

Built-In OpenGL Shading Language Variables and Functions

The OpenGL Shading Language, commonly called the “GLSL,” defines a number of variables for matching OpenGL state and large set of convenience functions. This appendix¹ provides a complete list of GLSL built-in variables and functions.

¹This appendix is adapted from *The OpenGL Shading Language Specification* (Versions 1.30 and 1.40) by John Kessenich.

The OpenGL Shading Language has been updated including deprecating and removing some features.

Version 1.30 of GLSL added numerous new functions and data types, while maintaining complete backward compatibility with previous versions of GLSL. No features were removed through deprecation, although various variables and state were deprecated.

Version 1.40, while aligning with the removed features of OpenGL, removed several features as well. Those removed features are made available again if the `GL_ARB_compatibility` extension is available with your implementation. Removed features (which are supported with the compatibility extension) are listed in the margin.

Variables

For OpenGL versions up to and including 3.0, much of the OpenGL state associated with the fixed-function pipeline that an application can configure is also available as variables from within a GLSL shader. The next sections describe all state variables available to shaders, and any restrictions on their use.

Vertex Shader Input Attribute Variables

Table I-1 lists the set of global attribute values reflecting the per-vertex input data that can be accessed exclusively from within a vertex shader. Almost all of these values were deprecated in Version 1.30.

Compatibility Extension	Variable Name	Type	Specifying Function	Description
gl_Vertex	<code>gl_Vertex</code>	<code>vec4</code>	glVertex	Vertex's world-space coordinate
gl_Color	<code>gl_Color</code>	<code>vec4</code>	glColor	Primary color value
gl_SecondaryColor	<code>gl_SecondaryColor</code>	<code>vec4</code>	glSecondaryColor	Secondary color value
gl_Normal	<code>gl_Normal</code>	<code>vec3</code>	glNormal	Lighting normal

Table I-1 Global Vertex Shader Attribute Variables

Variable Name	Type	Specifying Function	Description
<code>gl_MultiTexCoord<i>n</i></code>	<code>vec4</code>	<code>glMultiTexCoord(<i>n</i>, ...)</code>	Texture unit <i>n</i> 's texture coordinates, with <i>n</i> = 0 ... 7.
<code>gl_FogCoord</code>	<code>float</code>	<code>glFogCoord</code>	Fog coordinate
<code>gl_VertexID</code>	<code>int</code>	—	Index value of current vertex
<code>gl_InstanceID</code>	<code>int</code>	—	Instance index of current rendering primitive

Table I-1 (continued) Global Vertex Shader Attribute Variables

Vertex Shader Special Output Variables

The following variables are used in later stages of the vertex processing pipeline, and are only available in vertex shaders.

In particular, the `gl_Position` variable is the only required output of vertex shaders, and specifies the final, projected position of the input vertex. Generally, this is accomplished by directly multiplying the input vertex by the combined modelview and projection transformations, as in the following example:

```
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
```

Table I-2 enumerates the special output variables of a vertex shader.

Variable Name	Type	Description
<code>gl_Position</code>	<code>vec4</code>	Transformed vertex position.
<code>gl_PointSize</code>	<code>float</code>	Point size (in pixels) of output vertex. This value overrides the current OpenGL setting if <code>glEnable(GL_VERTEX_PROGRAM_POINT_SIZE)</code> ; has been called.
<code>gl_ClipDistance[]</code>	<code>float</code>	Distances from the vertex to each clipping plane. (This variable replaces <code>gl_ClipVertex</code> from GLSL 1.20.)

Compatibility Extension
gl_ClipVertex

Table I-2 Special Vertex Shader Output Variables

Vertex Shader Output Varying Variables

While user-defined varying variables are shared between vertex and fragment shaders, certain OpenGL-defined varying variables are accessible only in vertex shader, with their values being assigned to a unique set of (corresponding, but differently named) varying values that are available only in fragment shaders.

Table I-3 lists the variables that are only available in vertex shaders.

Compatibility Extension	Variable Name	Type	Description
gl_Front*Color gl_Back*Color gl_TexCoord gl_FogFragCoord	gl_FrontColor	vec4	Primary color to be used for front-facing primitives
	gl_BackColor	vec4	Primary color to be used for back-facing primitives
	gl_FrontSecondaryColor	vec4	Secondary color to be used for front-facing primitives
	gl_BackSecondaryColor	vec4	Secondary color to be used for back-facing primitives
	gl_TexCoord[n]	vec4	n^{th} Texture coordinates values
	gl_FogFragCoord	vec4	Fragment fog coordinate value

Table I-3 Varying Vertex Shader Output Variables

Table I-4 lists the set of input varying variables available in fragment shaders.

Compatibility Extension	Variable Name	Type	Description
gl_*Color gl_TexCoord gl_FogFragCoord	gl_Color	vec4	Primary color of the fragment
	gl_SecondaryColor	bool	Secondary color of the fragment
	gl_TexCoord[n]	vec4	n^{th} Texture coordinates for the fragment
	gl_FogFragCoord	float	Fragment's fog coordinate
	gl_PointCoord	vec2	Fragment's coordinates when point sprites are enabled

Table I-4 Varying Fragment Shader Input Variables

Built-In Implementation Constants

Most implementation-dependent maximum values are reflected as constant integer variables accessible in both vertex and fragment shaders. The variables listed in Table I-5 represent the minimum capabilities of an OpenGL implementation, and all are represented as a single constant integer expression.

Variable Name	Default Value	Description	Compatibility Extension
gl_MaxLights	8	Number of lights	gl_MaxClipPlanes gl_MaxVarying Floats
gl_MaxClipPlanes	6	Number of user-defined clipping planes	
gl_MaxTextureUnits	16	Number of texture units	
gl_MaxTextureCoords	2	Number of texture-coordinates supported.	
gl_MaxVertexAttribs	16	Number of vec4 sets of vertex attributes	
gl_MaxVertexUniformComponents	1024	Number of vec4 uniform variables available for a vertex shader	
gl_MaxVaryingFloats	64	Number of varying float variables	
gl_MaxVertexTextureImageUnits	16	Number of texture units available from within a vertex shader	
gl_MaxCombinedTextureImageUnits	16	Total number of texture units available for both vertex and fragment shaders	
gl_MaxTextureImageUnits	16	Number of texture units available from within a fragment shader	
gl_MaxFragmentUniformComponents	1024	Number of vec4 uniform variables available from a fragment shader	

Table I-5 Implementation Maximum Value Variables

Variable Name	Default Value	Description
gl_MaxDrawBuffers	8	Number of buffers available for simultaneous output from within a fragment shader using gl_FragData
gl_MaxClipDistances	8	Number of distances to user-defined clipping planes

Table I-5 (continued) Implementation Maximum Value Variables

Built-In Uniform State Variables

The following sets of variables are available in both vertex and fragment shaders, and reflect the OpenGL state settings specified by the application.

Note: All built-in uniform state, with the exception of `gl_DepthRange`, was deprecated in GLSL 1.30, and effectively removed. All of the functionality is available if your implementation includes the `GL_ARB_compatibility` extension.

Transformation Matrix State

The following variables represent the matrix transformation states of OpenGL. Various matrices are derived from the values set within an OpenGL application. For example, lighting normals (see “Define Normal Vectors for Each Vertex of Every Object” in Chapter 5) are transformed by the inverse-transpose of the upper-left 3×3 part of the modelview matrix, which is automatically computed for you when changes are made to the modelview matrix and presented using the `gl_NormalMatrix` variable.

Derived matrices (particularly those involving matrix inverses) may be undefined if the source matrix is ill-conditioned or singular.

Table I-6 lists the available matrices in GLSL.

