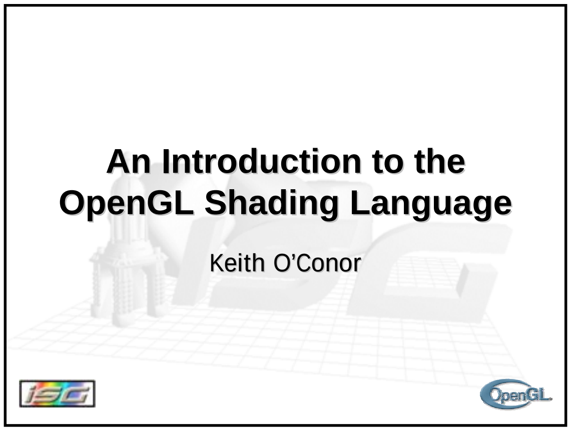
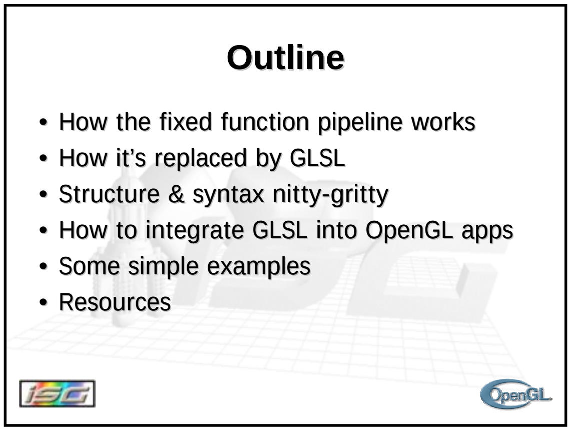


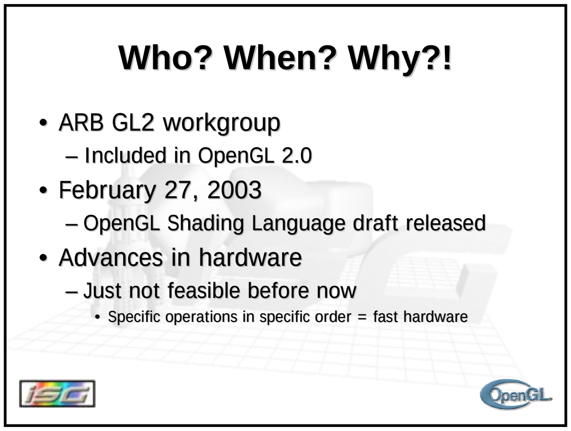
An Introduction to the OpenGL Shading Language

Keith O'Conor



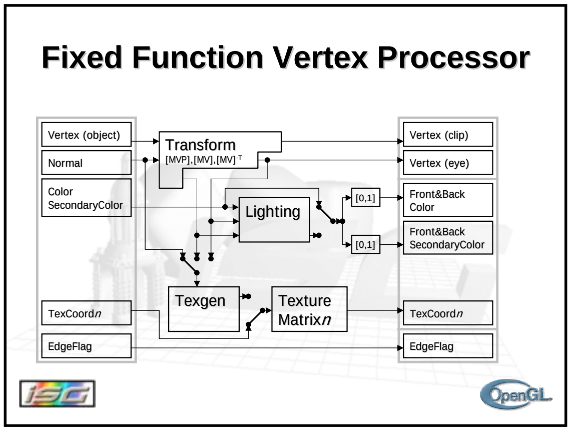
Outline

- How the fixed function pipeline works
- How it's replaced by GLSL
- Structure & syntax nitty-gritty
- How to integrate GLSL into OpenGL apps
- Some simple examples
- Resources



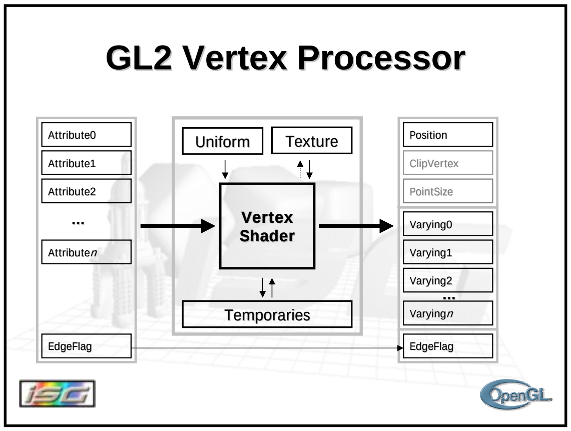
Who? When? Why?!

- ARB GL2 workgroup
  - Included in OpenGL 2.0
- February 27, 2003
  - OpenGL Shading Language draft released
- Advances in hardware
  - Just not feasible before now
    - Specific operations in specific order = fast hardware



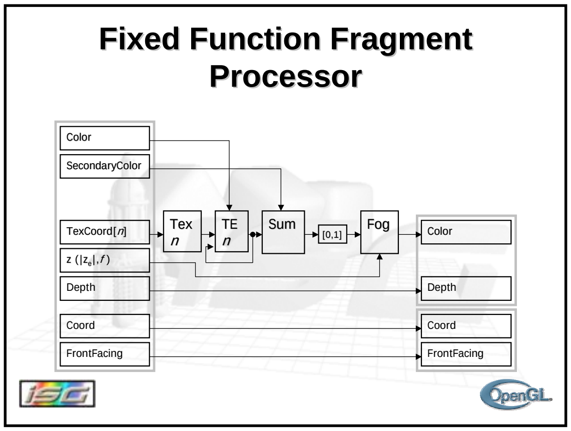
Fixed Function Vertex Processor

Vertex (object)
Normal
Color
SecondaryColor
TexCoord*n*
EdgeFlag
Transform
[MVP],[MV],[MV]$^{-T}$
Lighting
[0,1]
[0,1]
Texgen
Texture Matrix*n*
Vertex (clip)
Vertex (eye)
Front&Back Color
Front&Back SecondaryColor
TexCoord*n*
EdgeFlag



