OPENGL SHADERS

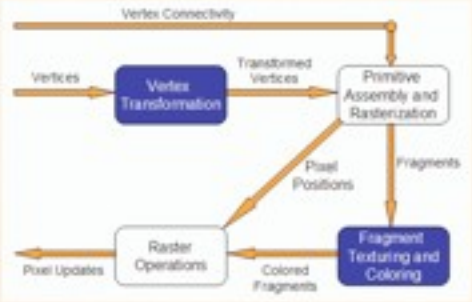
–

HTTP://WWW.LIGHTHOUSE3D.COM/TUTORIALS/GLSL-TUTORIAL/

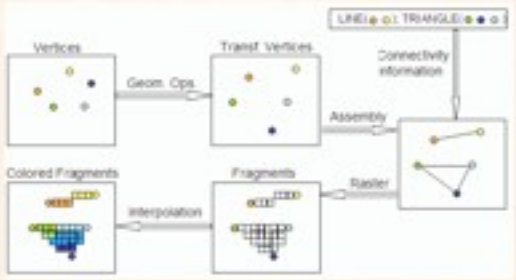
WHAT?

- GLSL 1.2 - GL Shading Language
- Cg - Nvidia's shading language
- ARB extensions (Architecture Review Board)
- OpenGL 2.0

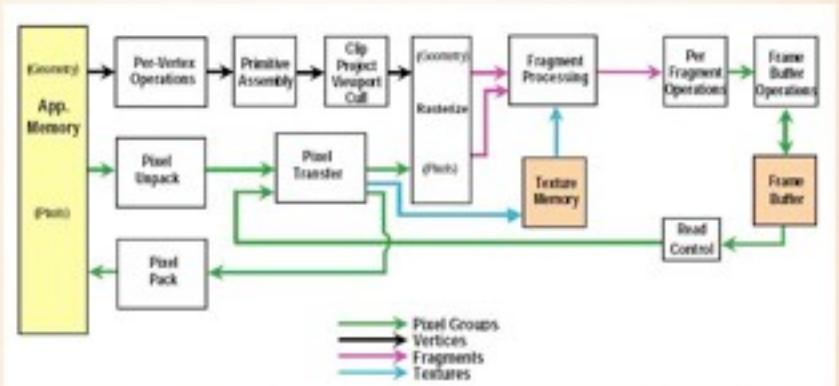
PIPELINE OVERVIEW



VISUAL PIPELINE



FIXED FUNCTION PIPELINE



FIXED V. PROGRAMMABLE

- standard OpenGL: Fixed-function pipeline
- add more user control & flexibility: programmable
- Pipeline processing - 2 stages
  - vertex processors
  - fragment processors

VERTEX PROCESSOR

- vertex shader executed once for each vertex
- vertex position transformation usually using the modelview and projection matrices
- normal transformation, normalization
- texture coordinate generation and transformation
- lighting per vertex
- color computation

FRAGMENT PROCESSOR

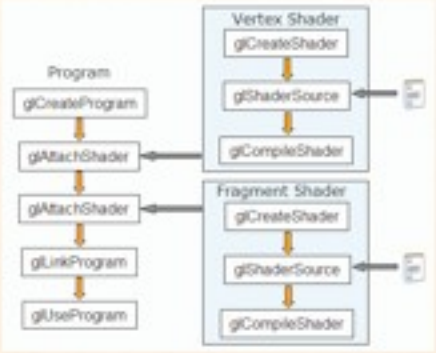
- fragment - per pixel data
- fragment shader executed once for each fragment
- computing colors and texture coordinates per pixel
- texture application
- fog computation
- computing normals for lighting per pixel
- can discard the fragment or compute color