Compatibility Extension
All fixed-function matrices

Variable Name	Type	Description
gl_ModelViewMatrix	mat4	Current modelview matrix
gl_ProjectionMatrix	mat4	Current projection matrix

Table I-6 Transformation Matrix Variables

Variable Name	Type	Description
gl_ModelViewProjectionMatrix	mat4	Product of the modelview and projection matrices
gl_TextureMatrix[n]	mat4	Texture matrix for texture unit n . The maximum dimension of the array is gl_MaxTextureCoords.
gl_NormalMatrix	mat3	Inverse-transpose of the upper 3×3 of the modelview matrix
gl_ModelViewMatrixInverse	mat4	Inverse of the modelview matrix
gl_ProjectionMatrixInverse	mat4	Inverse of the projection matrix
gl_ModelViewProjectionMatrixInverse	mat4	Inverse of combined modelview and projection matrices
gl_TextureMatrixInverse[n]	mat4	Inverse of texture matrix n
gl_ModelViewMatrixTranspose	mat4	Transpose of the modelview matrix
gl_ProjectionMatrixTranspose	mat4	Transpose of the projection matrix
gl_ModelViewProjectionMatrixTranspose	mat4	Inverse of the combined modelview and projection matrix
gl_TextureMatrixTranspose[n]	mat4	Transpose of texture matrix n
gl_ModelViewMatrixInverseTranspose	mat4	Inverse-transpose of the modelview matrix
gl_ProjectionMatrixInverseTranspose	mat4	Inverse-transpose of the projection matrix
gl_ModelViewProjectionMatrixInverseTranspose	mat4	Inverse-transpose of the combined modelview and projection matrix
gl_TextureMatrixInverseTranspose[n]	mat4	Inverse-transpose of texture matrix n

Table I-6 (continued) Transformation Matrix Variables

Lighting Normal Normalization State

Vertex normals can be normalized within a shader by using the **normalize()** function. However, if normals were uniformly scaled (all calls to **glScale()** had the same values for x , y , and z), they could be normalized by a simple scaling operation (thus saving the potentially expensive square root operation required by **normalize()**). The scaling factor is stored in:

```
uniform float gl_NormalScale;
```

and applied as

```
vec3 normal = gl_NormalScale * gl_Normal;
```

Depth Range State

Access to the depth range values (as set by **glDepthRange()**) can be accessed using:

```
struct gl_DepthRangeParameters {  
    float near;  
    float far;  
    float diff;  
};  
  
uniform gl_DepthRangeParameters gl_DepthRange;
```

Table I-7 provides descriptions for each of the depth range variables.

Variable Name	Type	Description
gl_DepthRange.near	float	Depth value that vertices at the near clipping plane are mapped to
gl_DepthRange.far	float	Depth value that vertices at the far clipping plane are mapped to
gl_DepthRange.diff	float	Difference between the near and far depth range values, computed as (far – near)

Table I-7 Depth Range Variables

User-Defined Clipping Planes

Access to the user-defined clipping planes (as specified by **glClipPlane()**) can be accessed in a shader through the array:

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

Compatibility Extension
gl_ClipPlanes

Point Parameter

Compatibility Extension
gl_Point

Parameters controlling the size and shading of points are controlled by the state set by calling **glPointParameter()**. The following structure allows access to all point parameter settings:

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

uniform gl_PointParameters gl_Point;
```

The variables associated with point sizes and parameters are listed in Table I-8.

Variable Name	Type	Description
gl_Point.size	float	Current point-size setting
gl_Point.sizeMin	float	Minimum value for derived point sizes
gl_Point.sizeMax	float	Maximum value for derived point sizes
gl_Point.fadeThresholdSize	float	Threshold value for selecting whether to fade a point's alpha value
gl_Point.distanceConstantAttenuation	float	Constant coefficient for point attenuation computations
gl_Point.distanceLinearAttenuation	float	Linear coefficient for point attenuation computations
gl_Point.distanceQuadraticAttenuation	float	Quadratic coefficient for point attenuation computations

Table I-8 Point Size and Attenuation Variables

Compatibility Extension
gl_FrontMaterial gl_BackMaterial

Material State

Material state, as specified by calling **glMaterial()**, can be queried by utilizing the values in the following structure, which stores material properties for both the front- and back-material state:

```

struct gl_MaterialParameters {
    vec4 emission;
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    float shininess;
};

uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;

```

Table I-9 lists the material variables and their descriptions.

Variable Name	Type	Description
gl_FrontMaterial.emission gl_BackMaterial.emission	vec4	Emission color for material
gl_FrontMaterial.ambient gl_BackMaterial.ambient	vec4	Ambient color for material
gl_FrontMaterial.diffuse gl_BackMaterial.diffuse	vec4	Diffuse color for material
gl_FrontMaterial.specular gl_BackMaterial.specular	vec4	Specular color for material
gl_FrontMaterial.shininess gl_BackMaterial.shininess	float	Shininess color for material

Table I-9 Lighting Material Variables

Lighting State

Compatibility Extension
gl_LightSource gl_LightModel gl_FrontLightProduct gl_BackLightProduct

The state associated with each light in OpenGL is stored in the following structure. For this section, the notation described in “The Mathematics of Lighting” on page 240 is used.

```

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];

```

Table I-10 provides a list of the parameters associated with each light source available in a shader.

Variable Name	Type	Description
<code>gl_LightSource[n].ambient</code>	vec4	Ambient color for light n
<code>gl_LightSource[n].diffuse</code>	vec4	Diffuse color for light n
<code>gl_LightSource[n].specular</code>	vec4	Specular color for light n
<code>gl_LightSource[n].position</code>	vec4	World-coordinate position for light n
<code>gl_LightSource[n].halfVector</code>	vec4	Lighting half-vector for light n
<code>gl_LightSource[n].spotDirection</code>	vec3	Spotlight direction for light n
<code>gl_LightSource[n].spotExponent</code>	float	Spotlight exponent for light n
<code>gl_LightSource[n].spotCutoff</code>	float	Spotlight cutoff angle for light n
<code>gl_LightSource[n].spotCosCutoff</code>	float	Cosine of the spotlight cutoff angle for light n
<code>gl_LightSource[n].constantAttenuation</code>	float	Constant coefficient for attenuation computation for light n
<code>gl_LightSource[n].linearAttenuation</code>	float	Linear coefficient for attenuation computation for light n
<code>gl_LightSource[n].quadraticAttenuation</code>	float	Quadratic coefficient for attenuation computation for light n

Table I-10 Light Source Variables

The following structure contains the values for the light-model parameters.

```

struct gl_LightModelParameters {
    vec4 ambient;
};

uniform gl_LightModelParameters gl_LightModel;

```

The values associated with the light model are described in Table I-11.

Variable Name	Type	Description
gl_LightModel.ambient	vec4	Light model's ambient color

Table I-11 Light Model Variables

To help reduce the computation required for computing the colors based on OpenGL's Phong lighting model, the non-light dependent terms of the lighting equation (the emission and ambient material terms) combined with the light model ambient value are cached in the following structure and are described in Table I-12.

```

struct gl_LightModelProducts {
    vec4 sceneColor;
};

uniform gl_LightModelProducts gl_FrontLightModelProduct;
uniform gl_LightModelProducts gl_BackLightModelProduct;

```

Variable Name	Type	Description
gl_FrontLightModelProduct.sceneColor	vec4	$\text{emission}_{\text{material}} + \text{ambient}_{\text{material}} *$
gl_BackLightModelProduct.sceneColor		$\text{ambient}_{\text{light model}}$

Table I-12 Cached Light Model Value Variables

Similarly, various products of constant factors of the material and lights are cached in the following structure and described in Table I-13.

```

struct gl_LightProducts
{
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
};

```

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

Variable Name	Type	Description
gl_FrontLightProduct[n].ambient gl_BackLightProduct[n].ambient	vec4	$\text{ambient}_{\text{material}} * \text{ambient}_{\text{light}}$
gl_FrontLightProduct[n].diffuse gl_BackLightProduct[n].diffuse	vec4	$(\max \{ \mathbf{L} \cdot \mathbf{n}, 0 \}) * \text{diffuse}_{\text{light}} * \text{diffuse}_{\text{material}}$
gl_FrontLightProduct[n].specular gl_BackLightProduct[n].specular	vec4	$(\max \{ \mathbf{s} \cdot \mathbf{n}, 0 \})^{\text{shininess}} * \text{specular}_{\text{light}} * \text{specular}_{\text{material}}$

Table I-13 Cached Light Product Value Variables

Texture Environment State

The texture environment color (as specified by **glTexEnv()**) for each texture unit is available by accessing:

```
uniform vec4 gl_TextureEnvColor[gl_MaxTextureImageUnits];
```

Texture Coordinate Generation State

The user-defined planes for texture coordinate generation are available by accessing the following sets of values, one for each set of texture coordinates:

```
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];
```

Compatibility Extension
gl_EyePlane* gl_ObjectPlane*

Fog

The state associated with fog and atmospheric effects (see “Fog” in Chapter 6) in OpenGL is stored in the following structure and described in Table I-14.

Compatibility Extension
gl_Fog

