The OpenGL Shading Language 1.50 Quick Reference Card

The OpenGL® Shading Language is several closely-related languages which are used to create shaders for each of the programmable processors contained in the OpenGL processing pipeline.

[n.n.n] and [Table n.n] refer to sections and tables in the specification at www.opengl.org/registry

Content shown in blue is removed from the OpenGL 3.2 core profile and present only in the OpenGL 3.2 compatibility profile.

Types [4.1.1-4.1.10]

Transparent Types

void	no function return value
bool	Boolean
int, uint	signed and unsigned integers
float	floating scalar
vec2, vec3, vec4	floating point vector
bvec2, bvec3, bvec4	Boolean vector
ivec2, ivec2, ivec3 **uvec2, uvec2, uvec3**	signed and unsigned integer vector
mat2, mat3, mat4	2x2, 3x3, 4x4 float matrix
mat2x2, mat2x3, mat2x4	2-column float matrix with 2, 3, or 4 rows
mat3x2, mat3x3, mat3x4	3-column float matrix with 2, 3, or 4 rows
mat4x2, mat4x3, mat4x4	4-column float matrix with 2, 3, or 4 rows

Floating-Point Sampler Types (Opaque)

sampler[1,2,3]D	access a 1D, 2D, or 3D texture
samplerCube	access cube mapped texture
sampler2DRect	access rectangular texture
sampler[1,2]DShadow	access 1D or 2D depth texture/comparison
sampler2DRectShadow	access rectangular texture/comparison
sampler[1,2]DArray	access 1D or 2D array texture
sampler[1,2]DArrayShadow	access 1D or 2D array depth texture/ comparison
samplerBuffer	access buffer texture
sampler2DMS	access 2D multi-sample texture
sampler2DMSArray	access 2D multi-sample array texture

Integer Sampler Types (Opaque)

isampler[1,2,3]D	access integer 1D, 2D, or 3D texture
isamplerCube	access integer cube mapped texture
isampler2DRect	access integer 2D rectangular texture
isampler[1,2]DArray	access integer 1D or 2D array texture
isamplerBuffer	access integer buffer texture
isampler2DMS	access integer 2D multi-sample texture
isampler2DMSArray	access int. 2D multi-sample array texture

Unsigned Integer Sampler Types (Opaque)

usampler[1,2,3]D	access unsigned int 1D, 2D, or 3D texture
usamplerCube	access unsigned int cube mapped texture
usampler2DRect	access unsigned int rectangular texture
usampler[1,2]DArray	access 1D or 2D array texture
usamplerBuffer	access unsigned integer buffer texture
usampler2DMS	access uint 2D multi-sample texture
usampler2DMSArray	access uint 2D multi-sample array texture

Implicit Conversions (*All others must use constructors*)

Expression type	*Implicitly converted to type*
int, uint	float
ivec2, uvec2	vec2
ivec3, uvec3	vec3
ivec4, uvec4	vec4

Aggregation of Basic Types

Arrays	**float**[3] foo; **float** foo[3]; * structures and blocks can be arrays * only 1-dimensional arrays supported * structure members can be arrays

Structures	**struct** *type-name* {
	members
	} *struct-name*[]; // optional variable declaration, // optionally an array

Blocks	**in/out/uniform** *block-name* { // interface matching by // block name
	optionally-qualified members
	} *instance-name*[]; // optional instance name, // optionally an array

Preprocessor [3.3]

Preprocessor Operators

Preprocessor operators follow C++ standards. Preprocessor expressions are evaluated according to the behavior of the host processor, not the processor targeted by the shader.

Preprocessor Directives

Each number sign (#) can be preceded in its line only by spaces or horizontal tabs.

*#define* *#undef* *#if* *#ifdef*
#ifndef *#else* *#elif* *#endif* *#error*
#pragma *#extension* *#version* *#line*

#version 150 #version 150 compatibility	"#version 150" is required in shaders using version 1.50 of the language. #version must occur in a shader before anything else other than white space or comments. Use "compatibility" to access features in the compatibility profile.
#extension *extension_name* : *behavior* #extension all : *behavior*	- *behavior*: require, enable, warn, disable - *extension_name*: the extension supported by the compiler, or "all"

Predefined Macros

__LINE__ __FILE__	Decimal integer constants	__VERSION__	Decimal integer, e.g.: 150

Qualifiers

Storage Qualifiers [4.3]

Variable declarations may have one storage qualifier.

none	(default) local read/write memory, or input parameter
const	compile-time constant, or read-only function parameter
in **centroid in**	linkage into a shader from previous stage (copied in) linkage with centroid based interpolation
out **centroid out**	linkage out of a shader to subsequent stage (copied out) linkage with centroid based interpolation
uniform	linkage between a shader, OpenGL, and the application

