Week 9: Introduction to VFX
Part 4: Programming Shaders

COMP270: Mathematics for 3D Worlds and Simulations

Objectives

- **Manipulate** custom meshes in a shader

Recap: Vertex Shader

- Takes in exactly one vertex
- Carry out operations on that vertex
- Store the result(s) as vertex attributes
- Vertex and attributes are returned back to the pipeline
- Typically used for deformation and animation

Unity and UE4 Vertex

- Unity has built in vertex types:
  - `appdata_base`: position, normal and one texture coordinate
  - `appdata_tan`: position, tangent, normal and one texture coordinate
  - `appdata_full`: position, tangent, normal, four texture coordinates and colour
- UE4 has a collection of Expression nodes that allow you to interact with vertices
  - Vector expressions
  - Coordinate expressions

Vertex Shader: GLSL

```
#version 330 core

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

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

out vec2 vertexTextureCoordOut;

void main() {
	mat4 mvpMatrix = projectionMatrix * viewMatrix * modelMatrix;
	vec4 mvpPosition = mvpMatrix * vec4(vertexPosition, 1.0f);
	vertexTextureCoordOut = vertexTextureCoord;
	gl_Position = mvpPosition;
}
```

Recap: Fragment Shader

- Takes in a pixel fragment (see rasterization)/output from the Vertex Shader
- Outputs colour and depth values
- Typically used for shading calculations and texturing

GLSL Example:

```
#version 330 core

in vec2 vertexTextureCoordOut;
out vec4 colour;
uniform sampler2D diffuseTexture;

void main() {
	colour = texture2D(diffuseTexture, vertexTextureCoordOut);
}
```