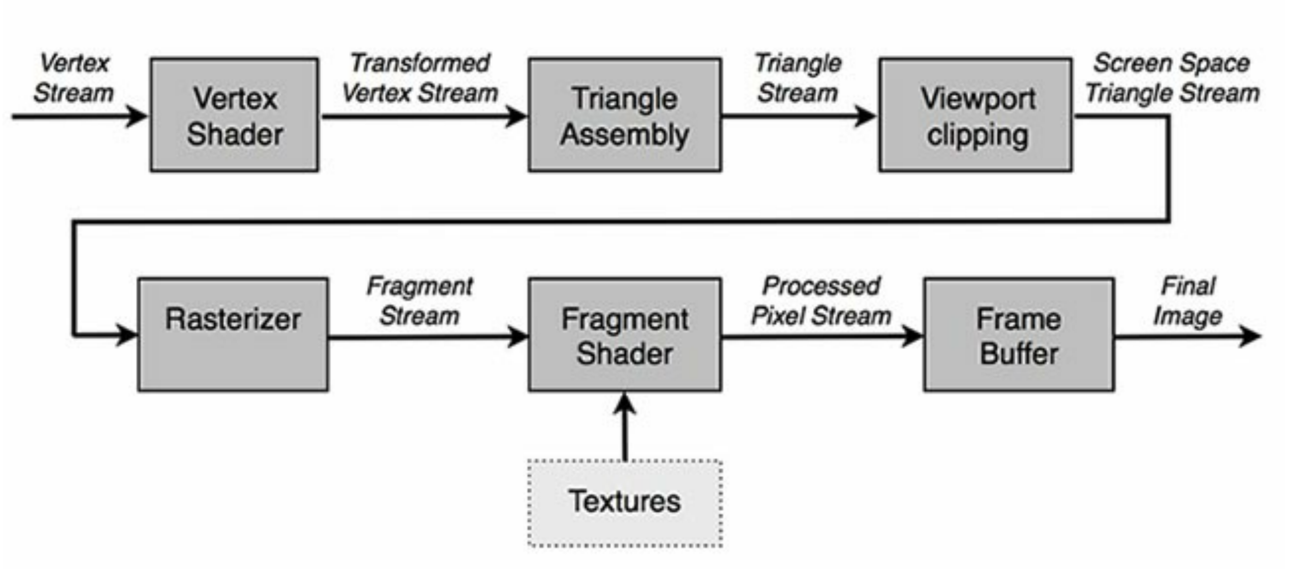
OpenGL Shader & GLSL

Shader is used to program the GPU rendering pipeline.
GLSL is a kind of shader language.

What can shader do?

There are lots of effects written by WebGL in fragment shader.
http://glslsandbox.com/?page=1

How it works?



[Rendering Pipeline Overview (https://www.opengl.org/wiki/Rendering_Pipeline_Overview)](https://www.opengl.org/wiki/Rendering_Pipeline_Overview)

Getting started

- Shader & Shader Program
- Uniform, Attribute, Varying (Type Qualifiers)
- Data Connection (VBO)
    - VBO
    - Uniform
    - Texture
- GLSL Syntax

Shader & Shader Program

1. `glCreateShader` to get a shader id.
2. `glShaderSource` bind shader source to a shader.

3. `glCompileShader` to compile a shader.
4. `glCreateProgram` to create a shader program.
5. `glAttachShader` to attach shaders onto shader program.
6. `glLinkProgram` to link program.
7. `glDetachShader` to detach shaders after a successful link.

```
GLuint vert = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vert, 1, "...void main(){...}", 0);
glCompileShader(vert);

GLuint frag = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(frag, 1, "...void main(){...}", 0);
glCompileShader(frag);

GLuint program = glCreateProgram();
glAttachShader(program, vert);
glAttachShader(program, frag);
glLinkProgram(program);
glDetachShader(program, vert);
glDetachShader(program, frag);
```

```
void display()
{
    glUseProgram(program);
    /* Shader program affect in this block. */
    glUseProgram(0);

    glUseProgram(another_program);
    /* Another shader program effect. */
    glUseProgram(0);
}
```

Type Qualifiers

- Uniform
  The value likes a constant in a draw call.

- Attribute (deprecated now)
  The value is different from each vertex.