Uniform [4.3.5]

Use to declare global variables with the same values across the entire primitive being processed. Uniform variables are read-only. Use uniform qualifiers with any basic data types or array of these, or when declaring a variable whose type is a structure, e.g.:

uniform **vec4** lightPosition;

Layout Qualifiers [4.3.8]

layout(*layout-qualifiers*) *block-declaration*
layout(*layout-qualifiers*) **in/out/uniform**
layout(*layout-qualifiers*) **in/out/uniform** *declaration*

Input Layout Qualifiers

Layout qualifier identifiers for geometry shader inputs:

points, lines, lines_adjacency, triangles, triangles_adjacency

Fragment shaders can have an input layout only for redeclaring the built-in variable gl_FragCoord with the layout qualifier identifiers:

origin_upper_left, pixel_center_integer

Output Layout Qualifiers

Layout qualifier identifiers for geometry shader outputs:

points, line_strip, triangle_strip,
max_vertices = *integer-constant*

Uniform-Block Layout Qualifiers

Layout qualifier identifiers for uniform blocks:

shared, packed, std140, row_major, column_major

Interpolation Qualifier [4.3.9]

Qualify outputs from vertex shader and inputs to fragment shader.

smooth	perspective correct interpolation
flat	no interpolation
noperspective	linear interpolation

The following predeclared variables can be redeclared with an interpolation qualifier:

Vertex language:	gl_FrontColor gl_BackColor gl_FrontSecondaryColor gl_BackSecondaryColor
Fragment language:	gl_Color gl_SecondaryColor

Parameter Qualifiers [4.4]

Input values are copied in at function call time, output values are copied out at function return time.

none	(default) same as **in**
in	for function parameters passed into a function
out	for function parameters passed back out of a function, but not initialized for use when passed in
inout	for function parameters passed both into and out of a function

Precision and Precision Qualifiers [4.5]

Precision qualifiers have no affect on precision; they aid code portability with OpenGL ES. They are:

highp, mediump, lowp

Precision qualifiers precede a floating point or integer declaration:

lowp float color;

A precision statement sets a default for subsequent declarations:

highp int;

Invariant Qualifiers Examples [4.6]

#pragma STDGL invariant(all)	force all output variables to be invariant
invariant gl_Position;	qualify a previously declared variable
invariant centroid out vec3 Color;	qualify as part of a variable declaration

Order of Qualification [4.7]

When multiple qualifications are present, they must follow a strict order. This order is as follows.

invariant, interpolation, storage, precision
storage, parameter, precision

Operators and Expressions

Operators [5.1]

Numbered in order of precedence. The relational and equality operators > < <= >= == != evaluate to a Boolean. To compare vectors component-wise, use functions such as lessThan(), equal(), etc.

1.	**( )**	parenthetical grouping
2.	**[ ]** **( )** **.** **++ --**	array subscript function call & constructor structure field or method selector, swizzler postfix increment and decrement
3.	**++ --** **+ - ~ !**	prefix increment and decrement unary
4.	*** / %**	multiplicative
5.	**+ -**	additive
6.	**<< >>**	bit-wise shift
7.	**< > <= >=**	relational
8.	**== !=**	equality
9.	**&**	bit-wise and
10.	**^**	bit-wise exclusive or
11.	**|**	bit-wise inclusive or
12.	**&&**	logical and
13.	**^^**	logical exclusive or
14.	**| |**	logical inclusive or
15.	**? :**	selection (Selects one entire operand. Use **mix**() to select individual components of vectors.)
16.	**=+** **= -=** ***= /=** **%= <<= >>=** **&= ^= |=**	assignment arithmetic assignments
17.	**,**	sequence

Vector Components [5.5]

In addition to array numeric subscript syntax (e,g,: v[0], v[i])), names of vector components are denoted by a single letter. Components can be swizzled and replicated, e.g.: pos.xx, pos.zy

{x, y, z, w}	Use when accessing vectors that represent points or normals
{r, g, b, a}	Use when accessing vectors that represent colors
{s, t, p, q}	Use when accessing vectors that represent texture coordinates

OpenGL Shading Language 1.50 Quick Reference Card

Built-In Inputs, Outputs, and Constants [7]

Vertex Language

in int gl_VertexID;
in int gl_InstanceID;
 	
in vec4 gl_Color;
in vec4 gl_SecondaryColor;
in vec3 gl_Normal;
in vec4 gl_Vertex;
in vec4 gl_MultiTexCoord{0-7};
in float gl_FogCoord;
	out gl_PerVertex {
	vec4 gl_Position;
	float gl_PointSize;
	float gl_ClipDistance[];
	vec4 gl_ClipVertex;
	};
	
	out vec4 gl_FrontColor;
	out vec4 gl_BackColor;
	out vec4 gl_FrontSecondaryColor;
	out vec4 gl_BackSecondaryColor;
	out vec4 gl_TexCoord[];
	out float gl_FogFragCoord;

