Graphics Hardware
Overview

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

- **What is a GPU**
- **Evolution of GPU**
- **GPU Design**
- **Modern Features**
  - Programmability!
Graphics Processing Unit

- 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
Big Name GPU

Graphics Hardware
Less Know, Equally Important

Imagination

POWERVR

Intel

Graphics Hardware  11/22/2010
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: PowerVR SGX5
## The GPU

### What does the GPU do?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application tasks (move objects according to application, move/aim camera)</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<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>CPU</td>
<td>GPU</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>
Graphics Accelerator

- A dedicated processor for graphics processing

Where’s the bottleneck?

Graphics Hardware
Graphics Bus Interface

- PCI based technology

![Diagram of Graphics Bus Interface]

- Graphics Memory/Frame buffer
- Graphics Processor
- Scan Controller
- Other Peripherals
- PCI Bus – 132 MB/s
- System Bus – 800MB/s

PCI based technology

Graphics Hardware
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

![Diagram showing components of AGP and their connections]

**AGP 1x: 518 MB/s**

**Fast!!!**
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

- **Mpixels/s**
- **Mvertices/s**
- **Mtransistors**

<table>
<thead>
<tr>
<th>Year</th>
<th>PCI (133 MB/s)</th>
<th>AGP (266 MB/s)</th>
<th>AGP2x (533 MB/s)</th>
<th>AGP4x (1.06 GB/s)</th>
<th>AGP8x (2.1 GB/s)</th>
<th>PCIe (4 GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>4 MB DirectX 1</td>
<td>32 MB DirectX 2</td>
<td>64 MB DirectX 5</td>
<td>128 MB DirectX 6</td>
<td>256 MB DirectX 8</td>
<td>512 MB DirectX 9</td>
</tr>
<tr>
<td>1996</td>
<td>4 MB OpenGL 1.1</td>
<td>32 MB OpenGL 1.2</td>
<td>64 MB OpenGL 1.3</td>
<td>128 MB OpenGL 1.4</td>
<td>256 MB OpenGL 1.5</td>
<td>512 MB OpenGL 1.5</td>
</tr>
</tbody>
</table>
Overview

- What is a GPU
- Evolution of GPU
- **GPU Design**
- Modern Features
  - Programmability!
Classic 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
  - 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

- **Geometry Processing**
  - Not in classic pipeline (optional in modern)
  - Creates new primitives
  - 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

[example from Mark Colbert]
example

![Diagram showing the process of 3D rendering with vertex processing, primitive assembly, rasterization & interpolation, fragment processing, and raster operations leading to 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();
...
```

[example from Mark Colbert]
example

Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

[example from Mark Colbert]
example

Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

d [example from Mark Colbert]
Program/API

Driver

GPU Front End

Vertex Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer(s)

screen space

[example from Mark Colbert]
example

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)
Overview

- What is a GPU
- Evolution of GPU
- GPU Design
- Modern Features
  - Programmability!
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>CPU</td>
<td>GPU</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>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Graphics Hardware
Shaders

- Simple programs that run on GPU
  - Definition of “simple” evolves quickly
- Three main types:
  - **Vertex Shaders**
    - Input: Vertex Data
    - Output: New Vertex Data
  - **Pixel /Fragment Shaders**
    - Input: Vertex Colors, Normals, etc
    - Output: Fragment Color
  - **Geometry Shaders**
    - Input: Mesh/Vertex Data
    - Output: New Mesh/Vertex Positions
GPU pipeline

Program/API

Driver

CPU

GPU Front End

GPU

Vertex Processing

Geometry Processing

Primitive Assembly

Rasterization & Interpolation

Fragment Processing

Raster Operations

Framebuffer

[example from Mark Colbert]
GPU pipeline

- **Vertex Processing**
  - Normally performs transformations
  - Programmable

- **Vertex Processor**
  - Data for interpolation:
    - POSITION
    - PSIZE
    - FOG
    - TEXCOORD[0-7]
    - COLOR[0-1]
  - Data for rasterization:
    - TEXCOORD[0-7]
    - COLOR[0-1]
GPU pipeline

- Geometry Processing
  - Optional, can create new geometry
  - Programmable

---

**Graphics Pipeline Diagram**

- **Geometry Processing**
- **Vertex Processing**
- **Textures**
- **Geometry Processor**
- **Shaders**
- **Points, Lines, Triangles**
**GPU pipeline**

- **Fragment Processing**
  - Programmable

Interpolated data:
- TEXCOORD[0-7]
- COLOR[0-1]

Rasterized data:
- DEPTH

Fragment Processor:
- Shader

Data for raster ops:
- COLOR[0-3]
- DEPTH

Textures:
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, NVIDIA only)
  - HLSL – High Level Shader Language (DirectX)
  - GLSL – OpenGL Shader Language (OpenGL)
GLSL

- Language targeted specifically at graphics programming
- Vectors built in:
  - i.e. vec2, vec3, vec4
  - `vec4 v = vec4(a,b,c,d);`
  - `v.xyzw, v.x, v.y, v.z`
  - `v.rgba, ....`
GLSL cont.

- Matrix built in…
  - e.g. float4x4
  - mat2, mat3, mat4
- Linear algebra built in…
  - dot product, matrix-vector multiplication
- Geometric processing…
  - distance, length, normal
- Most of math.h
  - Optimized version (2\textsuperscript{nd} order talyer approximation)
Vertex Shaders

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

- Tessellation (LOD)
- Particle effects
- Advanced clipping
- Multiple renders
Geometry Shaders
Geometry Shaders
Pixel Shaders

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

- Parametrically specify complex textures

Stages of a Procedural Brick Shader

- Graphics Hardware
Complex Examples

- Reflective bump-mapping
Far-cry Screen Shot
ATI Research Groups NPR
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 vertices or pixels
  - Known as SIMD (Single Instruction Multiple Data)
  - Or SIMT (Single Instruction Multiple Threads)
  - If-statements diminish effectiveness
- Best performing Shaders are aware of this.
Early Programmable Graphics Hardware

- Parallelism + pipelining: ATI Radeon 9700

4 vertex pipelines 8 pixel pipelines
Modern Programmable Hardware

- Unified Shaders (vertex, geometry, fragment)
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
  - OpenCL – CPU and GPU support
GPGPU – Not for Everything

- GPU threads are restrictive
  - Local memory is limited
  - Main memory access is expensive
  - Branching is expensive
  - Thread units must process same code path

- Unpredictable, varying code is not for GPUs
  - Huffman encoding/decoding
More GPGPU Links

- nVidia
- ATI
- General Purpose GPU Programming
  - [http://www.gpgpu.org](http://www.gpgpu.org)
- OpenCL
  - [http://www.khronos.org/opencl/](http://www.khronos.org/opencl/)