# OpenGL 3.2 API Quick Reference Card

**OpenGL®** is the only cross-platform graphics API that enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually-compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality.

- ***see FunctionName*** refers to functions on this reference card.
- **[n.n.n]** and **[Table n.n]** refer to sections and tables in the OpenGL 3.2 core specification.
- **[n.n.n]** refers to sections in the OpenGL Shading Language 1.50 specification.
- Content shown in blue is removed from the OpenGL 3.2 core profile and present only in the OpenGL 3.2 compatibility profile. Profile selection is made at context creation.
- **[n.n.n]** and **[Table n.n]** refer to sections and tables in the OpenGL 3.2 compatibility profile specification, and are shown only when they differ from the core profile.

Specifications are available at www.opengl.org/registry

## OpenGL Operation

### Floating-Point Numbers [2.1.2]

| | |
|---|---|
| **16-Bit** | 1-bit sign<br>5-bit exponent<br>10-bit mantissa |
| **Unsigned 11-Bit** | no sign bit<br>5-bit exponent<br>6-bit mantissa |
| **Unsigned 10-Bit** | no sign bit<br>5-bit exponent<br>5-bit mantissa |

### Command Letters [Table 2.1]

Letters are used in commands to denote types as shown below.

| | |
|---|---|
| **b** - byte (8 bits) | **ub** - ubyte (8 bits) |
| **s** - short (16 bits) | **us** - ushort (16 bits) |
| **i** - int (32 bits) | **ui** - uint (32 bits) |
| **f** - float (32 bits) | **d** - double (64 bits) |

## Vertex Specification

### Begin and End [2.6.1, 2.6.3]

Enclose coordinate sets between Begin/End pairs to construct geometric objects.

void **Begin**(enum *mode*);
void **End**(void);
*mode:* POINTS, LINE_STRIP, LINE_LOOP, LINES, POLYGON, QUAD_STRIP, QUADS, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES, LINES_ADJACENCY, LINE_STRIP_ADJACENCY, TRIANGLES_ADJACENCY, TRIANGLE_STRIP_ADJACENCY

### Polygon Edges [2.6.2]

Flag each edge of polygon primitives as either boundary or non-boundary.

void **EdgeFlag**(boolean *flag*);
void **EdgeFlagv**(boolean *flag);

### Vertex Specification [2.7]

Vertices have two, three, or four coordinates, and optionally a current normal, multiple current texture coordinate sets, multiple current generic vertex attributes, current color, current secondary color, and current fog coordinates.

void **Vertex{234}{sifd}**(T *coords*);
void **Vertex{234}{sifd}v**(T *coords*);
void **TexCoord{1234}{sifd}**(T *coords*);
void **TexCoord{1234}{sifd}v**(T *coords*);
void **MultiTexCoord{1234}{sifd}**(enum *texture*, T *coords*)
void **MultiTexCoord{1234}{sifd}v**(enum *texture*, T *coords*)
*texture:* TEXTURE*i* (where *i* is [0, MAX_TEXTURE_COORDS - 1])
void **Normal3{bsifd}**(T *coords*);
void **Normal3{bsifd}v**(T *coords*);
void **FogCoord{fd}**(T *coord*);
void **FogCoord{fd}v**(T *coord*);
void **Color{34}{bsifd ubusui}**(T *components*);
void **Color{34}{bsifd ubusui}v**(T *components*);
void **SecondaryColor3{bsifd ubusui}**(T *components*);
void **SecondaryColor3{bsifd ubusui}v**(T *components*);
void **Index{sifd ub}**(T *index*);
void **Index{sifd ub}v**(T *index*);
void **VertexAttrib{1234}{sfd}**(uint *index*, T *values*);
void **VertexAttrib{123}{sfd}v**(uint *index*, T *values*);
void **VertexAttrib4{bsifd ub us ui}v**(uint *index*, T *values*);
void **VertexAttrib4Nub**(uint *index*, T *values*);
void **VertexAttrib4N{bsi ub us ui}v**(uint *index*, T *values*);
void **VertexAttribI{1234}{i ui}**(uint *index*, T *values*);
void **VertexAttribI{1234}{i ui}v**(uint *index*, T *values*);
void **VertexAttribI4{bs ubus}v**(uint *index*, T *values*);

## GL Command Syntax [2.3]

GL commands are formed from a return type, a name, and optionally up to 4 characters (or character pairs) from the Command Letters table (above), as shown by the prototype below:

*return-type* **Name**{1234}{b s i f d ub us ui}{v} (*[args ,] T arg1 , . . . , T argN [, args]*);

The arguments enclosed in brackets ([args ,] and [, args]) may or may not be present.
The argument type T and the number N of arguments may be indicated by the command name suffixes. N is 1, 2, 3, or 4 if present, or else corresponds to the type letters from the Command Table (above). If "v" is present, an array of N items are passed by a pointer.

For brevity, the OpenGL documentation and this reference may omit the standard prefixes. The actual names are of the forms: glFunctionName(), GL_CONSTANT, GLtype

## Vertex Arrays [2.8]

Vertex data may be placed into arrays that are stored in the client address space or server address space.

