Programmable GPUs

Real Time Graphics

- Virtua Fighter 1995 (SEGA Corporation, NV1)
- Dead or Alive 2001 (Tecmo Corporation, Xbox (NV2A))
- Nalu 2004 (NVIDIA Corporation, GeForce 6)
- Human Head 2006 (NVIDIA Corporation, GeForce 7)
- Medusa 2008 (NVIDIA Corporation, GeForce GTX 200)
- Real-Time Dynamic Fracture 2013 (NVIDIA Corporation, GeForce GTX 700)

GPUs vs CPUs

- 4500 GFLOPS vs ~500 GFLOPS
- 290 GB/s vs 60 GB/s

Programmable Pipeline

- so far:
  - rendering pipeline = set of stages with fixed functionality

- now: programmable rendering pipeline!

- vertex shader
- fragment shader
Vertex Shader

- Run once for every vertex in your scene:
  - Common Functionality:
    - Performs viewing transforms (MVP)
    - Transforms texture coordinates
    - Calculates per-vertex lighting
  - A “vertex” is a malleable definition, you can pass in, and perform pretty much any operation you want

Fragment Shader

- Runs for all “initialized” fragments:
  - “initialized” → rendered to after rasterization
- Common Tasks:
  - texture mapping
  - Shading
- Synonymous with Pixel Shader

Vertex Shader - Applications

- deformable surfaces – on the fly vertex position computation
  - e.g. skinning

Fragment Shader - Applications

- GPU raytracing, NVIDIA
- Not really shaders, but very similar to NPR!
  - A Scanner Darkly, Warner Independent Pictures
Vertex & Fragment Shader

- massively parallel computing by parallelization
- same shader is applied to all data (vertices or fragments) – SIMD (single instruction multiple data)
- parallel programming issues:
  - main advantage: high performance
  - main disadvantage: no access to neighboring vertices/fragments

Shader Languages

- Many languages exist to write shaders:
  - GLSL – GL Shading Language (OpenGL)
  - HLSL – High Level Shading Language (Direct3D)
  - CG (Nvidia mid-level language for both)

GLSL

- We are using GLSL:
  - C-like programming language for GPUs
  - Highly Parallel (SIMD)
  - Differs greatly between versions

GLSL - Types

- Has all the basic C types
- Has "vector" types: vec2, vec3, vec4
- Has "matrix" types: mat2, mat3, mat4
- Has "sampler" types
  - Used for reading data from textures and framebuffers
  - (might be worthwhile looking into for Assignment 4)

Look at these links for more info:
- http://www.opengl.org/wiki/Data_Type_%28GLSL%29
- http://www.opengl.org/wiki/Sampler_%28GLSL%29#Sampler_types

GLSL – Integration into Opengl

- GLSL has some variables built in
  - These variables are always there and accessible in the corresponding shader

  - Vertex Shader
    - In: gl_Vertex (position), gl_Normal, gl_Color
    - Out: gl_Position

  - Fragment Shader
    - In: glFragCoord (fragment location), gl_Color
    - Out: gl_FragColor, gl_FragDepth
GLSL – Built in Variables

- Accessible in all shaders:
  - `gl_ModelViewMatrix`
  - `gl_ModelViewProjectionMatrix`
  - `gl_ProjectionMatrix`

- Here is a quick reference guide:
  - [http://mew.cx/glsl_quickref.pdf](http://mew.cx/glsl_quickref.pdf)

GLSL Example – Vertex Shader

- Vertex Shader: scale vertices

```glsl
#version 200

void main() {
    // scale passed in vertex
    vec4 a = gl_Vertex;
    a.x = a.x * 1.5;
    a.y = a.y * 1.5;
    // transform vertex
    gl_Position = gl_ModelViewProjectionMatrix * a;
}
```

GLSL – Uniform Variables

- Used to access data from the CPU on the GPU
- Need to be given a value from the OpenGL side

```glsl
#version 200

void main() {

    uniform float specIntensity;
    uniform vec4 specColor;
    uniform float t;
    uniform vec4 colors;

    // do something
}
```

GLSL Example – Fragment Shader

- Fragment Shader: color green

```glsl
#version 200

void main() {
    // color rendered fragments green
    gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

GLSL Example – Uniform Variables

- Within shader:

```glsl
#version 200

uniform float specIntensity;
uniform vec4 specColor;
uniform float t;
uniform vec4 colors;

void main() {
    // do something
}
```
GLSL – Samplers

- A type of uniform used to read from a texture within shaders
- There are different samplers for the different types of textures
- 2D textures store square textures
- Rectangle textures store non-square textures, such as the image being processed in A4

OpenGL Error Checking

- When Things go Wrong:
  - OpenGL won’t tell you
  - To ask, call glGetError()
  - Tells you the gl state (ok, error, etc)
  - For A4, this is all done for you, but you will need to break before the end of the program to read the output (in the black terminal)

OpenGL the old and the new

<table>
<thead>
<tr>
<th>GL 1.2 – 2.1</th>
<th>GL 3.0 – 4.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Shader</td>
<td>Vertex Shader</td>
</tr>
<tr>
<td>Pixel Shader</td>
<td>Tessellation (Control) Shader</td>
</tr>
<tr>
<td></td>
<td>Tessellation Evaluation/Hull Shader</td>
</tr>
<tr>
<td></td>
<td>Geometry Shader</td>
</tr>
<tr>
<td></td>
<td>Fragment Shader</td>
</tr>
<tr>
<td></td>
<td>... Compute Shader</td>
</tr>
</tbody>
</table>

OpenGL updated graphics pipeline

OpenGL 3.0+ changes

- Removed many of the GLSL built-in variables
- Removed GLSL/OpenGL built-in matrices
- Removed glVertex(), glColor, glTexCoord, glMaterial(), ...

OpenGL 3.0+ changes

- Why?
  - Efficiency
    - in most cases you don’t need everything
  - Control
    - with less dictated, shaders can be used to do more
OpenGL 3.0+ Advanced Pipeline

- Tesselation Control shader
  - Synonymous with Tessellation shader (d3d)
  - Subdivide geometry based on vertices

- Tesselation Evaluation
  - Synonymous with Hull shader (d3d)
  - Rearrange new vertices from tesselation control

- Geometry Shaders
  - Perform operations on groups of vertices

- Compute Shaders
  - Use the GPU to do math for you (no rendering)
  - This executes after the geometry shader, replacing the rest of the pipeline

GPGPU Applications

References and Resources

- [http://www.davidcornette.com/glsli/glsli.html](http://www.davidcornette.com/glsli/glsli.html)
- [http://nehe.gamedev.net/article/glsli%20an%20introduction/25007/](http://nehe.gamedev.net/article/glsli%20an%20introduction/25007/)
- [http://www.opengl.org/wiki/Data_Type_%28GLSL%29](http://www.opengl.org/wiki/Data_Type_%28GLSL%29)
- [http://www.opengl.org/wiki/Sampler_%28GLSL%29#Sampler_types](http://www.opengl.org/wiki/Sampler_%28GLSL%29#Sampler_types)
- [http://zach.in.tu-berlin.de/teaching/cg_literatur/glsli_tutorial/](http://zach.in.tu-berlin.de/teaching/cg_literatur/glsli_tutorial/)