Graphics Hardware

Instructor – Stephen J. Guy
Overview

- What is a GPU
- Evolution of GPU
- GPU Design
- Modern Features
  - Programmability!
- Programming Examples
Overview

- What is a GPU
- Evolution of GPU
- GPU Design
- Modern Features
  - Programmability!
- Programming Examples
What’s a GPU?

GPU = Graphics Processing Unit

CPU = Central Processing Unit

GPU : dedicated graphics rendering device

A GPU implements a number of graphics primitive operations in a way that makes running them much faster than drawing directly to the screen with the host CPU.
A simple graphics system

- Frame buffer can be part of the main memory

Similar layout to one in rasterization assignment

Problem?
Dedicated memory

- Video memory
  - On-board frame buffer
  - Much faster access

Can this be improved still?
Graphics Accelerator

- A dedicated processor for graphics processing

Diagram:
- CPU
- Main Memory
- System bus
- Graphics Memory/Frame buffer
- Graphics Processor
- Scan Controller

11/22/2010 Graphics Hardware
GPUs!

[Image of NVIDIA and ATI logos and graphics cards]
The GPU

- Traditional Requirements (until early 90s)
  - Drive display
  - Simple video decoding
- New developments
  - Feature sizes and pipeline
  - 3D Graphics Computations
    - Transformation and lighting
    - Per-pixel effects
  - High-definition TV
3D Graphics Card: ATI Radeon
### The GPU

**What does the GPU do?**

<table>
<thead>
<tr>
<th>Application tasks (move objects according to application, move/aim camera)</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene level calculations (object level culling, select detail level, create object mesh)</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>Transform</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td><strong>GPU</strong></td>
</tr>
<tr>
<td>Lighting</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td><strong>GPU</strong></td>
</tr>
<tr>
<td>Triangle Setup and Clipping</td>
<td><strong>CPU</strong></td>
<td><strong>Graphics Processor</strong></td>
<td><strong>Graphics Processor</strong></td>
<td><strong>GPU</strong></td>
</tr>
<tr>
<td>Rendering</td>
<td><strong>Graphics Processor</strong></td>
<td><strong>Graphics Processor</strong></td>
<td><strong>Graphics Processor</strong></td>
<td><strong>GPU</strong></td>
</tr>
</tbody>
</table>

| --- | --- | --- | --- |
Graphics Accelerator

- A dedicated processor for graphics processing

Where’s the bottleneck?
Graphics Bus Interface

- PCI based technology

![Diagram showing PCI bus interface with CPU, Main Memory, Graphics Memory, Scan Controller, Graphics Processor, Other Peripherals, System Bus, and PCI Bus connections.]
Graphics Bus Interface

- PCI Bus becomes the bottleneck!
  - Many devices are using it
  - There is a lot of stuff needs to be transmitted from main memory to graphics memory (geometry, textures, etc)
  - Example: 2M triangle, 90 Bytes each – 180MB > 132 MB (PCI bandwidth)
Accelerated Graphics Port (AGP)

- A dedicated bus that allows direct access of main memory

AGP 1x: 518 MB/s

Fast!!!

Graphics Memory/Frame buffer
Graphics Processor
Scan Controller

Other peripherals

PCI Bus – 132 MB/s

CPU

Main Memory
AGP

- AGP 1x is four times as fast compared to PCI!
  - Peaked at APG 8x
- No more local bus congestion!
- More geometry can be processed!
- Direct execution of many graphics operations from main memory
PCI Express

- Provide very high bandwidth to CPU
  - Standard bus for GPUs as of 2009
  - Throughput varies with PCIe version
Evolution of Performance