void **VertexPointer**(int *size*, enum *type*, sizei *stride*, void **pointer*);
*type:* SHORT, INT, FLOAT, HALF_FLOAT, DOUBLE
void **NormalPointer**(enum *type*, sizei *stride*, void **pointer*);
*type:* BYTE, SHORT, INT, FLOAT, HALF_FLOAT, DOUBLE
void **ColorPointer**(int *size*, enum *type*, sizei *stride*, void **pointer*);
*type:* BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, INT, UNSIGNED_INT, FLOAT, HALF_FLOAT, DOUBLE
void **SecondaryColorPointer**(int *size*, enum *type*, sizei *stride*, void **pointer*);
*type:* BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, INT, UNSIGNED_INT, FLOAT, HALF_FLOAT, DOUBLE
void **IndexPointer**(enum *type*, sizei *stride*, void **pointer*);
*type:* UNSIGNED_BYTE, SHORT, INT, FLOAT, DOUBLE
void **EdgeFlagPointer**(sizei *stride*, void **pointer*);
void **FogCoordPointer**(enum *type*, sizei *stride*, void **pointer*);
*type:* FLOAT, HALF_FLOAT, DOUBLE
void **TexCoordPointer**(int *size*, enum *type*, sizei *stride*, void **pointer*);
*type:* SHORT, INT, FLOAT, HALF_FLOAT, DOUBLE
void **VertexAttribPointer**(uint *index*, int *size*, enum *type*, boolean *normalized*, sizei *stride*, const void **pointer*);
*type:* BYTE, UNSIGNED_BYTE, SHORT, USHORT, INT, UINT, FLOAT, HALF_FLOAT, DOUBLE
void **VertexAttribIPointer**(uint *index*, int *size*, enum *type*, sizei *stride*, const void **pointer*);
*type:* BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, INT, UNSIGNED_INT
*index:* [0, MAX_VERTEX_ATTRIBS - 1]
void **EnableClientState**(enum *array*);
void **DisableClientState**(enum *array*);
*array:* VERTEX_ARRAY, NORMAL_ARRAY, COLOR_ARRAY, SECONDARY_COLOR_ARRAY, INDEX_ARRAY, EDGE_FLAG_ARRAY, FOG_COORD_ARRAY, TEXTURE_COORD_ARRAY
void **EnableVertexAttribArray**(uint *index*);
void **DisableVertexAttribArray**(uint *index*);
*index:* TEXTURE*i* (where *i* is [0, MAX_VERTEX_ATTRIBS - 1])
void **ClientActiveTexture**(enum *texture*);
void **ArrayElement**(int *i*);
**Enable/Disable**(PRIMITIVE_RESTART)
void **PrimitiveRestartIndex**(uint *index*);

### Drawing Commands [2.8.2] [2.8.1]

void **DrawArrays**(enum *mode*, int *first*, sizei *count*);
void **MultiDrawArrays**(enum *mode*, int **first*, sizei **count*, sizei *primcount*);
void **DrawElements**(enum *mode*, sizei *count*, enum *type*, void **indices*);
void **MultiDrawElements**(enum *mode*, sizei **count*, enum *type*, void ***indices*, sizei *primcount*);
void **DrawRangeElements**(enum *mode*, uint *start*, uint *end*, sizei *count*, enum *type*, void **indices*);
void **DrawArraysInstanced**(enum *mode*, int *first*, sizei *count*, sizei *primcount*);
void **DrawElementsInstanced**(enum *mode*, sizei *count*, enum *type*, const void **indices*, sizei *primcount*);
void **DrawElementsBaseVertex**(enum *mode*, sizei *count*, enum *type*, void **indices*, int *basevertex*);
void **DrawRangeElementsBaseVertex**(enum *mode*, uint *start*, uint *end*, sizei *count*, enum *type*, void **indices*, int *basevertex*);
void **DrawElementsInstancedBaseVertex**(enum *mode*, sizei *count*, enum *type*, const void **indices*, sizei *primcount*, int *basevertex*);
void **MultiDrawElementsBaseVertex**(enum *mode*, sizei **count*, enum *type*, void ***indices*, sizei *primcount*, int **basevertex*);
*mode:* POINTS, LINE_STRIP, LINE_LOOP, LINES, POLYGON, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES, QUAD_STRIP, QUADS, LINES_ADJACENCY, LINE_STRIP_ADJACENCY, TRIANGLES_ADJACENCY, TRIANGLE_STRIP_ADJACENCY
*type:* UNSIGNED_BYTE, UNSIGNED_SHORT, UNSIGNED_INT
void **InterleavedArrays**(enum *format*, sizei *stride*, void **pointer*);
*format:* V2F, V3F, C4UB_V2F, C4UB_V3F, C3F_V3F, N3F_V3F, C4F_N3F_V3F, T2F_V3F, T4F_V4F, T2F_C4UB_V3F, T2F_C3F_V3F, T2F_N3F_V3F, T2F_C4F_N3F_V3F, T4F_C4F_N3F_V4F

## Buffer Objects [2.9]

void **GenBuffers**(sizei *n*, uint **buffers*);
void **DeleteBuffers**(sizei *n*, const uint **buffers*);

### Creating and Binding Buffer Objects [2.9.1]

void **BindBuffer**(enum *target*, uint *buffer*);
*target:* ARRAY_BUFFER, COPY_READ_BUFFER, COPY_WRITE_BUFFER, ELEMENT_ARRAY_BUFFER, PIXEL_PACK_BUFFER, PIXEL_UNPACK_BUFFER, TEXTURE_BUFFER, TRANSFORM_FEEDBACK_BUFFER, UNIFORM_BUFFER
void **BindBufferRange**(enum *target*, uint *index*, uint *buffer*, intptr *offset*, sizeiptr *size*);
*target:* TRANSFORM_FEEDBACK_BUFFER, UNIFORM_BUFFER
void **BindBufferBase**(enum *target*, uint *index*, uint *buffer*);
*target:* ***see BindBufferRange***

### Creating Buffer Object Data Stores [2.9.2]

void **BufferData**(enum *target*, sizeiptr *size*, const void **data*, enum *usage*);
*usage:* STREAM_DRAW, STREAM_READ, STREAM_COPY, STATIC_DRAW, STATIC_READ, STATIC_COPY, DYNAMIC_DRAW, DYNAMIC_READ, DYNAMIC_COPY
void **BufferSubData**(enum *target*, intptr *offset*, sizeiptr *size*, const void **data*);
*target:* ***see BindBuffer***

### Mapping and Unmapping Buffer Data [2.9.3]

void ***MapBufferRange**(enum *target*, intptr *offset*, sizeiptr *length*, bitfield *access*);
*access:* The logical OR of MAP_READ_BIT, MAP_WRITE_BIT, MAP_INVALIDATE_RANGE_BIT, MAP_FLUSH_EXPLICIT_BIT, MAP_INVALIDATE_BUFFER_BIT, MAP_UNSYNCHRONIZED_BIT
void ***MapBuffer**(enum *target*, enum *access*);
*access:* READ_ONLY, WRITE_ONLY, READ_WRITE
void **FlushMappedBufferRange**(enum *target*, intptr *offset*, sizeiptr *length*);
*target:* ***see BindBuffer***
boolean **UnmapBuffer**(enum *target*);
*target:* ***see BindBuffer***

