CS418
GLSL Shader Programming Tutorial
(I)
Shader Introduction

Presented by : Wei-Wen Feng
3/12/2008

MP3

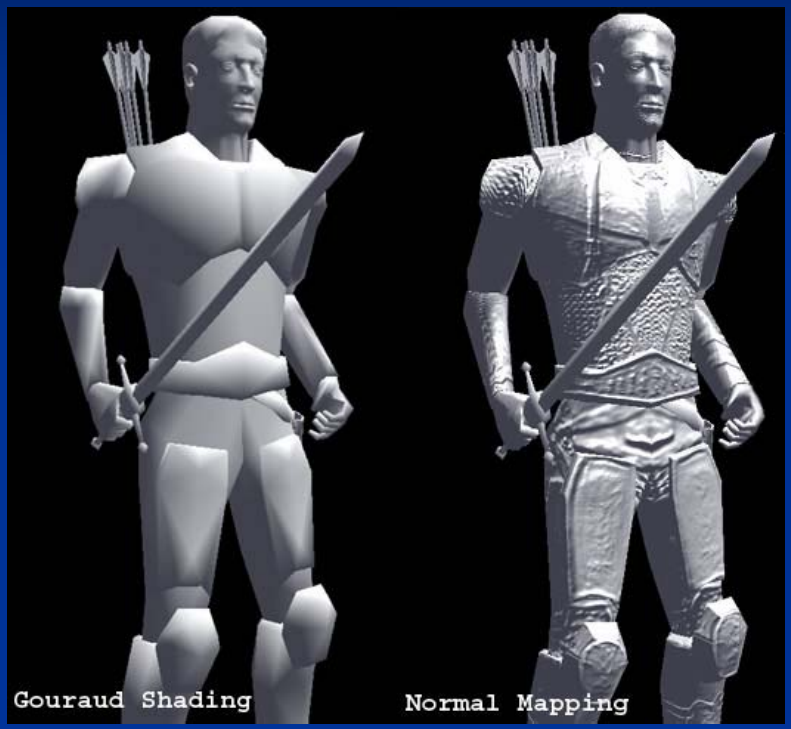
- MP3 will be available around Spring Break
  - Don't worry, you got two more weeks after break.
  - Use shader to do Phong shading, Vertex Transform, etc.
  - Shader programming might be tricky at first.
  - **Start earlier to setup the skeleton code and play with it.**

Shader Programming

- Programmable shader becomes important for real-time graphics application (Games, etc)

- Why you'd want to learn this
  - MP3, obviously ;-)
  - If you want to seriously do graphics
  - You can make use of it even for non-graphics application. → GPGPU

Shader Power Includes….



Normal Mapping

Real-Time Procedural Geometry
(Geforce 8800)

Shader Power Includes….

Comparison between ray-traced and rasterized image (photo: Intel)

Real-Time Ray Tracing

Real-Time Fluid Simulation

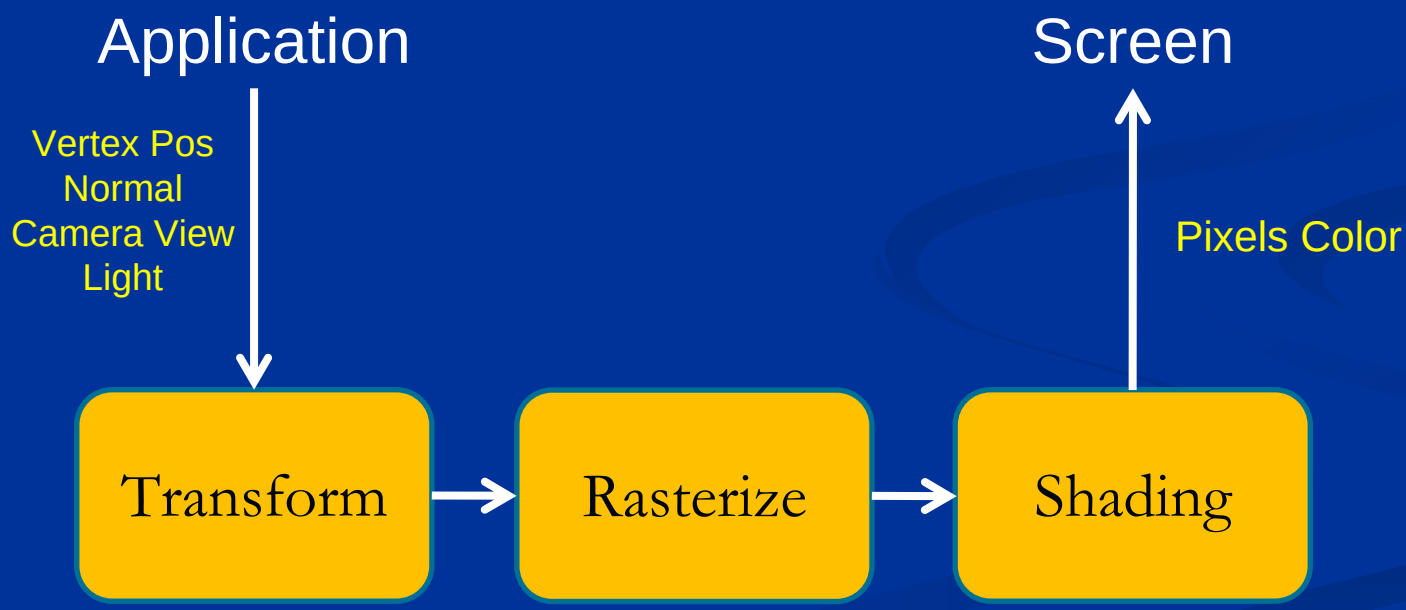
And a lot more !

