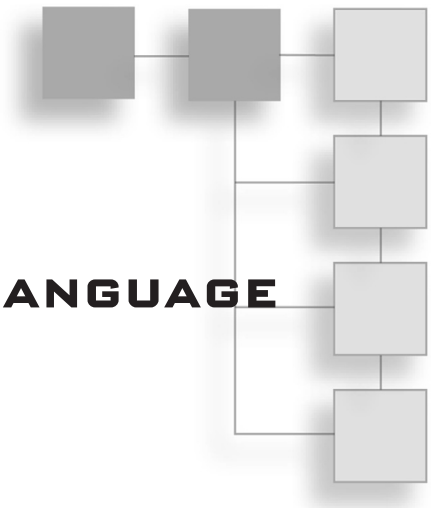
Appendix B



OpenGL Shading Language Reference

Rob Jones

Uniform Reference

The following lists all GLSL built-in uniform variables with their types. The built-in uniforms can be accessed from either vertex or fragment shaders and are automatically updated and maintained by the OpenGL implementation you are using.

Matrix State

```
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ProjectionMatrix;
uniform mat4 gl_ModelViewProjectionMatrix;
uniform mat4 gl_TextureMatrix[gl_MaxTextureCoords];
```

Derived Matrix State

These uniforms provide inverse and transposed versions of the matrices above. Poorly conditioned matrices may result in unpredictable values in their inverse forms.

```
uniform mat3 gl_NormalMatrix;  // transpose of the inverse of the
                               // upper leftmost 3×3 of gl_ModelViewMatrix
uniform mat4 gl_ModelViewMatrixInverse;
uniform mat4 gl_ProjectionMatrixInverse;
uniform mat4 gl_ModelViewProjectionMatrixInverse;
uniform mat4 gl_TextureMatrixInverse[gl_MaxTextureCoords];
uniform mat4 gl_ModelViewMatrixTranspose;
uniform mat4 gl_ProjectionMatrixTranspose;
```

```
uniform mat4 gl_ModelViewProjectionMatrixTranspose;
uniform mat4 gl_TextureMatrixTranspose[gl_MaxTextureCoords];
uniform mat4 gl_ModelViewMatrixInverseTranspose;
uniform mat4 gl_ProjectionMatrixInverseTranspose;
uniform mat4 gl_ModelViewProjectionMatrixInverseTranspose;
uniform mat4 gl_TextureMatrixInverseTranspose[gl_MaxTextureCoords];
```

Normal Scaling

```
uniform float gl_NormalScale;
```

Depth Range

These values are in window coordinates.

```
struct gl_DepthRangeParameters {
  float near; // n
  float far;  // f
  float diff; // f - n
};
uniform gl_DepthRangeParameters gl_DepthRange;
```

Clip Planes

```
uniform vec4 gl_ClipPlane[gl_MaxClipPlanes];
```

Point Size

```
struct gl_PointParameters {
  float size;
  float sizeMin;
  float sizeMax;
  float fadeThresholdSize;
  float distanceConstantAttenuation;
  float distanceLinearAttenuation;
  float distanceQuadraticAttenuation;
};
uniform gl_PointParameters gl_Point;
```

Material State

```
struct gl_MaterialParameters {
  vec4 emission;   // Ecm
  vec4 ambient;    // Acm
```

```
  vec4 diffuse;    // Dcm
  vec4 specular;   // Scm
  float shininess; // Srm
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
```

Light State

```
struct gl_LightSourceParameters {
  vec4 ambient;               // Acli
  vec4 diffuse;               // Dcli
  vec4 specular;              // Scli
  vec4 position;              // Ppli
  vec4 halfVector;            // Derived: Hi
  vec3 spotDirection;         // Sdli
  float spotExponent;         // Srli
  float spotCutoff;           // Crli, (range: [0.0,90.0], 180.0)
  float spotCosCutoff;        // Derived: cos(Crli), (range: [1.0,0.0],-1.0)
  float constantAttenuation;  // K0
  float linearAttenuation;    // K1
  float quadraticAttenuation; // K2
};
uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];
struct gl_LightModelParameters {
  vec4 ambient; // Acs
};
uniform gl_LightModelParameters gl_LightModel;
```

