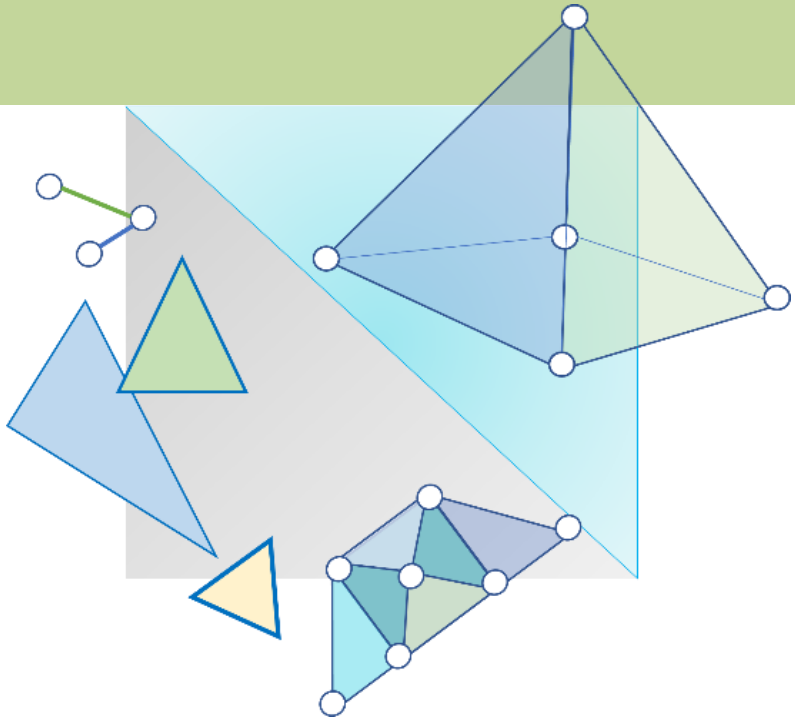
CITS3003 Graphics & Animation

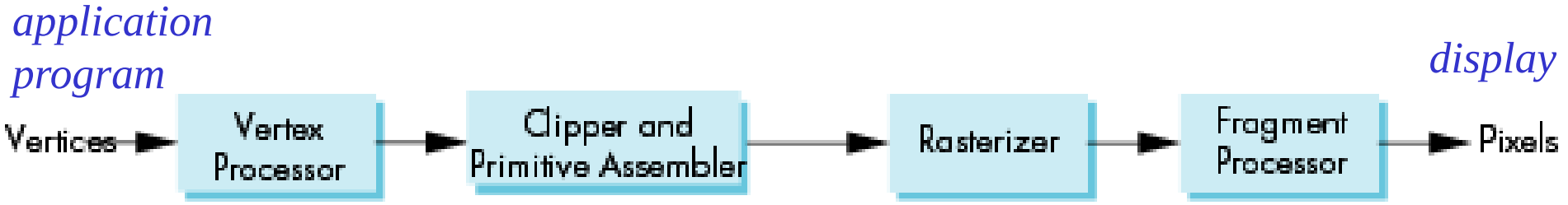
Lecture 6:
Vertex and Fragment
Shaders-2



Content

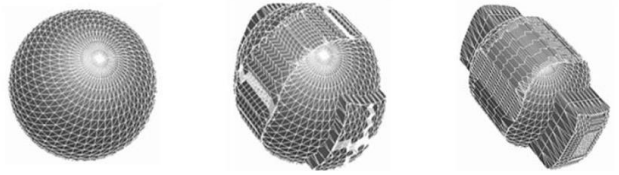
- Vertex Shader (continued from previous lecture)
- Examples of Vertex Shader
- Fragment Shader
- Examples of Fragment Shader
- How the application program and vertex shader work together

What vertex shader can do?
(Application perspective)

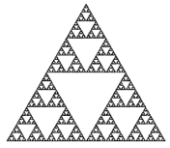


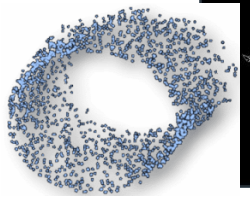
(Vertex shader) (Fragment shader)