### Copying Between Buffers [2.9.5]

void ***CopyBufferSubData**(enum *readtarget*, enum *writetarget*, intptr *readoffset*, intptr *writeoffset*, sizeiptr *size*);
*readtarget* and *writetarget:* ***see BindBuffer***

### Vertex Array Objects [2.10]

All states related to the definition of data used by the vertex processor is encapsulated in a vertex array object.

void **GenVertexArrays**(sizei *n*, uint **arrays*);
void **DeleteVertexArrays**(sizei *n*, const uint **arrays*);
void **BindVertexArray**(uint *array*);

### Buffer Object Queries [6.1.8] [6.1.14]

boolean **IsBuffer**(uint *buffer*);
void **GetBufferParameteriv**(enum *target*, enum *pname*, int **data*);
*pname:* BUFFER_SIZE, BUFFER_USAGE, BUFFER_ACCESS, BUFFER_ACCESS_FLAGS, BUFFER_MAPPED, BUFFER_MAP_POINTER, BUFFER_MAP_OFFSET, BUFFER_MAP_LENGTH
void **GetBufferSubData**(enum *target*, intptr *offset*, sizeiptr *size*, void **data*);
*target:* ***see BindBuffer***
void **GetBufferPointerv**(enum *target*, enum *pname*, void ***params*);
*target:* ***see BindBuffer***
*pname:* BUFFER_MAP_POINTER

### Vertex Array Object Queries [6.1.9] [6.1.15]

boolean **IsVertexArray**(uint *array*);

# OpenGL 3.2 API Quick Reference Card

## Rectangles, Matrices, Texture Coordinates

### Rectangles [2.11]

Specifiy rectangles as two corner vertices.

void **Rect{sifd}**(T *x1*, T *y1*, T *x2*, T *y2*);
void **Rect{sifd}v**(T *v1*[2], T *v2*[2]);

### Matrices [2.12.1]

void **MatrixMode**(enum *mode*);
*mode:* TEXTURE, MODELVIEW, COLOR, PROJECTION
void **LoadMatrix{fd}**(T *m*[16]);
void **MultMatrix{fd}**(T *m*[16]);
void **LoadTransposeMatrix{fd}**(T *m*[16]);
void **MultTransposeMatrix{fd}**(T *m*[16]);
void **LoadIdentity**(void);
void **Rotate{fd}**(T *θ*, T *x*, T *y*, T *z*);
void **Translate{fd}**(T *x*, T *y*, T *z*);
void **Scale{fd}**(T *x*, T *y*, T *z*);
void **Frustum**(double *l*, double *r*, double *b*, double *t*, double *n*, double *f*);
void **Ortho**(double *l*, double *r*, double *b*, double *t*, double *n*, double *f*);
void **PushMatrix**(void);
void **PopMatrix**(void);

### Generating Texture Coordinates [2.12.3]

void **TexGen{ifd}**(enum *coord*, enum *pname*, T *param*);
void **TexGen{ifd}v**(enum *coord*, enum *pname*, T *params);
*coord:* S, T, R, Q
*pname:* TEXTURE_GEN_MODE, OBJECT_PLANE, EYE_PLANE

## Viewport and Clipping

### Controlling the Viewport [2.16.1]

void **DepthRange**(clampd *n*, clampd *f*);
void **Viewport**(int *x*, int *y*, sizei *w*, sizei *h*);

### Clipping [2.23]

**Enable/Disable**(CLIP_DISTANCE*i*)
*i*: [0, MAX_CLIP_DISTANCES - 1]
void **ClipPlane**(enum *p*, double *eqn*[4]);
*p*: CLIP_PLANE*i* (where *i* is [0, MAX_CLIP_PLANES - 1])

## Lighting and Color

### Lighting/ Lighting Parameter Specification [2.13.1]

**Enable/Disable**(LIGHTING) (affects all lights)
**Enable/Disable**(LIGHT*i*) (affects individual lights)
void **Material{if}**(enum *face*, enum *pname*, T *param*);
void **Material{if}v**(enum *face*, enum *pname*, T *params*);
*face:* FRONT, BACK, FRONT_AND_BACK
*pname:* AMBIENT, DIFFUSE, AMBIENT_AND_DIFFUSE, SPECULAR, EMISSION, SHININESS, COLOR_INDEXES
void **Light{if}**(enum *light*, enum *pname*, T *param*);
void **Light{if}v**(enum *light*, enum *pname*, T *params*);
*light:* LIGHT*i* (where *i* >= 0)
*pname:* AMBIENT, DIFFUSE, SPECULAR, POSITION, SPOT_DIRECTION, SPOT_EXPONENT, SPOT_CUTOFF, CONSTANT_ATTENTUATION, LINEAR_ATTENUATION, QUADRATIC_ATTENUATION
void **LightModel{if}**(enum *pname*, T *param*);
void **LightModel{if}v**(enum *pname*, T *params*);
*pname:* LIGHT_MODEL_AMBIENT, LIGHT_MODEL_LOCAL_VIEWER, LIGHT_MODEL_TWO_SIDE, LIGHT_MODEL_COLOR_CONTROL

### ColorMaterial [2.13.3, 2.13.6]

**Enable/Disable**(COLOR_MATERIAL)
void **ColorMaterial**(enum *face*, enum *mode*);
*face:* FRONT, BACK, FRONT_AND_BACK
*mode:* EMISSION, AMBIENT, DIFFUSE, SPECULAR, AMBIENT_AND_DIFFUSE
void **ClampColor**(enum *target*, enum *clamp*);
*target:* CLAMP_VERTEX_COLOR
*clamp:* TRUE, FALSE, FIXED_ONLY

### Flatshading [2.18] [2.21]

void **ProvokingVertex**(enum *provokeMode*);
*provokeMode:* FIRST_VERTEX_CONVENTION, LAST_VERTEX_CONVENTION
void **ShadeModel**(enum *mode*);
*mode:* SMOOTH, FLAT

