Modern graphics hardware

- Hardware implementation of the rendering pipeline
- Programmability & “shaders”
  - Recent, last few years
  - At the vertex and pixel level

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 interpolation
- **Fragment:** texture evaluation, antialiasing
- **Dates:** 1992 - 2000

Fourth Generation - Programmability

- Programmable shading
- Image-based rendering
- Convergence of graphics and media processing
- Curved surfaces
### Questions?

### Modern Graphics Pipeline

**Application**
- Simulation
- Input event handlers
- Modify data structures
- Database traversal
- Primitive generation
- Utility functions

**Command**
- Command buffering
- Command interpretation
- Unpack and perform format conversion

```
Maintain graphics state
    glLoadIdentity();
    glClearIGl( GL_TRIANGLES, GL_FRONT );
    glColor3f( 0.0, 0.5, 0.0 );
    glTranslatef( 1.0, 0.0, 0.0 );
    glColor3f( 0.0, 0.5, 0.0 );
    g1Scale3f( 0.5, 0.0, 0.0 );
    glColor3f( 0.0, 0.5, 0.0 );
    glTranslatef( 1.0, 0.0, 0.0 );
    glColor3f( 0.5, 0.0, 0.0 );
    glLoadIdentity();
```

**Geometry**
- Evaluation of polynomials for curved surfaces
- Transform and projection
- Clipping, culling and primitive assembly

### Geometry

- Evaluation of polynomials for curved surfaces
- Transform and projection (object -> image space)
- Clipping, culling and primitive assembly
- Lighting (light sources and surface reflection)

- Texture coordinate generation
- Object-space triangles
- Screen-space lit triangles
### Rasterization

- Setup (per-triangle)
- Sampling (triangle = [fragments])
- Interpolation (interpolate colors and coordinates)

### Texture

- Texture transformation and projection
- Texture address calculation
- Texture filtering

### Fragment

- Texture combiners
- Texture Fragments
- Textured Fragments

### Display

- Gamma correction
- Analog to digital conversion

### Questions?
Programmable Graphics Hardware

- Geometry and pixel (fragment) stage become programmable
  - Elaborate appearance
  - More and more general-purpose computation (GPU hacking)

Vertex Shaders

- Vertex Shaders are both Flexible and Quick
  - Linear interpretation of vertex lighting values
  - Can be used to move/animate vertices

Pixel Shaders

- Pixel shaders have limited or no knowledge of neighbouring pixels
- Each pixel is calculated individually

Rich scene appearance

- Vertex shader
  - Geometry (skinning, displacement)
  - Setup interpolants for pixel shaders
- Pixel shader
  - Visual appearance
  - Also used for image processing and other GPU abuses
- Multipass
  - Render the scene or part of the geometry multiple times
  - E.g. shadow map, shadow volume
  - But also to get more complex shaders

How to program shaders?

- Assembly code
- Higher-level language and compiler (e.g. Cg, HLSL, GLSL)
- Send to the card like any piece of geometry
- Is usually modified/optimized by the driver
- We won’t talk here about other dirty driver tricks
What Does Cg look like?

**Assembly**

```
DOR R0 x, R1 x;
MULR R0 x, R0 xxx, R1 xyzz;
ADD R0 x, R0 x, 1.000000;
MULR R3 x, R3 yyyy, R0 xxzz;
SLD R4, R0 x, 0.000000 x;
ADD R4 x, R3 x, 0.000000 x;
MULR R2 x, R1 x, R2 yyyy;
ADD R2 x, R2 x, R2 yyyy;
MULR R0 x, R0 x, R2 yyyy;
MULR R0 x, R0 x, R3 yyyy;
ADD R0 x, R0 x, R3 yyyy;
MULR R0 x, R0 x, R3 yyyy;
ADD R0 x, R0 x, R2 yyyy;
MULR R0 x, R0 x, R2 yyyy;
```

**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
- HW 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)

Brushed Metal

- Procedural texture
- Anisotropic lighting

Melting Ice

- Procedural, animating texture
- Bumped environment map

Toon & Fur

Toon rendering without textures
Antialiasing
Great silhouettes without overdarkening

Vegetation & Thin Film

Translucence
Backlighting
Example of custom lighting
Simulates iridescence
General Purpose-computation on GPUs

• Hundreds of Gigaflops
  – Moore’s law cubed

• Becomes programmable
  – Code executed for each vertex or each pixel

• Use for general-purpose computation
  – But tedious, low level, hacky

• Performances not always as good as hoped for

Navier-Stokes on GPU [Bolz et al.]