Derived Light State

These uniforms provide values that are products of light and material values.

```
struct gl_LightModelProducts {
  vec4 sceneColor; // Derived. Ecm + Acm * Acs
};
uniform gl_LightModelProducts gl_FrontLightModelProduct;
uniform gl_LightModelProducts gl_BackLightModelProduct;
struct gl_LightProducts {
  vec4 ambient; // Acm * Acli
  vec4 diffuse; // Dcm * Dcli
  vec4 specular; // Scm * Scli
};
```

```
uniform gl_LightProducts gl_FrontLightProduct[gl_MaxLights];
uniform gl_LightProducts gl_BackLightProduct[gl_MaxLights];
```

Texture Environment and Generation

```
uniform vec4 gl_TextureEnvColor[gl_MaxTextureImageUnits];
uniform vec4 gl_EyePlaneS[gl_MaxTextureCoords];
uniform vec4 gl_EyePlaneT[gl_MaxTextureCoords];
uniform vec4 gl_EyePlaneR[gl_MaxTextureCoords];
uniform vec4 gl_EyePlaneQ[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneS[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneT[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneR[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneQ[gl_MaxTextureCoords];
```

Fog

```
struct gl_FogParameters {
  vec4 color;
  float density;
  float start;
  float end;
  float scale; // Derived: 1.0 / (end - start)
};
uniform gl_FogParameters gl_Fog;
```

C API Reference

This section provides a reference for the functions that have been added to OpenGL for use with the OpenGL Shading Language.

glAttachShader()

```
void glAttachShader(GLuint program, GLuint shader)
```

Parameters

`program`: Specifies the program object to which a shader object will be attached.

`shader`: Specifies the shader object that is to be attached.

Description

Attaches shader objects to program objects. This allows you to form a program that can be linked and then executed. You can attach multiple shaders to one program object.

Errors

GL_INVALID_VALUE is generated if either program or shader is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program is not of type GL_PROGRAM_OBJECT.

GL_INVALID_OPERATION is generated if shader is not of type GL_SHADER_OBJECT.

GL_INVALID_OPERATION is generated if shader is already attached to program.

GL_INVALID_OPERATION is generated if glAttachShader() is executed between the execution of glBegin() and the corresponding execution glEnd().

glBindAttribLocation()

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

Parameters

program: Specifies the handle of the program object in which the association is to be made.

index: Specifies the index of the generic vertex attribute to be bound.

name: Specifies a null terminated string containing the name of the vertex shader attribute variable to which index is to be bound.

Description

Used to associate a user-defined attribute in the program specificed with the index specified.

Binds come into effect on the next glLinkProgram() operation, and the user may not bind standard OpenGL attribute names using this function.

OpenGL makes a copy of name so the user is able to free it once the function has returned.

Errors

GL_INVALID_VALUE is generated if index is greater than or equal to GL_MAX_VERTEX_ATTRIBS.

GL_INVALID_OPERATION is generated if name starts with the reserved prefix "gl_".

GL_INVALID_VALUE is generated if program is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program is not of type GL_PROGRAM_OBJECT.

GL_INVALID_OPERATION is generated if glBindAttribLocation() is executed between the execution of glBegin() and the corresponding execution of glEnd().

glCompileShader()

```
void glCompileShader(GLuint shader)
```

Parameters

`shader`: Specifies the shader object to be compiled.

Description

Compiles the shader specified. The compilation state is stored as part of the shader object and may be queried via `glGetShaderiv()` using the parameter `GL_COMPILE_STATUS`.

Errors

`GL_INVALID_VALUE` is generated if shader is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `shader` is not of type `GL_SHADER_OBJECT`.

`GL_INVALID_OPERATION` is generated if `glCompileShader()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glCreateProgram()

```
GLuint glCreateProgram()
```

Description

Creates a program object and returns a handle to use to work with it. The returned handle can be used as a target for all operations that work on a program. The program object can be shared across OpenGL contexts. If shared, all the data attached to a program is also shared.

Errors

This function returns zero if an error occurs creating the program object.

`GL_INVALID_OPERATION` is generated if `glCreateProgram()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glCreateShader()

```
GLuint glCreateShader(GLenum shaderType)
```

Parameters

`shaderType`: Specifies the type of shader to be created. Must be either `GL_VERTEX_SHADER` or `GL_FRAGMENT_SHADER`.

Description

Creates a shader object of the requested type and returns a handle to use to work with it. The returned handle can be used as a target for all operations that work on a shader. The shader object can be shared across OpenGL contexts. If shared, all the data attached to a program is also shared.

Errors

This function returns zero if an error occurs creating the shader object.

`GL_INVALID_ENUM` is generated if shaderType is not an accepted value.

`GL_INVALID_OPERATION` is generated if `glCreateShader()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glDeleteProgram()

```
void glDeleteProgram(GLuint program)
```

Parameters

`program`: Specifies the program object to be deleted.