GL2 Vertex Processor

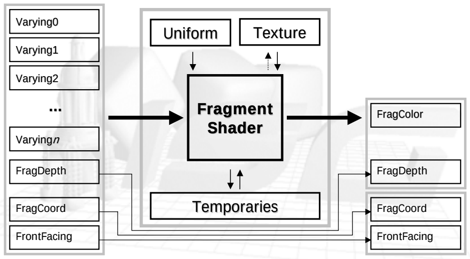
Attribute0
Attribute1
Attribute2
…
Attribute*n*
EdgeFlag
Uniform
Texture
Vertex Shader
Temporaries
Position
ClipVertex
PointSize
Varying0
Varying1
Varying2
…
Varying*n*
EdgeFlag



Fixed Function Fragment Processor

Color
SecondaryColor
TexCoord[*n*]
z (|z$_e$|,*f*)
Depth
Coord
FrontFacing
Tex *n*
TE *n*
Sum
[0,1]
Fog
Color
Depth
Coord
FrontFacing

GL2 Fragment Processor



In General…

- Vertex processes programmed
  - Vertex Transformation
  - Normal Transformation, Normalization
  - Lighting
  - Texture Coordinate Generation and Transformation
- Fragment processes programmed
  - Texture accesses & application
  - Fog

Previous programmability

- Texture Shaders
- Register Combiners
- Assembly programs
  - ARB_vertex_program
  - ARB_fragment_program
  - Messy!
- Needed general, readable & maintainable language

Types

`void`

`float  vec2  vec3  vec4`

`mat2  mat3  mat4`

`int  ivec2  ivec3  ivec4`

`bool  bvec2  bvec3  bvec4`

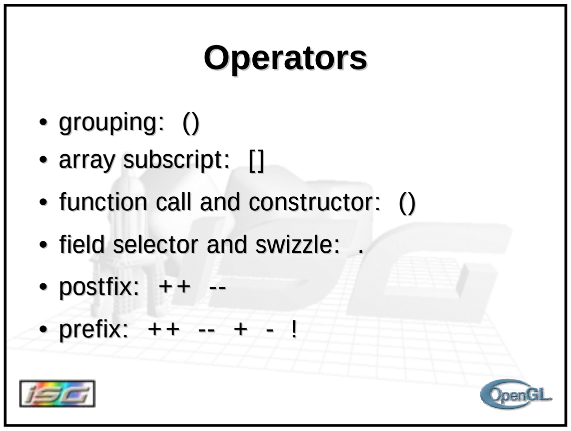
`sampler`*`n`*`D, samplerCube,`
`samplerShadow`*`n`*`D`

Types

- Structs
- Arrays
  - One dimensional
  - Constant size (ie `float array[4];`)
- Reserved types
  - **`half  hvec2  hvec3  hvec4`**
  - **`fixed  fvec2  fvec3  fvec4`**
  - **`double  dvec2  dvec3  dvec4`**

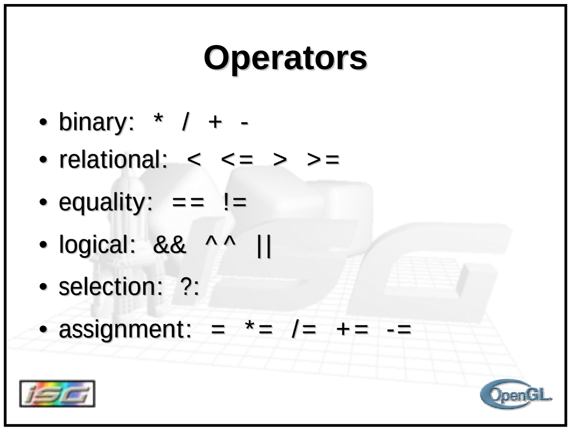
Type qualifiers

- attribute
  - Changes per-vertex
    - eg. position, normal etc.
- uniform
  - Does not change between vertices of a batch
    - eg light position, texture unit, other constants
- varying
  - Passed from VS to FS, interpolated
    - eg texture coordinates, vertex color



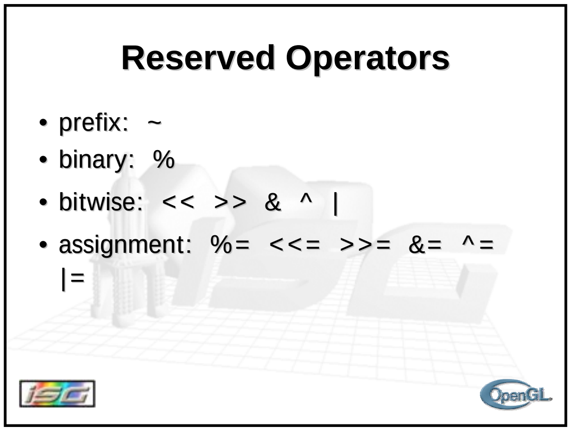
Operators

- grouping: ()
- array subscript: []
- function call and constructor: ()
- field selector and swizzle: .
- postfix: ++ --
- prefix: ++ -- + - !