Questions?

Graphics Hardware

• High performance through
  – Parallelism
  – Specialization
  – No data dependency
  – Efficient pre-fetching

Modern Graphics Hardware

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

• Programmable geometry and fragment stages

• 600 million vertices/second, 6 billion texels/second

• In the range of tera operations/second

• Floating point operations only

• Very little cache

Modern Graphics Hardware

• About 4-6 geometry units

• About 16 fragment units

• Deep pipeline (~800 stages)

• Tiling of screen (about 4x4)
  – Early z-rejection if entire tile is occluded

• Pixels rasterized by quads (2x2 pixels)
  – Allows for derivatives

• Very efficient texture pre-fetching
  – 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?

Architecture

6 vertex units
One big parallel rasterizer
16 texture units 
mipmap filtering
16 fragment units 
Cross-bar
16 raster operation units
z buffer, transbuffer
Screen-locked

Vertex shading unit (ATI X800)
- One 128-bit vector ALU and one 32-bit scalar ALU.
- Total of 12 instructions per clock
- 28 GFLOPs for the six units

Pixel shading unit (ATI X800)
- Two vector ALU & two scalar ALUs + texture addressing unit.
- Up to five floating-point instructions per cycle
- In total (16 units) 80 floating-point ops per clock, or 41.6 GFLOPs/sec from the pixel shaders alone.

Questions?
Bottlenecks?

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

Potential Bottlenecks

- 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
From Half Life 2 (Valve)

Radiosity Normal Mapping Shade Tree

Desired Image

Slide by Gary McTaggart (Valve)

Radiosity

Slide by Gary McTaggart (Valve)

Normal

Slide by Gary McTaggart (Valve)

Normal Mapped Radiosity

Slide by Gary McTaggart (Valve)

Albedo

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

Radiosity Normal Mapping Shade Tree

Cube Map Specular

Normal Mapped Specular

Specular Factor

Normal Mapped Specular
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”

Vertex Shader: Blendshapes (1/2)
- Collected from Maya “Blendshape” node
- 50 faces
  - 30 emotion faces (angry, happy, sad…)
  - 20 modifiers (left eyebrow up, right smirk …)
- Each target stored as difference vector
- A blendshape is a single multiply-add
  - Per active blend target
  - Per attribute
  - Result is a weighted sum of all active targets
- An active blendshape takes vertex attributes
  - 12 * (coordinate)
  - 6 * (coordinate + normal)
  - 4 * (coordinate + normal + tangent)

Shadow Volumes

Shadows in a Real Game Scene
Scene’s *Visible* Geometric Complexity

Wireframe shows geometric complexity of visible geometry

Primary light source location

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

Visible geometry

Shadow volume geometry

Abducted game images courtesy Joe Riedel at Contraband Entertainment

Situations When Shadow Volumes Are Too Expensive

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

Fuel game image courtesy Nathan d’Obrenan at Firetoad Software
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

• http://www.graphics.stanford.edu/courses/cs448a-01-fall/
• http://www.ati.com/developer/techpapers.html
• http://developer.nvidia.com/page/documentation.html
• http://download.nvidia.com/developer/SDK/Individual_Samples/samples.html
• http://developer.nvidia.com/page/tools.html

Hardware Shading for Artists

Computational Requirements

<table>
<thead>
<tr>
<th>Application</th>
<th>Per-Vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>ADD CMP MUL DIV SPE</td>
</tr>
<tr>
<td>Geometry</td>
<td>102 30 108 5 1</td>
</tr>
<tr>
<td>Rasterization</td>
<td>Per-Fragment</td>
</tr>
<tr>
<td>Text</td>
<td>ADD CMP MUL DIV SPE</td>
</tr>
<tr>
<td>Fragment</td>
<td>66 9 70 1 3</td>
</tr>
<tr>
<td>Display</td>
<td>Rough estimate</td>
</tr>
</tbody>
</table>

Slide from NVidia