### Queries [6.13]

void **GetLight{if}v**(enum *light*, enum *value*, T *data*);
void **GetMaterial{if}v**(enum *face*, enum *value*, T *data*);
*face:* FRONT, BACK

## Rendering Control and Queries

### Conditional Rendering [2.18]

void **BeginConditionalRender**(uint *id*, enum *mode*);
void **EndConditionalRender**(void);
*mode:* QUERY_WAIT, QUERY_NO_WAIT, QUERY_BY_REGION_WAIT, QUERY_BY_REGION_NO_WAIT

### Transform Feedback [2.19]

void **BeginTransformFeedback**(enum *primitiveMode*);
void **EndTransformFeedback**(void);
*primitiveMode:* TRIANGLES, LINES, POINTS
void **BindBufferRange**(enum *target*, uint *index*, uint *buffer*, intptr *offset*, sizeiptr *size*);
void **BindBufferBase**(enum *target*, uint *index*, uint *buffer*);
*target:* TRANSFORM_FEEDBACK_BUFFER

### Current Raster Position [2.24]

void **RasterPos{234}{sifd}**(T *coords*);
void **RasterPos{234}{sifd}v**(T *coords*);
void **WindowPos{23}{sifd}**(T *coords*);
void **WindowPos{23}{sifd}v**(const T *coords*);

### Asynchronous Queries [2.17]

void **BeginQuery**(enum *target*, uint *id*);
*target*: PRIMITIVES_GENERATED, SAMPLES_PASSED, TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN
void **EndQuery**(enum *target*);
void **GenQueries**(sizei *n*, uint *ids);
void **DeleteQueries**(sizei *n*, const uint *ids);

### Asynchronous State Queries [6.1.6] [6.1.12]

boolean **IsQuery**(uint *id*);
void **GetQueryiv**(enum *target*, enum *pname*, int *params);
*target:* SAMPLES_PASSED, PRIMITIVES_GENERATED, TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN
*pname:* CURRENT_QUERY, QUERY_COUNTER_BITS
void **GetQueryObjectiv**(uint *id*, enum *pname*, int *params);
void **GetQueryObjectuiv**(uint *id*, enum *pname*, uint *params);
*pname:* QUERY_RESULT, QUERY_RESULT_AVAILABLE

## Shaders and Programs

### Shader Objects [2.11.1] [2.14.1]

uint **CreateShader**(uint *type*);
*type:* VERTEX_SHADER, FRAGMENT_SHADER, GEOMETRY_SHADER
void **ShaderSource**(uint *shader*, sizei *count*, const char **string*, const int *length*);
void **CompileShader**(uint *shader*);
void **DeleteShader**(uint *shader*);

### Program Objects [2.11.2] [2.14.2]

uint **CreateProgram**(void);
void **AttachShader**(uint *program*, uint *shader*);
void **DetachShader**(uint *program*, uint *shader*);
void **LinkProgram**(uint *program*);
void **UseProgram**(uint *program*);
void **DeleteProgram**(uint *program*);

### Vertex Attributes [2.11.3] [2.14.3]

Vertex shaders operate on an array of 4-component items numbered from slot 0 to MAX_VERTEX_ATTRIBS - 1.

void **GetActiveAttrib**(uint *program*, uint *index*, sizei *bufSize*, sizei **length*, int **size*, enum **type*, char **name*);
**type* returns: FLOAT, FLOAT_VEC*n*, FLOAT_MAT*, INT, INT_VEC*n*, UNSIGNED_INT, UNSIGNED_INT_VEC*n*
int **GetAttribLocation**(uint *program*, const char **name*);
void **BindAttribLocation**(uint *program*, uint *index*, const char **name*);

### Uniform Variables [2.11.4] [2.14.4]

int **GetUniformLocation**(uint *program*, const char **name*);
uint **GetUniformBlockIndex**(uint *program*, const char **uniformBlockName*);
void **GetActiveUniformBlockName**(uint *program*, uint *uniformBlockIndex*, sizei *bufSize*, sizei **length*, char **uniformBlockName*);
void **GetActiveUniformBlockiv**(uint *program*, uint *uniformBlockIndex*, enum *pname*, int **params*);
*pname*: UNIFORM_BLOCK_BINDING, UNIFORM_BLOCK_DATA_SIZE, UNIFORM_BLOCK_NAME_LENGTH, UNIFORM_BLOCK_ACTIVE_UNIFORMS, UNIFORM_BLOCK_ACTIVE_UNIFORM_INDICES, UNIFORM_BLOCK_REFERENCED_BY_VERTEX_SHADER, UNIFORM_BLOCK_REFERENCED_BY_FRAGMENT_SHADER, UNIFORM_BLOCK_REFERENCED_BY_GEOMETRY_SHADER
void **GetUniformIndices**(uint *program*, sizei *uniformCount*, const char ***uniformNames*, uint **uniformIndices*);
void **GetActiveUniformName**(uint *program*, uint *uniformIndex*, sizei *bufSize*, sizei **length*, char **uniformName*);
void **GetActiveUniform**(uint *program*, uint *index*, sizei *bufSize*, sizei **length*, int **size*, enum **type*, char **name*);
void **GetActiveUniformsiv**(uint *program*, sizei *uniformCount*, const uint **uniformIndices*, enum *pname*, int **params*);
*pname*: UNIFORM_TYPE, UNIFORM_SIZE, UNIFORM_NAME_LENGTH, UNIFORM_BLOCK_INDEX, UNIFORM_OFFSET, UNIFORM_ARRAY_STRIDE, UNIFORM_MATRIX_STRIDE, UNIFORM_IS_ROW_MAJOR
*type*: FLOAT, FLOAT_VEC*n*, INT, INT_VEC*n*, UNSIGNED_INT, UNSIGNED_INT_VEC*n*, BOOL, BOOL_VEC*n*, FLOAT_MAT*, SAMPLER_*, INT_SAMPLER_*, UNSIGNED_INT_SAMPLER_*