Today's Agenda

- Shader Concept
- HelloShader
- Setup Your Shader
- Basic Data Types
- Uniform, Attributes, and Varying
- Examples
- Q & A

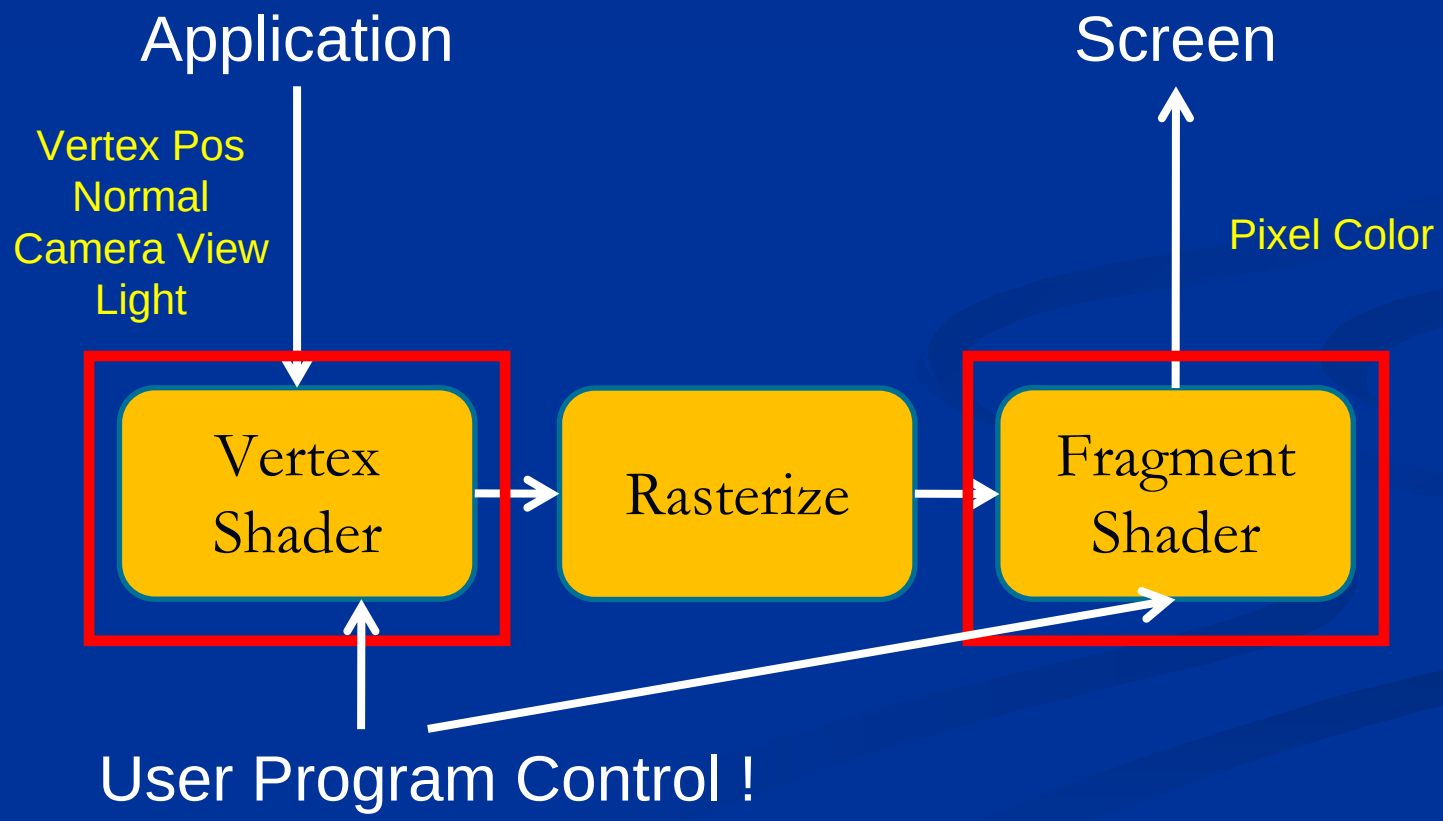
Shader Concept

- OpenGL Fixed Graphics Pipeline :



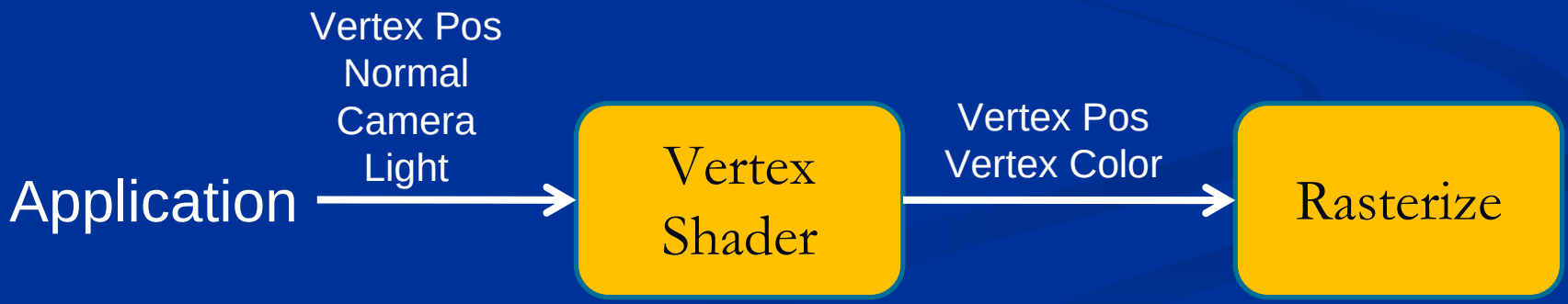
Shader Concept

- Programmable Shader :