```

struct gl_FogParameters {
    vec4 color;
    float density;
    float start;
    float end;
    float scale;
};

uniform gl_FogParameters gl_Fog;

```

Variable Name	Type	Description
gl_Fog.color	vec4	Fog color
gl_Fog.density	float	Fog density
gl_Fog.start	float	Starting distance for linear fog
gl_Fog.end	float	Ending distance for linear fog
gl_Fog.scale	float	Scaling factor for linear fog. This value is computed as $scale = \frac{1}{end - start}$

Table I-14 Fog Variables and Cached Values

Built-In Functions

Angle Conversion and Trigonometric Functions

Table I-15 describes the trigonometric functions available in GLSL shaders. Angles are measured in radians, and the following commands operate on each component separately. *TYPE* represents one of float, vec2, vec3, or vec4.

Function Syntax	Description
<i>TYPE</i> radians(<i>TYPE</i> degrees)	Returns $\left(\frac{\pi}{180}\right) \cdot \text{degrees}$
<i>TYPE</i> degrees(<i>TYPE</i> radians)	Returns $\left(\frac{180}{\pi}\right) \cdot \text{radians}$

Table I-15 Angle Conversion and Trigonometric Functions

Function Syntax	Description
<i>TYPE</i> sin (<i>TYPE angle</i>)	Returns the sine of <i>angle</i>
<i>TYPE</i> cos (<i>TYPE angle</i>)	Returns the cosine of <i>angle</i>
<i>TYPE</i> tan (<i>TYPE angle</i>)	Returns the tangent of <i>angle</i>
<i>TYPE</i> asin (<i>TYPE x</i>)	Returns the arcsine (\sin^{-1}) of <i>x</i> . The range of values returned by this function is $[-\pi/2, \pi/2]$, and the result is undefined if $ x > 1$.
<i>TYPE</i> acos (<i>TYPE x</i>)	Returns the arccosine (\cos^{-1}) of <i>x</i> . The range of values returned by this function is $[0, \pi]$, and the result is undefined if $ x > 1$.
<i>TYPE</i> atan (<i>TYPE y</i> , <i>TYPE x</i>)	Returns the arctangent (\tan^{-1}) of y/x . The signs of <i>x</i> and <i>y</i> are used to determine what quadrant the angle is in. The range of values returned by this function is $[-\pi, \pi]$, and the result is undefined if <i>x</i> and <i>y</i> are both 0.
<i>TYPE</i> atan (<i>TYPE y_over_x</i>)	Returns the arctangent (\tan^{-1}) of <i>y_over_x</i> . The range of values returned by this function is $[-\pi/2, \pi/2]$.
<i>TYPE</i> sinh (<i>TYPE x</i>)	Returns the hyperbolic sine of <i>x</i> , which is defined as $\frac{e^x - e^{-x}}{2}$
<i>TYPE</i> cosh (<i>TYPE x</i>)	Returns the hyperbolic cosine of <i>x</i> , which is defined as $\frac{e^x + e^{-x}}{2}$
<i>TYPE</i> tanh (<i>TYPE x</i>)	Returns the hyperbolic tangent of <i>x</i> , which is defined as $\frac{\sinh(x)}{\cosh(x)}$
<i>TYPE</i> asinh (<i>TYPE x</i>)	Returns the arc (inverse) hyperbolic sine of <i>x</i> .
<i>TYPE</i> acosh (<i>TYPE x</i>)	Returns the arc (inverse) hyperbolic cosine of <i>x</i> .
<i>TYPE</i> atanh (<i>TYPE x</i>)	Returns the arc (inverse) hyperbolic tangent of <i>x</i> .

Table I-15 (continued) Angle Conversion and Trigonometric Functions

Transcendental Functions

Table I-16 lists the logarithmic and exponential functions available in GLSL. The commands operate on each component separately. *TYPE* represents one of float, vec2, vec3, or vec4.

Function Syntax	Description
<i>TYPE</i> pow (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns xy . Results are undefined if $x < 0$, or if $x = 0$ and $y \leq 0$
<i>TYPE</i> exp (<i>TYPE</i> <i>x</i>)	Returns e^x .
<i>TYPE</i> log (<i>TYPE</i> <i>x</i>)	Returns $\ln(x)$. Results are undefined if $x \leq 0$
<i>TYPE</i> exp2 (<i>TYPE</i> <i>x</i>)	Returns 2^x .
<i>TYPE</i> log2 (<i>TYPE</i> <i>x</i>)	Returns $\log_2(x)$. Results are undefined if $x \leq 0$.
<i>TYPE</i> sqrt (<i>TYPE</i> <i>x</i>)	Returns \sqrt{x} . Results are undefined if $x \leq 0$.
<i>TYPE</i> inversesqrt (<i>TYPE</i> <i>x</i>)	Returns $\frac{1}{\sqrt{x}}$. Results are undefined if $x \leq 0$.

Table I-16 Transcendental Functions

Basic Numerical Functions

Table I-17 lists the basic numerical functions that GLSL supports. Each command operates on each component separately. *TYPE* represents one of float, vec2, vec3, or vec4. *iTYPE* represents one of int, ivec2, ivec3, or ivec4. Similarly, *uTYPE* represents one of uint, uvec2, uvec3, or uvec4.

Function Syntax	Description
<i>TYPE</i> abs (<i>TYPE</i> <i>x</i>) <i>iTYPE</i> abs (<i>iTYPE</i> <i>x</i>)	Returns $ x $
<i>TYPE</i> sign (<i>TYPE</i> <i>x</i>) <i>iTYPE</i> sign (<i>iTYPE</i> <i>x</i>)	Returns $\begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$

Table I-17 Basic Numerical Functions

Function Syntax	Description
<i>TYPE</i> floor (<i>TYPE</i> <i>x</i>)	Returns a value equal to the nearest integer that is less than or equal to <i>x</i>
<i>TYPE</i> ceil (<i>TYPE</i> <i>x</i>)	Returns a value equal to the nearest integer that is greater than or equal to <i>x</i>
<i>TYPE</i> fract (<i>TYPE</i> <i>x</i>)	Returns $x - \text{floor}(x)$
<i>TYPE</i> trunc (<i>TYPE</i> <i>x</i>)	Returns the nearest integer to <i>x</i> whose absolute value is not greater than the absolute value of <i>x</i>
<i>TYPE</i> round (<i>TYPE</i> <i>x</i>)	Returns the nearest integer to <i>x</i> rounded in an implementation-dependent manner, presumably using the fastest computational approach.
<i>TYPE</i> roundEven (<i>TYPE</i> <i>x</i>)	Returns the nearest even integer to <i>x</i> by adding 0.5. For example, 3.5 and 4.5, would both round to 4.0.
<i>TYPE</i> mod (<i>TYPE</i> <i>x</i> , float <i>y</i>) <i>TYPE</i> mod (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns the floating-point modulus: $x - y \cdot \text{floor}(x/y)$
<i>TYPE</i> modf (<i>TYPE</i> <i>x</i> , out <i>TYPE</i> <i>i</i>)	Returns the fractional part of <i>x</i> and sets <i>i</i> to the integer part of <i>x</i> . Both the return value and <i>i</i> will have the same sign as <i>x</i> .
<i>TYPE</i> min (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>) <i>TYPE</i> min (<i>TYPE</i> <i>x</i> , float <i>y</i>) <i>iTYPE</i> min (<i>iTYPE</i> <i>x</i> , <i>iTYPE</i> <i>y</i>) <i>iTYPE</i> min (<i>iTYPE</i> <i>x</i> , int <i>y</i>) <i>uTYPE</i> min (<i>uTYPE</i> <i>x</i> , <i>uTYPE</i> <i>y</i>) <i>uTYPE</i> min (<i>uTYPE</i> <i>x</i> , uint <i>y</i>)	Returns $x < y ? x : y$;
<i>TYPE</i> max (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>) <i>TYPE</i> max (<i>TYPE</i> <i>x</i> , float <i>y</i>) <i>iTYPE</i> max (<i>iTYPE</i> <i>x</i> , <i>iTYPE</i> <i>y</i>) <i>iTYPE</i> max (<i>iTYPE</i> <i>x</i> , int <i>y</i>) <i>uTYPE</i> max (<i>uTYPE</i> <i>x</i> , <i>uTYPE</i> <i>y</i>) <i>uTYPE</i> max (<i>uTYPE</i> <i>x</i> , uint <i>y</i>)	Returns $x > y ? x : y$;

Table I-17 (continued) Basic Numerical Functions

Function Syntax	Description
$TYPE$ clamp ($TYPE$ x , $TYPE$ $minVal$, $TYPE$ $maxVal$) $TYPE$ clamp ($TYPE$ x , float $minVal$, float $maxVal$) $iTYPE$ clamp ($iTYPE$ x , $iTYPE$ $minVal$, $iTYPE$ $maxVal$) $iTYPE$ clamp ($iTYPE$ x , int $minVal$, int $maxVal$) $uTYPE$ clamp ($uTYPE$ x , $uTYPE$ $minVal$, $uTYPE$ $maxVal$) $uTYPE$ clamp ($uTYPE$ x , uint $minVal$, uint $maxVal$)	Returns min(max(x, minVal), maxVal)
$TYPE$ mix ($TYPE$ x , $TYPE$ y , $TYPE$ a) $TYPE$ mix ($TYPE$ x , $TYPE$ y , float a)	Returns $x \cdot (1 - a) + y \cdot a$