Geometry Language

in gl_PerVertex {
vec4 gl_Position;
float gl_PointSize;
float gl_ClipDistance[];
} gl_in[];
 	
in int gl_PrimitiveIDIn;
	out gl_PerVertex {
	vec4 gl_Position;
	float gl_PointSize;
	float gl_ClipDistance[];
	};
	
	out int gl_PrimitiveID;
	out int gl_Layer;

Compatibility profile outputs from the Vertex Language are also available as deprecated inputs and outputs in the Geometry Language.

Fragment Language

in vec4 gl_FragCoord;
in bool gl_FrontFacing;
in float gl_ClipDistance[];
in vec2 gl_PointCoord;
in int gl_PrimitiveID;
	out float gl_FragDepth;

Built-In Constants With Minimum Values [7.4]

```
const int  gl_MaxClipDistances = 8;
const int  gl_MaxClipPlanes  = 8;
const int  gl_MaxDrawBuffers = 8;
```

Aggregate Operations and Constructors

Matrix Constructor Examples [5.4]

```
mat2(vec2, vec2);                  // one column per argument
mat3x2(vec2, vec2, vec2);          // column 1
mat2(float, float, float, float);  // column 2
mat2x3(vec2, float, vec2, float);  // column 2
mat4x4(mat3x3);                    // mat3x3 to upper left, set lower
                                   // right to 1, fill rest with zero
```

Array Constructor Example [5.4]

```
float c[3] = float[3](5.0, b + 1.0, 1.1);
```

Structure Constructor Example [5.4]

```
struct light {members; };
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

Matrix Components [5.6]

Access components of a matrix with array subscripting syntax. For example:

```
mat4 m;            // m represents a matrix
m[1] = vec4(2.0);  // sets second column to all 2.0
m[0][0] = 1.0;     // sets upper left element to 1.0
m[2][3] = 2.0;     // sets 4th element of 3rd column to 2.0
```

Examples of operations on matrices and vectors:

```
m = f * m;       // scalar * matrix component-wise
v = f * v;       // scalar * vector component-wise
v = v * v;       // vector * vector component-wise
m = m op m;      // matrix op matrix component-wise
m = m * m;       // linear algebraic multiply
m = v * m;       // row vector * matrix linear algebraic multiply
m = m * v;       // matrix * column vector linear algebraic multiply
f = dot(v, v);   // vector dot product
v = cross(v, v); // vector cross product
m = matrixCompMult(m, m);   // component-wise multiply
m = outerProduct(v, v);  // matrix product of column * row vector
```

Structure and Array Operations [5.7]

Select structure fields and the length() method of an array using the period (.) operator. Other operators include:

.	field or method selector
== !=	equality
=	assignment
[ ]	indexing (arrays only)

Array elements are accessed using the array subscript operator ([ ]). For example:

```
diffuseColor += lightIntensity[3] * NdotL;
```

Built-In Constants With Minimum Values (cont'd)