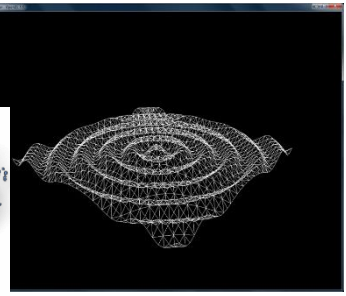
- Geometric transformations
  - Change location, rotation, scale of objects/camera
  - Apply 3D perspective transformation – make far objects smaller



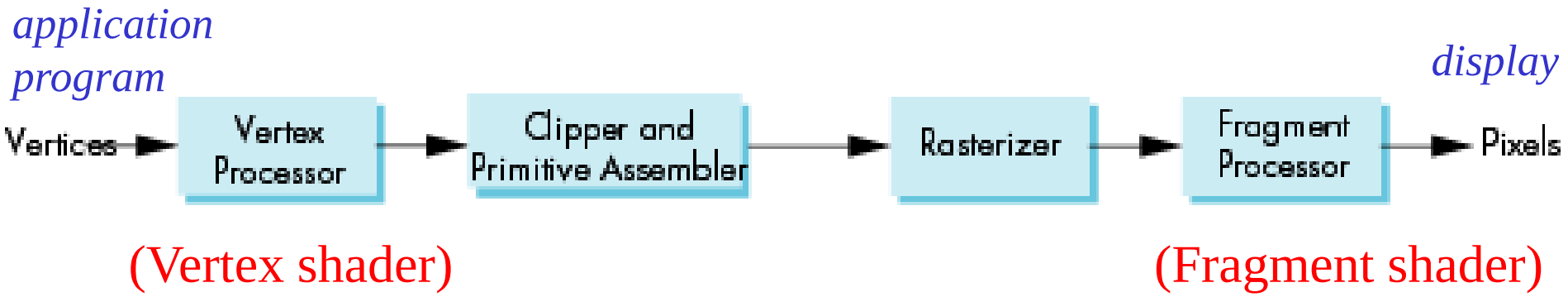
- Moving vertices
  - Perform morphing
  - Compute wave motion & deformation due to physical forces
  - Simulate particle effects – for fire, smoke, rain, waterfalls,
  - Compute fractals







What vertex shader can do?
(Application perspective)



- Lighting – vertex shader can also
  - Calculate shading color using light and surface properties
  - Calculate cartoon shading (for special effects)

Vertex Shader

- In the rendering pipeline, each vertex is processed independently.
- The vertex shader processes one vertex – it takes in one vertex from the vertex stream as input and generates the transformed vertex (optionally with attributes) to the output vertex stream.
- Multiple shader programs can be invoked and run in parallel to render complex scenes in real-time.

A Simple Vertex Shader

Recall from previous lecture…

OpenGL Version	GLSL Version
2.0	1.10
2.1	1.20
3.0	1.30
3.1	1.40
3.2	1.50

```
#version 150
```

GLSL version 1.50

input from application

```
in vec4 vPosition;
```

data type

must link to variable in application

```
void main(void)
{
    gl_Position = vPosition;
}
```

built-in variable

A more complex vertex shader

```
#version 150
in vec4 vPosition;
out vec4 color;
uniform vec3 theta;

void main()
{
    ….          // code omitted
    color = …..; // compute the out variable color
    gl_Position = vPosition; // may be a more complex expression
}
```

Vertex shader can produce output for the rasterizer and fragment shader further down the pipeline

Can also have uniform variable (details on a later slide)

Vertex Shader – Example 1

Below is a ***wave motion*** *vertex shader* example:

```
in vec4 vPosition;
uniform float xs, zs, // frequencies
uniform float h; // height scale
uniform float time; //time

void main()
{
  vec4 t = vPosition;
  t.y = vPosition.y + h*sin(time + xs*vPosition.x)
     + h*sin(time + zs*vPosition.z);

 gl_Position = t;
}
```

Remember: Uniform variables cannot be modified in the shader

Vertex Shader – Example 2

Below is a ***particle system*** example:

```
in vec3 vPosition;
uniform mat4 ModelViewProjectionMatrix;
uniform vec3 vel;
uniform float g, m, t;

void main() {
    vec3 object_pos;
    object_pos.x = vPosition.x + vel.x*t;
    object_pos.y = vPosition.y + vel.y*t + g/(2.0*m)*t*t;
    object_pos.z = vPosition.z + vel.z*t;

    gl_Position = ModelViewProjectionMatrix *
                  vec4(object_pos,1);
}
```