$TYPE$ mix ($TYPE$ x , $TYPE$ y , $bvec$ a)	Returns $a_i ? x_i : y_i$ for each component of a .
$TYPE$ step ($TYPE$ $edge$, $TYPE$ x) $TYPE$ step (float $edge$, $TYPE$ x)	Returns $x < edge ? 0.0 : 1.0$;
$TYPE$ smoothstep ($TYPE$ $edge0$, $TYPE$ $edge1$, $TYPE$ x) $TYPE$ smoothstep (float $edge0$, float $edge1$, $TYPE$ x)	Returns $\begin{cases} 0.0 & x \leq edge0 \\ t^3 - 2t^2 & edge0 < x < edge1 \\ 1.0 & x \geq edge1 \end{cases}$ where $t = \frac{x - edge0}{edge1 - edge0}$
$bvec$ isnan ($TYPE$ x)	Returns true if x is NaN (not a number), false otherwise.
$bvec$ isinf ($TYPE$ x)	Returns true if x is either positive or negative infinity (based on the underlying floating-point representation), false otherwise.

Table I-17 (continued) Basic Numerical Functions

Vector-Operation Functions

Compatibility Extension
ftransform()

Table I-18 describes the vector operations in GLSL. The following commands operate on their arguments as vectors. Where not explicitly specified, *TYPE* represents one of: float, vec2, vec3, or vec4.

Function Syntax	Description
float length (<i>TYPE</i> <i>x</i>)	Returns the length of vector <i>x</i> : sqrt ($x[0] \cdot x[0] + x[1] \cdot x[1] + \dots$)
float distance (<i>TYPE</i> <i>p0</i> , <i>TYPE</i> <i>p1</i>)	Returns the distance between <i>p0</i> and <i>p1</i> : length ($p0 - p1$)
float dot (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns the dot product of <i>x</i> and <i>y</i> : result = $x[0] \cdot y[0] + x[1] \cdot y[1] + \dots$
vec3 cross (vec3 <i>x</i> , vec3 <i>y</i>)	Returns the cross product of <i>x</i> and <i>y</i> , i.e., result.x = $x[1] \cdot y[2] - y[1] \cdot x[2]$ result.y = $x[2] \cdot y[0] - y[2] \cdot x[0]$ result.z = $x[0] \cdot y[1] - y[0] \cdot x[1]$
<i>TYPE</i> normalize (<i>TYPE</i> <i>x</i>)	Returns a vector in the same direction as <i>x</i> but with a length of 1.
vec4 ftransform ()	Returns the transformed input vertex such that it matches the output of the fixed-function vertex pipeline.
<i>TYPE</i> faceforward (<i>TYPE</i> <i>N</i> , <i>TYPE</i> <i>I</i> , <i>TYPE</i> <i>Nref</i>)	Returns dot (<i>Nref</i> , <i>I</i>) < 0 ? <i>N</i> : - <i>N</i>
<i>TYPE</i> reflect (<i>TYPE</i> <i>I</i> , <i>TYPE</i> <i>N</i>)	Returns the reflection direction for incident vector <i>I</i> , given the normalized surface orientation vector <i>N</i> : result = $I - 2 \cdot \mathbf{dot}(N, I) \cdot N$
<i>TYPE</i> refract (<i>TYPE</i> <i>I</i> , <i>TYPE</i> <i>N</i> , float <i>eta</i>)	Returns the refracted vector <i>R</i> , given the normalized incident vector <i>I</i> , normalized surface normal <i>N</i> , and ratio of indices of refraction <i>eta</i> . The refracted vector is computed in the following manner: $k = 1 - eta^2(1 - (\hat{N} \cdot \hat{I})^2)$ $\vec{R} = \begin{cases} 0 & k < 0 \\ (eta \cdot \hat{I} - eta\hat{N} \cdot \hat{I} + \sqrt{k}\hat{N}) & k > 0 \end{cases}$

Table I-18 Vector-Operation Functions

Matrix Functions

As compared to standard matrix multiplication, which is defined using the `*` operator, the `matrixCompMult()` routine, described in Table I-19, produces a component-wise multiplication of two equally dimensioned matrices.

Function Syntax	Description
<code>mat matrixCompMult(mat x, mat y)</code>	Multiply matrix x by matrix y component-wise, i.e., $result_{ij}$ is the scalar product of x_{ij} and y_{ij} .
<code>mat2 outerProduct(vec2 c, vec2 r)</code> <code>mat3 outerProduct(vec3 c, vec3 r)</code> <code>mat4 outerProduct(vec4 c, vec4 r)</code>	Generates a $c \times r$ matrix by computing the outer matrix product (also called the tensor product).
<code>mat2x3 outerProduct(vec3 c, vec2 r)</code> <code>mat3x2 outerProduct(vec2 c, vec3 r)</code>	
<code>mat2x4 outerProduct(vec4 c, vec2 r)</code> <code>mat4x2 outerProduct(vec2 c, vec4 r)</code>	
<code>mat3x4 outerProduct(vec4 c, vec3 r)</code> <code>mat4x3 outerProduct(vec3 c, vec4 r)</code>	
<code>mat2 transpose(mat2 m)</code> <code>mat3 transpose(mat3 m)</code> <code>mat4 transpose(mat4 m)</code>	Return the transpose of the matrix m . The input matrix is not modified.
<code>mat2x3 transpose(mat3x2 m)</code> <code>mat3x2 transpose(mat2x3 m)</code>	
<code>mat2x4 transpose(mat4x2 m)</code> <code>mat4x2 transpose(mat2x4 m)</code>	
<code>mat3x4 transpose(mat4x3 m)</code> <code>mat4x3 transpose(mat3x4 m)</code>	

Table I-19 Matrix Functions

Vector-Component Relational Functions

The following functions produce a vector of Boolean values representing the results of the specified operation conducted on each pair of components from the input vectors. *TYPE* represents any of *vec2*, *vec3*, *vec4*, *ivec2*, *ivec3*, *ivec4*, *uvec2*, *uvec3*, or *uvec4* in the prototypes below, or a Boolean vector when explicitly specified. The input vectors must be of the same type, and the Boolean vector returned, represented by *bvec* below, will have the same size as the input vectors.

Table I-20 describes the accepted vector functions.

Function Syntax	Description
<i>bvec lessThan</i> (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns the component-wise compare of $x < y$.
<i>bvec lessThanEqual</i> (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns the component-wise compare of $x \leq y$.
<i>bvec greaterThan</i> (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns the component-wise compare of $x > y$.
<i>bvec greaterThanEqual</i> (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>)	Returns the component-wise compare of $x \geq y$.
<i>bvec equal</i> (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>) <i>bvec equal</i> (<i>bvec</i> <i>x</i> , <i>bvec</i> <i>y</i>)	Returns the component-wise compare of $x == y$.
<i>bvec notEqual</i> (<i>TYPE</i> <i>x</i> , <i>TYPE</i> <i>y</i>) <i>bvec notEqual</i> (<i>bvec</i> <i>x</i> , <i>bvec</i> <i>y</i>)	Returns the component-wise compare of $x != y$.
<i>bool any</i> (<i>bvec</i> <i>x</i>)	Returns true if any component of <i>x</i> is true .
<i>bool all</i> (<i>bvec</i> <i>x</i>)	Returns true only if all components of <i>x</i> are true .
<i>bvec not</i> (<i>bvec</i> <i>x</i>)	Returns the component-wise logical complement of <i>x</i> .

Table I-20 Vector Component Operation Functions

Texture Lookup Functions

Texture lookup functions are available to both vertex and fragment shaders if supported by the underlying OpenGL implementation. All of these

functions rely on the use of *samplers*, which provide a handle for accessing the texture map specified by the application using the OpenGL API. The different types of samplers are detailed in “Accessing Texture Maps in Shaders” in Chapter 15.

In the following tables, sampler types (e.g., floating point, signed, and unsigned integer) are generically represented by “*SAMPLER*” with the specific texture type appended (e.g., *SAMPLER2DArray*). Likewise, optional parameters are denoted in square brackets ([...]).

The functions in Tables I-29 through I-32 provide access to textures through samplers, as set up through the OpenGL API. Texture properties such as size, pixel format, number of dimensions, filtering method, number of mip-map levels, depth comparison, and so on are also defined by OpenGL API calls. Such properties are taken into account as the texture is accessed via the built-in functions defined below.