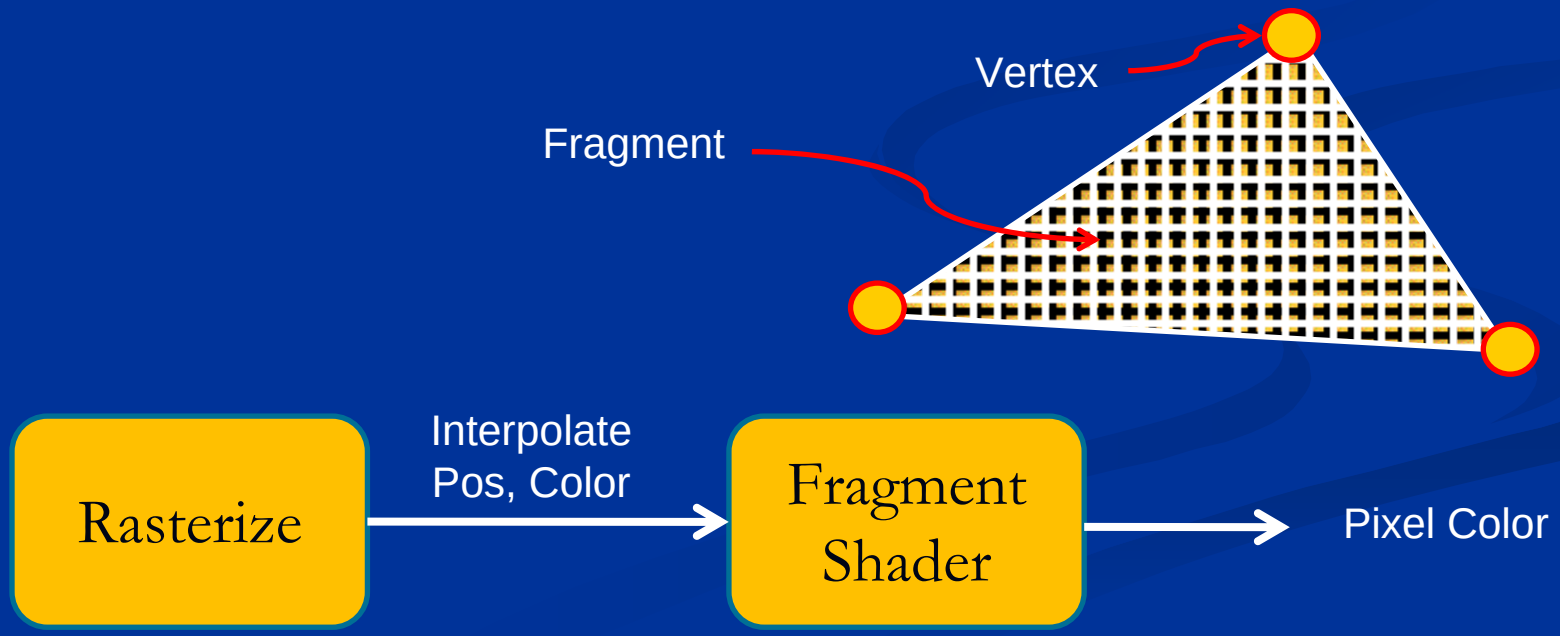
Shader Concept

- Vertex Shader
  - Execute on “Every” vertex.
  - Write your program to replace OpenGL fixed-function
  - You can transform vertex position, compute per-vertex attributes (Ex : color).

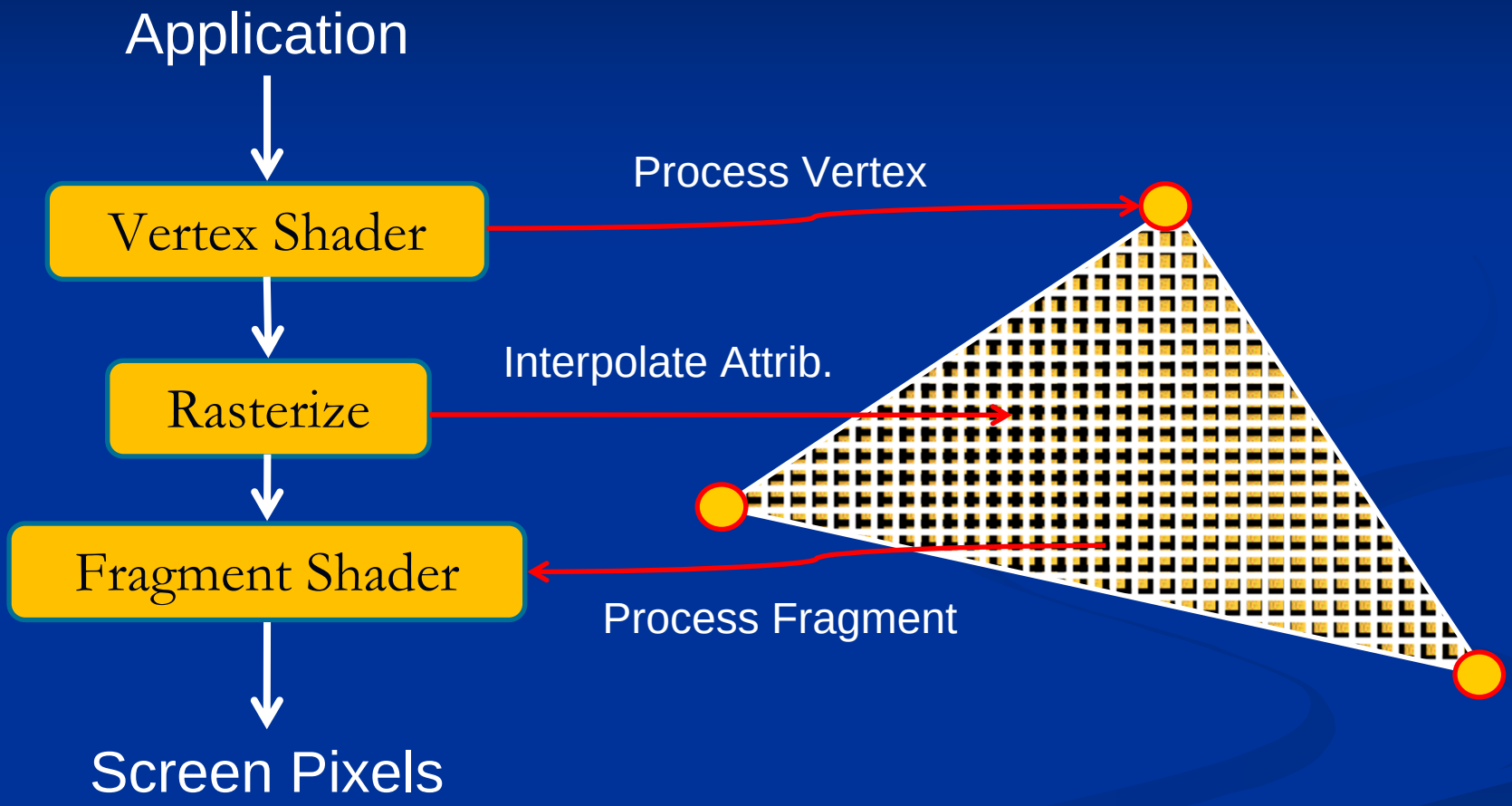


Shader Concept

- Fragment Shader (Pixel Shader)
  - Execute for each “Fragment”
  - “Fragments” are pixels from a triangle after rasterize.
  - Output final color on Screen.



Shader Concept



Shader Concept

- Shader Programming Language
  - HLSL : Microsoft DirectX → Game programming
  - Cg : nVidia graphics hardware → Vendor specific
  - GLSL : OpenGL 2.0 standard.
  - We will use GLSL for tutorial and MPs.

Shader Programming

- C-like syntax for programming.

