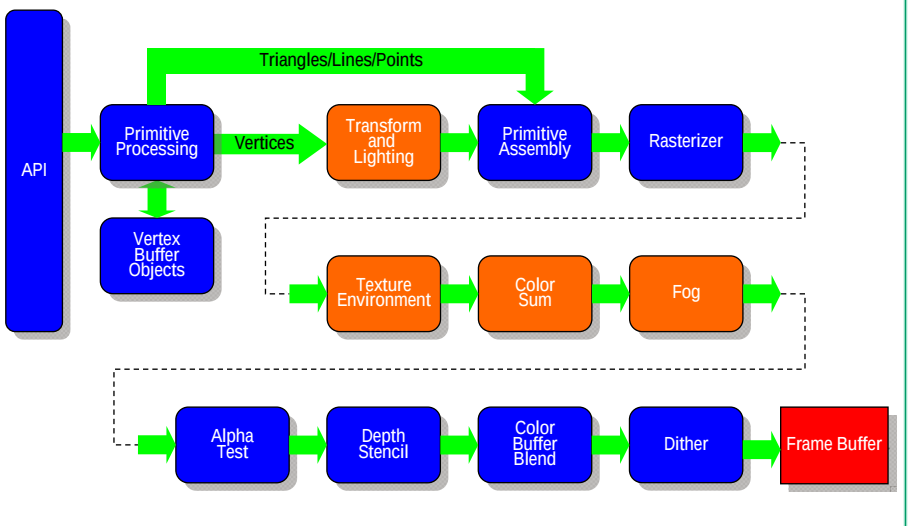
AMD
Smarter Choice

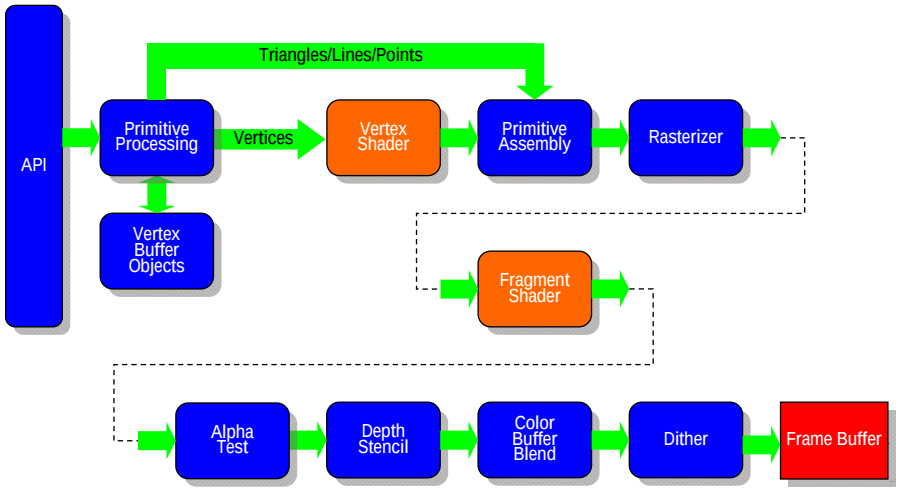
An Introduction to the
OpenGL Shading Language

