoint as light source
  – Tell OpenGL to render geometry
  – Store depth of result as texture

• Pass 2:
  – Setup GL state, setup viewpoint as eye
  – Set active shaders
    • Vertex shader computes light-space coordinates
    • Pixel shader performs lookup in shadow map
  – Tell OpenGL to render geometry

• Note: the CPU is in control of the pass graph
Shadow Volumes

Shadowed scene

Stencil buffer contents

green  = stencil value of 0
red = stencil value of 1
darker reds = stencil value > 1
Shadow Volumes vs. Shadow Maps

• Shadow mapping via projective texturing
  – The other prominent hardware-accelerated shadow technique

• Shadow mapping advantages
  – Requires no explicit knowledge of object geometry
  – No 2-manifold requirements, etc.
  – View independent

• Shadow mapping disadvantages
  – Sampling artifacts
  – Not omni-directional
Questions?
How to program shaders?

• Assembly code
• Higher-level language and compiler (e.g. Cg, HLSL, GLSLang)
• Send to the GPU like any piece of geometry
• (Heavily) optimized by the driver’s compiler
What does a shader look like?

**Assembly**

```
... RSQR R0.x, R0.x;
MULR R0.xyz, R0.xxxx, R4.xyz;
MOV R5.xyz, -R0.xyz;
MOV R3.xyz, -R3.xyz;
DP3R R3.x, R0.xyz, R3.xyz;
SLTR R4.x, R3.x, {0.000000}.x;
ADDR R3.x, {1.000000}.x, -R4.x;
MULR R3.xyz, R3.xxxx, R5.xyz;
MULR R0.xyz, R0.xxxx, R4.xxxx;
ADDR R0.xyz, R0.xxxx, R3.xxxx;
DP3R R1.x, R0.xyz, R1.xyz;
MAXR R1.x, {0.000000}.x, R1.x;
LG2R R1.x, R1.x;
MULR R1.x, {10.000000}.x, R1.x;
EX2R R1.x, R1.x;
MOV R1.xyz, R1.xxxx;
```

**Cg**

```
... COLOR cSpec = pow(max(0, dot(Nf, H)), phongExp).xxx;
COLOR cPlastic = Cd * (cAmbi + cDiff) + Cs * cSpec;
```

*Simple phong shader expressed in both assembly and Cg*
Cg Summary

- C-like language – expressive and efficient
- Hardware data types
- Vector and matrix operations
- Write separate vertex and fragment programs
- Connectors enable mix & match of programs by defining data flows
- Will be supported on any DX9 hardware
- Will support future HW (beyond NV30/DX9)
Questions?
General-purpose computation on GPUs

- Dense data-parallel processor (Tera-FLOPS)
- Increasingly programmable (Code executed for each vertex or each pixel)
- Use “shaders” for general-purpose computation
- Peak performance can be difficult to reach
- Parallelism = hard

Navier-Stokes on GPU [Bolz et al.]
Questions?
Graphics Hardware

• High performance through:
  – Parallelism
  – Specialization
  – Limited synchronization (no data dependency)
  – Efficient latency hiding (pre-fetching, threading)
Modern Graphics Hardware

• a.k.a. Graphics Processing Units (GPUs)

• Programmable geometry and fragment stages
• 600 million vertices/second, 6 billion texels/second
• Tera-operations/second
• Floating point operations only
• Very little (traditional) cache
Modern Graphics Hardware

- 100s of parallel shader pipelines
- Deep pipeline (~1k stages)
- Hierarchical tiling of screen (~4x4)
  - Early Z rejection if entire tile is occluded
- Pixels rasterized by quads (2x2 pixel groups)
  - Allows for derivatives
- Massive multithreading to hide texture latency
  - And smart memory layout
Why is it so fast?

• All transistors do computation, little cache
• Parallelism
• Specialization (rasterizer, texture filtering)
• Arithmetic intensity
• Deep pipeline, latency hiding, prefetching
• Little data dependency
• In general, memory-access patterns
Questions?
Traditional hardware architecture

- **Vertex units**: 
  - G
  - P
  - R
- **Fragment units**: 
  - T
  - F
  - P
- **Raster operation units**: 
  - D

- **One big parallel rasterizer**

- **Mipmap filtering**
  - Tex

- **Cross-bar**

- **Z-buffer, framebuffer**
  - Screen-locked
Unified shading architecture (NVIDIA G80)

- 128 SPs
- 3 ALUs/SP
- 1.5ghz
- $\approx$ 576 GFLOPs

- ~4k+ threads
- 85 GB/sec
Questions?
Bottlenecks?

