Announcements

• Prelim will be in homework hand back room after class
  – Not before

• Solutions at end of class
State of the art in GPUs

2015 FEATURES
UNREAL ENGINE 4
Computer Graphics System

- **CPU**
  - Input Devices
  - Network

- **Bus**
  - MC Vertices
  - SC Vertices
  - Pixel Parameters
  - Interpolated variables

- **Vertex Processor**
- **Fragment Processor**

- **Z-Buffer**
- **Double-buffered Framebuffers**
- **Texture Memory**

**Abbreviations:**
- MC = Model Coordinates
- SC = Screen Coordinates
- TC = Texture Coordinates
The Framebuffer

- RGB
  - floats for HDR and compute
- Alpha
  - transparency
- Z-buffer
  - hidden surface removal
- Double buffering
  - avoid tearing
Double buffering

- The monitor displays one image at a time
- Tearing/popping
- Use two buffers: one front and one back

As the front buffer is displayed...

the back buffer is where graphics data is sent to be rendered

When the back buffer is ready, the buffers are swapped
Buffers, buffers, buffers!!!

A-buffer - Carpenter, 1984
G-buffer - Saito & Takahashi, 1991
M-buffer - Schneider & Rossignac, 1995
P-buffer - Yuan & Sun, 1997
T-buffer - Hsiung, Thibadeau & Wu, 1990
W-buffer - 3dfx, 1996?
Z-buffer - Catmull, 1973 (?)
ZZ-buffer - Salesin & Stolfi, 1989

Accumulation Buffer - Haeberli & Akeley, 1990
Area Sampling Buffer - Sung, 1992
Back Buffer - Baum, Cohen, Wallace & Greenberg, 1986
Close Objects Buffer - Telea & van Overveld, 1997
Color Buffer
Compositing Buffer - Lau & Wiseman, 1994
Cross Scan Buffer - Tanaka & Takahashi, 1994
Delta Z Buffer - Yamamoto, 1991
Depth Buffer - 1984
Depth-Interval Buffer - Rossignac & Wu, 1989
Double Buffer - 1993

Escape Buffer - Hefting & Hart, 1995
Frame Buffer - Kajiya, Sutherland & Cheadle, 1975
Hierarchical Z-Buffer - Greene, 1993
Item Buffer - Weghorst, Hooper & Greenberg, 1984
Light Buffer - Haines & Greenberg, 1986
Mesh Buffer - Deering, 1995
Normal Buffer - Curington, 1985
Picture Buffer - Ollis & Borgwardt, 1988
Pixel Buffer - Peachey, 1987
Ray Distribution Buffer - Shinya, 1994
Ray-Z-Buffer - Lamparter, Muller & Winckler, 1990
Refreshing Buffer - Basil, 1977
Sample Buffer - Ke & Change, 1993
Shadow Buffer - GIMP, 1999
Sheet Buffer - Mueller & Crawfis, 1998
Stencil Buffer - 1992
Super Buffer - Gharachorloo & Pottle, 1985
Super-Plane Buffer - Zhou & Peng, 1992
Triple Buffer
Video Buffer - Scherson & Punte, 1987
Volume Buffer - Sramek & Kaufman, 1999

Source: Eric Haines
The Fragment Processor

- Fragment
  - Pixel to be
- Produce RGBA output
- Shader
  - Color computation
  - Texturing
  - Per-pixel lighting
  - Fog
  - Blending
  - Discarding fragments
The Rasterizer

- Screen space coordinates into lines, polys
- Interpolates
  - x, y
  - RGB
  - alpha
  - z
  - intensities
  - normals
  - texture coordinates
  - custom values given by shaders
Texture Mapping

• Workhorse
Vertex Processor

• Coordinates
  – in model units, out pixel units

• Shaders
  – Vertex transformations
  – Normal transformations, Normal normalization
  – Per-vertex lighting

• Fixed function
  – View volume clipping
  – Homogeneous division
  – Viewport mapping
  – Backface culling
# CPU and Bus

## PCI Express link performance[^21][^22]

<table>
<thead>
<tr>
<th>PCI Express version</th>
<th>Line code</th>
<th>Transfer rate[^a]</th>
<th>Bandwidth Per lane[^a]</th>
<th>Bandwidth In a x16 (16-lane) slot[^a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>8b/10b</td>
<td>2.5 GT/s</td>
<td>2 Gbit/s (250 MB/s)</td>
<td>32 Gbit/s (4 GB/s)</td>
</tr>
<tr>
<td>2.0</td>
<td>8b/10b</td>
<td>5 GT/s</td>
<td>4 Gbit/s (500 MB/s)</td>
<td>64 Gbit/s (8 GB/s)</td>
</tr>
<tr>
<td>3.0</td>
<td>128b/130b</td>
<td>8 GT/s</td>
<td>7.877 Gbit/s (984.6 MB/s)</td>
<td>126.032 Gbit/s (15.754 GB/s)</td>
</tr>
<tr>
<td>4.0</td>
<td>128b/130b</td>
<td>16 GT/s</td>
<td>15.754 Gbit/s (1969.2 MB/s)</td>
<td>252.064 Gbit/s (31.508 GB/s)</td>
</tr>
</tbody>
</table>