Benj Lipchak
Rob Simpson
Bill Licea-Kane

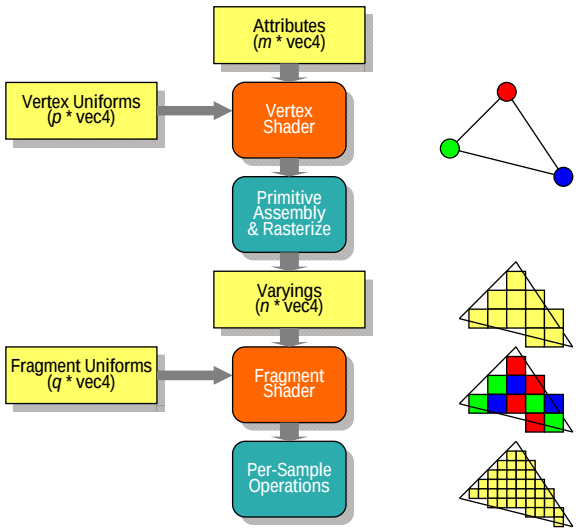
Fixed Functionality Pipeline



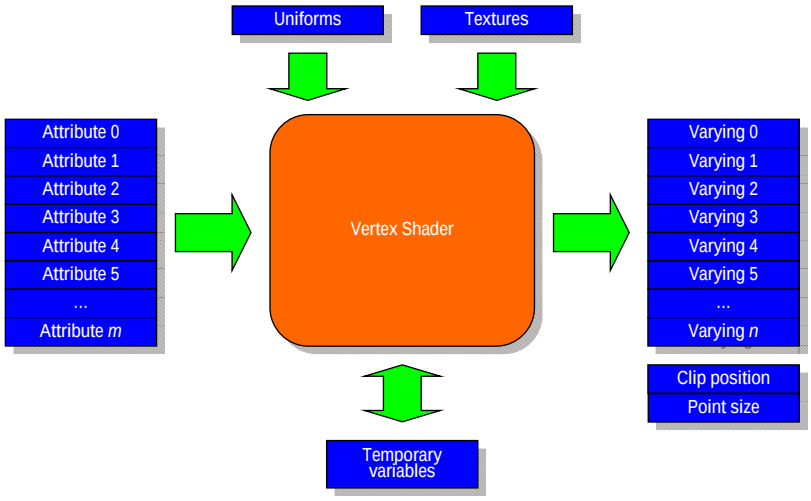
Programmable Shader Pipeline



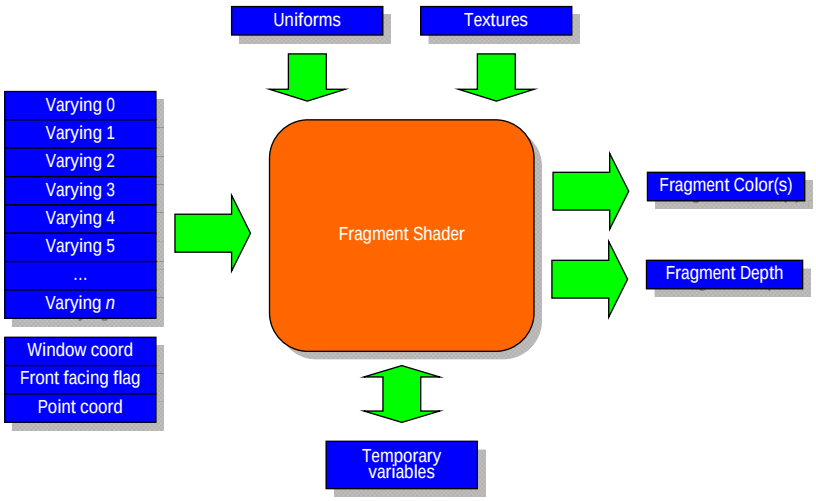
Programmer's Model



Vertex Shader Environment



Fragment Shader Environment



Precursors to GLSL

AMD Smarter Choice

Texture combiners

- EXT_texture_env_combine

Vendor-specific assembly-like programmable shaders

- EXT_vertex_shader
- ATI_fragment_shader, ATI_text_fragment_shader
- NV_*_program*

Standardized low-level programmable shaders

- ARB_vertex_program
- ARB_fragment_program

Not to be confused with GLSL extensions!

- GL_VERTEX_SHADER
- GL_FRAGMENT_SHADER

Hello World!

AMD Smarter Choice

```
void main(void)
{
    // This is our Hello World vertex shader

    // Standard MVP transform
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}

void main(void)
{
    // This is our Hello World fragment shader

    // Set to a constant color (hint: look at it upside down)
    gl_FragColor = vec4(0.7734);
}
```

Language Basics: variables types

Scalar

- `void` `float` `int` `bool`

Vector

- Floating point: `vec2` `vec3` `vec4`
- Integer: `ivec2` `ivec3` `ivec4`
- Boolean: `bvec2` `bvec3` `bvec4`

Matrix

- `mat2` `mat3` `mat4` == `mat2x2` `mat3x3` `mat4x4`
- `mat2x3` `mat2x4` `mat3x2` `mat3x4` `mat4x2` `mat4x3`

Containers

- Structures: `struct`
- Arrays: `[]`

Language Basics: storage qualifiers

`const`

- Local constants defined within shader

`uniform`

- Constant shader parameters that can be changed between draws
- Do not change per-vertex or per-fragment

`attribute`

- Per-vertex values (position, normal, color, etc.)