Description

Deletes the program object specified, including freeing memory and invalidating the handle. If the requested program is part of the current render state, then the program is flagged for deletion and will be deleted when it is next unbound. Any attached shader objects are automatically detached from the program upon deletion.

Errors

`GL_INVALID_VALUE` is generated if program is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `glDeleteProgram()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glDeleteShader()

```
void glDeleteShader(GLuint shader)
```

Parameters

`shader`: Specifies the shader object to be deleted.

Description

Deletes the shader object specified, including freeing memory and invalidating the handle. If the requested shader is attached to a program then it will be flagged for deletion but not until it is no longer attached to any program in any context.

Errors

`GL_INVALID_VALUE` is generated if shader is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `glDeleteShader()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glDetachShader()

```
void glDetachShader(GLuint program,GLuint shader)
```

Parameters

`program`: Specifies the program object from which to detach the shader object.

`shader`: Specifies the shader object to be detached.

Description

Detaches the requested shader from the requested program. If the shader is flagged for deletion and not attached to any other programs, it will be deleted.

Errors

`GL_INVALID_VALUE` is generated if either `program` or `shader` is a value that was not generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_OPERATION` is generated if `shader` is not a shader object.

`GL_INVALID_OPERATION` is generated if `shader` is not attached to `program`.

`GL_INVALID_OPERATION` is generated if `glDetachShader()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glEnableVertexAttribArray() / glDisableVertexAttribArray()

```
void glEnableVertexAttribArray(GLuint index)
void glDisableVertexAttribArray(GLuint index)
```

Parameters

`index`: Specifies the index of the generic vertex attribute to be enabled or disabled.

Description

Enables or disables a vertex array on the requested generic vertex attribute for use with the OpenGL vertex array functions.

Errors

`GL_INVALID_VALUE` is generated if `index` is greater than or equal to `GL_MAX_VERTEX_ATTRIBS`.

`GL_INVALID_OPERATION` is generated if `glEnableVertexAttribArray()` or `glDisableVertexAttribArray()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glGetActiveAttrib()

```
void glGetActiveAttrib(GLuint program, GLuint index, GLsizei bufSize,
                       GLsizei *length, GLint *size, GLenum *type,GLchar *name)
```

Parameters

`program`: Specifies the program object to be queried.

`index`: Specifies the index of the attribute variable to be queried.

`bufSize`: Specifies the maximum number of characters OpenGL is allowed to write in the character buffer indicated by `name`.

`length`: Returns the number of characters actually written by OpenGL in the string indicated by `name` (excluding the null terminator) if a value other than `NULL` is passed.

`size`: Returns the size of the attribute variable.

`type`: Returns the data type of the attribute variable.

`name`: Returns a null terminated string containing the name of the attribute variable.

Description

Extracts information about the currently active attribute specified by `index` in the speicified program. The function can return information about built-in attributes and user-defined attributes.

Errors

`GL_INVALID_VALUE` is generated if `program` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_VALUE` is generated if `index` is greater than or equal to the number of active attribute variables in `program`.

GL_INVALID_OPERATION is generated if glGetActiveAttrib() is executed between the execution of glBegin() and the corresponding execution of glEnd().

GL_INVALID_VALUE is generated if bufSize is less than zero.

glGetActiveUniform()

```
void glGetActiveUniform(GLuint program, GLuint index, GLsizei bufSize,
                        GLsizei *length, GLint *size, GLenum *type,GLchar *name)
```

Parameters

program: Specifies the program object to be queried.

index: Specifies the index of the uniform variable to be queried.

bufSize: Specifies the maximum number of characters OpenGL is allowed to write in the character buffer indicated by name.

length: Returns the number of characters actually written by OpenGL in the string indicated by name (excluding the null terminator) if a value other than NULL is passed.

size: Returns the size of the uniform variable.

type: Returns the data type of the uniform variable.

name: Returns a null terminated string containing the name of the uniform variable.

Description

Extracts information about the currently active uniform specified by index in the speicified program. The function can return information about built-in attributes and user-defined attributes.

Uniform variables declared as arrays or structures cannot be returned directly by this function; instead each subsection of the uniform variable must be queried.

Errors

GL_INVALID_VALUE is generated if program is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program is not a program object.

GL_INVALID_VALUE is generated if index is greater than or equal to the number of active uniform variables in program.

GL_INVALID_OPERATION is generated if glGetActiveUniform() is executed between the execution of glBegin() and the corresponding execution of glEnd().