```
const  int  gl_MaxTextureUnits  = 2;
const  int  gl_MaxTextureCoords  = 8;
const  int  gl_MaxGeometryTextureImageUnits  = 16;
const  int  gl_MaxTextureImageUnits   = 16;
const  int  gl_MaxVertexAttribs  = 16;
const  int  gl_MaxVertexTextureImageUnits  = 16;
const  int  gl_MaxCombinedTextureImageUnits  = 48;
const  int  gl_MaxGeometryVaryingComponents  = 64;
const  int  gl_MaxVaryingComponents  = 64;
const  int  gl_MaxVaryingFloats  = 64;
const  int  gl_MaxGeometryOutputVertices = 256;
const  int  gl_MaxFragmentUniformComponents = 1024;
const  int  gl_MaxGeometryTotalOutputComponents = 1024;
const  int  gl_MaxGeometryUniformComponents  = 1024;
const  int  gl_MaxVertexUniformComponents = 1024;
```

Statements and Structure

Iteration and Jumps [6]

Function Call	call by value, return

Iteration	for (;;) { break, continue }
	while ( ) { break, continue }
	do { break, continue } while ( );

Selection	if ( ) { }
	if ( ) { } else { }
	switch ( ) { case integer: ... break; ... default: ... }

Jump	break, continue, return (There is no 'goto')
Entry	void main()
Exit	return in main() discard // Fragment shader only

Built-In Functions

Angle & Trigonometry Functions [8.1]

Component-wise operation. Parameters specified as *angle* are assumed to be in units of radians. T is float, vec2, vec3, vec4.

T **radians**(T *degrees*)	degrees to radians
T **degrees**(T *radians*)	radians to degrees
T **sin**(T *angle*)	sine
T **cos**(T *angle*)	cosine
T **tan**(T *angle*)	tangent
T **asin**(T *x*)	arc sine
T **acos**(T *x*)	arc cosine
T **atan**(T *y*, T *x*) T **atan**(T *y_over_x*)	arc tangent
T **sinh**(T *x*)	hyperbolic sine
T **cosh**(T *x*)	hyperbolic cosine
T **tanh**(T *x*)	hyperbolic tangent
T **asinh**(T *x*)	hyperbolic sine
T **acosh**(T *x*)	hyperbolic cosine
T **atanh**(T *x*)	hyperbolic tangent

Exponential Functions [8.2]

Component-wise operation. T is float, vec2, vec3, vec4.

T **pow**(T *x*, T *y*)	x^y
T **exp**(T *x*)	e^x
T **log**(T *x*)	ln
T **exp2**(T *x*)	2^x
T **log2**(T *x*)	$\log_2$
T **sqrt**(T *x*)	square root
T **inversesqrt**(T *x*)	inverse square root

Common Functions [8.3]

Component-wise operation. T is float, vec2, vec3, vec4. Ti is int, ivec2, ivec3, ivec4. Tu is uint, uvec2, uvec3, uvec4. bvec is bvec2, bvec3, bvec4, bool.

T **abs**(T *x*) Ti **abs**(Ti *x*)	absolute value
T **sign**(T *x*) Ti **sign**(Ti *x*)	returns -1.0, 0.0, or 1.0
T **floor**(T *x*)	nearest integer <= *x*
T **trunc**(T *x*)	nearest integer with absolute value <= absolute value of *x*

(continued >)

Common Functions (Continued)

T **round**(T *x*)	nearest integer, implementation-dependent rounding mode
T **roundEven**(T *x*)	nearest integer, 0.5 rounds to nearest even integer
T **ceil**(T *x*)	nearest integer >= *x*
T **fract**(T *x*)	*x* - floor(*x*)
T **mod**(T *x*, float *y*) T **mod**(T *x*, T *y*)	modulus
T **modf**(T *x*, out T *i*)	separate integer and fractional parts
T **min**(T *x*, T *y*) T **min**(T *x*, float *y*) Ti **min**(Ti *x*, Ti *y*) Ti **min**(Ti *x*, int *y*) Tu **min**(Tu *x*, Tu *y*) Tu **min**(Tu *x*, uint *y*)	minimum value
T **max**(T *x*, T *y*) T **max**(T *x*, float *y*) Ti **max**(Ti *x*, Ti *y*) Ti **max**(Ti *x*, int *y*) Tu **max**(Tu *x*, Tu *y*) Tu **max**(Tu *x*, uint *y*)	maximum value
T **clamp**(T *x*, T *minVal*, T *maxVal*) T **clamp**(T *x*, float *minVal*, float *maxVal*) Ti **clamp**(Ti *x*, Ti *minVal*, Ti *maxVal*) Ti **clamp**(Ti *x*, int *minVal*, int *maxVal*) Tu **clamp**(Tu *x*, Tu *minVal*, Tu *maxVal*) Tu **clamp**(Tu *x*, uint *minVal*, uint *maxVal*)	min(max(*x*, *minVal*), *maxVal*)
T **mix**(T *x*, T *y*, T *a*) T **mix**(T *x*, T *y*, float *a*)	linear blend of *x* and *y*
T **mix**(T *x*, T *y*, bvec *a*)	true components in *a* select components from *y*, else from x