Texture Sizes Access Functions

The following routines will return the sizes (as a set of integers) for each dimension of a texture level (lod):

Function Syntax	Description	
<code>int textureSize(SAMPLER1D sampler, int lod)</code>	Returns the dimensions of the specified lod for the texture addressed by sampler. For multi-dimensional textures, the width, height, and depth of the texture (or the number of layers for array textures) are returned in that order.	
<code>ivec2 textureSize(SAMPLER2D sampler, int lod)</code>		
<code>ivec3 textureSize(SAMPLER3D sampler, int lod)</code>		
<code>ivec2 textureSize(SAMPLERCube sampler, int lod)</code>		
<code>int textureSize(SAMPLER1DShadow sampler, int lod)</code>		
<code>ivec2 textureSize(SAMPLER2DShadow sampler, int lod)</code>		
<code>ivec2 textureSize(SAMPLERCubeShadow sampler, int lod)</code>		
<code>ivec2 textureSize(SAMPLER1DArray sampler, int lod)</code>		
<code>ivec3 textureSize(SAMPLER2DArray sampler, int lod)</code>		
<code>ivec2 textureSize(SAMPLER1DArrayShadow sampler, int lod)</code>		
<code>ivec3 textureSize(SAMPLER2DArrayShadow sampler, int lod)</code>		
Added in GLSL 1.40:		
<code>ivec2 textureSize(SAMPLER2DRect sampler, int lod)</code>		
<code>ivec2 textureSize(SAMPLER2DRectShadow sampler, int lod)</code>		
<code>int textureSize(SAMPLERBuffer sampler, int lod)</code>		

Table I-21 Texture Size Access Functions

Basic Texture Access Functions

The following functions use the provided texture coordinates to retrieve a texel from the texture map associated with *sampler*.

All forms are accepted by fragment shaders. Functions with the *bias* parameter are not accepted in vertex shaders.

The optional *bias* parameter is used to alter which mipmap is sampled (see “Calculating the Mipmap Level” in Chapter 9). If a *bias* value is provided while accessing a texture that does not contain mipmaps, the texture is accessed directly, and the value is ignored. The *bias* forms of the texture functions are available only in fragment shaders.

Function Syntax	Description	
<code>gvec4 texture(SAMPLER1D sampler, float coord [, float bias])</code>	Samples the texture associated with <i>sampler</i> at the coordinate <i>coord</i> , adding <i>bias</i> to the computed mipmap level-of-detail.	
<code>gvec4 texture(SAMPLER2D sampler, vec2 coord [, float bias])</code>		
<code>gvec4 texture(SAMPLER3D sampler, vec3 coord [, float bias])</code>		
<code>gvec4 texture(SAMPLERCube sampler, vec3 coord [, float bias])</code>		
<code>float texture(SAMPLER1DShadow sampler, vec3 coord [, float bias])</code>		
<code>float texture(SAMPLER2DShadow sampler, vec3 coord [, float bias])</code>		
<code>float texture(SAMPLERCubeShadow sampler, vec4 coord [, float bias])</code>		
<code>gvec4 texture(SAMPLER1DArray sampler, vec2 coord [, float bias])</code>		
<code>gvec4 texture(SAMPLER2DArray sampler, vec3 coord [, float bias])</code>		
<code>float texture(SAMPLER1DArrayShadow sampler, vec3 coord [, float bias])</code>		
<code>float texture(SAMPLER2DArrayShadow sampler, vec4 coord)</code>		
Added in GLSL 1.40:		
<code>gvec4 texture(SAMPLER2DRect sampler, vec2 coord)</code>		
<code>float texture(SAMPLER2DRectShadow sampler, vec3 coord)</code>		

Table I-22 Basic Texture Access Functions

Texture Lookup Functions Using a Constant Coordinate Offset

Sample a texture using a constant offset that is applied to the texture coordinates provided. If a *bias* value is specified, it is added to the mipmap lod parameter before sampling. The *bias* form is not valid in vertex shaders.

Function Syntax	Description
<code>gvec4 textureOffset(SAMPLER1D sampler, float coord, int offset [, float bias])</code>	Sample a texture offsetting the provided coordinates. The range of offset values is implementation dependent, and may be queried by the application (not the shader, however) by querying the values <code>GL_MIN_PROGRAM_TEXEL_OFFSET</code> , and <code>GL_MAX_PROGRAM_TEXEL_OFFSET</code> .
<code>gvec4 textureOffset(SAMPLER2D sampler, vec2 coord, ivec2 offset [, float bias])</code>	
<code>gvec4 textureOffset(SAMPLER3D sampler, vec3 coord, ivec3 offset [, float bias])</code>	
<code>float textureOffset(SAMPLER1DShadow sampler, vec3 coord, int offset [, float bias])</code>	The offset value is not applied to the layer value for Array samplers.
<code>float textureOffset(SAMPLER2DShadow sampler, vec3 coord, ivec2 offset [, float bias])</code>	
<code>gvec4 textureOffset(SAMPLER1DArray sampler, vec2 coord, int offset [, float bias])</code>	
<code>gvec4 textureOffset(SAMPLER2DArray sampler, vec2 coord, ivec2 offset [, float bias])</code>	
<code>gvec4 textureOffset(SAMPLER1DArrayShadow sampler, vec2 coord, int offset [, float bias])</code>	
Added in GLSL 1.40:	
<code>gvec4 textureOffset(SAMPLER2DRect sampler, vec2 coord, ivec2 offset)</code>	
<code>float textureOffset(SAMPLER2DShadow sampler, vec3 coord, ivec2 offset)</code>	

Table I-23 Constant Texture-Coordinate Offset Texture Access Functions

Single Texel Fetch Functions

The following functions retrieve a single texel (without filtering) from a texture. The ***Offset** forms offset the supplied texture coordinates by an offset, similar to the **textureOffset()** functions.

Function Syntax	Description
<code>gvec4 texelFetch(SAMPLER1D sampler, int coord, int lod)</code>	Retrieve a single unfiltered texel from a texture map addressed by <i>coord</i> from the level-of-detail <i>lod</i> .
<code>gvec4 texelFetch(SAMPLER2D sampler, ivec2 coord, int lod)</code>	
<code>gvec4 texelFetch(SAMPLER3D sampler, ivec3 coord, int lod)</code>	
<code>gvec4 texelFetch(SAMPLER1DArray sampler, ivec2 coord, int lod)</code>	
<code>gvec4 texelFetch(SAMPLER2DArray sampler, ivec3 coord, int lod)</code>	
Added in GLSL 1.40:	
<code>gvec4 texelFetch(SAMPLER2DRect sampler, ivec2 coord)</code>	
<code>gvec4 texelFetch(SAMPLERBuffer sampler, int coord)</code>	
<code>gvec4 texelFetchOffset(SAMPLER1D sampler, int coord, int lod, int offset)</code>	Retrieve a single unfiltered texel from a texture map addressed by <i>coord</i> from the level-of-detail <i>lod</i> . Texture coordinates are offset using the values specified in <i>offset</i> .
<code>gvec4 texelFetchOffset(SAMPLER2D sampler, ivec2 coord, int lod, int offset)</code>	
<code>gvec4 texelFetchOffset(SAMPLER3D sampler, ivec3 coord, int lod, int offset)</code>	
<code>gvec4 texelFetchOffset(SAMPLER1DArray sampler, ivec2 coord, int lod, int offset)</code>	
<code>gvec4 texelFetchOffset(SAMPLER2DArray sampler, ivec3 coord, int lod, int offset)</code>	
<code>gvec4 texelFetchOffset(SAMPLER2DRect sampler, ivec2 coord, int offset)</code>	
<code>gvec4 texelFetchOffset(SAMPLERBuffer sampler, int coord, int offset)</code>	

Table I-24 Single Texel Texture Access Functions

Projected Texture Access Functions

The following functions retrieve a texel from the texture map associated with *sampler*. Before the texel is retrieved, all components of the input texture coordinate, *coord*, are divided by the last coordinate.