#### Loading Uniform Variables In Default Uniform Block

void **Uniform{1234}{if}**(int *location*, T *value*);
void **Uniform{1234}{if}v**(int *location*, sizei *count*, T *value*);
void **Uniform{1234}ui**(int *location*, T *value*);
void **Uniform{1234}uiv**(int *location*, sizei *count*, T *value*);
void **UniformMatrix{234}fv**(int *location*, sizei *count*, boolean *transpose*, const float **value*);
void **UniformMatrix{2x3,3x2,2x4,4x2,3x4,4x3}fv**( int *location*, sizei *count*, boolean *transpose*, const float **value*);

#### Uniform Buffer Object Bindings

void **UniformBlockBinding**(uint *program*, uint *uniformBlockIndex*, uint *uniformBlockBinding*);

### Varying Variables [2.11.6] [2.14.6]

void **TransformFeedbackVaryings**(uint *program*, sizei *count*, const char ***varyings*, enum *bufferMode*);
*bufferMode*: INTERLEAVED_ATTRIBS, SEPARATE_ATTRIBS
void **GetTransformFeedbackVarying**(uint *program*, uint *index*, sizei *bufSize*, sizei **length*, sizei **size*, enum **type*, char **name*);
**type* returns any of the scalar, vector, or matrix attribute types returned by GetActiveAttrib.

### Shader Execution (Validation) [2.11.7] [2.14.7]

void **ValidateProgram**(uint *program*);

### Geometry Shaders [2.12] [2.15]

**GetProgramiv**(uint *program*, GEOMETRY_INPUT_TYPE, int **params*)
**params* returns: POINTS, LINES, LINES_ADJACENCY, TRIANGLES, TRIANGLES_ADJACENCY
**GetProgramiv**(uint *program*, GEOMETRY_OUTPUT_TYPE, int **params*)
**params* returns: POINTS, LINE_STRIP, TRIANGLE_STRIP

### Fragment Shaders [3.9.2] [3.12.2]

void **BindFragDataLocation**(uint *program*, uint *colorNumber*, const char **name*);
int **GetFragDataLocation**(uint *program*, const char **name*);
*name*: null-terminated string

## Shader Queries

### Shader Queries [6.1.10] [6.1.16]

boolean **IsShader**(uint *shader*);
void **GetShaderiv**(uint *shader*, enum *pname*, int **params*);
*pname:* SHADER_TYPE, DELETE_STATUS, COMPILE_STATUS, INFO_LOG_LENGTH, SHADER_SOURCE_LENGTH
void **GetAttachedShaders**(uint *program*, sizei *maxCount*, sizei **count*, uint **shaders*);
void **GetShaderInfoLog**(uint *shader*, sizei *bufSize*, sizei **length*, char **infoLog*);
void **GetShaderSource**(uint *shader*, sizei *bufSize*, sizei **length*, char **source*);
void **GetVertexAttrib{dfi Ii Iui}v**(uint *index*, enum *pname*, double **params*);
*pname:* CURRENT_VERTEX_ATTRIB , VERTEX_ATTRIB_ARRAY_*x* (where *x* may be BUFFER_BINDING, ENABLED, SIZE, STRIDE, TYPE, NORMALIZED, INTEGER)
void **GetVertexAttribPointerv**(uint *index*, enum *pname*, void ***pointer*);
*pname:* VERTEX_ATTRIB_ARRAY_POINTER
void **GetUniform{if ui}v**(uint *program*, int *location*, T **params*)

### Program Queries [6.1.10] [6.1.16]

boolean **IsProgram**(uint *program*);
void **GetProgramiv**(uint *program*, enum *pname*, int **params*);
*pname:* DELETE_STATUS, LINK_STATUS, VALIDATE_STATUS, INFO_LOG_LENGTH, ATTACHED_SHADERS, GEOMETRY_INPUT_TYPE, GEOMETRY_VERTICES_OUT, GEOMETRY_OUTPUT_TYPE, ACTIVE_ATTRIBUTES, ACTIVE_ATTRIBUTE_MAX_LENGTH, ACTIVE_UNIFORMS, TRANSFORM_FEEDBACK_*, ACTIVE_UNIFORM_*
void **GetProgramInfoLog**(uint *program*, sizei *bufSize*, sizei **length*, char **infoLog*);

# OpenGL 3.2 API Quick Reference Card

## Rasterization [3]

**Enable/Disable**(*target*)
*target:* RASTERIZER_DISCARD, MULTISAMPLE

### Multisampling [3.3.1]

Use to antialias points, lines, polygons, bitmaps, and images.

**Enable/Disable**(MULTISAMPLE)

void **GetMultisamplefv**(enum *pname*, uint *index*, float **val*);
*pname:* SAMPLE_POSITION

### Points [3.4]

void **PointSize**(float *size*);

void **PointParameter{if}**(enum *pname*, T *param*);

void **PointParameter{if}v**(enum *pname*, const T *params*);
*pname:* POINT_SIZE_MIN, POINT_SIZE_MAX, POINT_DISTANCE_ATTENUATION POINT_FADE THRESHOLD_SIZE, POINT_SPRITE_COORD_ORIGIN
*param, params:* LOWER_LEFT, UPPER_LEFT, pointer to point fade threshold

**Enable/Disable**(VERTEX_PROGRAM_POINT_SIZE)

**Enable/Disable**(POINT_SMOOTH) *(Point antialias)*

**Enable/Disable**(POINT_SPRITE)

### Line Segments [3.5]

void **LineWidth**(float *width*);

**Enable/Disable**(LINE_SMOOTH) *(Line antialias)*

### Other Line Segments Features [3.5.2]

void **LineStipple**(int *factor*, ushort *pattern*);

**Enable/Disable**(LINE_STIPPLE)

### Stipple Query [6.1.5]

void **GetPolygonStipple**(void **pattern*);

### Polygons [3.6]

**Enable/Disable**(POLYGON_STIPPLE)

**Enable/Disable**(POLYGON_SMOOTH) *(Polygon antialias)*

void **FrontFace**(enum *dir*);
*dir:* CCW, CW

void **CullFace**(enum *mode*);
*mode:* FRONT, BACK, FRONT_AND_BACK

**Enable/Disable**(CULL_FACE)

### Stippling [3.6.2]

void **PolygonStipple**(ubyte **pattern*);