T **step**(T *edge*, T *x*) T **step**(float *edge*, T *x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep**(T *edge0*, T *edge1*, T *x*) T **smoothstep**(float *edge0*, float *edge1*, T *x*)	clip and smooth
bvec **isnan**(T *x*)	true if *x* is NaN
bvec **isinf**(T *x*)	true if *x* is positive or negative infinity

Geometric Functions [8.4]

These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

float **length**(T *x*)	length of vector
float **distance**(T *p0*, T *p1*)	distance between points
float **dot**(T *x*, T *y*)	dot product
vec3 **cross**(vec3 *x*, vec3 *y*)	cross product
T **normalize**(T *x*)	normalize vector to length 1
vec4 **ftransform**( )	invariant vertex transformation
T **faceforward**(T *N*, T *I*, T *Nref*)	returns *N* if dot(*Nref*, *I*) < 0, else -*N*
T **reflect**(T *I*, T *N*)	reflection direction I - 2 * dot(*N*,*I*) * *N*
T **refract**(T *I*, T *N*, float *eta*)	refraction vector

Matrix Functions [8.5]

Type mat is any matrix type.

mat **matrixCompMult**(mat *x*, mat *y*)	multiply *x* by *y* component-wise
mat*N* **outerProduct**(vec*N c*, vec*N r*)	where *N* is 2, 3, 4 : *c* * *r* outer product
mat*N*x*M* **outerProduct**(vec*M c*, vec*N r*)	where *N* != *M* and *N*, *M* = 2, 3, 4 : *c* * *r* outer product
mat*N* **transpose**(mat*N m*)	where *N* is 2, 3, 4 : transpose of *m*
mat*N*x*M* **transpose**(mat*M*x*N m*)	where *N* != *M* and *N*,*M* = 2, 3, 4 : transpose of *m*
float **determinant**(mat*N m*)	determinant of *m*
mat*N* **inverse**(mat*N m*)	where *N* is 2, 3, 4 : inverse of *m*

Vector Relational Functions [8.6]

Compare *x* and *y* component-wise. Sizes of the input and return vectors for any particular call must match. Type bvec is bvec*n*; vec is vec*n*; {ui}vec is {ui}vec*n* (where *n* is 2, 3, or 4). T is the union of vec and {ui}vec.

bvec **lessThan**(T *x*, T *y*)	<
bvec **lessThanEqual**(T *x*, T *y*)	<=
bvec **greaterThan**(T *x*, T *y*)	>
bvec **greaterThanEqual**(T *x*, T *y*)	>=
bvec **equal**(T *x*, T *y*) bvec **equal**(bvec *x*, bvec *y*)	==
bvec **notEqual**(T *x*, T *y*) bvec **notEqual**(bvec *x*, bvec *y*)	!=
bool **any**(bvec *x*)	true if any component of *x* is true
bool **all**(bvec *x*)	true if all components of *x* are true
bvec **not**(bvec *x*)	logical complement of *x*

The OpenGL Shading Language 1.50 Quick Reference Card

Derivative Functions [8.8]

Available only in fragment shaders. T is float, vec2, vec3, vec4.

T **dFdx**(T *p*)	derivative in *x*
T **dFdy**(T *p*)	derivative in *y*
T **fwidth**(T *p*)	sum of absolute derivative in *x* and *y*

Noise Functions [8.9]

Returns noise value. Available to fragment, geometry, and vertex shaders. T is float, vec2, vec3, vec4.

float **noise1**(T *x*)	
vec*n* **noise*n***(T *x*)	where *n* is 2, 3, or 4

Geometry Shader Functions [8.10]

Only available in geometry shaders.

void **EmitVertex**()	emits current values of output variables to the current output primitive
void **EndPrimitive**()	completes current output primitive and starts a new one

Texture Lookup Functions [8.7]

Available to vertex, geometry, and fragment shaders. gvec4 means vec4, ivec4, or uvec4. gsampler* means sampler*, isampler*, or usampler*.

Texture lookup, returning LOD if present:

int **textureSize**(gsampler1D *sampler*, int *lod*)

ivec2 **textureSize**(gsampler2D *sampler*, int *lod*)

ivec3 **textureSize**(gsampler3D *sampler*, int *lod*)

ivec2 **textureSize**(gsamplerCube *sampler*, int *lod*)

int **textureSize**(sampler1DShadow *sampler*, int *lod*)

ivec2 **textureSize**(sampler2DShadow *sampler*, int *lod*)

ivec2 **textureSize**(samplerCubeShadow *sampler*, int *lod*)

ivec2 **textureSize**(gsampler2DRect *sampler*)