<table>
<thead>
<tr>
<th>Year</th>
<th>Bus</th>
<th>Memory</th>
<th>DirectX</th>
<th>OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>PCI</td>
<td>4 MB</td>
<td>DirectX 1</td>
<td>OpenGL 1.1</td>
</tr>
<tr>
<td>1996</td>
<td>AGP</td>
<td>32 MB</td>
<td>DirectX 2</td>
<td>OpenGL 1.2</td>
</tr>
<tr>
<td>1997</td>
<td>AGP2x</td>
<td>64 MB</td>
<td>DirectX 3</td>
<td>OpenGL 1.3</td>
</tr>
<tr>
<td>1998</td>
<td>AGP4x</td>
<td>128 MB</td>
<td>DirectX 5</td>
<td>OpenGL 1.4</td>
</tr>
<tr>
<td>1999</td>
<td>AGP8x</td>
<td>256 MB</td>
<td>DirectX 6</td>
<td>OpenGL 1.5</td>
</tr>
<tr>
<td>2000</td>
<td>PCIe</td>
<td>512 MB</td>
<td>DirectX 7</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td></td>
<td>DirectX 8</td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td></td>
<td></td>
<td>OpenGL 1.3</td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td></td>
<td></td>
<td>DirectX 9</td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
<td></td>
<td>OpenGL 1.5</td>
<td></td>
</tr>
</tbody>
</table>
Overview

- What is a GPU
- Evolution of GPU
- **GPU Design**
- Modern Features
  - Programmability!
- Programming Examples
Recall the Generic Graphics Pipeline

3D Geometric Primitives

- Transform in 3D word coordinate system
- Illuminate according to lighting and reflectance
- Transform into 3D camera coordinate system
- Transform into 2D screen coordinate system
- Clip primitives outside camera’s view
- Draw Pixels (also texturing, hidden surface, ...)

Image
GPU pipeline

Program/API

Driver

GPU Front End

CPU

GPU

Bus

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer

(example from Mark Colbert)
GPU pipeline

- Program
  - Your Program
- API
  - Either OpenGL or DirectX Interface

--------

- Driver
  - Black-box
    - Implementations are Company Secrets
  - Largest Bottleneck in many GPU programs
GPU pipeline

- GPU Front End
  - Receives commands & data from driver
  - PCI Express helps at this stage

- Vertex Processing
  - Normally performs 3D→2D transformations
  - Programmable
GPU pipeline

- **Primitive Assembly**
  - Compiles Vertices into Points, Lines and/or Polygons
  - Link elements and set rasterizer

- **Rasterization**
  - Computes fragments (pixels)
    - Barycentric Coordinates
    - Depth, etc…
  - **Interpolation** (texture coordinates and colors)

- **Fragment Processing**
  - Per-pixel computations (e.g. lighting)
  - Programmable
GPU pipeline

- Depth Checking
  - Check framebuffer to see if lesser depth already exists (Z-Buffer)
  - Limited Programmability

- Blending
  - Use alpha channel to combine colors already in the framebuffer
  - Limited Programmability
GPU pipeline

Program/API

Driver

CPU

GPU

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer

(excerpt from Mark Colbert)
Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

Code Snippet

```c
....
glBegin(GL_TRIANGLES);
glTexCoord2f(1,0); glVertex3f(0,1,0);
glTexCoord2f(0,1); glVertex3f(-1,-1,0);
glTexCoord2f(0,0); glVertex3f(1,-1,0);
glEnd();
...
```
Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

[example from Mark Colbert]
Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

viewing frustum

[example from Mark Colbert]
example from Mark Colbert
Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

Framebuffer

[example from Mark Colbert]
Overview

- What is a GPU
- Evolution of GPU
- GPU Design
- Modern Features
  - Programmability!
- Programming Examples
Programmable GPU

Why do we want programmability?