GL_INVALID_VALUE is generated if bufSize is less than zero.

glGetAttachedShaders()

```
void glGetAttachedShaders(GLuint program, GLsizei maxCount, GLsizei *count,
                          GLuint *shaders)
```

Parameters

`program`: Specifies the program object to be queried.

`maxCount`: Specifies the size of the array for storing the returned object names.

`count`: Returns the number of names actually returned in `objects`.

`shaders`: Specifies an array that is used to return the names of attached shader objects.

Description

Returns up to `maxCount` handles to attached shader object for a given program object into memory specified by shaders. The amount written is returned in `count`.

Errors

`GL_INVALID_VALUE` is generated if `program` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_VALUE` is generated if `maxCount` is less than zero.

`GL_INVALID_OPERATION` is generated if `glGetAttachedShaders()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glGetAttribLocation()

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

Parameters

`program`: Specifies the program object to be queried.

`name`: Points to a null terminated string containing the name of the attribute variable whose location is to be queried.

Description

Returns the location of the named attribute for the given program object. The name cannot start with a prefix of "gl_"; if it does, −1 is returned.

Errors

`GL_INVALID_OPERATION` is generated if `program` is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program is not a program object.

GL_INVALID_OPERATION is generated if program has not been successfully linked.

GL_INVALID_OPERATION is generated if glGetAttribLocation() is executed between the execution of glBegin() and the corresponding execution of glEnd().

glGetProgram()

```
void glGetProgramiv(GLuint program, GLenum pname, GLint *params)
```

Parameters

program: Specifies the program object to be queried.

pname: Specifies the object parameter. Accepted symbolic names are GL_DELETE_STATUS, GL_LINK_STATUS, GL_VALIDATE_STATUS, GL_INFO_LOG_LENGTH, GL_ATTACHED_SHADERS, GL_ACTIVE_ATTRIBUTES, GL_ACTIVE_ATTRIBUTE_MAX_LENGTH, GL_ACTIVE_UNIFORMS, and GL_ACTIVE_UNIFORM_MAX_LENGTH.

params: Returns the requested object parameter.

Description

Returns details about a program object based on the given pname. If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_VALUE is generated if program is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program does not refer to a program object.

GL_INVALID_ENUM is generated if pname is not an accepted value.

GL_INVALID_OPERATION is generated if glGetProgram() is executed between the execution of glBegin() and the corresponding execution of glEnd().

glGetProgramInfoLog()

```
void glGetProgramInfoLog(GLuint program, GLsizei maxLength, GLsizei *length,
                         GLchar *infoLog)
```

Parameters

program: Specifies the program object whose information log is to be queried.

maxLength: Specifies the size of the character buffer for storing the returned information log.

length: Returns the length of the string returned in infoLog (excluding the null terminator).

infoLog: Specifies an array of characters that is used to return the information log.

Description

Returns up to `maxLength` characters from the named program's information log into `infoLog`. The length of the characters written is returned in `length`. Because the text returned in the log isn't consistent across vendors, the information returned shouldn't be relied upon beyond debugging and development.

Errors

`GL_INVALID_VALUE` is generated if `program` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_VALUE` is generated if `maxLength` is less than zero.

`GL_INVALID_OPERATION` is generated if `glGetProgramInfoLog()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glGetShader()

```
void glGetShaderiv(GLuint shader, GLenum pname, GLint *params)
```

Parameters

`shader`: Specifies the shader object to be queried.

`pname`: Specifies the object parameter. Accepted symbolic names are `GL_SHADER_TYPE`, `GL_DELETE_STATUS`, `GL_COMPILE_STATUS`, `GL_INFO_LOG_LENGTH`, and `GL_SHADER_SOURCE_LENGTH`.

`params`: Returns the requested object parameter.

Description

Returns details about a shader object based on the given `pname`.

Errors

`GL_INVALID_VALUE` is generated if `shader` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `shader` does not refer to a shader object.

`GL_INVALID_ENUM` is generated if `pname` is not an accepted value.

`GL_INVALID_OPERATION` is generated if `glGetShader()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glGetShaderInfoLog()

```
void glGetShaderInfoLog(GLuint shader, GLsizei maxLength, GLsizei *length,
                        GLchar *infoLog)
```

Parameters

`shader`: Specifies the shader object whose information log is to be queried.

`maxLength`: Specifies the size of the character buffer for storing the returned information log.

`length`: Returns the length of the string returned in `infoLog` (excluding the null terminator).

