Computer Graphics Hardware
An Overview
Graphics System

Input devices

CPU/Memory

GPU

Monitor
Raster Graphics System

- Raster: An array of picture elements
- Based on raster-scan TV technology
- The screen (and a picture) consists of discrete pixels, and each pixel has a small display area

![Diagram of a raster graphics system]

- Frame buffer
- DAC
- Video controller
- x, y

A
Frame Buffer

- Frame buffer: the memory to hold the pixel properties (color, alpha, depth, stencil mask, etc)

- Properties of a frame buffer that affect the graphics performance:
  - Size: screen resolution
  - Depth: color level
    - 1 bit/pixel: black and white
    - 8 bits/pixel: 256 levels of gray or color pallet index
    - 24 bits/pixel: 16 million colors
  - Speed: refresh speed
A (way too) simple graphics system

Frame buffer can be part of the main memory

Problem?
Dedicated memory

Video memory: On-board frame buffer: much faster to access
A dedicated processor for graphics processing
Graphics Bus Interface

PCI based technology

- Graphics Memory/Frame buffer
- Graphics Processor
- Scan Controller
- Other Peripherals
- System Bus
- CPU
- Main Memory
- PCIe (8 GB/s)
Graphics Accelerators
What do GPUs do?

- Graphics processing units (GPUs) are massively parallel processors
  - Process geometry/pixels and produce images to be displayed on the screen
  - Can also be used to perform general purpose computation (via CUDA/OpenGL)
- Evolved from simple video scan controllers, to special purpose processors that implement a simple pipeline with fixed graphics functionality, to complex many-core architectures that contain several deep parallel pipelines
  - The latest GPU (Kepler GK110) contains 15x192 cores and 7.1 billions transistors
  - A graphics card can easily have more than 2GB of video memory
CPUs vs. GPUs (2005)

Pentium Extreme Edition 840
- 3.2 GHz Dual Core
- 230M Transistors
- 90nm process
- 206 mm^2
- 2 x 1MB Cache
- 25.6 GFlops

GeForce 7800 GTX
- 430 MHz
- 302M Transistors
- 110nm process
- 326 mm^2
- 313 GFlops (shader)
- 1.3 TFlops (total)
nVidia G80 GPU (2006)

- 128 streaming floating point processors @1.5Ghz
- 1.5 Gb Shared RAM with 86Gb/s bandwidth
- 500 Gflop on one chip (single precision)
nVidia G80 GPU

Application → Data Assembler → Vtx Thread Issue → Setup / Rstr / ZCull → Prim Thread Issue → Frag Thread Issue → Thread Processor → L1 → TF → L2 → FB → Framebuffer

- Vertex assembly → Vertex operations → Primitive assembly → Primitive operations → Rasterization → Fragment operations
nVidia Fermi GPU (2009)
## nVidia Fermi GPU (2009)

<table>
<thead>
<tr>
<th>GPU</th>
<th>G80</th>
<th>GT200</th>
<th>Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>681 million</td>
<td>1.4 billion</td>
<td>3.0 billion</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>128</td>
<td>240</td>
<td>512</td>
</tr>
<tr>
<td>Double Precision Floating Point Capability</td>
<td>None</td>
<td>30 FMA ops / clock</td>
<td>256 FMA ops / clock</td>
</tr>
<tr>
<td>Single Precision Floating Point Capability</td>
<td>128 MAD ops / clock</td>
<td>240 MAD ops / clock</td>
<td>512 FMA ops / clock</td>
</tr>
<tr>
<td>Special Function Units (SFUs) / SM</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Warp schedulers (per SM)</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Shared Memory (per SM)</td>
<td>16 KB</td>
<td>16 KB</td>
<td>Configurable 48 KB or 16 KB</td>
</tr>
<tr>
<td>L1 Cache (per SM)</td>
<td>None</td>
<td>None</td>
<td>Configurable 16 KB or 48 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>None</td>
<td>None</td>
<td>768 KB</td>
</tr>
<tr>
<td>ECC Memory Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concurrent Kernels</td>
<td>No</td>
<td>No</td>
<td>Up to 16</td>
</tr>
<tr>
<td>Load/Store Address Width</td>
<td>32-bit</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
</tbody>
</table>
nVidia Kepler GK110 (2012)

Architecture
- 7.1B Transistors
- 15 SMX units
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5
- PCI Express Gen3
CPU/GPU Performance Gap

[Graph showing the performance gap between AMD (GPU), NVIDIA (GPU), and Intel (CPU) from 2001 to 2009. The graph indicates a significant increase in GFLOPS for many-core GPUs compared to multicore CPUs. The data courtesy of John Owens.]
Why are GPU’s so fast?