`varying`

- Values output by the vertex shader, input by the fragment shader
- Interpolated during rasterization

Language Basics: operators

Grouping, function/constructor	`()`
Array/component indexing	`[]`
Component/member selection	`.`
Unary	`++ -- + - !`
Binary	`* / + -`
Relational	`< <= > >= == !=`
Logical	`&& ^^ ||`
Ternary conditional	`?:`
Assignment	`= *= /= += -=`
Sequence	`,`

Language Basics: constructors

- Used to initialize a structure or built-in type
  - Built-in type initialization:

```
vec3 myRGB = vec3(0.25, 0.5, 0.75);
```

  - Structure initialization:

```
struct S { int a; float b; };
S s = S(2, 3.5);
```

- Provide enough components of correct type

```
vec2 myYZ = vec2(0.5, 0.75);
vec4 myPos = vec4(0.25, myYZ, 1.0);
```

- Also provides explicit type conversions – no casting in GLSL!
  - Only int to float implicit conversions are allowed

```
float numTexels = countTexels();
if (!bool(numTexels)) discard; // non-zero value -> true
```

Vector components

```
vec2 v2;
vec3 v3;
vec4 v4;

v2.x   // is a float
v2.z   // wrong: undefined for type
v4.rgba // is a vec4
v4.stp  // is a vec3
v4.b    // is a float
v4.xy   // is a vec2
v4.xgp  // wrong: mismatched component sets
```


Language Basics: swizzles

- Components from {xyzw}, {rgba}, or {stpq}
- Writemask or swizzle during assignment

```
vec4 foo = vec4(1.0);
foo.xyz = vec3(0.25, 0.5, 0.75);
foo.wzyx = foo;  // reverse the components
```

- Swizzle or replicate components on right hand side

```
foo = foo.wzyx;  // another way to reverse components
foo = foo.xxyy;  // components reusable on right side
v2.yyyy          // wrong: too many components for type
```

- Use indexing for vector and matrix component selection

```
mat4 myMatrix = mat4(1.0);
foo.x = foo[2];          // same as foo.x = foo.z;
foo = myMatrix[0];       // first column of matrix
foo.x = myMatrix[0][0];  // first column, first row
```

Language Basics: flow control

- `for while do`
  - Loops can have `break`, `continue`
- `if else`
- Function calls
  - Can have `return`
- The above can all be nested!
- Note: no unstructured jumps (a.k.a `goto`)
- `discard`
  - Only available in fragment shaders
  - "Kills" the fragment, no further processing in the pipeline

Language Basics: VS built-in variables

Inputs

- `attribute vec4 gl_Vertex`
- `attribute vec3 gl_Normal`
- `attribute vec4 gl_Color`
- `attribute vec4 gl_SecondaryColor`
- `attribute vec4 gl_MultiTexCoord`*n* (0-7)
- `attribute float gl_FogCoord`

Outputs

- `vec4 gl_Position`: **must be written!**
- `float gl_PointSize`
- `vec4 gl_ClipVertex`
- `varying vec4 gl_FrontColor`
- `varying vec4 gl_BackColor`
- `varying vec4 gl_FrontSecondaryColor`
- `varying vec4 gl_BackSecondaryColor`
- `varying vec4 gl_TexCoord[`*n*`]`
- `varying float gl_FogFragCoord`

Language Basics: FS built-in variables

Inputs

- `vec4 gl_FragCoord`
- `bool gl_FrontFacing`
- `varying vec4 gl_Color`
- `varying vec4 gl_SecondaryColor`
- `varying vec4 gl_TexCoord[n]`
- `varying float gl_FogFragCoord`
- `varying vec2 gl_PointCoord`