`infoLog`: Specifies an array of characters that is used to return the information log.

Description

Returns up to `maxLength` characters from the named shader's information log into `infoLog`. The length of the characters written is returned in `length`. Because the text returned in the log isn't consistent across vendors, the information returned shouldn't be relied upon beyond debugging and development.

Errors

`GL_INVALID_VALUE` is generated if `shader` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `shader` is not a shader object.

`GL_INVALID_VALUE` is generated if `maxLength` is less than zero.

`GL_INVALID_OPERATION` is generated if `glGetShaderInfoLog()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glGetShaderSource()

```
void glGetShaderSource(GLuint shader, GLsizei bufSize,GLsizei *length,
                       GLchar *source)
```

Parameters

`shader`: Specifies the shader object to be queried.

`bufSize`: Specifies the size of the character buffer for storing the returned source code string.

`length`: Returns the length of the string returned in `source` (excluding the null terminator).

`source`: Specifies an array of characters that is used to return the source code string.

Description

Returns up to `bufSize` characters from the named shader's source, which would have been previously set with `glShaderSource()`. The length of the characters written is returned in `length`.

Errors

GL_INVALID_VALUE is generated if shader is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if shader is not a shader object.

GL_INVALID_VALUE is generated if bufSize is less than zero.

GL_INVALID_OPERATION is generated if glGetShaderSource() is executed between the execution of glBegin() and the corresponding execution of glEnd().

glGetUniform()

```
void glGetUniformfv(GLuint program, GLint location, GLfloat *params)
void glGetUniformiv(GLuint program, GLint location, GLint *params)
```

Parameters

program: Specifies the program object to be queried.

location: Specifies the location of the uniform variable to be queried.

params: Returns the value of the specified uniform variable.

Description

Retrieves the value previously set for a uniform value at a given location in a given program.

Errors

GL_INVALID_VALUE is generated if program is not a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program is not a program object.

GL_INVALID_OPERATION is generated if program has not been successfully linked.

GL_INVALID_OPERATION is generated if location does not correspond to a valid uniform variable location for the specified program object.

GL_INVALID_OPERATION is generated if glGetUniform() is executed between the execution of glBegin() and the corresponding execution of glEnd().

glGetUniformLocation()

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

Parameters

program: Specifies the program object to be queried.

name: Points to a null terminated string containing the name of the uniform variable whose location is to be queried.

Description

Retrieves the location for a named uniform. This location can be used to set or retrieve the value of the uniform.

If name isn't an active uniform for the named program or starts with "gl_", then −1 is returned.

Errors

`GL_INVALID_VALUE` is generated if `program` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_OPERATION` is generated if `program` has not been successfully linked.

`GL_INVALID_OPERATION` is generated if `glGetUniformLocation()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glGetVertexAttrib()

```
void glGetVertexAttribdv(GLuint index, GLenum pname, GLdouble *params)
void glGetVertexAttribfv(GLuint index, GLenum pname, GLfloat *params)
void glGetVertexAttribiv(GLuint index, GLenum pname, GLint *params)
```

Parameters

`index`: Specifies the generic vertex attribute parameter to be queried.

`pname`: Specifies the symbolic name of the vertex attribute parameter to be queried. `GL_VERTEX_ATTRIB_ARRAY_ENABLED`, `GL_VERTEX_ATTRIB_ARRAY_SIZE`, `GL_VERTEX_ATTRIB_ARRAY_STRIDE`, `GL_VERTEX_ATTRIB_ARRAY_TYPE`, `GL_VERTEX_ATTRIB_ARRAY_NORMALIZED`, and `GL_CURRENT_VERTEX_ATTRIB` are accepted.

`params`: Returns the requested data.

Description

Returns data about the requested vertex attribute in the currently active program. If an error is generated, no change is made to the contents of `params`.

Errors

`GL_INVALID_VALUE` is generated if `index` is greater than or equal to `GL_MAX_VERTEX_ATTRIBS`.

`GL_INVALID_ENUM` is generated if `pname` is not an accepted value.

`GL_INVALID_OPERATION` is generated if `index` is zero and `pname` is `GL_CURRENT_VERTEX_ATTRIB`.

glGetVertexAttribPointer()

```
void glGetVertexAttribPointerv(GLuint index, GLenum pname, GLvoid **pointer)
```

Parameters

`index`: Specifies the generic vertex attribute parameter to be queried.

`pname`: Specifies the symbolic name of the generic vertex attribute parameter to be queried. Must be `GL_VERTEX_ATTRIB_ARRAY_POINTER`.