- More difficult to program than usual C/C++
  - No “printf” type debug. You only see “Colors”.
  - Respect rules for input/output and data access.
  - Think in parallel. Same code will apply to all vertices/fragments.

Hello-Shader

```
// Vertex Shader
void main()
{
    gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

```
// Fragment Shader
void main()
{
    gl_FragColor = vec4(1, 0, 0, 1);
}
```

Setup Your Shader

- We need OpenGL 2.0 for GLSL functionality
  - It's not out of box in Visual Studio.
  - MS has no interest to include OpenGL 2.0 in VS.
- OpenGL Extension
  - OpenGL keeps evolve new functions.
  - Extension is used before whole API revision.
- GLEW : OpenGL extension Wrangler
  - http://glew.sourceforge.net/index.html
  - Easy way to setup GL extension libs.
  - Access GL shader APIs through it.

GLEW

- Include “glew.h” instead of “gl.h”
- glewInit()
  - Call it right after you setup GL context.
  - Or it will crash on you.
  - After that, you can call extension APIs as usual.

```
#include <GL/glew.h>
#include <GL/glut.h>
...
glutInit(&argc, argv);
glutCreateWindow(“Test");

GLenum err = glewInit();
```

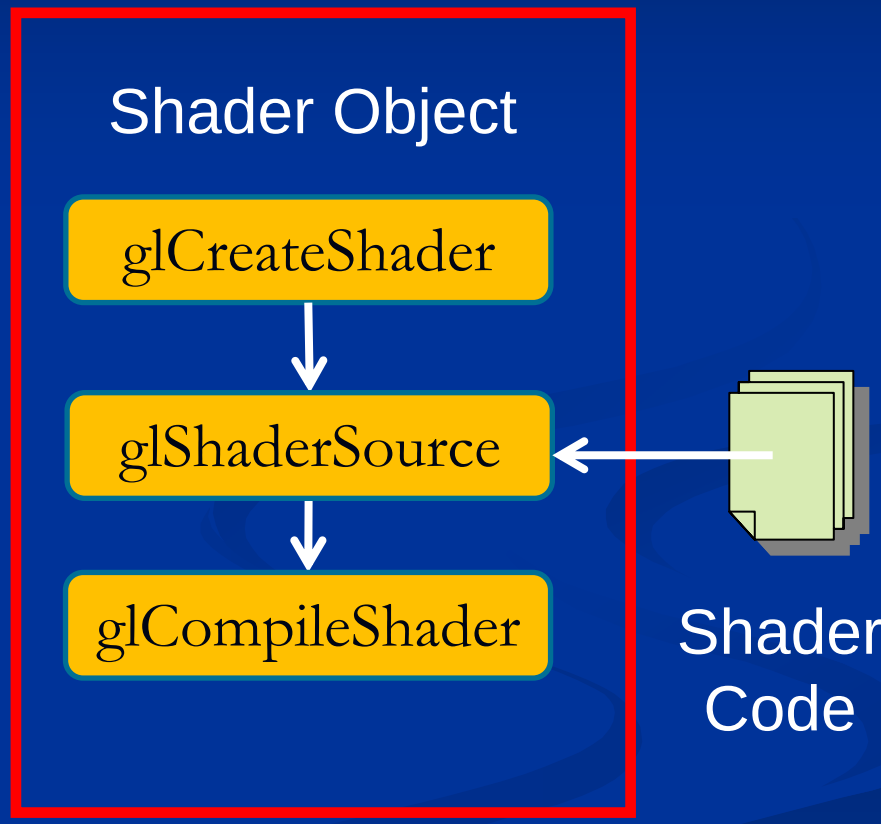
Setup Shader Programming

- Like OpenGL lighting/texture, its setup is a bit messy. But it's a one-time headache.

- Basic flow :
  - Write your shader code in a text editor.
  - Make shader setup in OpenGL code.
  - Load / build shader source code from file.
  - Apply shader program to your rendering pipeline.

Setup Shader

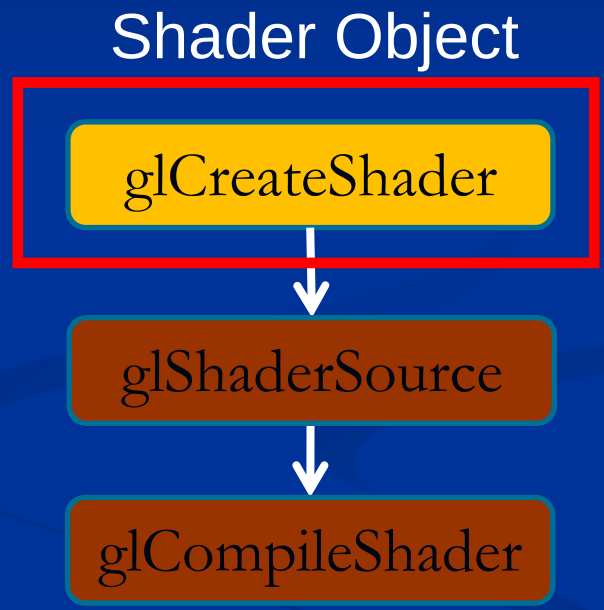
- Shader
  - Vertex shader
  - Fragment shader
- Create a shader
  - Create object
  - Set source code
  - Compile code



Setup Shader

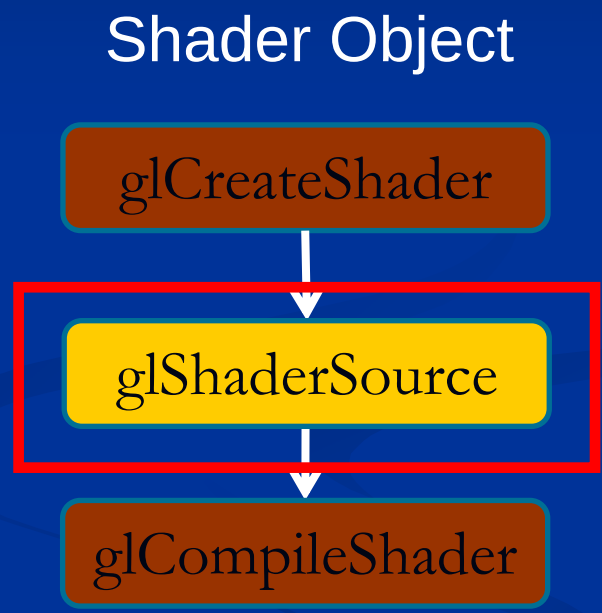
- glCreateShader(GLenum Type)
- Create a shader container for use.
- Return the handle (ID) to a new shader.

- Type :
  - GL_VERTEX_SHADER
  - GL_FRAGMENT_SHADER



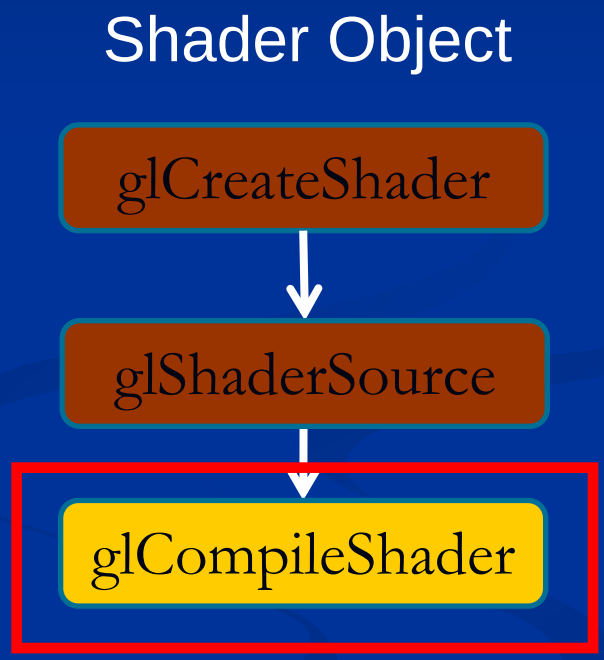
Setup Shader

- glShaderSource(int **shaderobject**,
  int **numOfStrings**,
  const char ****strings**,
  int ***lenOfStrings**)
- Specify source code for a shader.
- What I usually do :