Outputs

- `vec4 gl_FragColor`
- `vec4 gl_FragData[n]`
- `float gl_FragDepth`

Built-in variables

Attributes & uniforms

For ease of programming

OpenGL state mapped to variables

Some special variables are required to be written to, others are optional

Special built-ins

AMD Smarter Choice

Vertex shader

```
vec4  gl_Position;      // must be written
vec4  gl_ClipPosition;  // may be written
float gl_PointSize;     // may be written
```

Fragment shader

```
float gl_FragColor;     // may be written
float gl_FragDepth;     // may be read/written
vec4  gl_FragCoord;     // may be read
bool  gl_FrontFacing;   // may be read
```


Attributes

AMD Smarter Choice

Built-in

```
attribute vec4  gl_Vertex;
attribute vec3  gl_Normal;
attribute vec4  gl_Color;
attribute vec4  gl_SecondaryColor;
attribute vec4  gl_MultiTexCoordn;
attribute float gl_FogCoord;
```

User-defined

```
attribute vec3  myTangent;
attribute vec3  myBinormal;
Etc…
```

Built-in Uniforms

AMD Smarter Choice

```
uniform   mat4  gl_ModelViewMatrix;
uniform   mat4  gl_ProjectionMatrix;
uniform   mat4  gl_ModelViewProjectionMatrix;
uniform   mat3  gl_NormalMatrix;
uniform   mat4  gl_TextureMatrix[n];

struct gl_MaterialParameters {
  vec4  emission;
  vec4  ambient;
  vec4  diffuse;
  vec4  specular;
  float shininess;
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
```


Built-in Uniforms

AMD Smarter Choice

```
struct gl_LightSourceParameters {
  vec4  ambient;
  vec4  diffuse;
  vec4  specular;
  vec4  position;
  vec4  halfVector;
  vec3  spotDirection;
  float spotExponent;
  float spotCutoff;
  float spotCosCutoff;
  float constantAttenuation
  float linearAttenuation
  float quadraticAttenuation
};
Uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];
```

Built-in Varyings

```
varying   vec4  gl_FrontColor      // vertex
varying   vec4  gl_BackColor;      // vertex
varying   vec4  gl_FrontSecColor;  // vertex
varying   vec4  gl_BackSecColor;   // vertex

varying   vec4  gl_Color;          // fragment
varying   vec4  gl_SecondaryColor; // fragment

varying   vec4  gl_TexCoord[];     // both
varying   float gl_FogFragCoord;   // both
```


Language Basics: function calls

Special storage qualifiers apply to function parameters, e.g.:

```
bool f(in vec2 inputVec, out float retVal)
{
    ...
}
```

in: Parameter is copied in to the function but not copied out (default)

const in: Parameter is copied in to the function and cannot change

out: Parameter is copied out of the function but not copied in

inout: Parameter is both copied in and copied out

Notes

- Recursion is strictly forbidden!
- Functions can return a value or **void**

Built-in functions

AMD Smarter Choice

Angles & Trigonometry

- **radians, degrees, sin, cos, tan, asin, acos, atan**

Exponentials

- **pow, exp2, log2, sqrt, inversesqrt**

Common

- **abs, sign, floor, ceil, fract, mod, min, max, clamp**

Built-in functions

AMD Smarter Choice

Interpolations

- **mix**(x,y,a) **x*(1.0-a) + y*a**)
- **step**(edge,x) **x <= edge ? 0.0 : 1.0**
- **smoothstep**(edge0,edge1,x)
  - **zero if x <= edge0,**
  - **1 if x >= edge1**
  - **performs smooth Hermite interpolation between 0 and 1 when edge0 < x < edge1.**

Built-in functions

AMD Smarter Choice

Geometric

- **length, distance, cross, dot, normalize, faceForward, reflect**

Matrix

- **matrixCompMult**

Vector relational

- **lessThan, lessThanEqual, greaterThan, greaterThanEqual, equal, notEqual, notEqual, any, all**

