COMP3320 Introduction to OpenGL

Alex Biddulph

The University of Newcastle, Australia

Based on the work provided at www.learnopengl.com

Semester 2, 2021

More on Shaders

Typical structure of a GLSL shader

```
#version version_number
in type in_variable_name;

out type out_variable_name;

uniform type uniform_name;

void main() {
  // process input(s) and do
  // some weird graphics stuff
  type weird_results = weird_processing;

  // output processed stuff to output variable
  out_variable_name = weird_results;
}
```

More on Shaders

- The first line specifies OpenGL version. In these examples we use version 3.3 using the core profile

  ```
  #version 330 core
  ```

- We can then specify inputs, outputs, and uniforms in any order.
- type can be any of the following with n and m one of $2, 3, \text{or } 4$

  ```
  bool    int    uint   float   double   bvecn
  ivecn   uvecn  vecn   dvecn   matn     matnxm
  ```

- Arrays of these types are also permitted
- Vectors can be *swizzled*

  ```
  vec3 pos;
  vec2 pos2 = pos.xy;
  vec4 pos4 = pos2.xyyx;
  vec3 pos3 = pos4.wzy;
  ```

- Finally we have the main function. This function must be in every shader

Shader Inputs and Outputs

- Inputs and outputs allow you to pass data in to and get data out of your shaders. They also facilitate passing data between shader stages
- In a vertex shader the inputs are also known as vertex attributes. OpenGL guarantees the existence of at least 16 4-component vertex attributes.

Examples

Query `GL_MAX_VERTEX_ATTRIBS` using ☞`glGetIntegerv` to find the maximum available on your hardware

- Built-in vertex shader output ☞`gl_Position`
- Fragments shaders require a `vec4` colour output variable

Shader Inputs and Outputs

- Variable declarations can be decorated with the ☞layout specifier
  ```
  layout(location = 0) in vec3 aPosition;
  ```
- Location layout simplifies linking vertex attributes and fragment shader outputs in the main program
- If an input and an output share the same type and name in two consecutive shaders, OpenGL will link them together

```
// In the vertex shader
out vec4 frag_colour;
// In the fragment shader
in vec4 frag_colour;
```

Uniforms

- Allows for setting arbitrary data in a shader program
- Global to a shader program
- Can be accessed by any shader at any stage in the program
- Maintain their value until you change it
- If you declare a uniform and don't use it in your GLSL shader code it will be silently removed

Examples

Find the location of a uniform in the shader program using ☞`glGetUniformLocation`

Examples

Set the value of a uniform using ☞`glUniform`