  - Edit and save shader code to text file.
  - Load the text file and store them in ***char array***
  - Call glShaderSource and pass in strings loaded from file.
- Don't try to specify a **vertex shader** code to a **fragment shader** object.



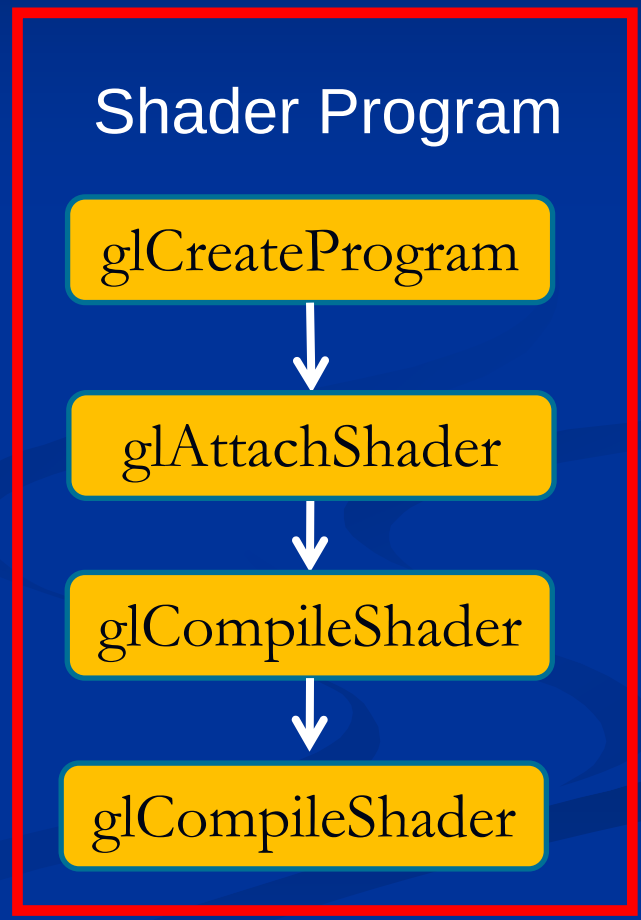
Setup Shader

- glCompileShader(int **shaderobject**)
- Compile the source code in shader object before you can use it.
- Be sure to check for compiling error.
- Use glGetShaderInfoLog to get compiler errors for shader.
- If you use a faulty shader, OpenGL might reset to fixed pipeline, or ….crash.



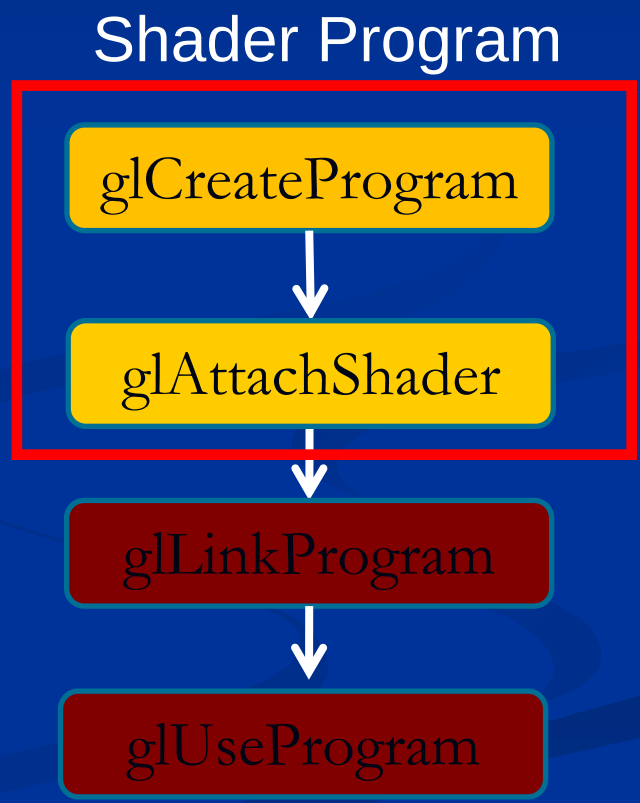
Setup Program

- Shader Program
  - A container holds multiple shaders.
  - OpenGL will execute your "program" on graphics card.
  - You don't need to have both "VS" and "FS" in program.
  - Fixed pipeline will be used in place of missing shader.



Setup Program

- glCreateProgram(int program)
  - Create a container and return its ID.
- glAttachShader(int program, int shader)
  - Add your shader objects to program.
  - More than one shaders can be added.



Shader1 :
```
main()
{
// do transform
}
```

Shader2 :
```
HelpFunc()
{
// do something
}
```

Shader3 :
```
main()
{
// do lighting
}
```

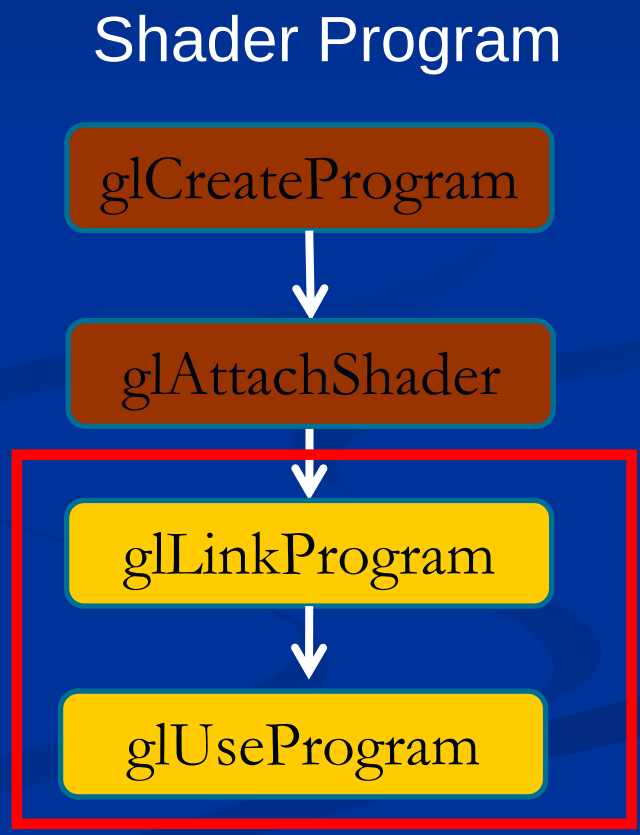
Attach Shader1 & Shader2 →OK

Attach Shader1 & Shader3 → Error !

Setup Program

- glLinkProgram(int program)
  - Link shader objects in a program.
  - Like your C/C++ compiler !
  - You can change source code later, program recompile automatically.

- glUseShader(int program)
  - Apply this program in your rendering.
  - You can create/link multiple programs and switch them.
  - Ex :



GetErrors

- glGetError, gluErrorString
  - Check for OpenGL error, and get Error message.
- glGetShaderInfoLog, glGetProgramInfoLog
  - Check if shader compile correctly.
  - You can't run a C/C++ program with compile error. So does shader.
- We put together some help functions in skeleton code.
  - printShaderInfoLog → shader compile msg
  - printProgramInfoLog → shader program link msg
  - printOpenGLError → OpenGL error

  - Use them, they can save you lots of time !

Source Code