[^21]: [Reference 21]
[^22]: [Reference 22]
Computer Graphics System

Input Devices → CPU → Bus → Vertex Processor → Rasterizer → Fragment Processor → Texture Memory → Double-buffered Framebuffers → Z-Buffer

MC = Model Coordinates
SC = Screen Coordinates
TC = Texture Coordinates
GPUs Faster than Moore’s Law

One-pixel polygons (~10M polygons @ 30Hz)

Slope ~2.4x/year
(Moore’s Law ~ 1.7x/year)

Graph courtesy of Professor John Poulton  (from Eric Haines)
GPU Parallelism

• GPUs are SIMD machines

• They exploit 2 types of parallelism
  – Data: (vertex, triangle, fragment) parallelism
    ▪ Process k triangles in parallel, m fragments in parallel
  – Task: pipeline
    ▪ Pipeline in GPUs up to 800-1000 clocks long (compare to 10-20 on CPUs)
Multi-Threaded SIMD

- Very fine grain threads

- Latency
  - Hide latency by switching to other threads
  - Shared register file (very large, 65k 32-bit registers now)
  - Also prefetching
Architectural Trends

- More general purpose
- More shaders: vertex, pixel, geometry, tessellation
- Longer shaders
  - Length of shaders: 16, 128, … unbounded
- More bits
  - More texturing: more, bigger, and greater precision
  - Better floating point
  - Better HDR support
- More SIMD cores
  - More parallelism
### GTX TITAN GPU Engine Specs:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Cores</td>
<td>2688</td>
</tr>
<tr>
<td>Base Clock (MHz)</td>
<td>837</td>
</tr>
<tr>
<td>Boost Clock (MHz)</td>
<td>876</td>
</tr>
<tr>
<td>Texture Fill Rate (billion/sec)</td>
<td>187.5</td>
</tr>
</tbody>
</table>

### GTX TITAN Memory Specs:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Clock</td>
<td>6.0 Gbps</td>
</tr>
<tr>
<td>Standard Memory Config</td>
<td>6144 MB</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>GDDR5</td>
</tr>
<tr>
<td>Memory Interface Width</td>
<td>384-bit GDDR5</td>
</tr>
<tr>
<td>Memory Bandwidth (GB/sec)</td>
<td>288.4</td>
</tr>
</tbody>
</table>

### GTX TITAN Support:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Important Technologies</td>
<td>GPU Boost 2.0, PhysX, TXAA, NVIDIA G-SYNC-ready, SHIELD-ready</td>
</tr>
<tr>
<td>Other Supported Technologies</td>
<td>3D Vision, CUDA, Adaptive VSync, FXAA, NVIDIA Surround, SLI-ready</td>
</tr>
<tr>
<td>OpenGL</td>
<td>4.4</td>
</tr>
<tr>
<td>Bus Support</td>
<td>PCI Express 3.0</td>
</tr>
<tr>
<td>Certified for Windows 7, Windows 8, Windows Vista, or Windows XP</td>
<td>Yes</td>
</tr>
<tr>
<td>3D Vision Ready</td>
<td>Yes</td>
</tr>
<tr>
<td>Microsoft DirectX</td>
<td>12 API</td>
</tr>
<tr>
<td>Blu Ray 3D</td>
<td>Yes</td>
</tr>
<tr>
<td>3D Gaming</td>
<td>Yes</td>
</tr>
<tr>
<td>3D Vision Live (Photos and Videos)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
OpenGL 4.2+

Diagram showing the flow of data and control from the Vertex Shader to the Fragment Shader, highlighting the Fixed Function and Programmable components.
GPU Pipeline

• Vertex shader
  – Model and View Transform
  – Vertex Shading

• Tessellation Shader
  – Create subdivision surfaces

• Geometry Shader
  – Create/destroy primitives

• Fragment Shader
  – Fully general and really powerful
Tessellation Shaders

• Adaptive subdivision
  – Based on size, curvature, screen space extent

• Coarse models with
  – GPU compression
  – detailed displacement maps w/o detailed geometry
  – subdivision rules
  – adapt quality to level of detail
    ▪ smoother silhouettes
  – Terrain proof of concept, Demo
The dark ages (early-mid 1990’s), when there were only frame buffers for normal PC’s.

Some accelerators were no more than a simple chip that sped up linear interpolation along a single span, so increasing fill rate.

This is where pipelines start for PC commodity graphics, prior to Fall of 1999.

This part of the pipeline reaches the consumer level with the introduction of the NVIDIA GeForce256.

Hardware today has moved traditional application processing into the graphics accelerator.
Era of GPUs

Nvidia's GeForce 256 was the first graphics chip to actually be called a GPU, based on the addition of a hardware-based transformation and lighting engine (T&L).

This engine allowed the graphics chip to undertake the heavily floating-point intensive calculations of transforming the 3D objects and scenes – and their associated lighting – into the 2D representation of the rendered image. Previously, this computation was undertaken by the CPU, which could easily bottleneck with the workload, and tended to limit available detail.