- Varying (deprecated now)
  The value can be changed in shader pipeline.

```
// vertex shader old syntex
uniform mat4 ModelViewMatrix;
uniform float delta;

attribute vec3 Position;
attribute vec3 Normal;

varying vec3 vNormal;
varying vec2 vTexcoord;
varying float beta;

void main() {
    //...
}
```

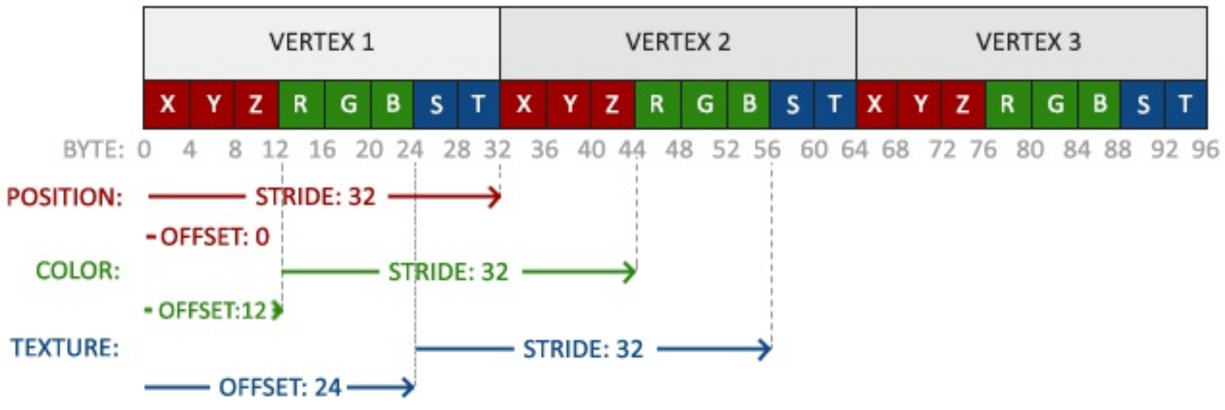
```
// fragment shader old syntex
uniform float gamma;

varying vec3 vNormal;

void main() {
    //...
}
```

Data connection (VBO)

Vertex Buffer Object


VERTEX 1
VERTEX 2
VERTEX 3
X Y Z R G B S T X Y Z R G B S T X Y Z R G B S T
BYTE: 0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96
POSITION:
STRIDE: 32
OFFSET: 0
COLOR:
STRIDE: 32
OFFSET:12
TEXTURE:
STRIDE: 32
OFFSET: 24

```
glGenBuffers(1, &vboName);
glBindBuffer(GL_ARRAY_BUFFER, vboName);

VertexAttribute *vertices = //...
glBufferData(GL_ARRAY_BUFFER,
    sizeof(VertexAttribute) * vertices_length,
    vertices,
    GL_STATIC_DRAW);

glEnableVertexAttribArray(0);
glVertexAttribPointer(0,
    3,
    GL_FLOAT,
    GL_FALSE,
    sizeof(VertexAttribute),
    (void*)(offsetof(VertexAttribute, position)));
```

```
// vertex shader modern syntex
#version 450
layout(location = 0) in vec3 pos;

void main() {
    gl_Position = vec4(pos, 1.0);
}
```

:::warning
`0` in line10, 11 of GL code and `= 0` in line 3 above should be the same.
:::

[offsetof (http://www.cplusplus.com/reference/cstddef/offsetof)](http://www.cplusplus.com/reference/cstddef/offsetof)

[glGenBuffers (https://www.opengl.org/sdk/docs/man4/html/glGenBuffers.xhtml)](https://www.opengl.org/sdk/docs/man4/html/glGenBuffers.xhtml)

[glBindBuffer (https://www.opengl.org/sdk/docs/man4/html/glBindBuffer.xhtml)](https://www.opengl.org/sdk/docs/man4/html/glBindBuffer.xhtml)

[glBufferData (https://www.opengl.org/sdk/docs/man4/html/glBufferData.xhtml)](https://www.opengl.org/sdk/docs/man4/html/glBufferData.xhtml)

[glEnableVertexAttribArray (https://www.opengl.org/sdk/docs/man4/html/glEnableVertexAttribArray.xhtml)](https://www.opengl.org/sdk/docs/man4/html/glEnableVertexAttribArray.xhtml)

[glVertexAttribPointer (https://www.opengl.org/sdk/docs/man4/html/glVertexAttribPointer.xhtml)](https://www.opengl.org/sdk/docs/man4/html/glVertexAttribPointer.xhtml)