Vertex Shader

```
in  vec4 vPosition;
out vec4 color;
uniform vec3 theta;

void main()
{
    ….          // code omitted
    color = …..;
    gl_Position = vPosition;
}
```

```
attribute vec4 vPosition;
varying vec4 color;
uniform vec3 theta;

void main()
{
    ….          // code omitted
    color = …..;
    gl_Position = vPosition;
}
```

Older Version

What can Fragment Shader do?
(Application perspective)

- Per fragment lighting calculations

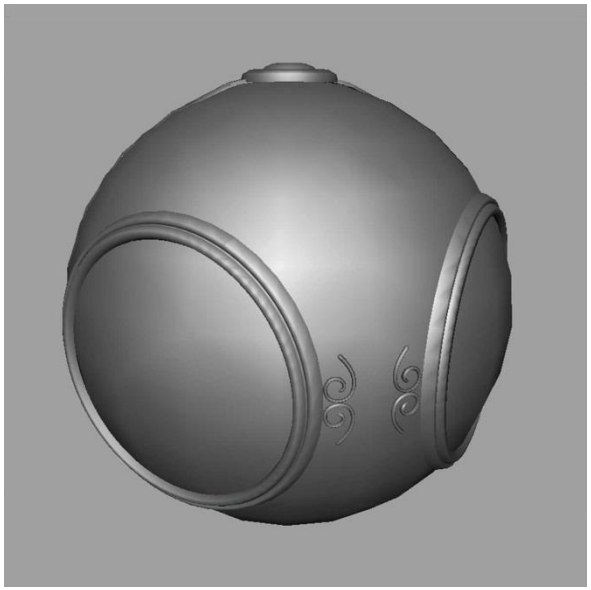
  (recall that a *fragment* is a potential pixel that not only has location coordinates but also has colour, depth, and alpha values)

per vertex lighting

per fragment lighting

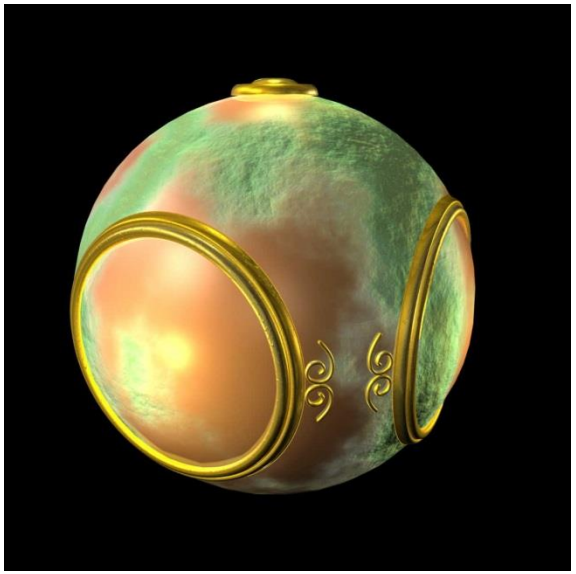
What can Fragment Shader do?
(Application perspective)

- Texture mapping can be done at the fragment shader also.



smooth shading

environment mapping



bump mapping

What can Fragment Shader do?
(Application perspective)

- Lighting calculation
  - Per fragment lighting
- Texture mapping, including
  - Environment mapping
  - Bump mapping

What can Fragment Shader do?
(Application perspective)

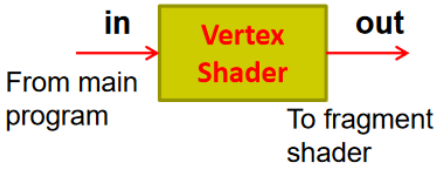
Recall from a previous lecture…

```
#version 150

out vec4 fragcolor;

void main(void) {
  fragcolor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

declare that *fragcolor* is an output variable of the shader

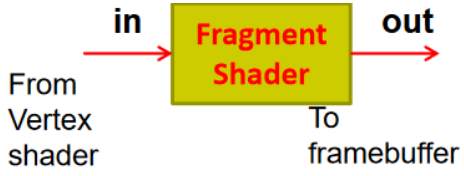
fragcolor must be computed and output

Example: Vertex and fragment shaders




```
#version 150

const vec4 red =
   vec4(1.0, 0.0, 0.0, 1.0);
in vec4 vPosition;
out vec4 color_out;

void main(void)
{
  gl_Position = vPosition;
  color_out = red;
}
```

Vertex shader