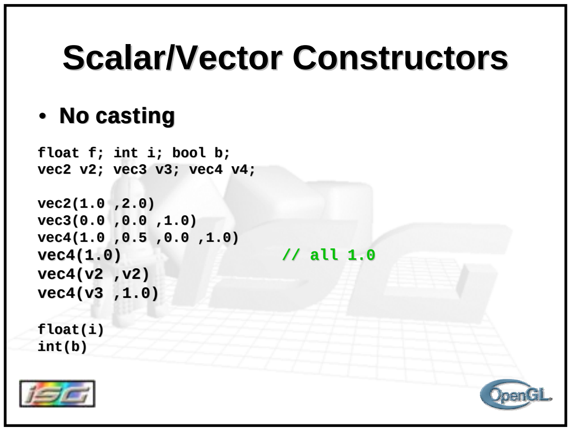
Operators

- binary: * / + -
- relational: < <= > >=
- equality: == !=
- logical: && ^^ ||
- selection: ?:
- assignment: = *= /= += -=



Reserved Operators

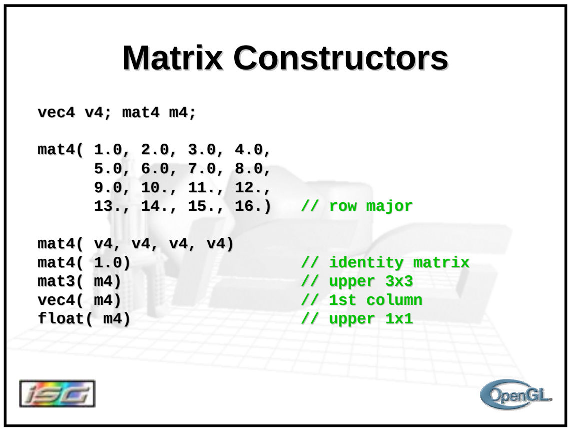
- prefix: ~
- binary: %
- bitwise: << >> & ^ |
- assignment: %= <<= >>= &= ^= |=



Scalar/Vector Constructors

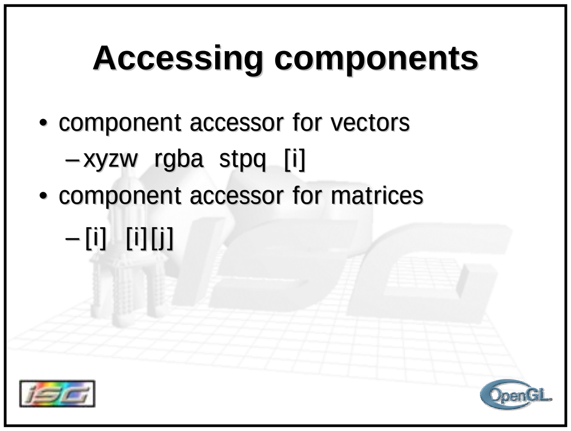
- **No casting**

```
float f; int i; bool b;
vec2 v2; vec3 v3; vec4 v4;

vec2(1.0 ,2.0)
vec3(0.0 ,0.0 ,1.0)
vec4(1.0 ,0.5 ,0.0 ,1.0)
vec4(1.0)                   // all 1.0
vec4(v2 ,v2)
vec4(v3 ,1.0)

float(i)
int(b)
```



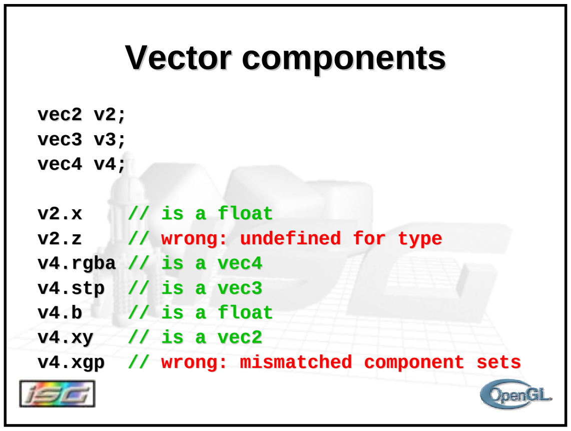
Matrix Constructors

```
vec4 v4; mat4 m4;

mat4( 1.0, 2.0, 3.0, 4.0,
      5.0, 6.0, 7.0, 8.0,
      9.0, 10., 11., 12.,
      13., 14., 15., 16.)   // row major

mat4( v4, v4, v4, v4)
mat4( 1.0)                  // identity matrix
mat3( m4)                   // upper 3x3
vec4( m4)                   // 1st column
float( m4)                  // upper 1x1
```



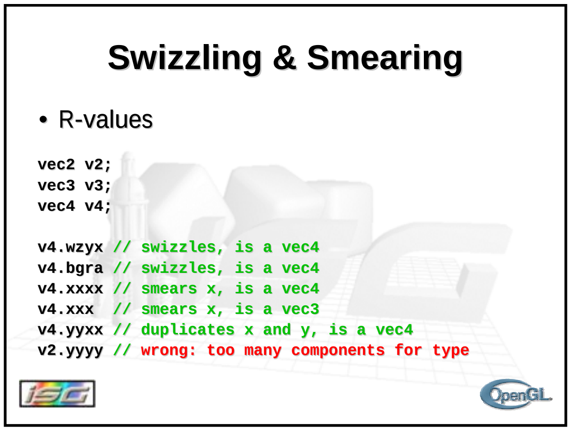
Accessing components

- component accessor for vectors
  - xyzw rgba stpq [i]
- component accessor for matrices
  - [i] [i][j]



