Topics

The fixed-function graphics pipeline
Programmable stages
  - Vertex shaders
  - Fragment shaders
GL shading language (GLSL)
Mapping other applications to GPUs
A Trip Down The Graphics Pipeline

J. Blinn

A Trip Down the Graphics Pipeline

Command

Vertex

Assembly

Rasterization

Fragment

Texture

FB ops

Display
Command Processor

Command queue
Command interpretation
Unpack and perform format conversion
Maintain graphics state

Vertex (per-vertex)

Vertex transformation
Normal transformation
Texture coordinate generation
Texture coordinate transformation
Lighting (light sources and surface reflection)

Object-space triangles  Screen-space shaded triangles
Primitive Assembly

Combine transformed/shaded vertices into primitives
- 1 vert -> point
- 2 verts -> line
- 3 verts -> triangle

Clipping
Perspective projection
Transform to window coordinates (viewport)
Determine orientation (CW/CCW)
Back-face cull

Rasterization

Setup (per-triangle)
Sampling (triangle = {fragments})
Interpolation (interpolate colors and coordinates)
Textures are arrays indexed by floats (Sampler)
Texture address calculation
Texture bilinear interpolation and filtering

Fragment
Combine texture sampler outputs
Per-fragment shading
Framebuffer Operations

Operation
- Test window ownership
- Test scissor and stencil mask
- Test alpha
- Test depth

Blending or compositing

Framebuffer Pixels

Display

Gamma correction
Analog to digital conversion

Framebuffer Pixels

Light
Programmable Stages

Programmable Graphics Pipeline

- Command
  - Vertex
  - Assembly
  - Rasterization
  - Fragment
  - FB ops
  - Display

- Texture

Programmable stage
Programmable Graphics Pipeline

- Command
  - Vertex
    - Assembly
      - Rasterization
        - Fragment
          - FB ops
            - Display

Programmable stage

Shader Program Architecture

- Inputs
  - Registers
    - Shader Program
      - Outputs
        - Texture
          - Constant

CS148 Lecture 15
Pat Hanrahan, Fall 2011
What’s in a GPU?

NVIDIA GeForce GTX 480

What’s in a GPU?
GLSL
OpenGL Shading Language

Simple Vertex and Fragment Shaders

// simple.vert
void main()
{
    gl_Position =
        gl_ModelViewMatrix *
            gl_ProjectionMatrix * gl_Vertex;
    gl_Normal = gl_NormalMatrix * gl_Normal;
    gl_FrontColor = gl_Color;
    gl_BackColor = gl_Color;
}

// simple.frag
void main()
{
    gl_FragColor = gl_Color;
}
Uniform Variables

uniform variables are changed at most once per geometric primitive

// Predefined OpenGL state
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ProjectionMatrix;
uniform mat4 gl_NormalMatrix;

// User-defined
uniform float time;

Attribute Variables

attribute variables are properties of a vertex
They are the inputs of the vertex shader

attribute vec4 gl_Color;
attribute vec4 gl_Normal;
attribute vec4 gl_Vertex;

N. B. that points, vectors, normals and colors are all vec4
Varying Variables

attribute variables are the vertex shader inputs

The outputs of the vertex shader are varying

```cpp
attribute vec4 gl_Color;
varying vec4 gl_FrontColor;
varying vec4 gl_BackColor;

// vert shader
void main() {
    gl_FrontColor = gl_Color;
}
```

Varying Variables

varying variables are interpolated across the triangle

gl_Color is set to gl_FrontColor or gl_BackColor depending on whether the triangle is front-facing or back-facing

```cpp
varying vec4 gl_Color;
vec4 gl_FragColor;

void main() {
    gl_FragColor = gl_Color;
}
```
Vectors

Constructors

```cpp
vec3 V3 = vec3(1.0, 2.0, 3.0);
vec4 V4 = vec4(V3, 4.0);
```

Swizzling

```cpp
vec2 V2 = V4.xy;
vec4 V4Reverse = V4.wzyx;
vec4 res = V4.xyzw + V4.xxxx;
```

Basic Vector Operators

```cpp
float res = dot(V4, V4Reverse);
vec3 res = cross(V3, vec3(1.0, 0.0, 0.0));
```

Textures

```cpp
uniform sampler2D SomeTexture;

void main()
{
    vec4 SomeTextureColor =
        texture2D(SomeTexture, vec2(0.5, 0.5));
}
```

N. B. Textures coordinates are from (0, 0) to (1, 1)
The OpenGL Pipeline in GLSL - Vertex

Built-in attributes

<table>
<thead>
<tr>
<th>GLSL</th>
<th>OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vec4 gl_Vertex</code></td>
<td><code>glVertex*()</code></td>
</tr>
<tr>
<td><code>vec4 gl_Color</code></td>
<td><code>glColor*()</code></td>
</tr>
<tr>
<td><code>vec4 gl_SecondaryColor</code></td>
<td><code>glSecondaryColor*()</code></td>
</tr>
<tr>
<td><code>vec4 gl_Normal</code></td>
<td><code>glNormal()</code></td>
</tr>
<tr>
<td><code>vec4 gl_MultiTexCoord0</code></td>
<td><code>glMultiTexCoord()</code></td>
</tr>
</tbody>
</table>

OpenGL in GLSL - Fragment

Built-in varying

| `vec4 gl_Position`          |
| `vec4 gl_FrontColor, gl_BackColor` |
| `vec4 gl_FrontSecondaryColor,` |
| `gl_BackSecondaryColor`     |
| `vec4 gl_TexCoord[n]`       |
| `vec4 gl_FragCoord`         |

Outputs

| `vec4 gl_FragColor` |
| `vec4 gl_FragDepth` |
**Communicating with GLSL**

**Can extend uniform state**

```cpp
uniform float x;
addr = GetUniformLocation(program, "x");
glUniform1f(addr, value);
```

**Can extend attribute (inside glBegin/glEnd)**

```cpp
uniform float y;
addr = GetAttributeLocation(program,"y");
glVertexAttrib1f(addr, value);
```

---

**Limitations**

**Memory**
- No access to neighboring fragments
- Limited stack space, instruction count
- Cannot read and write framebuffer

**Performance**
- Branching support is limited and slow
- Graphics card will timeout if code takes too long
- Variable support across different graphics cards
GPU Computing

How to Get a TeraFLOP

16 cores x 32 SIMD functional units x 2 flops/cycle x 1 GHz = 1 TFLOP
Why GPU Computing?

Beyond basic graphics
- Collision detection
- Fluid and cloth
- Ray tracing

Beyond graphics
- Protein folding
- Speech recognition
- Fourier transforms

Check out CUDA and OpenCL

Intel Core i7 975 NVIDIA GeForce GTX 285