- Entertainment Industry has driven the economy of these chips?
  - Males age 15-35 buy $10B in video games / year
- Moore’s Law ++
- Simplified design (stream processing)
- Single-chip designs.
Modern GPU has more ALU’s

Figure 1-2. The GPU Devotes More Transistors to Data Processing
A Specialized Processor

- Very Efficient For
  - Fast Parallel Floating Point Processing
  - Single Instruction Multiple Data Operations
  - High Computation per Memory Access

- Not As Efficient For
  - Double Precision
  - Logical Operations on Integer Data
  - Branching-Intensive Operations
  - Random Access, Memory-Intensive Operations
The Rendering Pipeline

The process to generate two-dimensional images from given virtual cameras and 3D objects

The pipeline stages implement various core graphics rendering algorithms

Why should you know the pipeline?
- Necessary for programming GPUs
- Understand various graphics algorithms
- Analyze performance bottleneck

Diagram:
- host interface
- vertex processing
- triangle setup
- pixel processing
- memory interface
The Rendering Pipeline

- The basic construction – three conceptual stages
- Each stage is a pipeline and runs in parallel
- Graphics performance is determined by the slowest stage
- Modern graphics systems:
  - Software
  - Hardware

Diagram:
- Application
  - Geometry
    - Rasteriazer
      - Image
Host Interface

- The host interface is the communication bridge between the CPU and the GPU
- It receives commands from the CPU and also pulls geometry information from system memory
- It outputs a *stream* of vertices in object space with all their associated information (normals, texture coordinates, per vertex color etc)
Vertex Processing

- The vertex processing stage receives vertices from the host interface in object space and outputs them in screen space.
- This may be a simple linear transformation, or a complex operation involving morphing effects.
- Normals, texcoords etc are also transformed.
- No new vertices are created in this stage, and no vertices are discarded (input/output has 1:1 mapping).

```
host interface → vertex processing → triangle setup → pixel processing → memory interface
```
Triangle setup

- In this stage geometry information becomes raster information (screen space geometry is the input, pixels are the output)
- Prior to rasterization, triangles that are backfacing or are located outside the viewing frustrum are rejected
- Some GPUs also do some hidden surface removal at this stage
Triangle Setup (cont)

- A fragment is generated if and only if its center is inside the triangle.
- Every fragment generated has its attributes computed to be the perspective correct interpolation of the three vertices that make up the triangle.
Each fragment provided by triangle setup is fed into fragment processing as a set of attributes (position, normal, texcoord etc), which are used to compute the final color for this pixel.

The computations taking place here include texture mapping and math operations.

Typically the bottleneck in modern applications
Memory Interface

- Fragment colors provided by the previous stage are written to the framebuffer.
- Used to be the biggest bottleneck before fragment processing took over.
- Before the final write occurs, some fragments are rejected by the zbuffer, stencil and alpha tests.
- On modern GPUs, z and color are compressed to reduce framebuffer bandwidth (but not size).
Programmability in the GPU

- Vertex and fragment processing, and now triangle setup, are programmable.
- The programmer can write programs that are executed for every vertex as well as for every fragment.
- This allows fully customizable geometry and shading effects that go well beyond the generic look and feel of older 3D applications.
The Graphics Pipeline

Application Stage → 3D Triangles → Geometry Stage → 2D Triangles → Rasterization Stage → Pixels

For each triangle vertex:
- Transform 3D position into screen position
- Compute attributes

For each triangle:
- Rasterize triangle
- Interpolate vertex attributes across triangle
- Shade pixels
- Resolve visibility
Diagram of a modern GPU

Input from CPU

Host interface

Vertex processing

Triangle setup

Pixel processing

Memory Interface

64bits to memory

64bits to memory

64bits to memory

64bits to memory
GPU Architecture Progression

1999
- Multi-texture, 32b rendering

2001
- Programmable vertex, 3D textures, shadow maps, multisampling

2003
- Fragment programs, Color and depth compression

2005
- Transparency antialiasing

2007
- Double Precision

1998
- 16-bit depth, Color, and textures

2000
- T&L, cube maps, Texture compression, Anisotropic filtering

2002
- Early z-cull, Dual-monitor

2004
- Flow control, FP textures, VTF

2006
- Unified shader, geometry shader, CUDA/C

Ballistics

Far Cry

(courtesy: nvidia)
Modern GPU’s: Unified Architecture

Discrete Design

Shader A

Shader B

Shader C

Shader D

Unified Design

Vertex shaders, pixel shaders, etc. become *threads* running different programs on flexible cores

(courtesy: nvidia)
Why unify?

Vertex Shader

Pixel Shader
Idle hardware

Heavy Geometry
Workload Perf = 4

Vertex Shader
Idle hardware

Pixel Shader

Heavy Pixel
Workload Perf = 8

(courtesy: nvidia)
Why unify?

Unified Shader

- Vertex Workload
- Pixel

Heavy Geometry
Workload Perf = 11

Unified Shader

- Pixel Workload
- Vertex

Heavy Pixel
Workload Perf = 11

(courtesy: nvidia)
The Quest for Realism