### Polygon Rasterization & Depth Offset [3.6.3] [3.6.4]

void **PolygonMode**(enum *face*, enum *mode*);
*face:* FRONT, BACK, FRONT_AND_BACK
*mode*: POINT, LINE, FILL

void **PolygonOffset**(float *factor*, float *units*);

**Enable/Disable**(*target*)
*target:* POLYGON_OFFSET_POINT, POLYGON_OFFSET_LINE, POLYGON_OFFSET_FILL

### Pixel Rectangles [3.7]

void **PixelStore{if}**(enum *pname*, T *param*);
*pname:* UNPACK_*x* (where *x* may be SWAP_BYTES, LSB_FIRST, ROW_LENGTH, SKIP_ROWS, SKIP_PIXELS, ALIGNMENT, IMAGE_HEIGHT, SKIP_IMAGES)

### Pixel Transfer Modes [3.7.3]

void **PixelTransfer{if}**(enum *param*, T *value*);
*param:* MAP_COLOR, MAP_STENCIL, INDEX_SHIFT, INDEX_OFFSET, *x*_SCALE, DEPTH_SCALE, *x*_BIAS, DEPTH_BIAS, or another value from [Table 3.2]

void **PixelMap{ui us f}v**(enum *map*, sizei *size*, T *values*);
*map:* PIXEL_MAP_{I, S, R, G, B, A}_TO_{I, S, R, G, B, A} [Table 3.3]

### Enumerated Queries [6.1.3]

void **GetPixelMap{ui us f}v**(enum *map*, T *data*);
*map:* PIXEL_MAP_{I, S, R, G, B, A}_TO_{I, S, R, G, B, A} [Table 3.3]

### Color Table Specification [3.7.3]

void **ColorTable**(enum *target*, enum *internalformat*, sizei *width*, enum *format*, enum *type*, void **data*);
*target:* {PROXY_}COLOR_TABLE, POST_CONVOLUTION_COLOR_TABLE, POST_COLOR_MATRIX_COLOR_TABLE [Table 3.4]
*internalformat:* One of the formats in [Table 3.16] or [Tables 3.17-3.19] except the RED, RG, DEPTH_COMPONENT, and DEPTH_STENCIL base and sized internal formats in those tables, all sized internal formats with non-fixed internal data types as discussed in [3.9], and sized internal format RGB9_E5.
format: RED, GREEN, BLUE, ALPHA, RG, RGB, RGBA, BGRA, LUMINANCE, LUMINANCE_ALPHA
type: {UNSIGNED_}BYTE/SHORT/INT*, {HALF_}FLOAT [Table 3.5]

**Enable/Disable**(POST_COLOR_MATRIX_COLOR_TABLE)

void **ColorTableParameter{if}v**(enum *target*, enum *pname*, T *params*);
*target:* COLOR_TABLE, POST_CONVOLUTION_COLOR_TABLE, POST_COLOR_MATRIX_COLOR_TABLE
*pname:* COLOR_TABLE_SCALE, COLOR_TABLE_BIAS

### Alternate Color Table Specification Commands

void **CopyColorTable**(enum *target*, enum *internalformat*, int *x*, int *y*, sizei *width*);

void **ColorSubTable**(enum *target*, sizei *start*, sizei *count*, enum *format*, enum *type*, void **data*);

void **CopyColorSubTable**(enum *target*, sizei *start*, int *x*, int *y*, sizei *count*);
*target* and *pname*: ***see ColorTableParameter{if}v***

### Color Table Query [6.1.7]

void **GetColorTable**(enum *target*, enum *format*, enum *type*, void **table*);
*target:* COLOR_TABLE, POST_CONVOLUTION_COLOR_TABLE, POST_COLOR_MATRIX_COLOR_TABLE
*format* and *type*: ***See GetTexImage***, except *format* cannot be DEPTH_COMPONENT

void **GetColorTableParameter{if}v**(enum *target*, enum *pname*, T *params*);
*target:* {PROXY_}COLOR_TABLE, {PROXY_}POST_CONVOLUTION_COLOR_TABLE, {PROXY_}POST_COLOR_MATRIX_COLOR_TABLE
*pname:* COLOR_TABLE_*x* (where *x* may be SCALE, BIAS, FORMAT, COLOR_TABLE_WIDTH, RED_SIZE, GREEN_SIZE, BLUE_SIZE, ALPHA_SIZE, LUMINANCE_SIZE, INTENSITY_SIZE)

### Convolution Filter Specification [3.7.3]

**Enable/Disable**(POST_CONVOLUTION_COLOR_TABLE)

void **ConvolutionFilter2D**(enum *target*, enum *internalformat*, sizei *width*, sizei *height*, enum *format*, enum *type*, void **data*);
*target:* CONVOLUTION_2D
*internalformat:* ***see ColorTable***
*format:* RED, GREEN, BLUE, ALPHA, RG, RGB, RGBA, BGRA, LUMINANCE, LUMINANCE_ALPHA, RED_INTEGER, GREEN_INTEGER, BLUE_INTEGER, ALPHA_INTEGER, RG_INTEGER, RGB_INTEGER, RGBA_INTEGER, BGR_INTEGER, BGRA_INTEGER
*type:* {UNSIGNED_}BYTE/SHORT/INT*, {HALF_}FLOAT

void **ConvolutionParameter{if}v**(enum *target*, enum *pname*, T *params*);
*target:* CONVOLUTION_2D
*pname:* CONVOLUTION_FILTER_SCALE, CONVOLUTION_FILTER_BIAS

void **ConvolutionFilter1D**(enum *target*, enum *internalformat*, sizei *width*, enum *format*, enum *type*, void **data*);
*target:* CONVOLUTION_1D
*internalformat, format,* and *type*: ***see ConvolutionFilter2D***

void **SeparableFilter2D**(enum *target*, enum *internalformat*, sizei *width*, sizei *height*, enum *format*, enum *type*, void **row*, void **column*);
*target:* SEPARABLE_2D
*internalformat, format,* and *type*: ***see ConvolutionFilter2D***