Vector components

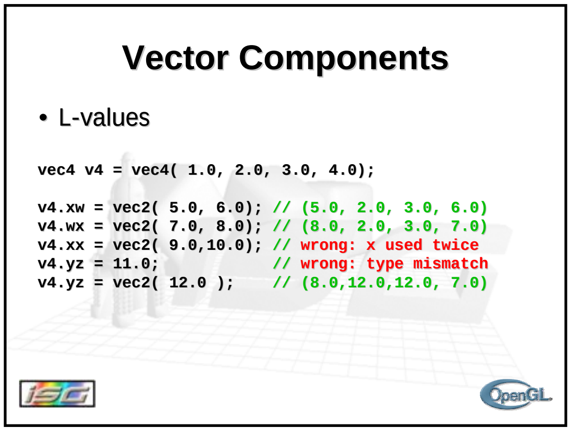
```
vec2 v2;
vec3 v3;
vec4 v4;

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
```



Swizzling & Smearing

- R-values

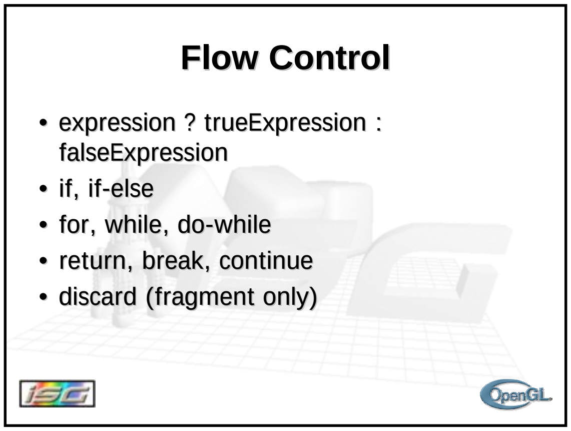
```
vec2 v2;
vec3 v3;
vec4 v4;

v4.wzyx // swizzles, is a vec4
v4.bgra // swizzles, is a vec4
v4.xxxx // smears x, is a vec4
v4.xxx  // smears x, is a vec3
v4.yyxx // duplicates x and y, is a vec4
v2.yyyy // wrong: too many components for type
```



Vector Components

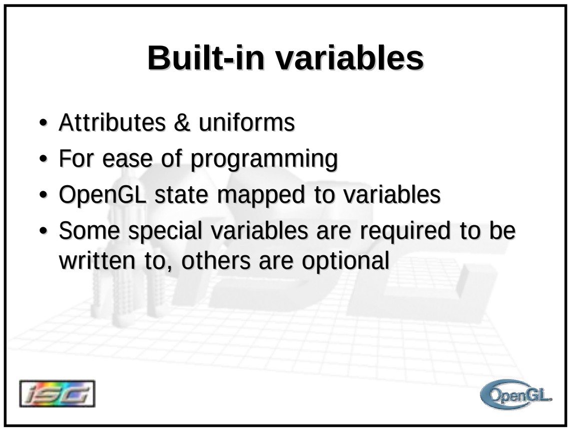
- L-values

```
vec4 v4 = vec4( 1.0, 2.0, 3.0, 4.0);

v4.xw = vec2( 5.0, 6.0); // (5.0, 2.0, 3.0, 6.0)
v4.wx = vec2( 7.0, 8.0); // (8.0, 2.0, 3.0, 7.0)
v4.xx = vec2( 9.0,10.0); // wrong: x used twice
v4.yz = 11.0;            // wrong: type mismatch
v4.yz = vec2( 12.0 );    // (8.0,12.0,12.0, 7.0)
```



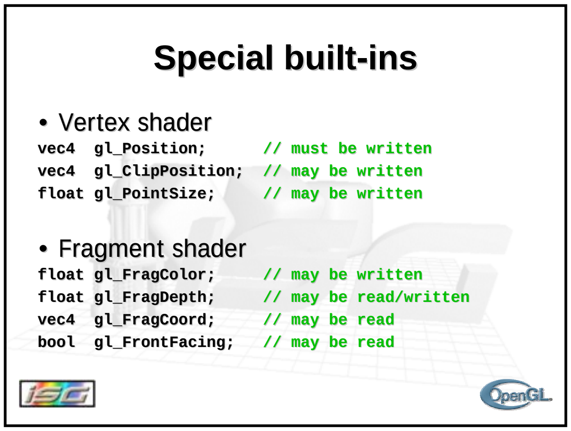
Flow Control

- expression ? trueExpression : falseExpression
- if, if-else
- for, while, do-while
- return, break, continue
- discard (fragment only)



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



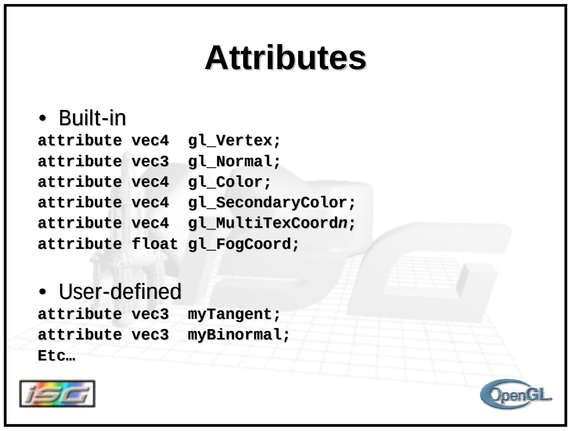
Special built-ins

- 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
```



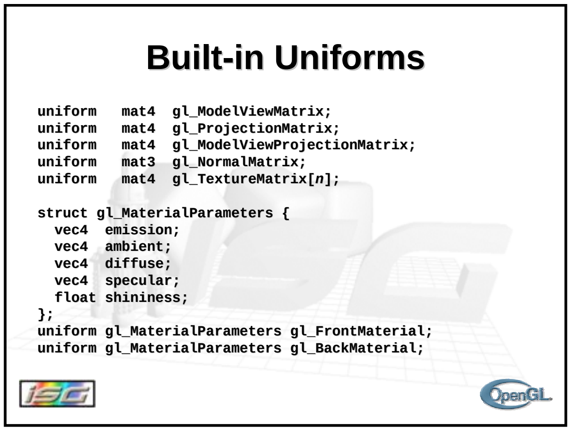
Attributes

- 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;
```