Built-in functions

AMD Smarter Choice

Texture

- **texture1D, texture2D, texture3D, textureCube**
- **texture1DProj, texture2DProj, texture3DProj, textureCubeProj**
- **shadow1D, shadow2D, shadow1DProj, shadow2Dproj**

Vertex

- **ftransform**

Starter Shaders: color manipulation

AMD Smarter Choice

```
// simple.fs
//
// copy primary color

void main(void)
{
    // Copy the primary color
    gl_FragColor = gl_Color;
}


// colorinvert.fs
//
// invert like a color negative

void main(void)
{
    // invert color components
    gl_FragColor.rgb = 1.0 - gl_Color.rgb;
    gl_FragColor.a = 1.0;
}
```


Starter Shaders: color manipulation

AMD Smarter Choice

```
// grayscale.fs
//
// convert RGB to grayscale

void main(void)
{
    // Convert to grayscale using NTSC conversion weights
    float gray = dot(gl_Color.rgb, vec3(0.299, 0.587, 0.114));

    // replicate grayscale to RGB components
    gl_FragColor = vec4(gray, gray, gray, 1.0);
}

// sepia.fs
//
// convert RGB to sepia tone

void main(void)
{
    // Convert to grayscale using NTSC conversion weights
    float gray = dot(gl_Color.rgb, vec3(0.299, 0.587, 0.114));

    // convert grayscale to sepia
    gl_FragColor = vec4(gray * vec3(1.2, 1.0, 0.8), 1.0);
}
```

Starter Shaders: color manipulation

AMD Smarter Choice

```
// heatsig.fs
//
// map grayscale to heat signature

uniform sampler1D sampler0;

void main(void)
{
    // Convert to grayscale using NTSC conversion weights
    float gray = dot(gl_Color.rgb, vec3(0.299, 0.587, 0.114));

    // look up heatsig value
    gl_FragColor = texture1D(sampler0, gray);
}
```


Starter Shaders: color manipulation

AMD Smarter Choice

```
// fog.fs
//
// per-pixel fog

uniform float density;

void main(void)
{
    const vec4 fogColor = vec4(0.5, 0.8, 0.5, 1.0);

    // calculate 2nd order exponential fog factor
    // based on fragment's Z distance
    const float e = 2.71828;
    float fogFactor = (density * gl_FragCoord.z);
    fogFactor *= fogFactor;
    fogFactor = clamp(pow(e, -fogFactor), 0.0, 1.0);

    // Blend fog color with incoming color
    gl_FragColor = mix(fogColor, gl_Color, fogFactor);
}
```

Starter Shaders: convolution

```
// passthrough.fs
//
// pass through a single texel value

uniform sampler2D sampler0;

void main(void)
{
    gl_FragColor = texture2D(sampler0, gl_TexCoord[0].st);
}
```


Starter Shaders: convolution

```
// blur.fs
//
// blur (low-pass) 3x3 kernel

uniform sampler2D sampler0;
uniform vec2 tc_offset[9];

void main(void)
{
    vec4 sample[9];

    for (int i = 0; i < 9; i++)
    {
        sample[i] = texture2D(sampler0,
                              gl_TexCoord[0].st + tc_offset[i]);
    }

//   1 2 1
//   2 1 2   / 13
//   1 2 1

    gl_FragColor = (sample[0] + (2.0*sample[1]) + sample[2] +
                    (2.0*sample[3]) + sample[4] + (2.0*sample[5]) +
                    sample[6] + (2.0*sample[7]) + sample[8]) / 13.0;
}
```

Starter Shaders: convolution

Blur	1 2 1 2 1 2 / 13 1 2 1
Sharpen	-1 -1 -1 -1 9 -1 -1 -1 -1
LaPlacian	-1 -1 -1 -1 8 -1 -1 -1 -1
Dilation	max(kernel)
Erosion	min(kernel)

Starter Shaders: vertex shaders

```
// simple.vs
//
// Generic vertex transformation,
// copy primary color

void main(void)
{
    // normal MVP transform
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

    // Copy the primary color
    gl_FrontColor = gl_Color;
}
```