```
vtx_shader = glCreateShader(GL_VERTEX_SHADER);

vs = textFileRead("vs_basic.vert");

const char * vv = vs;

glShaderSource(vtx_shader, 1, &vv,NULL);

glCompileShader(vtx_shader);
printShaderInfoLog(vtx_shader) // check compile msg

shader_program = glCreateProgram();
glAttachShader(shader_program,vtx_shader);

glLinkProgram(shader_program);
printProgramInfoLog(shader_program); // check link msg
glUseProgram(shader_program);
```

Basic Data Type

- GLSL is like C, but with more vector/matrix data types built-in.
- vec2, vec3, vec4 : float vector data
  - ivec for “int” vector.
- mat2, mat3, mat4 : float matrix data
- Each vertex/fragment shader must have an entry point
  - **“void main()”**, just like C !.

```
void main()
{
  mat4 M;
  vec3 t;
  t = vec3(0,0,1);
}
```

Accessing Vector

```
void main()
{
   mat4 M;
   vec3 t;
   t = vec3(0,0,1);
   t[0] = 3; // Now t = (3,0,1)
   t.x = 1;   // t = (1,0,1)
   t.yz = vec2(1,2); // t = (1,1,2)
   t.zx = vec2(3,0); // t = (0,1,3)
}
```

However, don't try to break its logic :

V.xzz = vec3(1,2,3); // Err!

- Swizzling operator let you access particular parts of vec
  - (x,y,z,w) = (r,g,b,a) = (s,t,p,q)
  - V.x = V.r = V.s = V[0]
- Access parts of a vec
  - V.xy = vec2(4,2)
- Can do it in any order :
  - V.zx = vec2(3,0)
- Or repeat the elements
  - V.xyz = U.xxx

Control Flow

- C-like expression for program control
  - if (bool expression) ... else
  - for (int i=0 ;i<n ;i++) loop
  - do…while (bool expression)

  - These conditional branch is much more expensive than in CPU. → Don't overuse it, especially in fragment shader.

Function

- C-like function declaration
- void func_name(in vec3 pin, out vec3 pout)
  - However, no C-like reference to get return values.
    - void func_name(vec3& out) → No available
  - Qualifiers for function parameters :
    - in : for input
    - out : for return values
    - inout : for both input & return values
  - By default, they are input parameters.

Useful Functions

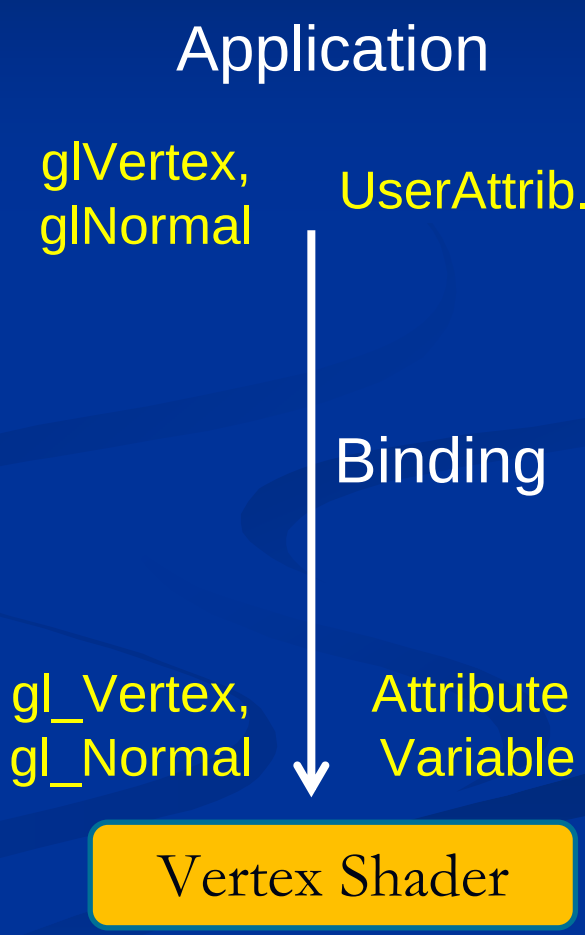
- normalize(v) : unitize a vector
- dot(u,v) : dot product
- ftransform() : return transformed vertex positions (Vertex shader only !)
- reflect(v,n) : get reflected direction (useful for specular !)

- I skip texture access for now. We will discuss it next time.

Variable Types : Attribute

