GPU Vertex Shader Example

```
#version 330 core

// Vertex attributes
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 color;

// Uniforms
uniform mat4 uniform_mvp;

void main(void) {
// Required output
    gl_Position = uniform_mvp * vec4(position, 1.0);
}
```

CPU Side:

Step 1. Create data structure

```
struct VertexStruct
{
    glm::vec3 position;          // 12 bytes
    glm::vec3 color;             // 12 bytes
};
```

Step 2. Create CPU data buffer (explicitly or obj)

```
// Create data here (e.g. store in a std::vector)
VertexStruct bottom_left; VertexStruct bottom_right;
VertexStruct top;

bottom_left.position     = glm::vec3(-1.0f, -1.0f, 0.0f);
bottom_right.position    = glm::vec3( 1.0f, -1.0f, 0.0f);
top.position             = glm::vec3( 0.0f,  1.0f, 0.0f);
bottom_left.color        = glm::vec3( 1.0f,  0.0f, 0.0f);
bottom_right.color       = glm::vec3( 0.0f,  1.0f, 0.0f);
top.color                = glm::vec3( 0.0f,  0.0f, 1.0f);

// add vertex data in CPU buffer in counter-clockwise order
std::vector<VertexStruct> triangle_data;
triangle_data.push_back(bottom_left);
triangle_data.push_back(bottom_right);
triangle_data.push_back(top);
```

Step 3. Load buffer to GPU

```
// Create a vertex array object
GLuint vao_triangle = 0;
glGenVertexArrays(1, &vao_triangle);          // Generate
glBindVertexArray(vao_triangle);              // Bind

// Create a vertex buffer object
GLuint vbo_triangle = 0;
glGenBuffers(1, &vbo_triangle);               // Generate
glBindBuffer(GL_ARRAY_BUFFER, vbo_triangle); // Bind

// size: 3 elements * 24 bytes each = 72 bytes
int size = triangle_data.size() * sizeof(VertexStruct);

// Pass data to GPU
glBufferData(GL_ARRAY_BUFFER, size, &triangle_data[0],
GL_STATIC_DRAW);

// Data structure
int stride = sizeof(VertexStruct);
// Inform OpenGL about the location of the attributes
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, stride, 0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, stride,
(GLvoid*)(3 * sizeof(GLfloat)));

// make them active
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);

// set the VAO inactive
glBindVertexArray(0);
```

Step 3. Draw

```
/* bind shaders, pass uniform parameters
...
...
*/
glBindVertexArray(vao_triangle);
glDrawArrays(GL_TRIANGLES, 0, triangle_data.size());
glBindVertexArray(0);
```

GPU Side: Assign the corresponding attributes in the vertex shader

```
#version 330 core

// Vertex attributes
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 color;

// Uniforms
uniform mat4 uniform_mvp;

void main(void) {
// Required output
    gl_Position = uniform_mvp * vec4(position, 1.0);
}
```