`pointer`: Returns the requested data.

Description

Returns data about the requested vertex attribute in the currently active program. The pointer returned is client-side state. The initial value for each pointer is `NULL`.

Errors

`GL_INVALID_VALUE` is generated if index is greater than or equal to `GL_MAX_VERTEX_ATTRIBS`.

`GL_INVALID_ENUM` is generated if `pname` is not an accepted value.

glIsProgram()

```
GLboolean glIsProgram(GLuint program)
```

Parameters

`program`: Specifies a potential program object.

Description

Returns a Boolean to indicate if the specified handle is a program (`GL_TRUE`) or not (`GL_FALSE`). If the passed handle is zero or a nonzero value that isn't a program, then `GL_FALSE` is returned.

Errors

`GL_INVALID_OPERATION` is generated if `glIsProgram()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

No error is generated if `program` is not a valid program object name.

glIsShader()

```
GLboolean glIsShader(GLuint shader)
```

Parameters

`program`: Specifies a potential shader object.

Description

Returns a boolean to indicate if the specified handle is a shader (`GL_TRUE`) or not (`GL_FALSE`). If the passed handle is zero or a non-zero value which isn't a shader then `GL_FALSE` is returned.

Errors

`GL_INVALID_OPERATION` is generated if `glIsShader()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

No error is generated if `shader` is not a valid shader object name.

glLinkProgram()

```
void glLinkProgram(GLuint program)
```

Parameters

`program`: Specifies the handle of the program object to be linked.

Description

Performs a link operation on the named program object. If any vertex shaders are attached they are linked into a vertex program. If any fragment shaders are attached they are linked into a fragment program.

The status of the link operation is held as part of the object state.

A link operation clears the named program's information log of any previous details. A failed link results in an unworking program, even if previous links were successful.

Errors

`GL_INVALID_VALUE` is generated if `program` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_OPERATION` is generated if `glLinkProgram()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glShaderSource()

```
void glShaderSource(GLuint shader, GLsizei count, const GLchar **string,
                    const GLint *length)
```

Parameters

`shader`: Specifies the handle of the shader object whose source code is to be replaced.

`count`: Specifies the number of elements in the `string` and `length` arrays.

`string`: Specifies an array of pointers to strings containing the source code to be loaded into the shader.

`length`: Specifies an array of string lengths.

Description

Sets the source code for a shader to the code containted in the passed array of strings. Any code previously stored in the shader is lost and replaced by the new incoming code. The OpenGL implementation copies the strings into the shader so the user is free to delete the source code once it has been uploaded to a shader object.

Errors

`GL_INVALID_VALUE` is generated if `shader` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `shader` is not a shader object.

`GL_INVALID_VALUE` is generated if `count` is less than zero.

`GL_INVALID_OPERATION` is generated if `glShaderSource()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glUniform()

```
void glUniform1f(GLint location, GLfloat v0)
void glUniform2f(GLint location, GLfloat v0, GLfloat v1)
void glUniform3f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2)
void glUniform4f(GLint location, GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3)
void glUniform1i(GLint location, GLint v0)
void glUniform2i(GLint location, GLint v0, GLint v1)
void glUniform3i(GLint location, GLint v0, GLint v1, GLint v2)
void glUniform4i(GLint location, GLint v0, GLint v1, GLint v2, GLint v3)
void glUniform1fv(GLint location, GLsizei count, const GLfloat *value)
void glUniform2fv(GLint location, GLsizei count, const GLfloat *value)
void glUniform3fv(GLint location, GLsizei count, const GLfloat *value)
void glUniform4fv(GLint location, GLsizei count, const GLfloat *value)
void glUniform1iv(GLint location, GLsizei count, const GLint *value)
void glUniform2iv(GLint location, GLsizei count, const GLint *value)
void glUniform3iv(GLint location, GLsizei count, const GLint *value)
void glUniform4iv(GLint location, GLsizei count, const GLint *value)
```

```
void glUniformMatrix2fv(GLint location, GLsizei count, GLboolean transpose,
                        const GLfloat *value)
void glUniformMatrix3fv(GLint location, GLsizei count, GLboolean transpose,
                        const GLfloat *value)
void glUniformMatrix4fv(GLint location, GLsizei count, GLboolean transpose,
                        const GLfloat *value)
```

Parameters

`location`: Specifies the location of the uniform variable to be modified.

`v0, v1, v2, v3`: Specifies the new values to be used for the specified uniform variable.