```
#version 150
in vec4 color_out;
out vec4 fragcolor;

void main(void) {
  fragcolor = color_out;
}
// in pre-OpenGL 3.2
// versions, use built-in:
// gl_FragColor = color_out;
```

Fragment shader

out variables declared in the vertex shader must be *in* variables in the fragment shader

Example: Vertex and fragment shaders

Older versions

```
const vec4 red =
   vec4(1.0, 0.0, 0.0, 1.0);
attribute vec4 vPosition;
varying vec4 color_out;

void main(void)
{
  gl_Position = vPosition;
  color_out = red;
}
```

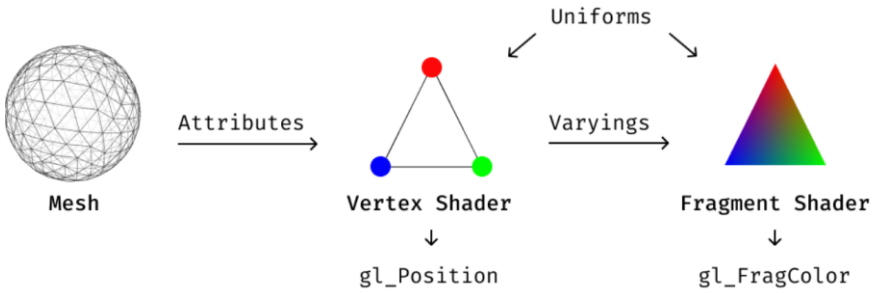
Vertex shader

```
varying vec4 color_out;

void main(void) {
 gl_FragColor = color_out;
}
```

Built-in variable

Fragment shader

varying variables declared in the vertex shader must be *varying* variable in the fragment shader

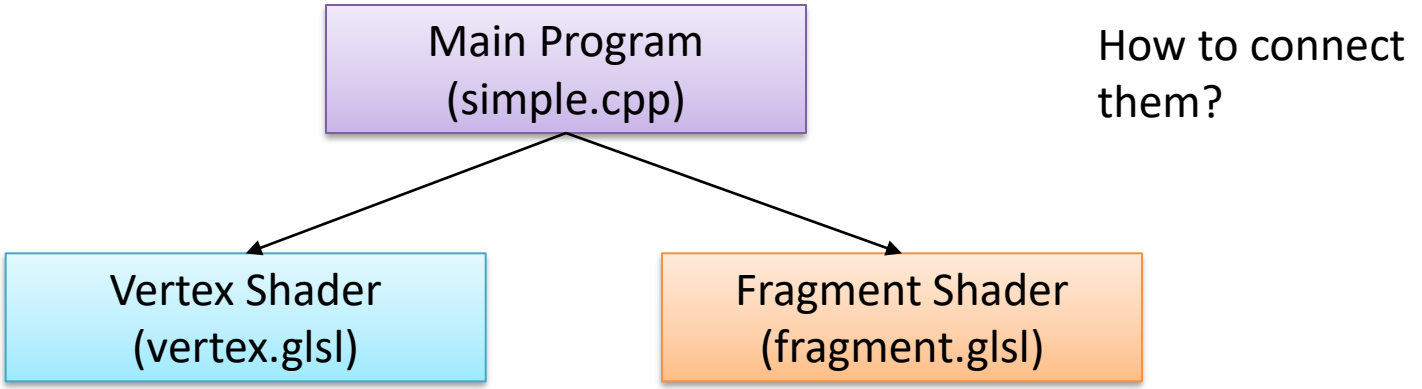


Shaders and Application Program Must Work Together

- For each variable declared using the qualifier attribute (or in) in the vertex shader, the application needs to know how to link to it.
  - All attribute variable names are stored in a table.
  - The application program can get an **index** for each attribute variable from the table.
- Similarly for uniform variables.

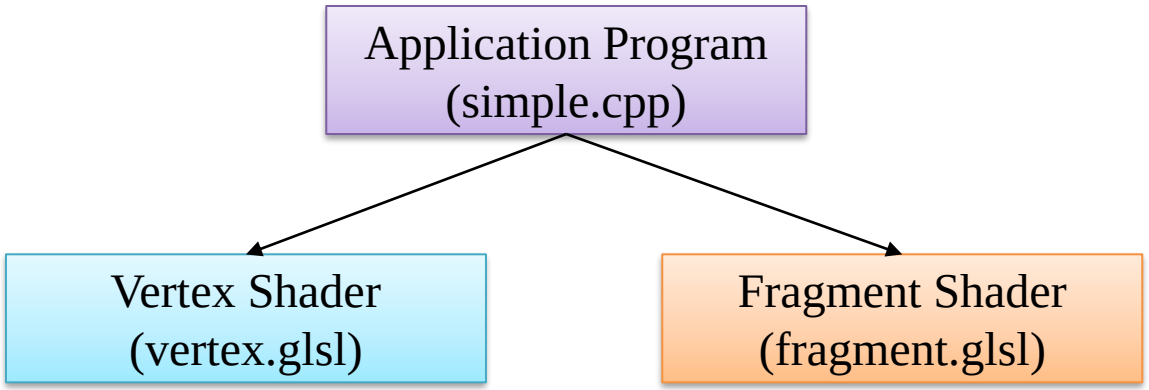
Shaders and Application Program Must Work Together

- For each variable declared using the qualifier attribute (or in) in the vertex shader, the application needs to know how to link to it.



Shaders and Application Program Must Work Together

- For each variable declared using the qualifier attribute (or in) in the vertex shader, the application needs to know how to link to it.

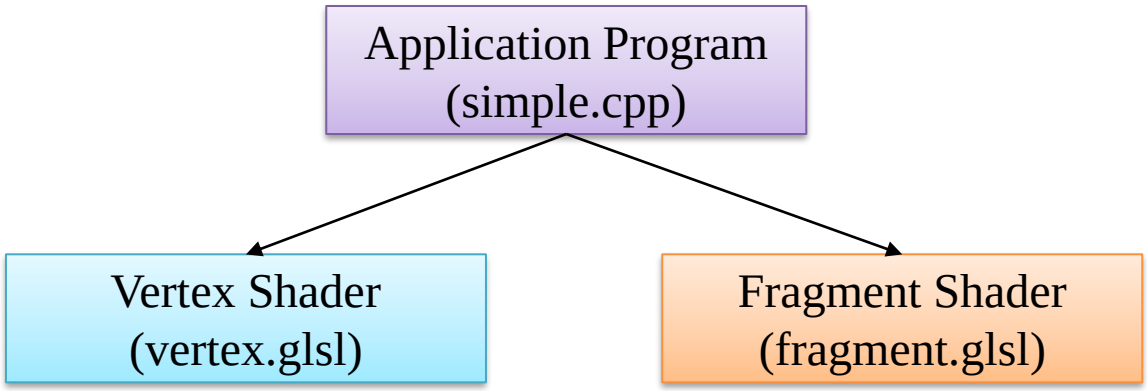


initShader( )
connects main
Program to shader files