Data connection (Uniform)

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45.0, 1.0, 1e-2, 1e2);
GLfloat mtx[16];
glGetFloatv (GL_PROJECTION_MATRIX, mtx);
GLint loc = glGetUniformLocation(program, "Projection");

glUseProgram(program);
    glUniformMatrix4fv(loc, 1, GL_FALSE, mtx);
glUseProgram(NULL);
```

:::warning
`"Projection"` in line 6 correspond with `Projection` in line 5 in shader.
:::

```
// vertex shader modern syntex
#version 450
layout(location = 0) in vec3 pos;

uniform mat4 Projection;

void main() {
    gl_Position = Projection * vec4(pos, 1.0);
}
```

---

glGet (https://www.opengl.org/sdk/docs/man4/html/glGet.xhtml)
glUniform (https://www.opengl.org/sdk/docs/man4/html/glUniform.xhtml)
glGetUniformLocation
(https://www.opengl.org/sdk/docs/man4/html/glGetUniformLocation.xhtml)
gluPerspective (https://www.opengl.org/sdk/docs/man2/xhtml/gluPerspective.xml)

Data connection (Texture)

```
GLuint tex = glGenTextures(...);
/* You can handle it yourself,
 *   or use the GLMtexture.id by glm library */
glEnable(GL_TEXTURE_2D);

GLint loc = glGetUniformLocation(program, "MyTexture_1");

glUseProgram(program);
    glActiveTexture(GL_TEXTURE0 + 0);
    glBindTexture(GL_TEXTURE_2D, tex);

    glUniform1i(loc, 0);

    draw_call();
    glBindTexture(GL_TEXTURE_2D, NULL);
glUseProgram(NULL);
```

:::warning
`+ 0` in line 9 correspond with `0` in line 12.
:::

```
// fragment shader modern syntex
#version 450
layout(binding = 0) uniform sampler2D MyTexture_1;

in vec2 texcoord;
out vec4 outColor;

void main() {
    outColor = texture2D(MyTexture_1, texcoord);
}
```

:::warning
`= 0` in line 3 correspond with above code.
:::

GLSL Syntax

Basic variable types

```
vec2, vec3, vec4, ...
mat2, mat3, mat4, ...
float, int, bool, ...
sampler2D, ...
```

Basic functions

```
max, min, sin, cos, pow, log, ..
dot, normalize, reflect, ...
transpose, inverse, ...
+, -, *, /
```

Basic manipulation

```
vec3 c8 = vec3(1.0, 1.0, 9.0);
vec4 v2 = (c8 * 7.63).xyxz + vec4(5.0, 5.0, 6.0, 6.0);
vec3 c1 = normalize(v2.rgb);
mat3x2 m1 = mat3x2(c8, c1); // column major
mat4 m2 = mat4(v2, vec4(1.0), vec4(c8, 0.0), c1.gggg);
mat3 m3 = mat3(m2); // get left top of mat4
ivec2 iv1 = ivec2(c8.z, c1.r);
```

Vertex shader

- `gl_Position` is need
- `attribute` is alias to `in`
- `varying` is alias to `out`
- use `flat` for `out` if flat interpolation
    - e.g. `flat out float delta;`

Fragment shader

- the LAST `out vec4` in main loop is need for color to frame-buffer
- `varying` is alias to `in`

Example

```
// outline vertex shader
#version 450

layout(location = 5) in vec4 in_Pos;
layout(location = 6) in vec4 in_Norm;

uniform mat4 MV;
uniform mat4 P;

out vec3 normal;

void main() {
    gl_Position = P * MV * in_Pos;
    normal = vec3(/* calculate normal after modelview transform */);
}
```

```
// outline fragment shader
#version 450

in vec3 normal;
out vec4 outColor;

void main() {
    if(abs(normal.z) < 0.3) {
        outColor = vec4(1.0);
    } else {
        outColor = vec4(1.0, vec2(0.0), 1.0);
    }
}
```

{%youtube Nmxfli_PY0U %}

Resources

Models (http://graphics.cs.williams.edu/data/meshes.xml)
OpenGL tutorial (http://learnopengl.com/)
OpenGL tutorial (CN) (https://learnopengl-cn.github.io/)
OBJ file format (https://en.wikipedia.org/wiki/Wavefront_.obj_file)
OpenGL wiki & GLSL common mistakes
(https://www.opengl.org/wiki/GLSL_:_common_mistakes)