All forms are accepted by fragment shaders. Functions with the *bias* parameter are not accepted in vertex shaders.

Once again, the optional *bias* parameter is used to alter which mipmap is sampled (see “Calculating the Mipmap Level” in Chapter 9). If a *bias* value is provided while accessing a texture that does not contain mipmaps, the texture is accessed directly and the value is ignored. The *bias* forms of the texture functions are available only in fragment shaders.

The ***Offset** forms apply the specified offset, similar to the operation of the **texelOffset()** routines, to the provided coordinates before the projective divide and texture sampling.

Function Syntax	Description
<code>gvec4 textureProj(SAMPLER1D sampler, vec2 coord [, float bias])</code>	Sample a texture with projection. Texture coordinates are divided by the last component of <i>coord</i> before sampling. For shadow maps, the third component of the divided coordinate is used as the shadow map reference value.
<code>gvec4 textureProj(SAMPLER1D sampler, vec4 coord [, float bias])</code>	
<code>gvec4 textureProj(SAMPLER2D sampler, vec3 coord [, float bias])</code>	
<code>gvec4 textureProj(SAMPLER2D sampler, vec4 coord [, float bias])</code>	
<code>gvec4 textureProj(SAMPLER3D sampler, vec4 coord [, float bias])</code>	
<code>float textureProj(SAMPLER1DShadow sampler, vec4 coord [, float bias])</code>	
<code>gvec4 textureProj(SAMPLER2D sampler, vec4 coord [, float bias])</code>	
Added in GLSL 1.40:	
<code>gvec4 textureProj(SAMPLER2DRect sampler, vec3 coord)</code>	
<code>gvec4 textureProj(SAMPLER2DRect sampler, vec4 coord)</code>	
<code>float textureProj(SAMPLER2DRectShadow sampler, vec4 coord)</code>	

Table I-25 Projected Texture Access Functions

Function Syntax	Description
gvec4 textureProjOffset (<i>SAMPLER1D</i> sampler, vec2 coord, int offset [, float bias])	Sample a texture with projection. Texture coordinates are divided by the last component of coord before sampling. For shadow maps, the third component of the divided coordinate is used as the shadow map reference value.
gvec4 textureProjOffset (<i>SAMPLER1D</i> sampler, vec4 coord, int offset [, float bias])	
gvec4 textureProjOffset (<i>SAMPLER2D</i> sampler, vec3 coord, ivec2 offset [, float bias])	
gvec4 textureProjOffset (<i>SAMPLER2D</i> sampler, vec4 coord, ivec2 offset [, float bias])	
gvec4 textureProjOffset (<i>SAMPLER3D</i> sampler, vec4 coord, ivec3 offset [, float bias])	
float textureProjOffset (<i>SAMPLER1DShadow</i> sampler, vec4 coord, int offset [, float bias])	
gvec4 textureProjOffset (<i>SAMPLER2D</i> sampler, vec4 coord, ivec2 offset [, float bias])	
Added in GLSL 1.40:	
gvec4 textureProjOffset (<i>SAMPLER2DRect</i> sampler, vec3 coord, ivec2 offset)	
gvec4 textureProjOffset (<i>SAMPLER2DRect</i> sampler, vec4 coord, ivec2 offset)	
float textureProjOffset (<i>SAMPLERRectShadow</i> sampler, vec4 coord, ivec2 offset)	

Table I-25 (continued) Projected Texture Access Functions

Explicit Texture LOD Access Functions

The following functions retrieve texels only from a single mipmap level explicitly specified by the *lod* parameter.

The ***Offset** forms apply the specified offset, similar to the operation of the **texelOffset()** routines, to the provided coordinates before texture sampling.

Function Syntax	Description
<code>gvec4 textureLod(SAMPLER1D sampler, float coord, float lod)</code>	Sample from the single mipmap level specified by <i>lod</i> . These calls also set all partial derivatives associated with mipmap-level interpolation to zero.
<code>gvec4 textureLod(SAMPLER2D sampler, vec2 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLER3D sampler, vec3 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLERCube sampler, vec3 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLER1DShadow sampler, vec3 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLER2DShadow sampler, vec3 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLER1DArray sampler, vec2 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLER2DArray sampler, vec2 coord, float lod)</code>	
<code>gvec4 textureLod(SAMPLER1DArrayShadow sampler, vec2 coord, float lod)</code>	
<code>gvec4 textureLodOffset(SAMPLER1D sampler, float coord, float lod, int offset)</code>	
<code>gvec4 textureLodOffset(SAMPLER2D sampler, vec2 coord, float lod, ivec2 offset)</code>	
<code>gvec4 textureLodOffset(SAMPLER3D sampler, vec3 coord, float lod, ivec3 offset)</code>	
<code>float textureLodOffset(SAMPLER1DShadow sampler, vec3 coord, float lod, ivec offset)</code>	
<code>float textureLodOffset(SAMPLER2DShadow sampler, vec3 coord, float lod, ivec offset)</code>	
<code>gvec4 textureLodOffset(SAMPLER1DArray sampler, vec2 coord, float lod, ivec offset)</code>	
<code>gvec4 textureLodOffset(SAMPLER2DArray sampler, vec2 coord, float lod, ivec offset)</code>	
<code>float textureLodOffset(SAMPLER1DArrayShadow sampler, vec2 coord, float lod, ivec offset)</code>	

Table I-26 Explicit Texture LOD Access Functions

Explicit Gradient Texture Access Functions

The following functions sample a texture using the user-supplied gradients in the mipmap level determination. The ***Offset** forms apply the specified offset, similar to the operation of the `texelOffset()` routines, to the provided coordinates before texture sampling.

Function Syntax	Description	
<code>gvec4 textureGrad(SAMPLER1D sampler, float coord, float dPdx, float dPdy)</code>	Sample a texture using the provided explicit gradients. The partial derivatives are with respect to the window <i>x</i> and <i>y</i> coordinates.	
<code>gvec4 textureGrad(SAMPLER2D sampler, vec2 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>gvec4 textureGrad(SAMPLER3D sampler, vec3 coord, vec3 dPdx, vec3 dPdy)</code>		
<code>gvec4 textureGrad(SAMPLERCube sampler, vec3 coord, vec3 dPdx, vec3 dPdy)</code>		
<code>float textureGrad(SAMPLER1DShadow sampler, vec3 coord, float dPdx, float dPdy)</code>		
<code>float textureGrad(SAMPLER2DShadow sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>float textureGrad(SAMPLERCubeShadow sampler, vec4 coord, vec3 dPdx, vec3 dPdy)</code>		
<code>gvec4 textureGrad(SAMPLER1DArray sampler, vec2 coord, float dPdx, float dPdy)</code>		
<code>gvec4 textureGrad(SAMPLER2DArray sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>float textureGrad(SAMPLER1DArrayShadow sampler, vec3 coord, float dPdx, float dPdy)</code>		
<code>float textureGrad(SAMPLER2DShadow sampler, vec4 coord, vec2 dPdx, vec2 dPdy)</code>		
Added in GLSL 1.40:		
<code>gvec4 textureGrad(SAMPLER2DRect sampler, vec2 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>float textureGrad(SAMPLER2DRectShadow sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>		

Table I-27 Explicit Gradient Texture Access Functions

Function Syntax	Description	
<code>gvec4 textureGradOffset(SAMPLER1D sampler, float coord, float dPdx, float dPdy)</code>	Sample a texture using the provided explicit gradient values. The provided offset is applied to the texture coordinates before sampling.	
<code>gvec4 textureGradOffset(SAMPLER2D sampler, vec2 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>gvec4 textureGradOffset(SAMPLER3D sampler, vec3 coord, vec3 dPdx, vec3 dPdy)</code>		