```
// Load shaders and use the resulting shader program
    GLuint program = InitShader( "vertex.glsl", "fragment.glsl" );
    glUseProgram( program );
```

Shaders and Application Program Must Work Together

- For each variable declared using the qualifier attribute (or in) in the vertex shader, the application needs to know how to link to it.

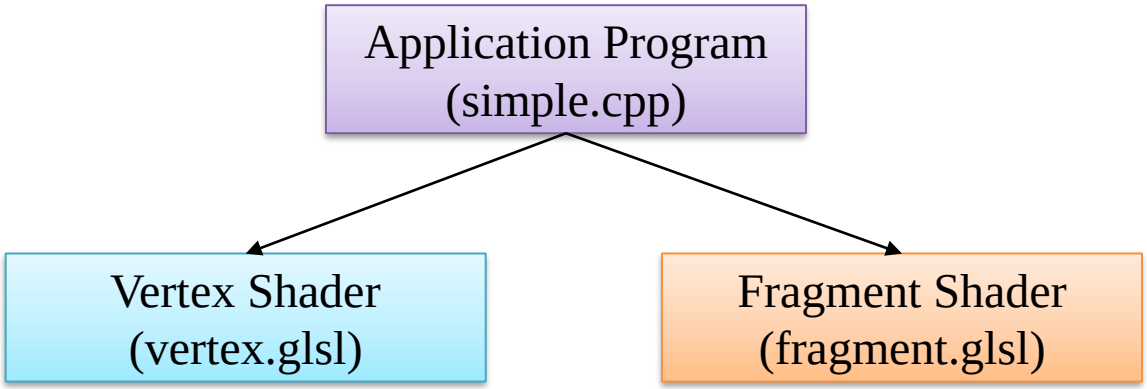


initShader( )

- Connects and links vertex, fragment shaders
- Links variables in application with variables in shaders
  - Vertex attributes
  - Uniform variables