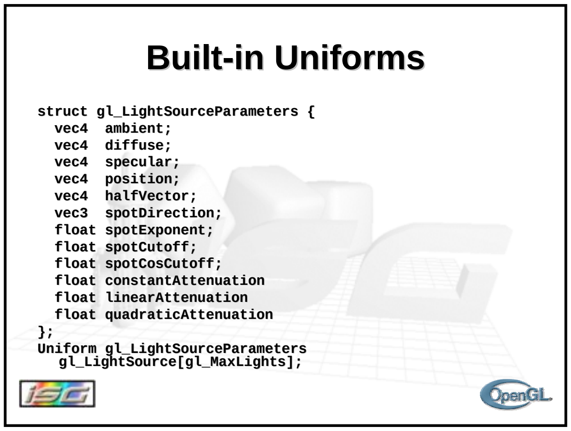
- User-defined

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



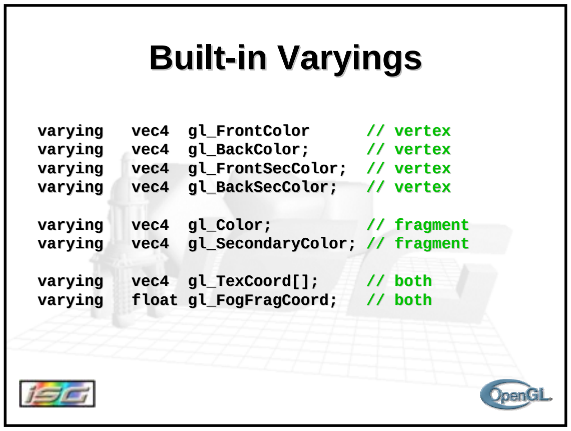
Built-in Uniforms

```
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;
```



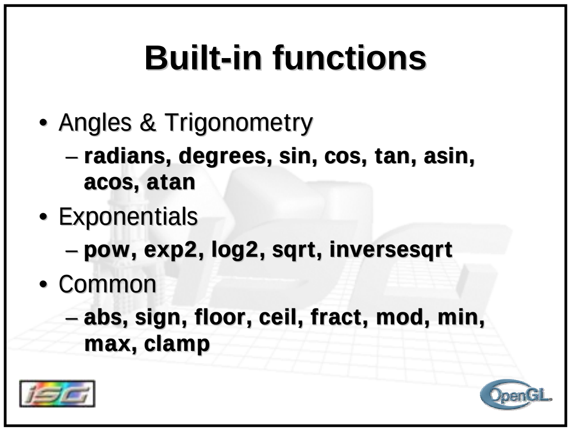
Built-in Uniforms

```
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];
```



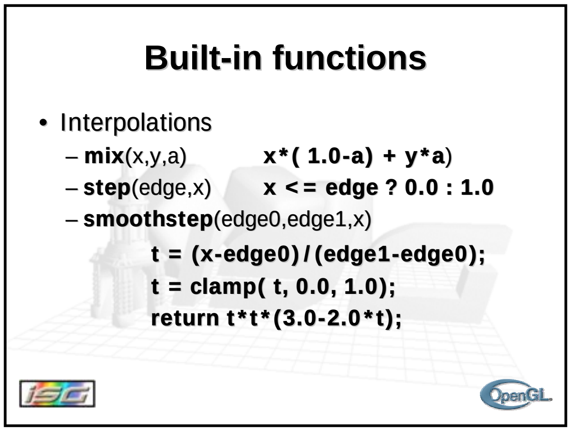
Built-in Varyings

```
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
```



Built-in functions

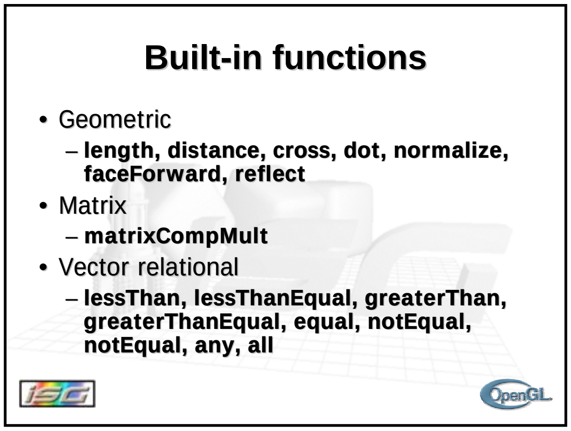
- Angles & Trigonometry
  - **radians, degrees, sin, cos, tan, asin, acos, atan**
- Exponentials
  - **pow, exp2, log2, sqrt, inversesqrt**
- Common
  - **abs, sign, floor, ceil, fract, mod, min, max, clamp**