<code>gvec4 textureGradOffset(SAMPLERCube sampler, vec3 coord, vec3 dPdx, vec3 dPdy)</code>		
<code>float textureGradOffset(SAMPLER1DShadow sampler, vec3 coord, float dPdx, float dPdy)</code>		
<code>float textureGradOffset(SAMPLER2DShadow sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>float textureGradOffset(SAMPLERCubeShadow sampler, vec4 coord, vec3 dPdx, vec3 dPdy)</code>		
<code>gvec4 textureGradOffset(SAMPLER1DArray sampler, vec2 coord, float dPdx, float dPdy)</code>		
<code>gvec4 textureGradOffset(SAMPLER2DArray sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>float textureGradOffset(SAMPLER1DArrayShadow sampler, vec3 coord, float dPdx, float dPdy)</code>		
<code>float textureGradOffset(SAMPLER2DShadow sampler, vec4 coord, vec2 dPdx, vec2 dPdy)</code>		
Added in GLSL 1.40:		
<code>gvec4 textureGradOffset(SAMPLER2DRect sampler, vec2 coord, vec2 dPdx, vec2 dPdy)</code>		
<code>float textureGradOffset(SAMPLER2DShadow sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>		

Table I-27 **(continued)** Explicit Gradient Texture Access Functions

Combination Texture Access Functions

The following routines provide combinations of the other routines for texture access. For example, the `textureProjLod()` function does a projective texture sample on a single mipmap level. As always, the `*Offset()` variants apply a coordinate offset before sampling the texture, similar to `textureOffset()`.

Function Syntax	Description
<code>gvec4 textureProjLod(SAMPLER1D sampler, vec2 coord, float lod)</code>	Sample a texture using a projective lookup within an explicit LOD.
<code>gvec4 textureProjLod(SAMPLER1D sampler, vec4 coord, float lod)</code>	
<code>gvec4 textureProjLod(SAMPLER2D sampler, vec3 coord, float lod)</code>	
<code>gvec4 textureProjLod(SAMPLER2D sampler, vec4 coord, float lod)</code>	
<code>gvec4 textureProjLod(SAMPLER3D sampler, vec4 coord, float lod)</code>	
<code>float textureProjLod(SAMPLER1DShadow sampler, vec4 coord, float lod)</code>	
<code>float textureProjLod(SAMPLER2DShadow sampler, vec4 coord, float lod)</code>	Sample a texture using a projective lookup within an explicit LOD, offsetting the supplied coordinates.
<code>gvec4 textureProjLodOffset(SAMPLER1D sampler, vec2 coord, float lod, int offset)</code>	
<code>gvec4 textureProjLodOffset(SAMPLER1D sampler, vec4 coord, float lod, int offset)</code>	
<code>gvec4 textureProjLodOffset(SAMPLER2D sampler, vec3 coord, float lod, ivec2 offset)</code>	
<code>gvec4 textureProjLodOffset(SAMPLER2D sampler, vec4 coord, float lod, ivec2 offset)</code>	
<code>gvec4 textureProjLodOffset(SAMPLER3D sampler, vec4 coord, float lod, ivec3 offset)</code>	
<code>float textureProjLodOffset(SAMPLER1DShadow sampler, vec4 coord, float lod, int offset)</code>	
<code>float textureProjLodOffset(SAMPLER2DShadow sampler, vec4 coord, float lod, ivec2 offset)</code>	

Table I-28 Projected, LOD, and Gradient Combined Texture Access Functions

Function Syntax	Description
<code>gvec4 textureProjGrad(SAMPLER1D sampler, vec2 coord, float dPdx, float dPdy)</code>	Sample a texture using a projective lookup with an explicit gradient.
<code>gvec4 textureProjGrad(SAMPLER1D sampler, vec4 coord, float dPdx, float dPdy)</code>	
<code>gvec4 textureProjGrad(SAMPLER2D sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>	
<code>gvec4 textureProjGrad(SAMPLER2D sampler, vec4 coord, vec2 dPdx, vec2 dPdy)</code>	
<code>gvec4 textureProjGrad(SAMPLER3D sampler, vec4 coord, vec3 dPdx, vec3 dPdy)</code>	
<code>float textureProjGrad(SAMPLER1DShadow sampler, vec4 coord, float dPdx, float dPdy)</code>	
<code>float textureProjGrad(SAMPLER2DShadow sampler, vec4 coord, float dPdx, float dPdy)</code>	
Added in GLSL 1.40:	
<code>gvec4 textureProjGrad(SAMPLER2DRect sampler, vec3 coord, vec2 dPdx, vec2 dPdy)</code>	
<code>gvec4 textureProjGrad(SAMPLER2DRect sampler, vec4 coord, vec2 dPdx, vec2 dPdy)</code>	
<code>float textureProjGrad(SAMPLER2DRectShadow sampler, vec4 coord, float dPdx, float dPdy)</code>	

Table I-28 (continued) Projected, LOD, and Gradient Combined Texture Access Functions

Function Syntax	Description
<code>gvec4 textureProjGradOffset(SAMPLER1D sampler, vec2 coord, float dPdx, float dPdy, int offset)</code>	Sample a texture using a projective lookup with an explicit gradient., offsetting the supplied coordinates.
<code>gvec4 textureProjGradOffset(SAMPLER1D sampler, vec4 coord, float dPdx, float dPdy, int offset)</code>	
<code>gvec4 textureProjGradOffset(SAMPLER2D sampler, vec3 coord, vec2 dPdx, vec2 dPdy, ivec2 offset)</code>	
<code>gvec4 textureProjGradOffset(SAMPLER2D sampler, vec4 coord, vec2 dPdx, vec2 dPdy, ivec2 offset)</code>	
<code>gvec4 textureProjGradOffset(SAMPLER3D sampler, vec4 coord, vec3 dPdx, vec3 dPdy, ivec3 offset)</code>	
<code>float textureProjGradOffset(SAMPLER1DShadow sampler, vec4 coord, float dPdx, float dPdy, int offset)</code>	
<code>float textureProjGradOffset(SAMPLER2DShadow sampler, vec4 coord, vec2 dPdx, vec2 dPdy, ivec2 offset)</code>	
Added in GLSL 1.40:	
<code>gvec4 textureProjGradOffset(SAMPLER2DRect sampler, vec3 coord, vec2 dPdx, vec2 dPdy, ivec2 offset)</code>	
<code>gvec4 textureProjGradOffset(SAMPLER2DRect sampler, vec4 coord, vec2 dPdx, vec2 dPdy, ivec2 offset)</code>	
<code>float textureProjGradOffset(SAMPLER2DRectShadow sampler, vec4 coord, vec2 dPdx, vec2 dPdy, ivec2 offset)</code>	

Table I-28 (continued) Projected, LOD, and Gradient Combined Texture Access Functions

Deprecated Texture Access Functions

The following set of functions were deprecated in GLSL Version 1.30, but had not been removed from the language as of the time of this book's publication.

Projective Texture Access Functions

The following functions retrieve a texel from the texture map associated with *sampler*. Before the texel is retrieved, all components of the input texture coordinate, *coord*, are divided by the last coordinate.

Function Syntax	Description
vec4 texture1D (sampler1D <i>sampler</i> , float <i>coord</i>)	Samples a one-dimensional texture map, with optional mipmap-level biasing
vec4 texture1D (sampler1D <i>sampler</i> , float <i>coord</i> , float <i>bias</i>)	
vec4 texture2D (sampler2D <i>sampler</i> , vec2 <i>coord</i>)	Samples a two-dimensional texture map, with optional mipmap-level biasing
vec4 texture2D (sampler2D <i>sampler</i> , vec2 <i>coord</i> , float <i>bias</i>)	
vec4 texture3D (sampler3D <i>sampler</i> , vec3 <i>coord</i>)	Samples a three-dimensional texture map, with optional mipmap-level biasing
vec4 texture3D (sampler3D <i>sampler</i> , vec3 <i>coord</i> , float <i>bias</i>)	

Table I-29 Deprecated Basic Texture Access Functions

All forms are accepted by fragment shaders. Functions with the *bias* parameter are not accepted in vertex shaders.

Once again, The optional *bias* parameter is used to alter which mipmap is sampled (see “Calculating the Mipmap Level” in Chapter 9). If a *bias* value is provided while accessing a texture that does not contain mipmaps, the texture is accessed directly and the value is ignored. The *bias* forms of the texture functions are only available in fragment shaders.

