Shading with OpenGL 3.3

Alexander Christensen

Department of Computer Science
University of Copenhagen

2018

Overview

1. Shading pipeline
2. glsl - OpenGL Shading Language
   - Runtime requirements
   - Ensure parallelism
3. Shader programs
   - Vertex shader
   - Fragment shader
4. Examples

Shading pipeline

TODO: Include pipeline picture!
Where are the scan conversion algorithms run?
What is rasterization?
What is a shader?

glsl - OpenGL Shading Language

History with OpenGL (fixed-function pipeline vs. programmable shaders)

glsl is a DSL for writing programmable shaders

glsl has a C-like language syntax

Shaders are purely run on the GPU!

glsl - OpenGL Shading Language

Supported expressions:

- primitive data types: `float, int, uint, ...`
- vector/matrix data types: `mat2, mat3, mat4, vec2, vec3, vec4, ...`
- special types: `struct, enum`
- functions

glsl - OpenGL Shading Language

Important guarantee: determinable running time!

- No while loops
- No recursion

Remarks:

Shader execution calls are independent and run in parallel
Not possible to read/modify return values of other shader calls in shader code
Execution units on GPU's typically cannot do branch-prediction very well!

Shader programs - Vertex shader

- Per-vertex processing
- Built-in variables: `gl_Position`
- Can modify position.
- Outputs a vertex position which must be in normalized coordinates (NDC)

Shader programs - Fragment shader

Per-fragment processing.

Each fragment is typically the size of a pixel

Built-in variables: `gl_FragCoord`, `gl_FrontFacing`, `gl_PointCoord`

Outputs a color value (r, g, b, a), in normalized coordinates ($[0, 1]$).

Shader variables - Attributes

Attributes are used in vertex shaders:

```
layout (location = 0) in vec3 vertexPosition;
layout (location = 1) in vec2 texCoord;
```

Data, such as vertex positions, are buffered to the GPU. Attributes are pointers to this data.

```
GLfloat data[] = {
// vertexPosition    texCoord
0.0f, 0.5f, -0.3f,   0.0f, 0.0f,
// ...
}
```

How many times is it run?

Imagine a triangle, in NDC, with coordinates
$(-1,-1,0), (-1,1,0), (1,1,0)$.
Assume application window of size 800×600
$\rightarrow$ 3 vertex shader calls
$\rightarrow$ $\approx (800 \cdot 600)/2 = 240000$ fragment shader calls!

Creating a shader program

Debugging

Black screen
Check variables before they are sent to the GPU

Example 1 - static rendering

Example 2 - animation

Summary

References