ivec2 **textureSize**(sampler2DRectShadow *sampler*)

ivec2 **textureSize**(gsampler1DArray *sampler*, int *lod*)

ivec3 **textureSize**(gsampler2DArray *sampler*, int *lod*)

ivec2 **textureSize**(sampler1DArrayShadow *sampler*, int *lod*)

ivec3 **textureSize**(sampler2DArrayShadow *sampler*, int *lod*)

int **textureSize**(gsamplerBuffer *sampler*)

ivec2 **textureSize**(gsampler2DMS *sampler*)

ivec2 **textureSize**(gsampler2DMSArray *sampler*)

Texture lookup:

gvec4 **texture**(gsampler1D *sampler*, float *P* [, float *bias*])

gvec4 **texture**(gsampler2D *sampler*, vec2 *P* [, float *bias*])

gvec4 **texture**(gsampler3D *sampler*, vec3 *P* [, float *bias*])

gvec4 **texture**(gsamplerCube *sampler*, vec3 *P* [, float *bias*])

float **texture**(sampler{1,2}DShadow *sampler*, vec3 *P* [, float *bias*])

float **texture**(samplerCubeShadow *sampler*, vec4 *P* [, float *bias*])

gvec4 **texture**(gsampler1DArray *sampler*, vec2 *P* [, float *bias*])

gvec4 **texture**(gsampler2DArray *sampler*, vec3 *P* [, float *bias*])

float **texture**(sampler1DArrayShadow *sampler*, vec3 *P* [, float *bias*])

float **texture**(sampler2DArrayShadow *sampler*, vec4 *P*)

gvec4 **texture**(gsampler2DRect *sampler*, vec2 *P*)

float **texture**(sampler2DRectShadow *sampler*, vec3 *P*)

Texture lookup with projection:

gvec4 **textureProj**(gsampler1D *sampler*, vec{2,4} *P* [, float *bias*])

gvec4 **textureProj**(gsampler2D *sampler*, vec{3,4} *P* [, float *bias*])

gvec4 **textureProj**(gsampler3D *sampler*, vec4 *P* [, float *bias*])

float **textureProj**(sampler{1,2}DShadow *sampler*, vec4 *P* [, float *bias*])

gvec4 **textureProj**(gsampler2DRect *sampler*, vec{3,4} *P*)

float **textureProj**(sampler2DRectShadow *sampler*, vec4 *P*)

Texture lookup with explicit LOD:

gvec4 **textureLod**(gsampler1D *sampler*, float *P*, float *lod*)

gvec4 **textureLod**(gsampler2D *sampler*, vec2 *P*, float *lod*)

gvec4 **textureLod**(gsampler3D *sampler*, vec3 *P*, float *lod*)

gvec4 **textureLod**(gsamplerCube *sampler*, vec3 *P*, float *lod*)

float **textureLod**(sampler{1,2}DShadow *sampler*, vec3 *P*, float *lod*)

gvec4 **textureLod**(gsampler1DArray *sampler*, vec2 *P*, float *lod*)

gvec4 **textureLod**(gsampler2DArray *sampler*, vec3 *P*, float *lod*)

float **textureLod**(sampler1DArrayShadow *sampler*, vec3 *P*, float *lod*)

Texture lookup with offset:

gvec4 **textureOffset**(gsampler1D *sampler*, float *P*, int *offset* [, float *bias*])

gvec4 **textureOffset**(gsampler2D *sampler*, vec2 *P*, ivec2 *offset* [, float *bias*])

gvec4 **textureOffset**(gsampler3D *sampler*, vec3 *P*, ivec3 *offset* [, float *bias*])

gvec4 **textureOffset**(gsampler2DRect *sampler*, vec2 *P*, ivec2 *offset*)

float **textureOffset**(sampler2DRectShadow *sampler*, vec3 *P*, ivec2 *offset*)

float **textureOffset**(sampler1DShadow *sampler*, vec3 *P*, int *offset* [, float *bias*])

float **textureOffset**(sampler2DShadow *sampler*, vec3 *P*, ivec2 *offset* [, float *bias*])

gvec4 **textureOffset**(gsampler1DArray *sampler*, vec2 *P*, int *offset* [, float *bias*])

gvec4 **textureOffset**(gsampler2DArray *sampler*, vec3 *P*, ivec2 *offset* [, float *bias*])

float **textureOffset**(sampler1DArrayShadow *sampler*, vec3 *P*, int *offset* [, float *bias*])

Fetch a single texel:

gvec4 **texelFetch**(gsampler1D *sampler*, int *P*, int *lod*)

gvec4 **texelFetch**(gsampler2D *sampler*, ivec2 *P*, int *lod*)

gvec4 **texelFetch**(gsampler3D *sampler*, ivec3 *P*, int *lod*)

gvec4 **texelFetch**(gsampler2DRect *sampler*, ivec2 *P*)

gvec4 **texelFetch**(gsampler1DArray *sampler*, ivec2 *P*, int *lod*)

gvec4 **texelFetch**(gsampler2DArray *sampler*, ivec3 *P*, int *lod*)

gvec4 **texelFetch**(gsamplerBuffer *sampler*, int *P*)

