GPU
Graphics Processing Unit
Scene Transformations

Lighting & Shading

Viewing Transformations

Rasterization

GPUs evolved as hardware and software algorithms evolve.
Early Graphics

- Originally, no specialized graphics hardware
- All processing in software on CPU,
- Results transmitted to frame buffer
  - first, external frame buffers
  - later, internal frame buffers.

Diagram:

1. CPU
2. Frame buffer
3. Display
More detailed pipeline

Simple functionality transferred to specialized hardware.

- Geometry data
- Transform & lighting
- Culling, perspective divide, viewport mapping
- Rasterization
- Simple texturing
- Depth test
- Frame buffer blending
Add more functionality to GPU.

Simple functionality transferred to specialized hardware.

Geometry data

Transform & lighting

Culling, perspective divide, viewport mapping

Rasterization

Simple texturing

Depth test

Frame buffer blending
Fixed function GPU pipeline

- Pipeline implemented in hardware
- Each stage does fixed task
- Tasks are parameterized
- Inflexible – fixed, parameterized functions
- Vector-matrix operations (some parallelism).
Technology advances

- Hardware gets cheaper, smaller, and more powerful
- Parallel architectures develop
- Graphics processing get more sophisticated (environmental mapping, displacement mapping, sub-surface scattering)
- Need more flexibility in GPUs
Vertex shader

- Graphics systems: convert everything to triangles
- Pass vertices, normals, colors, texture coordinates to GPU processor
- GPU: vertex-based computations,
  - Independent of other vertices
- Later, assemble into triangles.
Fragment shader

- Fragment is triangle clipped to pixel
  - Interpolate values
- Multiple textures, Alpha, stencil, depth
  - Independent of other fragments
- Blend with contents of frame buffer.
Introduce parallelism: add multiple units

Geometry data

Vertex Shader
Vertex Shader
Vertex Shader

Culling, perspective divide, viewport mapping

Rasterization

Fragment Shader
Fragment Shader
Fragment Shader

Alpha test, depth test, stencil test

Frame buffer blending
Shading language

- Shade trees -> Pixar’s Renderman shader
Shader Language

- Low level (like assembler) but high-level language compilers: nVidia’s Cg
- 4 component floating point data type
- SIMD
Cg: C-based graphics program

- Array & structures
- Flow control
- Vectors & matrices
- No memory allocation, file I/O
Power

• GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine.

• The raw computational power of a GPU dwarfs that of the most powerful CPU, and the gap is steadily widening.

• GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine.
Next: unify shaders

• One set of shaders
• Allocate to either vertices or fragments
Pipeline evolved
Evolved pipeline
GPGPU

• Make GPU more general – adapt certain types of programs to it’s pipelined, parallel architecture

• Single GeForce 8800 chip achieves a sustained 330 billion floating-point operations per second (Gflops) on simple benchmarks

• Cost-effective: graphics driving demand up, supply up, price down for GPUs

• Finding uses in non-graphics applications.
# GeForce 8800 GTX

![GeForce 8800 GTX](image)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>575MHz</td>
</tr>
<tr>
<td>Multi-Processors</td>
<td>128</td>
</tr>
<tr>
<td>Shader</td>
<td>1350MHz</td>
</tr>
<tr>
<td>Memory</td>
<td>900MHz</td>
</tr>
<tr>
<td>Memory</td>
<td>768MB GDDR3</td>
</tr>
</tbody>
</table>

$599 e-tail
More general: NVIDIA’s CUDA

- More general data parallel model
- Decompose across threads
- Sharing and communication between threads..