### Alternate Convolution Filter Specification Commands

void **CopyConvolutionFilter2D**(enum *target*, enum *internalformat*, int *x*, int *y*, sizei *width*, sizei *height*);
*target:* CONVOLUTION_2D
*internalformat:* ***see ConvolutionFilter2D***

void **CopyConvolutionFilter1D**(enum *target*, enum *internalformat*, int *x*, int *y*, sizei *width*);
*target:* CONVOLUTION_1D
*internalformat:* ***see ConvolutionFilter2D***

### Convolution Query [6.1.8]

void **GetConvolutionFilter**(enum *target*, enum *format*, enum *type*, void **image*);
*target:* CONVOLUTION_1D, CONVOLUTION_2D
*format* and *type*: ***See GetTexImage***, except *format* cannot be DEPTH_COMPONENT

void **GetSeparableFilter**(enum *target*, enum *format*, enum *type*, void **row*, void **column*, void **span*);
*target:* SEPARABLE_2D
*format* and *type*: ***See GetTexImage***

void **GetConvolutionParameter{if}v**(enum *target*, enum *pname*, T *params*);
*target:* CONVOLUTION_1D, CONVOLUTION_2D, SEPARABLE_2D
*pname:* {MAX_}CONVOLUTION_WIDTH, {MAX_}CONVOLUTION_HEIGHT, CONVOLUTION_*x* (where *x* may be BORDER_COLOR, BORDER_MODE, FILTER_SCALE, FILTER_BIAS, FORMAT)

### Histogram Table Specification [3.7.3]

**Enable/Disable**(HISTOGRAM)

void **Histogram**(enum *target*, sizei *width*, enum *internalformat*, boolean *sink*);
*target:* HISTOGRAM, PROXY_HISTOGRAM
*internalformat:* ***see ColorTable***

### Histogram Query [6.1.9]

void **GetHistogram**(enum *target*, boolean *reset*, enum *format*, enum *type*, void **values*);
*target:* HISTOGRAM
*format* and *type*: ***See GetTexImage***, except *format* cannot be DEPTH_COMPONENT

void **ResetHistogram**(enum *target*);
*target:* HISTOGRAM

void **GetHistogramParameter{if}v**(enum *target*, enum *pname*, T *params*);
*target:* HISTOGRAM, PROXY_HISTOGRAM
*pname:* HISTOGRAM_*x* (where *x* may be FORMAT, WIDTH, RED_SIZE, GREEN_SIZE, BLUE_SIZE, ALPHA_SIZE, LUMINANCE_SIZE, SINK)

### Minmax Table Specification [3.7.3]

**Enable/Disable**(MINMAX)

void **Minmax**(enum *target*, enum *internalformat*, boolean *sink*);
*target:* MINMAX
*internalformat:* ***see ColorTable***, except INTENSITY base and sized internal formats

### Minmax Query [6.1.10]

void **GetMinmax**(enum *target*, boolean *reset*, enum *format*, enum *type*, void **values*);
*target:* MINMAX
*format* and *type*: ***See GetTexImage***, except *format* cannot be DEPTH_COMPONENT

void **ResetMinmax**(enum *target*);
*target:* MINMAX

void **GetMinmaxParameter{if}v**(enum *target*, enum *pname*, T *params*);
*target:* MINMAX
*pname:* MINMAX_FORMAT, MINMAX_SINK

### Rasterization of Pixel Rectangles [3.7.5]

void **DrawPixels**(sizei *width*, sizei *height*, enum *format*, enum *type*, void **data*);
*format:* {COLOR|STENCIL}_INDEX, DEPTH_COMPONENT, DEPTH_STENCIL, RED, GREEN, BLUE, ALPHA, RG, RGB, RGBA, BGR, BGRA, LUMINANCE{_ALPHA} [Table 3.6] (*_INTEGER formats are not supported)
type: UNSIGNED_BYTE, BITMAP, BYTE, UNSIGNED_SHORT, SHORT, UNSIGNED_INT, INT, HALF_FLOAT, FLOAT, or another value from [Table 3.5]

void **ClampColor**(enum *target*, enum *clamp*);
*target:* CLAMP_READ_COLOR, CLAMP_FRAGMENT_COLOR
*clamp:* TRUE, FALSE, FIXED_ONLY

void **PixelZoom**(float *zx*, float *zy*);

### Pixel Transfer Operations [3.7.6]

void **ConvolutionParameter{if}**(enum *target*, enum *pname*, T *param*);
*target:* CONVOLUTION_1D, CONVOLUTION_2D, SEPARABLE_2D
*pname:* CONVOLUTION_BORDER_MODE
*param:* REDUCE, CONSTANT_BORDER, REPLICATE_BORDER

### Bitmaps [3.8]

void **Bitmap**(sizei *w*, sizei *h*, float *xb0*, float *yb0*, float *xbi*, float *ybi*, ubyte **data*);

## Texturing [3.8] [3.9]

void **ActiveTexture**(enum *texture*);
*texture:* TEXTURE*i* (where *i* is [0, MAX_COMBINED_TEXTURE_IMAGE_UNITS - 1])

### Texture Image Specification [3.8.1] [3.9.1]

void **TexImage3D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, enum *format*, enum *type*, void **data*);
*target:* {PROXY_}TEXTURE_3D, {PROXY_}TEXTURE_2D_ARRAY
*internalformat:* ALPHA, DEPTH_COMPONENT, DEPTH_STENCIL, LUMINANCE, LUMINANCE_ALPHA, INTENSITY, RED, RG, RGB, RGBA; a sized internal format from [Tables 3.12-3.13] [Tables 3.17-3.19]; COMPRESSED_RED_RGTC1, COMPRESSED_SIGNED_RED_RGTC1, COMPRESSED_RG_RGTC2, COMPRESSED_SIGNED_RG_RGTC2, or a generic compressed format from [Table 3.14] [Table 3.20]
*format:* COLOR_INDEX, DEPTH_COMPONENT, DEPTH_STENCIL, RED, GREEN, BLUE, ALPHA, RG, RGB, RGBA, BGR, BGRA, LUMINANCE, LUMINANCE_ALPHA, RED_INTEGER, GREEN_INTEGER, BLUE_INTEGER, ALPHA_INTEGER, RG_INTEGER, RGB_INTEGER, RGBA_INTEGER, BGR_INTEGER, BGRA_INTEGER, or one of the values from [Table 3.5] [Table 3.8]
type: UNSIGNED_BYTE, BITMAP, BYTE, UNSIGNED_SHORT, SHORT, UNSIGNED_INT, INT, HALF_FLOAT, FLOAT, or a value from [Table 3.5]