```
attribute vec4 velocity;
void main()
{
   gl_Position = gl_Vertex + velocity*0.1;
}
```

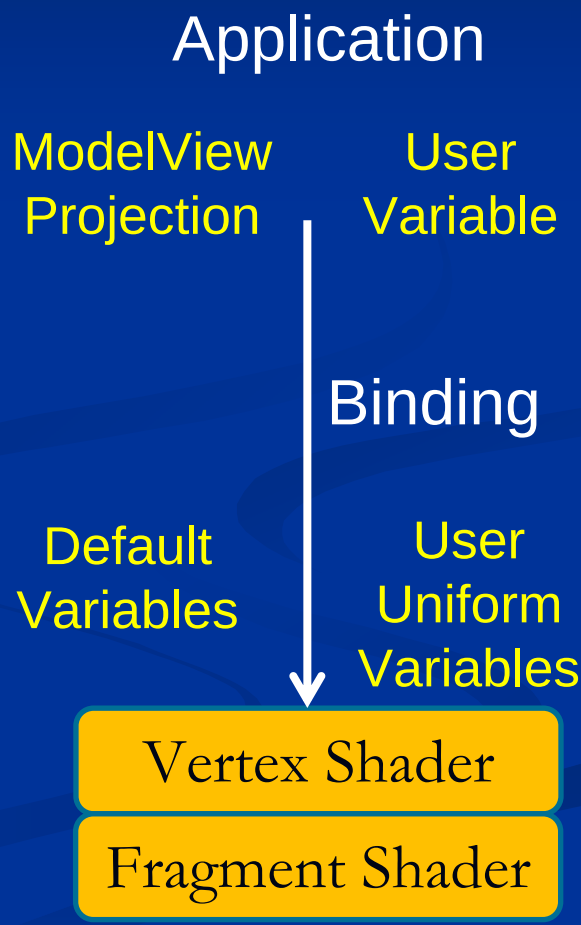
- Attribute variable :
  - Change at each vertex (Position, Color, etc)
  - Define your own attributes in OpenGL.
- Useful default binding :
  - gl_Vertex → glVertex
  - gl_Normal →glNormal
  - gl_Color →glColor



Variable Types : Uniform

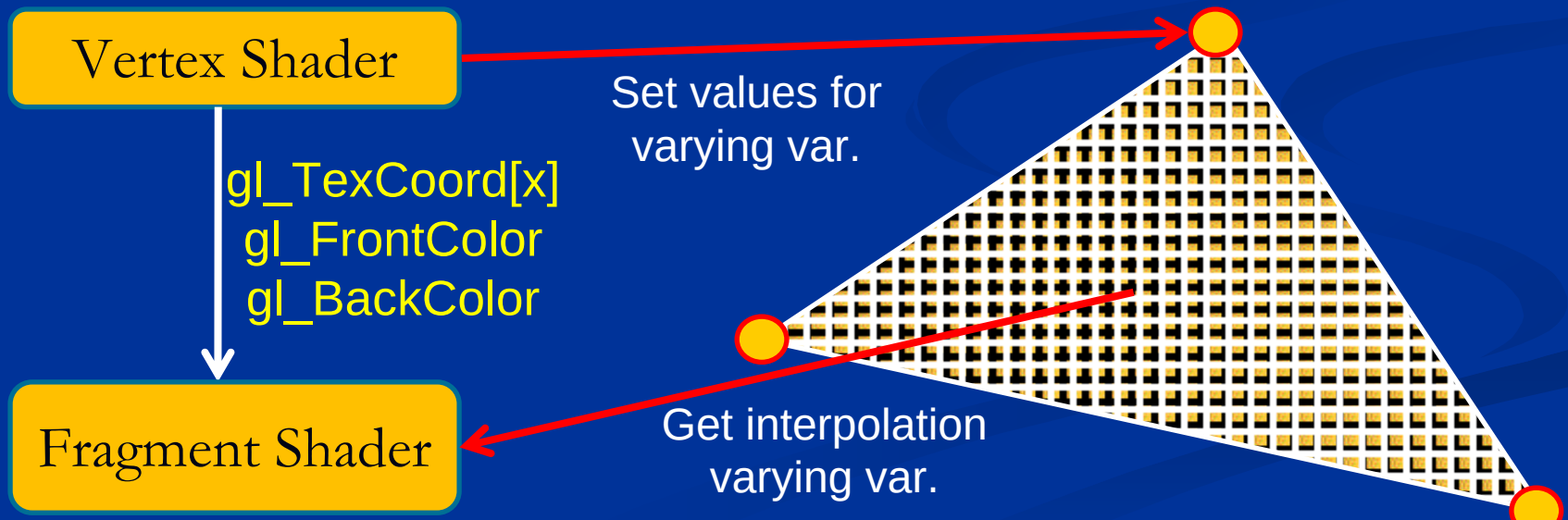
```
attribute vec4 velocity;
uniform float time;
void main()
{
   mat4 m = gl_ModelViewProjectionMatrix;
   gl_Position = m*gl_Vertex + velocity*time;
}
```

- Uniform variable :
  - Information that is constant for an object
  - You can set it for both vertex/fragment shader
- Useful default variable:
  - gl_ModelViewProjectionMatrix
  - gl_NormalMatrix → Transform normal



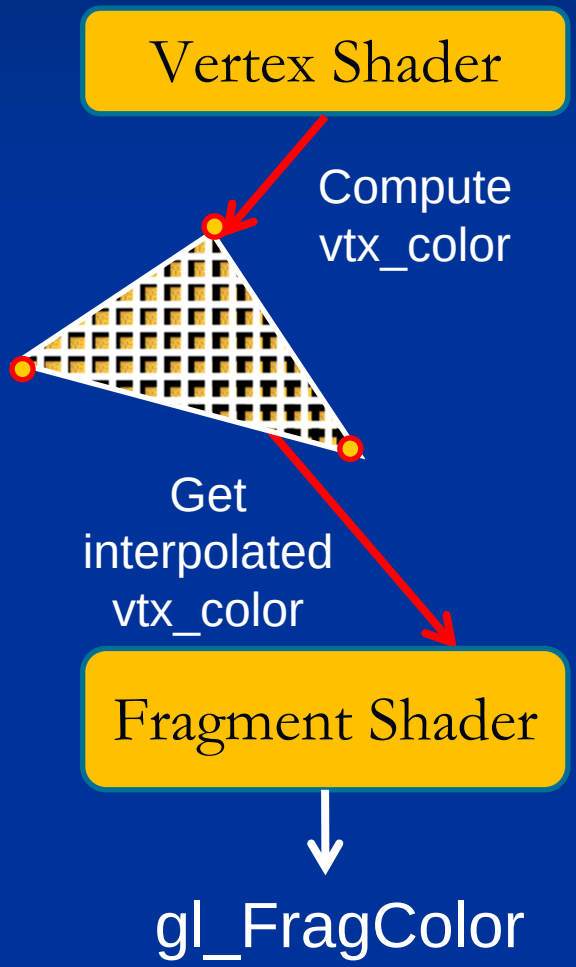
Variable Types : Varying

- Varying variable :
  - Passing data from Vertex shader to Fragment shader.
  - Data that will be interpolated during rasterization.



Variable Types : Varying

A Simple Gouraud Shading

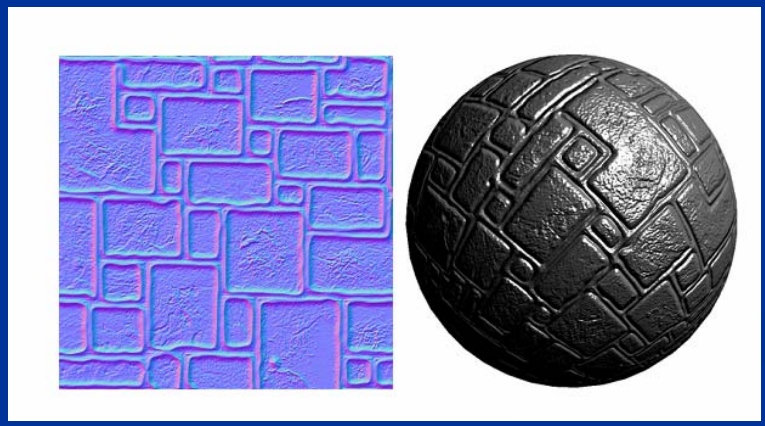


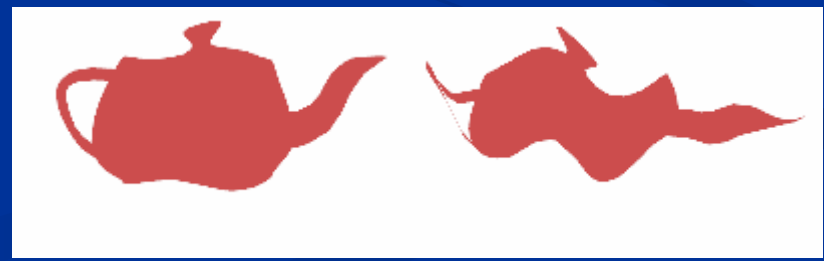