Starter Shaders: vertex shaders

AMD Smarter Choice

```
// diffuse.vs
//
// Generic vertex transformation,
// diffuse lighting based on one
// white light

uniform vec3 lightPos[1];

void main(void)
{
    // normal MVP transform
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

    vec3 N = normalize(gl_NormalMatrix * gl_Normal);
    vec4 V = gl_ModelViewMatrix * gl_Vertex;
    vec3 L = normalize(lightPos[0] - V.xyz);

    // output the diffuse color
    float NdotL = dot(N, L);
    gl_FrontColor = gl_Color * vec4(max(0.0, NdotL));
}
```


Example: Fragment Shader

AMD Smarter Choice

```
varying vec4 diffuseColor;

varying vec3 lightVector;

varying vec3 fragNormal;


void main(){


   float perFragmentLighting=max(dot(lightVector,fragNormal),0.0);


   gl_FragColor = diffuseColor * lightingFactor;


}
```

Starter Shaders: vertex shaders

AMD Smarter Choice

```
// ptsize.vs
//
// Generic vertex transformation,
// attenuated point size

void main(void)
{
    // normal MVP transform
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

    vec4 V = gl_ModelViewMatrix * gl_Vertex;

    gl_FrontColor = gl_Color;

    // calculate point size based on distance from eye
    float ptSize = length(V);
    ptSize = ptSize * ptSize * ptSize;
    gl_PointSize = 20000000.0 / ptSize;
}
```


Starter Shaders: vertex shaders

AMD Smarter Choice

```
// stretch.vs
//
// Generic vertex transformation,
// followed by squash/stretch

uniform vec3 lightPos[1];
uniform vec3 squashStretch;

void main(void)
{
    // normal MVP transform, followed by squash/stretch
    vec4 stretchedCoord = gl_Vertex;
    stretchedCoord.xyz *= squashStretch;
    gl_Position = gl_ModelViewProjectionMatrix * stretchedCoord;

    ...
}
```

Basic method

2 basic object types

- Shader object
- Program object

Create Vertex & Fragment Shader Objects

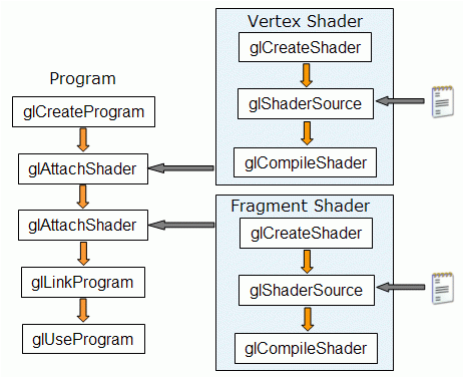
Compile both

Create program object & attach shaders

Link program

Use program

Creating Shaders



Compiling

AMD Smarter Choice

```
void glShaderSource(GLuint shader, GLsizei nstrings, const GLchar **strings,
   const GLint *lengths)
    //if lengths==NULL, assumed to be null-terminated

void glCompileShader (GLuint shader);
```


Attaching & Linking

AMD Smarter Choice

```
void glAttachShader(GLuint program, GLuint shader);
    //twice, once for vertex shader & once for fragment shader

void glLinkProgram(GLuint program);
   //program now ready to use

void glUseProgram(GLuint program);
   //switches on shader, bypasses FFP
   //if program==0, shaders turned off, returns to FFP
```

In short…

```
GLuint programObject;

GLuint vertexShaderObject;

GLuint fragmentShaderObject;


unsigned char *vertexShaderSource =
   readShaderFile(vertexShaderFilename);

unsigned char *fragmentShaderSource =
   readShaderFile(fragmentShaderFilename);


programObject=glCreateProgram ();

vertexShaderObject=glCreateShader (GL_VERTEX_SHADER);

fragmentShaderObject=glCreateShader (GL_FRAGMENT_SHADER);
```


Example

```
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);
}
```

Other functions

AMD Smarter Choice

Clean-up

```
void glDetachObject (GLuint container, GLuint attached);
void glDeleteObject (GLuint object);
```

Info Log

```
void glGetInfoLog (GLuint object,     GLsizei maxLength,
GLsizei *length,  GLchar *infoLog);
```