Function Syntax	Description
vec4 texture1DProj (sampler1D <i>sampler</i> , vec2 <i>coord</i>)	Samples a one-dimensional texture map, with optional mipmap-level biasing. The input texture coordinate is divided by the last texture coordinates (<i>coord.s</i> for the vec2 call; <i>coord.q</i> for the vec4 call).
vec4 texture1DProj (sampler1D <i>sampler</i> , vec4 <i>coord</i>)	
vec4 texture1DProj (sampler1D <i>sampler</i> , vec2 <i>coord</i> , float <i>bias</i>)	
vec4 texture1DProj (sampler1D <i>sampler</i> , vec4 <i>coord</i> , float <i>bias</i>)	

Table I-30 Projective Texture Access Functions

Function Syntax	Description
vec4 texture2DProj (sampler2D <i>sampler</i> , vec3 <i>coord</i>)	Samples a two-dimensional texture map, with optional mipmap-level biasing. The input texture coordinate is divided by the last texture coordinates (<i>coord.t</i> for the vec2 call; <i>coord.q</i> for the vec4 call).
vec4 texture2DProj (sampler2D <i>sampler</i> , vec4 <i>coord</i>)	
vec4 texture2DProj (sampler2D <i>sampler</i> , vec3 <i>coord</i> , float <i>bias</i>)	
vec4 texture2DProj (sampler2D <i>sampler</i> , vec4 <i>coord</i> , float <i>bias</i>)	
vec4 texture3DProj (sampler3D <i>sampler</i> , vec4 <i>coord</i>)	Samples a three-dimensional texture map, with optional mipmap-level biasing. The input texture coordinate is divided by the last texture coordinates (<i>coord.p</i> for the vec2 call; <i>coord.q</i> for the vec4 call).
vec4 texture3DProj (sampler3D <i>sampler</i> , vec4 <i>coord</i> , float <i>bias</i>)	

Table I-30 Projective Texture Access Functions

Vertex Shader Texture Access Functions

Texel retrieval in vertex shaders does not include the automatic computation of a mipmap level, as happens in a fragment shader. The following functions allow for the sampling of mipmapped textures by vertex shaders by providing an mipmap level-of-detail, or *lod* value (see “Calculating the Mipmap Level” in Chapter 9 for how to compute the level-of-detail value).

The vertex shader versions of the texture access functions are available in non-projective and projective forms (those calls that include “**Proj**” in the name. The input texture coordinates for the projective forms are processed as described in “Projective Texture Access Functions” on page 33.

Function Syntax	Description
vec4 texture1DLod (sampler1D <i>sampler</i> , float <i>coord</i> , float <i>lod</i>)	Samples a one-dimensional texture map by specifying the mipmap level-of-detail.
vec4 texture1DProjLod (sampler1D <i>sampler</i> , vec2 <i>coord</i> , float <i>lod</i>)	
vec4 texture1DProjLod (sampler1D <i>sampler</i> , vec4 <i>coord</i> , float <i>lod</i>)	

Table I-31 Vertex Shader Texture Access Functions

Function Syntax	Description
vec4 texture2DLod (sampler2D <i>sampler</i> , vec2 <i>coord</i> , float <i>lod</i>)	Samples a two-dimensional texture map by specifying the mipmap level-of-detail.
vec4 texture2DProjLod (sampler2D <i>sampler</i> , vec3 <i>coord</i> , float <i>lod</i>)	
vec4 texture2DProjLod (sampler2D <i>sampler</i> , vec4 <i>coord</i> , float <i>lod</i>)	
vec4 texture3DLod (sampler3D <i>sampler</i> , vec3 <i>coord</i> , float <i>lod</i>)	Samples a three-dimensional texture map by specifying the mipmap level-of-detail.
vec4 texture3DProjLod (sampler3D <i>sampler</i> , vec4 <i>coord</i> , float <i>lod</i>)	

Table I-31 (continued) Vertex Shader Texture Access Functions

Cube-Map Texture Access Functions

Retrieve the texel at input texture coordinate, *coord*, in the cube map texture currently bound to *sampler*. The direction of *coord* is used to select which face to do a two-dimensional texture lookup in.

Function Syntax	Description
vec4 textureCube (samplerCube <i>sampler</i> , vec3 <i>coord</i>)	Samples a cube-mapped texture, with optional <i>bias</i> (fragment shader only) or <i>lod</i> parameters.
vec4 textureCube (samplerCube <i>sampler</i> , vec3 <i>coord</i> , float <i>bias</i>)	
vec4 textureCubeLod (samplerCube <i>sampler</i> , vec3 <i>coord</i> , float <i>lod</i>)	

Table I-32 Cube-Map Texture Access Functions

Shadow-Map Texture Access Functions

If a non-shadow texture call is made to a sampler that represents a depth texture with depth comparisons turned on, then results are undefined. If a shadow texture call is made to a sampler that represents a depth texture with depth comparisons turned off, then results are undefined. If a shadow texture call is made to a sampler that does not represent a depth texture, then results are undefined.

Function Syntax	Description
vec4 shadow1D (sampler1DShadow <i>sampler</i> , vec3 <i>coord</i>)	Sample a shadow depth map. The functions utilizing the <i>bias</i> parameter are only available in fragment shaders. The LOD and projective versions of the functions process texture coordinates similarly to other LOD and projective functions, respectively.
vec4 shadow2D (sampler2DShadow <i>sampler</i> , vec3 <i>coord</i>)	
vec4 shadow1D (sampler1DShadow <i>sampler</i> , vec3 <i>coord</i> , float <i>bias</i>)	
vec4 shadow2D (sampler2DShadow <i>sampler</i> , vec3 <i>coord</i> , float <i>bias</i>)	
vec4 shadow1DProj (sampler1DShadow <i>sampler</i> , vec4 <i>coord</i>)	
vec4 shadow2DProj (sampler2DShadow <i>sampler</i> , vec4 <i>coord</i>)	
vec4 shadow1DProj (sampler1DShadow <i>sampler</i> , vec4 <i>coord</i> , float <i>bias</i>)	
vec4 shadow2DProj (sampler2DShadow <i>sampler</i> , vec4 <i>coord</i> , float <i>bias</i>)	
vec4 shadow1DLod (sampler1DShadow <i>sampler</i> , vec3 <i>coord</i> , float <i>lod</i>)	
vec4 shadow2DLod (sampler2DShadow <i>sampler</i> , vec3 <i>coord</i> , float <i>lod</i>)	
vec4 shadow1DProjLod (sampler1DShadow <i>sampler</i> , vec4 <i>coord</i> , float <i>lod</i>)	
vec4 shadow2DProjLod (sampler2DShadow <i>sampler</i> , vec4 <i>coord</i> , float <i>lod</i>)	

Table I-33 Shadow-Map Texture Access Functions

Fragment Processing Functions

Table I-34 describes the functions for accessing a fragment's derivative, which are computed as the fragments of the underlying geometric primitive are rasterized.

Function Syntax	Description
<i>TYPE</i> dFdx (<i>TYPE</i> <i>p</i>)	Returns the derivative in the x-direction.
<i>TYPE</i> dFdy (<i>TYPE</i> <i>p</i>)	Returns the derivative in the y-direction
<i>TYPE</i> fwidth (<i>TYPE</i> <i>p</i>)	Returns $ dFdx(p) + dFdy(p) $

Table I-34 Fragment Derivative Functions

Noise Functions

The stochastic noise functions described in Table I-35 are available in both vertex and fragment shaders. The computed noise values returned are pseudo-random, but repeatable provided the same random number seed is used.

Function Syntax	Description
float noise1 (<i>TYPE</i> <i>x</i>)	Returns a 1D noise value based on the input value <i>x</i> .
vec2 noise2 (<i>TYPE</i> <i>x</i>)	Returns a 2D noise value based on the input value <i>x</i> .
vec3 noise3 (<i>TYPE</i> <i>x</i>)	Returns a 3D noise value based on the input value <i>x</i> .
vec4 noise4 (<i>TYPE</i> <i>x</i>)	Returns a 4D noise value based on the input value <i>x</i> .

Table I-35 Random-Noise Generation Functions