Built-in functions

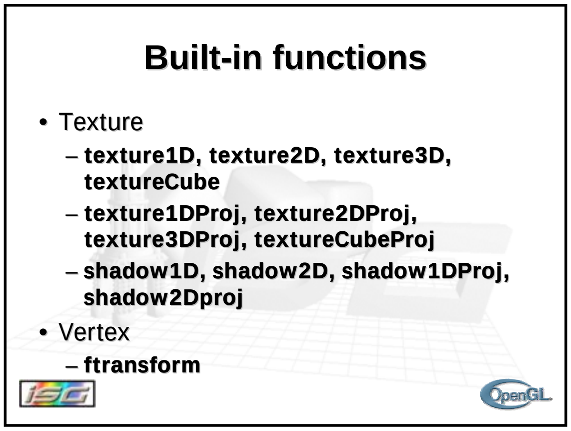
- Interpolations
  - **mix**(x,y,a)   **x*(1.0-a) + y*a**)
  - **step**(edge,x)   **x <= edge ? 0.0 : 1.0**
  - **smoothstep**(edge0,edge1,x)

```
t = (x-edge0)/(edge1-edge0);
t = clamp( t, 0.0, 1.0);
return t*t*(3.0-2.0*t);
```



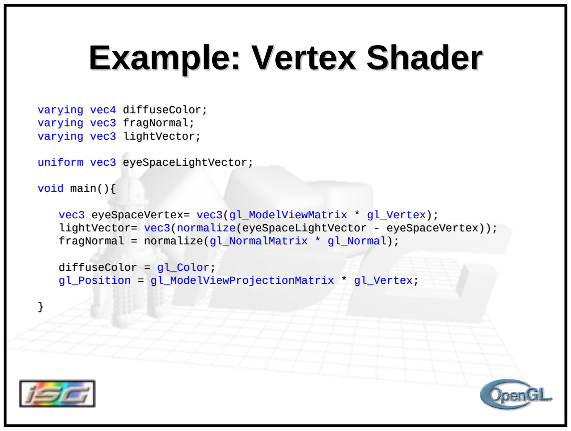
Built-in functions

- Geometric
  - **length, distance, cross, dot, normalize, faceForward, reflect**
- Matrix
  - **matrixCompMult**
- Vector relational
  - **lessThan, lessThanEqual, greaterThan, greaterThanEqual, equal, notEqual, notEqual, any, all**



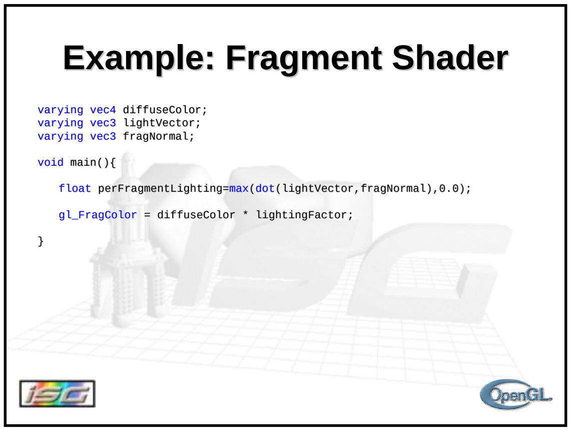
Built-in functions

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



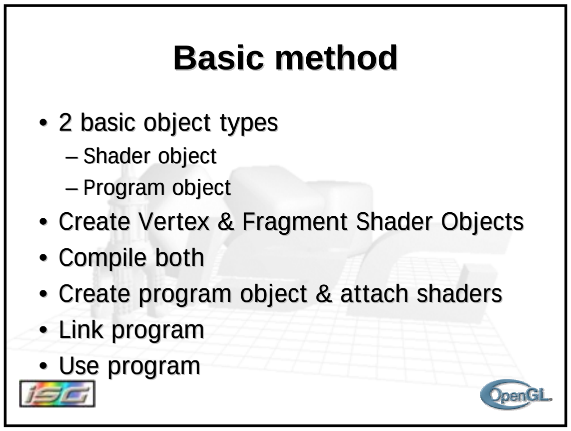
Example: Vertex Shader

```
varying vec4 diffuseColor;
varying vec3 fragNormal;
varying vec3 lightVector;

uniform vec3 eyeSpaceLightVector;

void main(){

   vec3 eyeSpaceVertex= vec3(gl_ModelViewMatrix * gl_Vertex);
   lightVector= vec3(normalize(eyeSpaceLightVector - eyeSpaceVertex));
   fragNormal = normalize(gl_NormalMatrix * gl_Normal);

   diffuseColor = gl_Color;
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

}
```



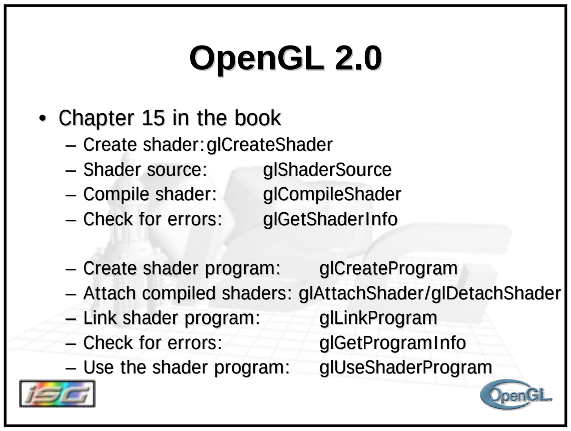
Example: Fragment Shader

```
varying vec4 diffuseColor;
varying vec3 lightVector;
varying vec3 fragNormal;

void main(){

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

   gl_FragColor = diffuseColor * lightingFactor;

}
```



Basic method

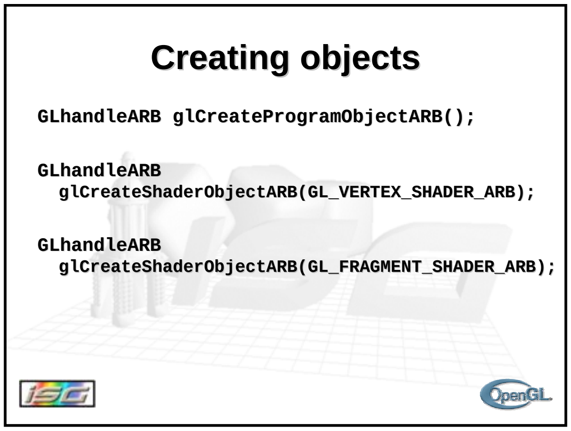
- 2 basic object types
  - Shader object
  - Program object
- Create Vertex & Fragment Shader Objects
- Compile both
- Create program object & attach shaders
- Link program
- Use program