`count`: Specifies the number of elements that are to be modified (this should be 1 if the targeted uniform variable is not an array, 1 or more if it is an array).

`value`: Specifies a pointer to an array of `count` values that will be used to update the specified uniform variable.

`transpose`: Specifies whether to transpose the matrix as the values are loaded into the uniform variable.

Description

Allows the user to set the contents of a location corresponding to an active uniform variable for the active program.

`glUniform1i()` and `glUniform1iv()` are the only two functions that can be used to load uniform variables defined as sampler types. Loading samplers with any other function will cause a `GL_INVALID_OPERATION` error.

If the `count` value is greater than 1 and the uniform isn't an array, then a `GL_INVALID_OPERATION` error is generated and the uniform will remain unchanged.

Errors

`GL_INVALID_OPERATION` is generated if there is no current program object.

`GL_INVALID_OPERATION` is generated if the size of the uniform variable declared in the shader does not match the size indicated by the `glUniform()` command.

`GL_INVALID_OPERATION` is generated if one of the integer variants of this function is used to load a uniform variable of type float, vec2, vec3, vec4, or an array of these, or if one of the floating-point variants of this function is used to load a uniform variable of type int, ivec2, ivec3, or ivec4, or an array of these.

`GL_INVALID_OPERATION` is generated if `location` is an invalid uniform `location` for the current program object and location is not equal to −1.

GL_INVALID_VALUE is generated if count is less than zero.

GL_INVALID_OPERATION is generated if count is greater than 1 and the indicated uniform variable is not an array variable.

GL_INVALID_OPERATION is generated if a sampler is loaded using a command other than glUniform1i() and glUniform1iv().

GL_INVALID_OPERATION is generated if glUniform() is executed between the execution of glBegin() and the corresponding execution of glEnd().

glUseProgram()

```
void glUseProgram(GLuint program)
```

Parameters

program: Specifies the handle of the program object whose executables are to be used as part of current rendering state.

Description

Installs the requested program as part of the render state.

If the program contains a valid GL_VERTEX_SHADER, then OpenGL's vertex fixed function pipeline operations are disabled and the user is responsible for reproducing any needed functionality.

If the program contains a valid GL_FRAGMENT_SHADER, then OpenGL's fragment fixed function pipeline operations are disabled and the user is responsible for reproducing any needed functionality.

While a program is installed as part of the render state you can continue to update the disabled fixed functionality via normal OpenGL calls.

Errors

GL_INVALID_VALUE is generated if program is neither zero nor a value generated by OpenGL.

GL_INVALID_OPERATION is generated if program is not a program object.

GL_INVALID_OPERATION is generated if program could not be made part of current state.

GL_INVALID_OPERATION is generated if glUseProgram is executed between the execution of glBegin() and the corresponding execution of glEnd().

glValidateProgram()

```
void glValidateProgram(GLuint program)
```

Parameters

`program`: Specifies the handle of the program object to be validated.

Description

Checks to see if, given the current OpenGL state, the specified program can be executed. Any information generated will be stored in the program's information log.

The status of the validation process is stored as part of the program object and can be retrived via `glGetProgram()`. This value will be `GL_TRUE` if successful and `GL_FALSE` if not.

If validation is successful then the program can run in the current OpenGL state; otherwise it can't execute.

Errors

`GL_INVALID_VALUE` is generated if `program` is not a value generated by OpenGL.

`GL_INVALID_OPERATION` is generated if `program` is not a program object.

`GL_INVALID_OPERATION` is generated if `glValidateProgram()` is executed between the execution of `glBegin()` and the corresponding execution of `glEnd()`.