- The bottleneck determines overall throughput
- In general, the bottleneck varies over the course of an application and even over a frame
- Getting good performance is all about finding and eliminating bottlenecks in the pipeline
Potential Bottlenecks

On-Chip Cache Memory
- Vertex Shading (T&L)
  - Pre-TnL cache
  - Post-TnL cache
- Triangle Setup
- Rasterization
- Fragment Shading and Raster Operations

Video Memory
- Geometry
- Command s
- Textures
- Frame Buffer

System Memory
- CPU

AGP transfer limited
CPU limited
Texture b/w limited
Frame buffer b/w limited
Vertex transformation limited
Setup limited
Raster limited
Fragment shader limited
Rendering pipeline bottlenecks

• The term “transform/vertex/geometry bound” often means the bottleneck is “anywhere before the rasterizer”

• The term “fill/raster bound” often means the bottleneck is “anywhere after setup for rasterization” (computation of edge equations)

• Can be both transform and fill bound over the course of a single frame!
Questions?
Shader zoo
Layering

// bowling pin, based on RenderMan example
(CIRCLE over (BRUNS over BASE)) * MARKS * Cd + Cs
From Half Life 2 (Valve)

Desired Image

Slide by Gary McTaggart (Valve)
Radiosity
Normal
Normal Mapped Radiosity
Albedo
Albedo * Normal Mapped Radiosity
Radiosity Normal Mapping Shade Tree

Slide by Gary McTaggart (Valve)
Cube Map Specular
Normal Mapped Specular
Specular Factor
Normal Mapped Specular
Normal Mapped Specular * Specular Factor
Final Result
Radiosity Normal Mapping Shade Tree

Cube Map
Specular Factor
Normal
Lightmaps
Albedo

Slide by Gary McTaggart (Valve)
Radiosity
Normal Map
Albedo
Albedo * Normal Mapped Radiosity
Normal Mapped Specular
Specular Factor
Normal Mapped Specular * Specular Factor

Slide by Gary McTaggart (Valve)
Albedo * Normal Mapped Radiosity
Final Result
Refraction mapping (multipass)
Image processing

- Start with ordinary model
  - Render to backbuffer
- Render parts that are the sources of glow
  - Render to offscreen texture
- Blur the texture
- Add blur to the scene
More glow

- From “Tron”
Shadows in a Real Game Scene

Abducted game images courtesy Joe Riedel at Contraband Entertainment
Scene’s *Visible* Geometric Complexity

Wireframe shows geometric complexity of visible geometry
Blow-up of Shadow Detail

Notice cable shadows on player model

Notice player’s own shadow on floor
Scene’s Shadow Volume Geometric Complexity

Wireframe shows geometric complexity of shadow volume geometry

Shadow volume geometry projects away from the light source
Visible Geometry vs. Shadow Volume Geometry

Typically, shadow volumes generate considerably more pixel updates than visible geometry.
Other Example Scenes (1 of 2)

Dramatic chase scene with shadows

*Abducted* game images courtesy Joe Riedel at Contraband Entertainment

MIT EECS 6.837, Durand
Situations when Shadow Volumes are too expensive

Chain-link fence's shadow appears on truck & ground with shadow maps

Chain-link fence is shadow volume nightmare!

Chain-link fence's shadow appears on truck & ground with *shadow maps*

*Fuel game image courtesy Nathan d’Obrenan at Firetoad Software*
• http://www.graphics.stanford.edu/courses/cs448a-01-fall/
• http://www.ati.com/developer/techpapers.html
Hardware Shading for Artists

Anisotropic Hair Shafts

Reflective Catchlights

Expressive Blendshape Animation

Colored Translucence

Skin Oil Variations

Goosebumps

Subsurface Blood Layers

Robust Skeletal Animation

MIT EECS 6.837, Durand

Slide from NVidia