gvec4 **texelFetch**(gsampler2DMS *sampler*, ivec2 *P*, int *sample*)

gvec4 **texelFetch**(gsampler2DMSArray *sampler*, ivec3 *P*, int *sample*)

Fetch a single texel, with offset:

gvec4 **texelFetchOffset**(gsampler1D *sampler*, int *P*, int *lod*, int *offset*)

gvec4 **texelFetchOffset**(gsampler2D *sampler*, ivec2 *P*, int *lod*, ivec2 *offset*)

gvec4 **texelFetchOffset**(gsampler3D *sampler*, ivec3 *P*, int *lod*, ivec3 *offset*)

gvec4 **texelFetchOffset**(gsampler2DRect *sampler*, ivec2 *P*, ivec2 *offset*)

gvec4 **texelFetchOffset**(gsampler1DArray *sampler*, ivec2 *P*, int *lod*, int *offset*)

gvec4 **texelFetchOffset**(gsampler2DArray *sampler*, ivec3 *P*, int *lod*, ivec2 *offset*)

Projective texture lookup with offset:

gvec4 **textureProjOffset**(gsampler1D *sampler*, vec{2,4} *P*, int *offset* [, float *bias*])

gvec4 **textureProjOffset**(gsampler2D *sampler*, vec{3,4} *P*, ivec2 *offset* [, float *bias*])

gvec4 **textureProjOffset**(gsampler3D *sampler*, vec4 *P*, ivec3 *offset* [, float *bias*])

gvec4 **textureProjOffset**(gsampler2DRect *sampler*, vec{3,4} *P*, ivec2 *offset*)

float **textureProjOffset**(sampler2DRectShadow *sampler*, vec4 *P*, ivec2 *offset*)

float **textureProjOffset**(sampler1DShadow *sampler*, vec4 *P*, int offset [, float *bias*])

float **textureProjOffset**(sampler2DShadow *sampler*, vec4 *P*, ivec2 *offset* [, float *bias*])

Offset texture lookup with explicit LOD:

gvec4 **textureLodOffset**(gsampler1D *sampler*, float *P*, float *lod*, int *offset*)

gvec4 **textureLodOffset**(gsampler2D *sampler*, vec2 *P*, float *lod*, ivec2 *offset*)

gvec4 **textureLodOffset**(gsampler3D *sampler*, vec3 *P*, float *lod*, ivec3 *offset*)

float **textureLodOffset**(sampler1DShadow *sampler*, vec3 *P*, float *lod*, int *offset*)

float **textureLodOffset**(sampler2DShadow *sampler*, vec3 *P*, float *lod*, ivec2 *offset*)

gvec4 **textureLodOffset**(gsampler1DArray *sampler*, vec2 *P*, float *lod*, int *offset*)

gvec4 **textureLodOffset**(gsampler2DArray *sampler*, vec3 *P*, float *lod*, ivec2 *offset*)

float **textureLodOffset**(sampler1DArrayShadow *sampler*, vec3 *P*, float *lod*, int *offset*)

Projective texture lookup with explicit LOD:

gvec4 **textureProjLod**(gsampler1D *sampler*, vec{2,4} *P*, float *lod*)

gvec4 **textureProjLod**(gsampler2D *sampler*, vec{3,4} *P*, float *lod*)

gvec4 **textureProjLod**(gsampler3D *sampler*, vec4 *P*, float *lod*)

float **textureProjLod**(sampler{1,2}DShadow *sampler*, vec4 *P*, float *lod*)

Offset projective texture lookup with explicit LOD:

gvec4 **textureProjLodOffset**(gsampler1D *sampler*, vec{2,4} *P*, float *lod*, int *offset*)

gvec4 **textureProjLodOffset**(gsampler2D *sampler*, vec{3,4} *P*, float *lod*, ivec2 *offset*)

gvec4 **textureProjLodOffset**(gsampler3D *sampler*, vec4 *P*, float *lod*, ivec3 *offset*)

float **textureProjLodOffset**(sampler1DShadow *sampler*, vec4 *P*, float *lod*, int *offset*)

float **textureProjLodOffset**(sampler2DShadow *sampler*, vec4 *P*, float *lod*, ivec2 *offset*)

Texture lookup with explicit gradient:

gvec4 **textureGrad**(gsampler1D *sampler*, float *P*, float *dPdx*, float *dPdy*)

gvec4 **textureGrad**(gsampler2D *sampler*, vec2 *P*, vec2 *dPdx*, vec2 *dPdy*)

gvec4 **textureGrad**(gsampler3D *sampler*, vec3 *P*, vec3 *dPdx*, vec3 *dPdy*)

gvec4 **textureGrad**(gsamplerCube *sampler*, vec3 *P*, vec3 *dPdx*, vec3 *dPdy*)

gvec4 **textureGrad**(gsampler2DRect *sampler*, vec2 *P*, vec2 *dPdx*, vec2 *dPdy*)

float **textureGrad**(sampler2DRectShadow *sampler*, vec3 *P*, vec2 *dPdx*, vec2 *dPdy*)

float **textureGrad**(sampler1DShadow *sampler*, vec3 *P*, float *dPdx*, float *dPdy*)

float **textureGrad**(sampler2DShadow *sampler*, vec3 *P*, vec2 *dPdx*, vec2 *dPdy*)