OpenGL 2.0

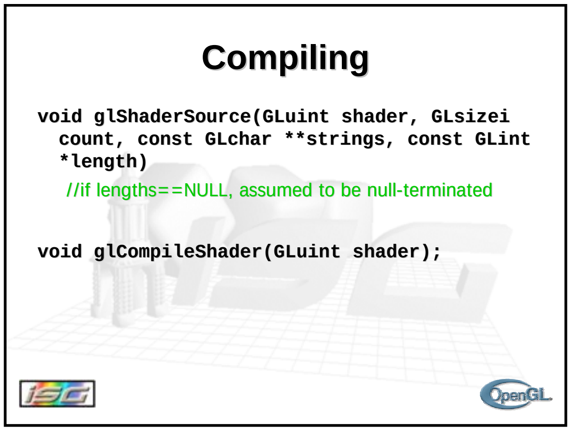
- Chapter 15 in the book
  - Create shader: glCreateShader
  - Shader source: glShaderSource
  - Compile shader: glCompileShader
  - Check for errors: glGetShaderInfo

  - Create shader program: glCreateProgram
  - Attach compiled shaders: glAttachShader/glDetachShader
  - Link shader program: glLinkProgram
  - Check for errors: glGetProgramInfo
  - Use the shader program: glUseShaderProgram



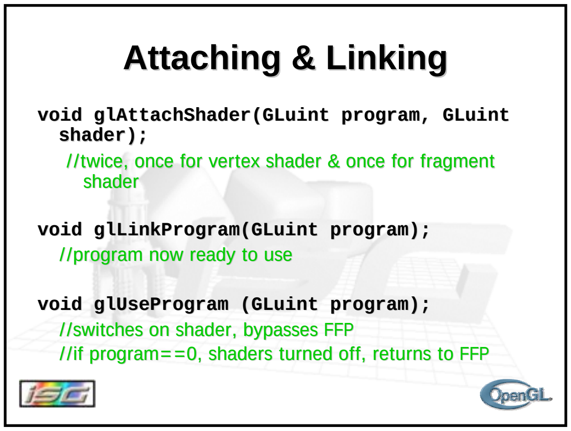
Creating objects

```
GLhandleARB glCreateProgramObjectARB();

GLhandleARB
  glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);

GLhandleARB
  glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
```



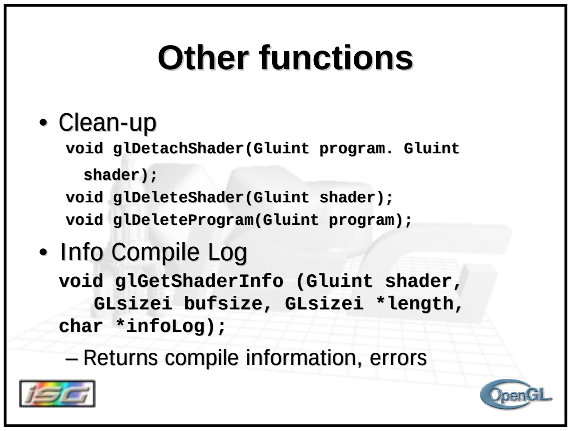
Compiling

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

void glCompileShader(GLuint shader);
```



Attaching & Linking

```
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
```



Other functions

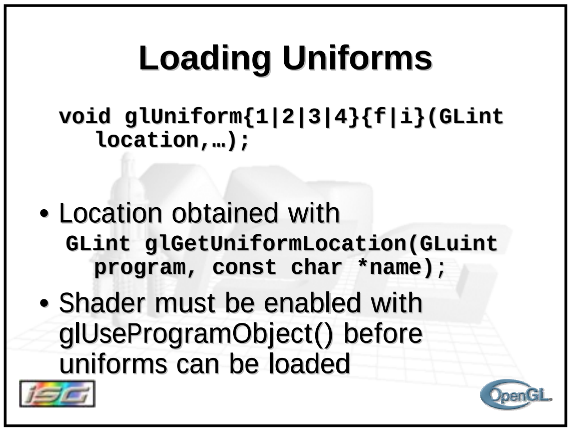
- Clean-up

```
void glDetachShader(Gluint program. Gluint
  shader);
void glDeleteShader(Gluint shader);
void glDeleteProgram(Gluint program);
```

- Info Compile Log