- Returns compile & linking information, errors

Loading Uniforms

AMD Smarter Choice

```
void glUniform{1|2|3|4}{f|i} (GLint location,…);
```

Location obtained with

```
GLint glGetUniformLocation (GLuint program, const GLuint
  *name);
```

Shader must be enabled with glUseProgramObject () before uniforms can be loaded

Loading Attributes

```
void glVertexAttrib{1234}{sfd} (GLint     index,…);
```

Index obtained with

```
GLint glGetAttribLocation (GLuint program, const GLuint
  *name);
```

Alternate method

```
void glBindAttribLocation (GLuint program, GLuint index,
  const GLuint *name);
```

- Program must be linked **after** binding attrib locations

Loading Textures

Bind textures to different units as usual

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D,myFirstTexture);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D,mySecondTexture);
```

Then load corresponding sampler with texture unit that texture is bound to

```
glUniform1i (glGetUniformLocation ( programObject,"myFirstSampler"),0);
glUniform1i (glGetUniformLocation ( programObject,"mySecondSampler"),1);
```

Ivory – vertex shader

AMD Smarter Choice

```
uniform vec4 lightPos;

varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    vec4 vert = gl_ModelViewMatrix * gl_Vertex;

    normal   = gl_NormalMatrix * gl_Normal;
    lightVec = vec3(lightPos - vert);
    viewVec  = -vec3(vert);
}
```


Ivory – fragment shader

AMD Smarter Choice

```
varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(){
      vec3 norm = normalize(normal);

      vec3 L = normalize(lightVec);
      vec3 V = normalize(viewVec);
      vec3 halfAngle = normalize(L + V);

      float NdotL = dot(L, norm);
   float NdotH = clamp(dot(halfAngle, norm), 0.0, 1.0);

   // "Half-Lambert" technique for more pleasing diffuse term
     float diffuse  = 0.5 * NdotL + 0.5;
     float specular = pow(NdotH, 64.0);

     float result = diffuse + specular;

     gl_FragColor = vec4(result);
}
```

Gooch – vertex shader

AMD Smarter Choice

```
uniform vec4 lightPos;

varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    vec4 vert = gl_ModelViewMatrix * gl_Vertex;

    normal   = gl_NormalMatrix * gl_Normal;
    lightVec = vec3(lightPos - vert);
    viewVec  = -vec3(vert);
}
```


Gooch – fragment shader

AMD Smarter Choice

```
uniform vec3 ambient;

varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(){
    const float b = 0.55;
    const float y = 0.3;
    const float Ka = 1.0;
    const float Kd = 0.8;
    const float Ks = 0.9;

    vec3 specularcolor = vec3(1.0, 1.0, 1.0);

    vec3 norm = normalize(normal);
    vec3 L = normalize (lightVec);
    vec3 V = normalize (viewVec);
    vec3 halfAngle = normalize (L + V);
```

Gooch – fragment shader (2)

AMD Smarter Choice

```
    vec3 orange = vec3(.88,.81,.49);
    vec3 purple = vec3(.58,.10,.76);

    vec3 kCool = purple;
    vec3 kWarm = orange;

    float NdotL = dot(L, norm);
    float NdotH = clamp(dot(halfAngle, norm), 0.0, 1.0);
    float specular = pow(NdotH, 64.0);

    float blendval  = 0.5 * NdotL + 0.5;
    vec3 Cgooch = mix(kWarm, kCool, blendval);

    vec3 result = Ka * ambient + Kd * Cgooch + specularcolor * Ks * specular;

    gl_FragColor = vec4(result, 1.0);
}
```


Useful References

AMD Smarter Choice

http://www.3dshaders.com/

- Home page for the “orange book” focused solely on GLSL

http://www.opengl.org/sdk/

- OpenGL SDK, including links to the below resources

http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf

- one double-sided page cheat sheet to GLSL – indispensible!

http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

- This is the ultimate authority: the GLSL specification document

http://www.opengl.org/sdk/docs/books/SuperBible/

- Full reference and tutorial to OpenGL 2.1
- All sample code downloadable for Windows, Mac OS X, and Linux