glVertexAttrib()

```
void glVertexAttrib1f(GLuint index, GLfloat v0)
void glVertexAttrib1s(GLuint index, GLshort v0)
void glVertexAttrib1d(GLuint index, GLdouble v0)
void glVertexAttrib2f(GLuint index, GLfloat v0, GLfloat v1)
void glVertexAttrib2s(GLuint index, GLshort v0, GLshort v1)
void glVertexAttrib2d(GLuint index, GLdouble v0, GLdouble v1)
void glVertexAttrib3f(GLuint index, GLfloat v0, GLfloat v1, GLfloat v2)
void glVertexAttrib3s(GLuint index, GLshort v0, GLshort v1, GLshort v2)
void glVertexAttrib3d(GLuint index, GLdouble v0, GLdouble v1, GLdouble v2)
void glVertexAttrib4f(GLuint index, GLfloat v0, GLfloat v1, GLfloat v2,
                      GLfloat v3)
void glVertexAttrib4s(GLuint index, GLshort v0, GLshort v1, GLshort v2,
                      GLshort v3)
void glVertexAttrib4d(GLuint index, GLdouble v0, GLdouble v1, GLdouble v2,
                      GLdouble v3)
void glVertexAttrib4Nub(GLuint index, GLubyte v0, GLubyte v1, GLubyte v2,
                        GLubyte v3)
void glVertexAttrib1fv(GLuint index, const GLfloat *v)
void glVertexAttrib1sv(GLuint index, const GLshort *v)
void glVertexAttrib1dv(GLuint index, const GLdouble *v)
```

```
void glVertexAttrib2fv(GLuint index, const GLfloat *v)
void glVertexAttrib2sv(GLuint index, const GLshort *v)
void glVertexAttrib2dv(GLuint index, const GLdouble *v)

void glVertexAttrib3fv(GLuint index, const GLfloat *v)
void glVertexAttrib3sv(GLuint index, const GLshort *v)
void glVertexAttrib3dv(GLuint index, const GLdouble *v)

void glVertexAttrib4fv(GLuint index, const GLfloat *v)
void glVertexAttrib4sv(GLuint index, const GLshort *v)
void glVertexAttrib4dv(GLuint index, const GLdouble *v)
void glVertexAttrib4iv(GLuint index, const GLint *v)
void glVertexAttrib4bv(GLuint index, const GLbyte *v)

void glVertexAttrib4ubv(GLuint index, const GLubyte *v)
void glVertexAttrib4usv(GLuint index, const GLushort *v)
void glVertexAttrib4uiv(GLuint index, const GLuint *v)

void glVertexAttrib4Nbv(GLuint index, const GLbyte *v)
void glVertexAttrib4Nsv(GLuint index, const GLshort *v)
void glVertexAttrib4Niv(GLuint index, const GLint *v)
void glVertexAttrib4Nubv(GLuint index, const GLubyte *v)
void glVertexAttrib4Nusv(GLuint index, const GLushort *v)
void glVertexAttrib4Nuiv(GLuint index, const GLuint *v)
```

Parameters

`index`: Specifies the index of the generic vertex attribute to be modified.

`v0, v1, v2, v3`: Specifies the new values to be used for the specified vertex attribute.

`v`: Specifies a pointer to an array of values to be used for the generic vertex attribute.

Description

Allows you to set the value of generic attributes for the currently active program.

The letters s, f, i, d, ub, us, and ui indicate the type of data passed.

The functions with "N" in the name normalize the data passed into the range indicated by their type; signed are normalized into [−1,1] and unsigned types are normalized into [0,1].

`glVertexAttrib()` can be issued at any time.

Errors

GL_INVALID_VALUE is generated if index is greater than or equal to GL_MAX_VERTEX_ATTRIBS.

glVertexAttribPointer()

```
void glVertexAttribPointer(GLuint index, GLint size, GLenum type,
                           GLboolean normalized, GLsizei stride,
                           const GLvoid *pointer)
```

Parameters

index: Specifies the index of the generic vertex attribute to be modified.

size: Specifies the number of values for each element of the generic vertex attribute array. Must be 1, 2, 3, or 4.

type: Specifies the data type of each component in the array. Symbolic constants GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, and GL_DOUBLE are accepted.

normalized: Specifies whether fixed-point data values should be normalized (GL_TRUE) or converted directly as fixed-point values (GL_FALSE) when they are accessed.

stride: Specifies the byte offset between consecutive attribute values. If stride is zero (the initial value), the attribute values are understood to be tightly packed in the array.

pointer: Specifies a pointer to the first component of the first attribute value in the array.

Description

This function allows you to set up the location and data format of a vertex-array-like structure to be used with generic vertex attributes. It works in much the same way as the standard OpenGL vertex-array-related functions do.

If normalized is set to GL_TRUE, then the data is remapped into a normalized range based on its type; unsigned types are mapped into [0,1], and signed types are mapped into [−1,1].

Execution of glVertexAttribPointer() is not allowed between the execution of glBegin() and glEnd(), but an error may or may not be generated. If no error is generated, the operation is undefined.

Errors

GL_INVALID_VALUE is generated if index is greater than or equal to GL_MAX_VERTEX_ATTRIBS.

GL_INVALID_VALUE is generated if size is not 1, 2, 3, or 4.

GL_INVALID_ENUM is generated if type is not an accepted value.

GL_INVALID_VALUE is generated if stride is negative.