```
void glGetShaderInfo (Gluint shader,
    GLsizei bufsize, GLsizei *length,
char *infoLog);
```

  – Returns compile information, errors



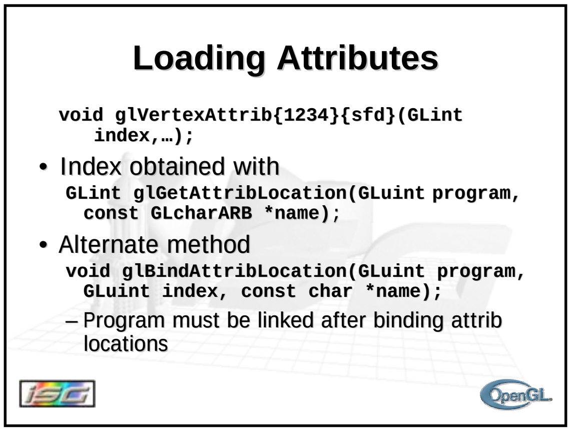
Loading Uniforms

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

- Location obtained with

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

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



Loading Attributes

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

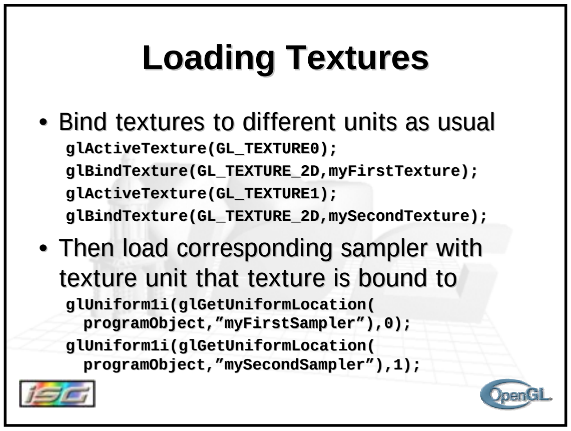
- Index obtained with

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

- Alternate method

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

  – Program must be linked after binding attrib locations



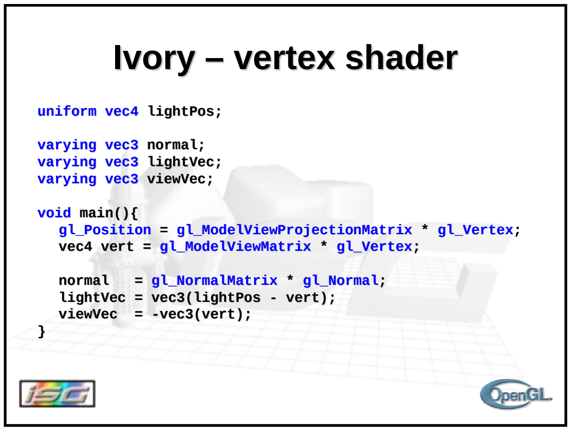
Loading Textures

- Bind textures to different units as usual

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

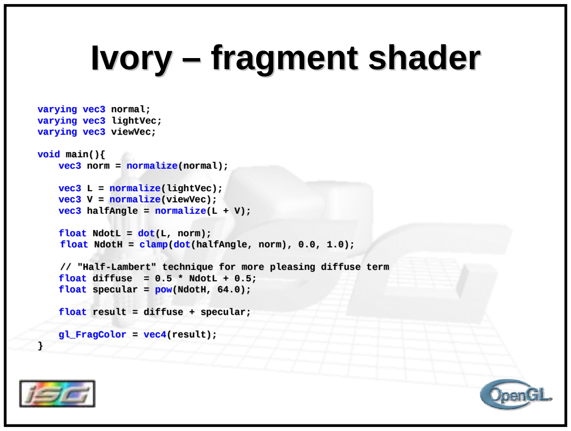
- Then load corresponding sampler with texture unit that texture is bound to

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



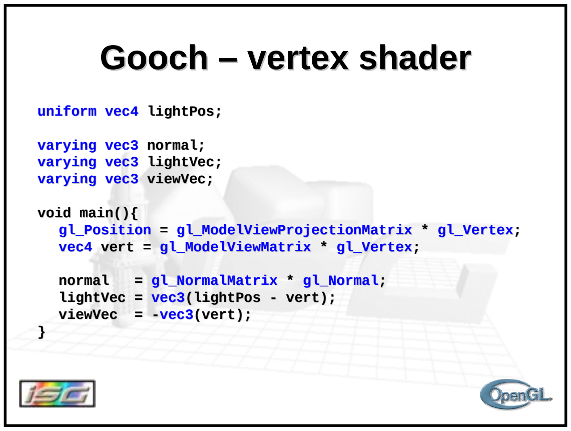
Ivory – vertex shader

```
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);
}
```



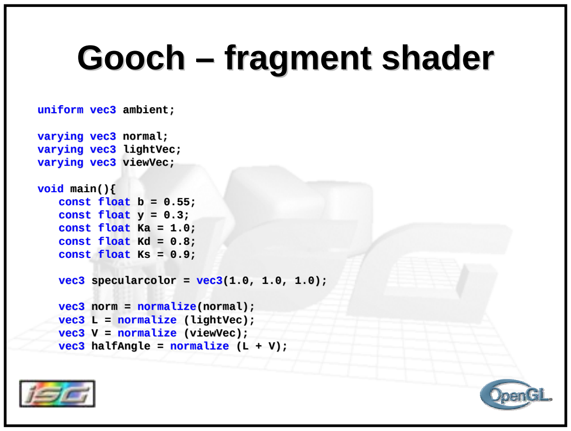
Ivory – fragment shader

```
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);
}
```



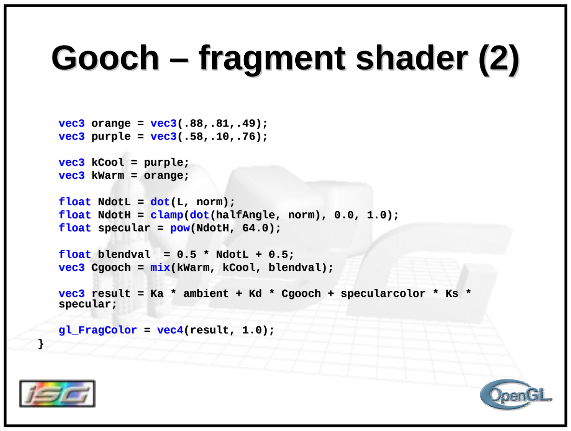
Gooch – vertex shader

```
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

```
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)

```
   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);
}
```