```
// Vertex Shader Program
varying vec4 vtx_color; // to fragment shader
void main()
{
   gl_Position = gl_Vertex;
   vtx_color = dot(gl_Normal,LightDir);
}
```

```
// Fragment Shader Program
varying vec4 vtx_color; // from vertex shader
void main()
{
    gl_FragColor =  vtx_color; // output vtx_color
}
```

Setting Variables

- OpenGL already set up some default variables for you.
  - Uniform : gl_LightSource, gl_***Matrix, etc
  - Attributes : gl_Vertex, gl_Normal, etc.
- You may want to define your own variables
  - Attributes : tangent, binormal direction
  - Uniform : animation time





Setting Uniform

- int glGetUniformLocation(int prog, int Name)
  - Return Location(ID) to a uniform variable
  - Prog : shader program
  - Name : variable name in shader code
- int glUniform (int loc, float value)
  - Set value for Uniform var. in shader

```
//OpenGL code
LocA = glGetUniformLocation(prog,"time");
glUniform1f(LocA,1.0f);

LocB = glGetUniformLocation(prog,"vel");
glUniform3f(LocB, 0.0, 1.0, 0.5);
```

```
//Shader code
uniform float time;
uniform vec3 vel;
void main
{
gl_Position = vel*time;
}
```

Setting Attributes

- int glGetAttribLocation(prog, int name)
  - Return Location(ID) to a attribute variable
  - Prog : Shader program, Name : Attribute name in shader
- int glVertexAttrib
  - Similar to other calls like glColor, glNormal.

```
//OpenGL code
Loc = glGetAttribLocation(prog,"scale");

glBegin(GL_TRIANGLES);
for (int i=0;i<3;i++)
{
   glVertexAttrib1f(Loc,Scale[i]);
   glVertex3fv(Pos[i]);
}
glEnd();
```

```
//Shader code
attribute float scale;
void main
{
gl_Position = gl_Vertex*scale;
}
```

Examples

- Per-pixel phong lighting & vertex animation
  - MP3 requirements
  - Animation by offseting vertex position in vertex shader with sinusoidal function.
  - Per-pixel lighting calculated in fragment shader.

Examples

- Toon Shading
  - Use fixed color to replace original shading color.

Coding for GPU

- Think in parallel when writing shader code
  - The same program run on every vertices/fragments.
  - CPU is not like CPU, don't abuse "if..then..else".

- Coding & Debugging :
  - Always make sure shader is compiled correctly.
  - Make use of default variables like gl_Vertex, gl_Normal, gl_XXMatrix to save your time.
  - Check variables in question → Output it as color.

Coding for GPU

- Common Mistakes :
  - Nothing change when you modify shader code ?
    - Maybe OpenGL already go back to fixed pipeline.
    - Which means something wrong in your shader code
  - Use wrong names in OpenGL to refer Shader variables.
    - That variable has no values.
  - Mismatch vector length
    - gl_Normal : vec3
    - gl_Vertex : vec4

Q&A

- Summary
  - Install GLEW before building sample codes.
  - Try it earlier in case of compatibility issues.
  - MP3 won't be difficult once you get everything setup.