```
// Load shaders and use the resulting shader program
    GLuint program = InitShader( "vertex.glsl", "fragment.glsl" );
    glUseProgram( program );
```

Shaders and Application Program Must Work Together



initShader( )

- Connects and links vertex, fragment shaders
- Links variables in application with variables in shaders
  - Vertex attributes
  - Uniform variables

Compiler puts all variables declared in shader into a table

Vertex shader file

```
in vec4 vPosition
```

Location of vPosition

Variable
Variable 1
vPosition
.....
Variable N

Shaders and Application Program Must Work Together

Vertex shader file

```
in vec4 vPosition
```

Location of vPosition →

Variable
Variable 1
vPosition
.....
Variable N

1-Get index/location of vertex attribute **vPosition**

```
GLuint vPos = glGetAttribLocation( program, "vPosition" );
glEnableVertexAttribArray( vPos );
```

2-Enable vertex array attribute at index/location of **vPosition**

3-Describe the form of data in the vertex array

```
glVertexAttribPointer( vPos, 3, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0) );
```

- Address in the buffer, where data begins
- Location of vPosition
- 3 (x,y,z) floats per vertex
- Data no normalized (0-1 range)
- Stride/Data is contiguous

Reference to *attribute* (or *in*) variables – An Example

- **In application program (in function init()):**

```
#define BUFFER_OFFSET( offset )
        ((GLvoid*) (offset))

GLuint loc = glGetAttribLocation( program, "vPosition" );
glEnableVertexAttribArray( loc );
glVertexAttribPointer( loc, 3, GL_FLOAT, GL_FALSE, 0,
                BUFFER_OFFSET(0) );
```

The application program can refer to the vertex attribute via this index

- **In vertex shader:**

```
in vec3 vPosition;
```

Must be the same

Reference to *attribute* (or *in*) variables – Another Example

- **In application program (in function init()):**

```
GLuint loc, loc2;

loc = glGetAttribLocation(program, "vPosition");
glEnableVertexAttribArray(loc);
glVertexAttribPointer(loc, 3, GL_FLOAT, GL_FALSE, 0,
    BUFFER_OFFSET(0));

loc2 = glGetAttribLocation(program, "vColor");
glEnableVertexAttribArray(loc2);
glVertexAttribPointer(loc2, 3, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(sizeofpoints));
```

// vPosition and vColor are *in* variables in the vertex shader

- **In vertex shader:**

```
in vec3 vPosition;
in vec3 vColor;
```

Reference to *uniform* Variables – An Example

Sometimes we want to connect variables in OpenGL application to uniform variable in shader

- **In application program (init()):**

```
/* my_angle set in application */
GLfloat my_angle;
my_angle = 5.0 /* or some other value */
```

- Declare a variable in the application program
- Assign it a value

```
GLuint angleParam;
angleParam = glGetUniformLocation(myProgObj,"angle");
```

- find location of shader "**angle**" variable in linker table

```
glUniform1f(angleParam, my_angle);
```

- Connect: location of shader variable shader "**angle**" to application variable "my_angle"

The data type must be consistent

- **In vertex shader:**

```
uniform float angle;
```

- Declare a uniform variable in the shader

Reference to *uniform* Variables – An Example

Sometimes we want to connect variables in OpenGL application to uniform variable in shader

- **In application program (init()):**

```
/* my_angle set in application */
GLfloat my_angle;
my_angle = 5.0 /* or some other value */

GLuint angleParam;
angleParam = glGetUniformLocation(myProgObj,"angle");

glUniform1f(angleParam, my_angle);
```

- **In vertex shader:**

```
uniform float angle;
```

This line appears in the **display** callback function also, as the new value of *my_angle* computed in the application program for every frame needs to be copied to the vertex shader.

Further Reading

"Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL" by Edward Angel and Dave Shreiner, 6th Ed, 2012

- Sec2. 2.8.2-2.8.5
  The Vertex Shader …The InitShader Function
- Sec 3.12.2 Uniform Variables

A good reference on OpenGL shaders:
http://antongerdelan.net/opengl/shaders.html