float **textureGrad**(samplerCubeShadow *sampler*, vec4 *P*, vec3 *dPdx*, vec3 *dPdy*)

gvec4 **textureGrad**(gsampler1DArray *sampler*, vec2 *P*, float *dPdx*, float *dPdy*)

gvec4 **textureGrad**(gsampler2DArray *sampler*, vec3 *P*, vec2 *dPdx*, vec2 *dPdy*)

float **textureGrad**(sampler1DArrayShadow *sampler*, vec3 *P*, float *dPdx*, float *dPdy*)

float **textureGrad**(sampler2DArrayShadow *sampler*, vec4 *P*, vec2 *dPdx*, vec2 *dPdy*)

Texture lookup with explicit gradient and offset:

gvec4 **textureGradOffset**(gsampler1D *sampler*, float *P*, float *dPdx*, float *dPdy*, int *offset*)

gvec4 **textureGradOffset**(gsampler2D *sampler*, vec2 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

gvec4 **textureGradOffset**(gsampler3D *sampler*, vec3 *P*, vec3 *dPdx*, vec3 *dPdy*, ivec3 *offset*)

gvec4 **textureGradOffset**(gsampler2DRect *sampler*, vec2 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

float **textureGradOffset**(sampler2DRectShadow *sampler*, vec3 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

float **textureGradOffset**(sampler1DShadow *sampler*, vec3 *P*, float *dPdx*, float *dPdy*, int *offset*)

float **textureGradOffset**(sampler2DShadow *sampler*, vec3 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

float **textureGradOffset**(samplerCubeShadow *sampler*, vec4 *P*, vec3 *dPdx*, vec3 *dPdy*, ivec2 *offset*)

gvec4 **textureGradOffset**(gsampler1DArray *sampler*, vec2 *P*, float *dPdx*, float *dPdy*, int *offset*)

gvec4 **textureGradOffset**(gsampler2DArray *sampler*, vec3 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

float **textureGradOffset**(sampler1DArrayShadow *sampler*, vec3 *P*, float *dPdx*, float *dPdy*, int *offset*)

float **textureGradOffset**(sampler2DArrayShadow *sampler*, vec4 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

Projective texture lookup with explicit gradient:

gvec4 **textureProjGrad**(gsampler1D *sampler*, vec{2,4} *P*, float *dPdx*, float *dPdy*)

gvec4 **textureProjGrad**(gsampler2D *sampler*, vec{3,4} *P*, vec2 *dPdx*, vec2 *dPdy*)

gvec4 **textureProjGrad**(gsampler3D *sampler*, vec4 *P*, vec3 *dPdx*, vec3 *dPdy*)

gvec4 **textureProjGrad**(gsampler2DRect *sampler*, vec{3,4} *P*, vec2 *dPdx*, vec2 *dPdy*)

float **textureProjGrad**(sampler2DRectShadow *sampler*, vec4 *P*, vec2 *dPdx*, vec2 *dPdy*)

float **textureProjGrad**(sampler1DShadow *sampler*, vec4 *P*, float *dPdx*, float *dPdy*)

float **textureProjGrad**(sampler2DShadow *sampler*, vec4 *P*, vec2 *dPdx*, vec2 *dPdy*)

Projective texture lookup with explicit gradient and offset:

gvec4 **textureProjGradOffset**(gsampler1D *sampler*, vec{2,4} *P*, float *dPdx*, float *dPdy*, int *offset*)

gvec4 **textureProjGradOffset**(gsampler2D *sampler*, vec{3,4} *P*, vec2 *dPdx*, vec2 *dPdy*, vec2 *offset*)

gvec4 **textureProjGradOffset**(gsampler2DRect *sampler*, vec{3,4} *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

float **textureProjGradOffset**(sampler2DRectShadow *sampler*, vec4 *P*, vec2 *dPdx*, vec2 *dPdy*, ivec2 *offset*)

gvec4 **textureProjGradOffset**(gsampler3D *sampler*, vec4 *P*, vec3 *dPdx*, vec3 *dPdy*, vec3 *offset*)

float **textureProjGradOffset**(sampler1DShadow *sampler*, vec4 *P*, float *dPdx*, float *dPdy*, int *offset*)

float **textureProjGradOffset**(sampler2DShadow *sampler*, vec4 *P*, vec2 *dPdx*, vec2 *dPdy*, vec2 *offset*)

OpenGL is a registered trademark of Silicon Graphics International, used under license by Khronos Group.

The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices.

See www.khronos.org to learn more about the Khronos Group.
See www.opengl.org to learn more about OpenGL.

Reference card production by Miller & Mattson www.millermattson.com