<table>
<thead>
<tr>
<th>Application tasks (move objects according to application, move/aim camera)</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene level calculations (object level culling, select detail level, create object mesh)</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>Transform</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>Lighting</td>
<td>CPU</td>
<td>CPU</td>
<td>GPU</td>
<td></td>
</tr>
<tr>
<td>Triangle Setup and Clipping</td>
<td>CPU</td>
<td>Graphics Processor</td>
<td>Graphics Processor</td>
<td>GPU</td>
</tr>
</tbody>
</table>

Shaders

- Small programs that run on GPU
  - Definition of “small” evolves quickly
- Two main types:
  - Vertex Shaders
    - Input: Vertex Positions
    - Output: New Vertex Positions
  - Pixel /Fragment Shaders
    - Input: Vertex Colors, Normals, etc
    - Output: Pixel Color
- A quickly evolving field
  - Geometry Shaders slowly becoming common
    - Input: Mesh/Vertex Positions
    - Output: New Mesh/Vertex Positions
GPU pipeline

1. Program/API
2. Driver
3. GPU Front End
4. GPU
5. CPU
6. Bus
7. Vertex Processing
8. Primitive Assembly
9. Rasterization & Interpolation
10. Fragment Processing
11. Raster Operations
12. Framebuffer

[example from Mark Colbert]
GPU pipeline

- Vertex Processing
  - Normally performs transformations
  - Programmable

Vertex Processing

- Vertex
  - for interpolation
  - for rasterization

- shader
  - POSITION
  - PSIZE
  - FOG
  - TEXCOORD[0-7]
  - COLOR[0-1]

- textures

- vertex
  - POSITION,
  - NORMAL,
  - BINORMAL*,
  - TANGENT*,
  - TEXCOORD[0-7],
  - COLOR[0-1],
  - PSIZE
GPU pipeline

- Fragment Processing
  - Programmable
Programming Shaders

- Can be written in assembly language for specific GPU
- High level languages exist
  - Better portability
  - Slightly slower
  - Much easier to code
- Most are small variants on C
  - Cg – C for Graphics (DirectX and OpenGL)
  - HLSL – High Level Shader Language (DirectX)
  - GLSL – OpenGL Shader Language (OpenGL)
Language targeted specifically at graphics programming

Vectors built in:

i.e. float2, float3, float4

float4 v = float4(a,b,c,d);

v[0], v[1], v[2], or v[3]

v.xyzw, v.xxxx, v.wyzx

v.rgba, ....
Matrix built in…
- e.g. float4x4
- float4 v = float4(a,b,c,d); //Matrix is 4 vectors
  float4x4 m = float4x4(v,v,v,v);

Linear algebra built in…
- dot product, matrix-vector multiplication

Geometric processing…
- distance, length, normal, refact

Most of math.h
- Optimized version (2nd order talyer approximation)
Vertex Shaders

- Example Time!

- Cg Vertex Shader
Vertex Shaders

- Commonly used for
  - Environmental effects
  - Complex character animation
  - Procedural deformation
  - Motion blur
  - Custom Lighting
Pixel Shaders

- Example Time!

- Cg Pixel Shader
What Pixel Shaders Can Do

- Bump Mapping
- Per-pixel lighting
- Cartoon rendering
- Isotropic BRDF (Bi-Directional Reflective Distribution Function) Based Lighting
- GPU ray-tracing
More Shaders

- Parametrically specify complex textures

Stages of a Procedural Brick Shader
Complex Examples

- Reflective bump-mapping
Far-cry Screen Shot
GPGPU

- How does GPU growth compare to GPU?
Why are GPUs so fast!?

- Hardware is designed to exploit parallelism!
- Same exact shader running over millions of vertexies or pixels
  - Known as SIMD (Single Instruction Multiple Data)
  - If-statements diminish effectiveness
- Best performing Shaders are aware of this.
Modern Graphics Hardware

- Parallelism + pipelining: ATI Radeon 9700

- 4 vertex pipelines
- 8 pixel pipelines
GPGPU – General Purpose GPU

- Program graphics card for non-graphics
- GPGPU programs must be aware of SIMD
- nVidia has popular language specifically for this
  - CUDA
  - nvidia.com/cuda for more details
More GPGPU Links

- nVidia

- ATI
  - http://www.ati.com/developer/

- General Purpose GPU Programming
  - http://www.gpgpu.org
Review

- What’s Video Memory?