SETUP FOR GLSL

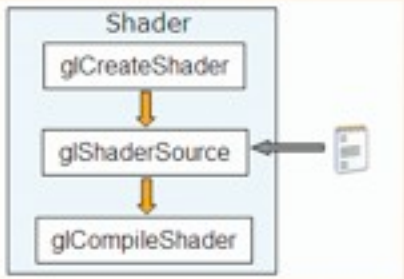
- Each shader like a C module
  - compiled separately
  - linked to OpenGL program
  - example brick shader:

ptgmedia.pearsoncmg.com/images/0321334892/samplechapter/rost_ch06.pdf

PROCESS OVERVIEW



CREATING SHADERS

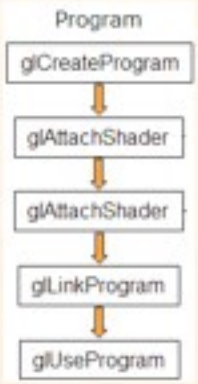


PROCESS OVERVIEW

```
void glShaderSource(GLuint shader, int numOfStrings, const char **strings, int *lenOfStrings);
```

- shader – the handler to the shader.
- numOfStrings – the number of strings in the array.
- strings – the array of strings.
- lenOfStrings – an array with the length of each string, or NULL, meaning that the strings are NULL terminated.

INCORPORATING SHADERS




```
void setShaders() {

    char *vs,*fs;

    v = glCreateShader(GL_VERTEX_SHADER);
    f = glCreateShader(GL_FRAGMENT_SHADER);

    vs = textFileRead("toon.vert");
    fs = textFileRead("toon.frag");

    const char * vv = vs;
    const char * ff = fs;

    glShaderSource(v, 1, &vv,NULL);
    glShaderSource(f, 1, &ff,NULL);

    free(vs);free(fs);

    glCompileShader(v);
    glCompileShader(f);

    p = glCreateProgram();

    glAttachShader(p,v);
    glAttachShader(p,f);

    glLinkProgram(p);
    glUseProgram(p);
}
```

DEBUGGING

Is hard
- no printf
- functions to test compilation & linking
e.g.

```
void glGetShaderiv(GLuint object, GLenum type, int *param);
```

Can fetch an 'infoLog' to get more about errors

GLSL VARIABLES

Read-only in shader

value set by program

Uniform

defined for a primitive (outside glBegin-glEnd)

not on a per Vertex basis

Attribute

on a per Vertex basis - for vertex shaders

GLSL VARIABLES

bool	vec{2,3,4}	mat2	const
float	bvec{2,3,4}	mat3	attribute
int	ivec{2,3,4}	mat4	uniform
			varying

sampler{1D,2D,3D,Cube,1DShadow,2DShadow}

GLSL STATEMENTS

if - else	
for loop	continue
while	break
do-while	discard

each shader - one main()

declare in / out parameters

BRICK VERTEX SHADER

```
uniform vec3 LightPosition;
const float SpecularContribution = 0.3;
const float DiffuseContribution = 1.0 - SpecularContribution;
varying float LightIntensity;
varying vec2 MCposition;

void main() {
    vec3 ecPosition = vec3(gl_ModelViewMatrix * gl_Vertex);
    vec3 tnorm = normalize(gl_NormalMatrix * gl_Normal);
    vec3 lightVec = = normalize(LightPosition - ecPosition);
    vec3 reflectVec = reflect(-lightVec, tnorm);
    vec3 viewVec = normalize(-ecPosition);
    float diffuse = max(dot(lightVec, tnorm), 0.0);
    float spec = 0.0;

    if (diffuse > 0.0) {
        spec = max(dot(reflectVec, viewVec), 0.0);
        spec = pow(spec, 16.0);
    }
    LightIntensity = DiffuseContribution * diffuse + SpecularContribution * spec;
    MCposition   = gl_Position   = gl_Vertex.xy; ftransform();
}
```

BRICK FRAGMENT SHADER

```
uniform vec3 BrickColor, MortarColor;
uniform vec2 BrickSize;
uniform vec2 BrickPct;
varying vec2 MCposition;
varying float LightIntensity;

void main()
{
    vec3 color;
    vec2 position, useBrick;
    position = MCposition / BrickSize;
    if (fract(position.y * 0.5) > 0.5) position.x += 0.5;
    position = fract(position);
    useBrick = step(position, BrickPct);
    color = mix(MortarColor, BrickColor, useBrick.x * useBrick.y);
    color *= LightIntensity;
    gl_FragColor = vec4(color, 1.0);
}
```