void **TexImage2D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, int *border*, enum *format*, enum *type*, void **data*);
*target:* {PROXY_}TEXTURE_2D, {PROXY_}TEXTURE_1D_ARRAY, {PROXY_}TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_*, PROXY_TEXTURE_CUBE_MAP
*internalformat, format,* and *type*: ***See TexImage3D***

void **TexImage1D**(enum *target*, int *level*, int *internalformat*, sizei *width*, int *border*, enum *format*, enum *type*, void **data*);
*target:* TEXTURE_1D, PROXY_TEXTURE_1D
*type:* UNSIGNED_BYTE, BITMAP, BYTE, UNSIGNED_SHORT, SHORT, UNSIGNED_INT, INT, HALF_FLOAT, FLOAT, or another value from [Table 3.2] [Table 3.5]
*internalformat* and *format*: ***See TexImage3D***

### Alt. Texture Image Specification Commands [3.8.2] [3.9.2]

void **CopyTexImage2D**(enum *target*, int *level*, enum *internalformat*, int *x*, int *y*, sizei *width*, sizei *height*, int *border*);
*target:* TEXTURE_2D, TEXTURE_1D_ARRAY, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP*
*internalformat:* ***See TexImage2D***

void **CopyTexImage1D**(enum *target*, int *level*, enum *internalformat*, int *x*, int *y*, sizei *width*, int *border*);
*target:* TEXTURE_1D
*internalformat:* ***See TexImage1D***

*(Continued >)*

# OpenGL 3.2 API Quick Reference Card

## Texturing (continued)

void **TexSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, enum *type*, void **data*);
*target:* TEXTURE_3D, TEXTURE_2D_ARRAY
*format* and *type:* *See TexImage3D*

void **TexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, enum *type*, void **data*);
*target:* TEXTURE_2D, TEXTURE_1D_ARRAY, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_*
*format* and *type:* *See TexImage2D*

void **TexSubImage1D**(enum *target*, int *level*, int *xoffset*, sizei *width*, enum *format*, enum *type*, void **data*);
*target:* TEXTURE_1D
*format:* *See TexImage1D*
*type:* BYTE, UNSIGNED_BYTE*, SHORT, UNSIGNED_SHORT*, INT, UNSIGNED_INT*, HALF_FLOAT, FLOAT*

void **CopyTexSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, int *x*, int *y*, sizei *width*, sizei *height*);
*target:* TEXTURE_3D, TEXTURE_2D ARRAY

void **CopyTexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *x*, int *y*, sizei *width*, sizei *height*);
*target:* TEXTURE_2D, TEXTURE_1D_ARRAY, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP*

void **CopyTexSubImage1D**(enum *target*, int *level*, int *xoffset*, int *x*, int *y*, sizei *width*);
*target:* TEXTURE_1D

### Compressed Texture Images [3.8.3] [3.9.3]

void **CompressedTexImage3D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, sizei *imageSize*, void **data*);
*target:* *See TexImage3D*
*internalformat:* COMPRESSED_RED_RGTC1_RED, COMPRESSED_SIGNED_RED_RGTC1_RED, COMPRESSED_RG_RGTC2_RG, COMPRESSED_SIGNED_RG_RGTC2

void **CompressedTexImage2D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, int *border*, sizei *imageSize*, void **data*);
*target:* *See TexImage2D* (Compressed rectangular texture formats not supported.)
*internalformat:* *See CompressedTexImage3D*

void **CompressedTexImage1D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, int *border*, sizei *imageSize*, void **data*);
*target:* TEXTURE_1D, PROXY_TEXTURE_1D
*internalformat:* *See CompressedTexImage3D*

void **CompressedTexSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, sizei *imageSize*, void **data*);
*target:* TEXTURE_3D, TEXTURE_2D ARRAY
*format:* *See TexImage3D*

void **CompressedTexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, sizei *imageSize*, void **data*);
*target:* TEXTURE_2D, TEXTURE_1D_ARRAY, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_*
*format:* *See TexImage2D*

void **CompressedTexSubImage1D**(enum *target*, int *level*, int *xoffset*, sizei *width*, enum *format*, sizei *imageSize*, void **data*);
*target:* TEXTURE_1D
*format:* *See TexImage1D*

### Multisample Textures [3.8.4] [3.9.4]

void **TexImage3DMultisample**(enum *target*, sizei *samples*, int *internalformat*, sizei *width*, sizei *height*, sizei *depth*, boolean *fixedsamplelocations*);
*target:* TEXTURE_2D_MULTISAMPLE_ARRAY, PROXY_TEXTURE_2D_MULTISAMPLE_ARRAY
*internalformat:* ALPHA, RED, RG, RGB, RGBA, DEPTH_COMPONENT, DEPTH_STENCIL, STENCIL_INDEX, or the sized internal formats corresponding to these base formats

void **TexImage2DMultisample**(enum *target*, sizei *samples*, int *internalformat*, sizei *width*, sizei *height*, boolean *fixedsamplelocations*);
*target:* TEXTURE_2D_MULTISAMPLE, PROXY_TEXTURE_2D_MULTISAMPLE
*internalformat:* *See TexImage3DMultisample*

### Buffer Textures [3.8.5] [3.9.5]

void **TexBuffer**(enum *target*, enum *internalformat*, uint *buffer*);
*target:* TEXTURE_BUFFER
*internalformat:* R8, R16, R16F, R32F, R8I, R16I, R32I, R8U1, R16UI, R32UI, RG8, RG26, RG16F, RG32F, RG8I, RG16I, RG32I, RG8UI, RG16UI, RG32UI, RGBA8, RGBA16, RGBA16F, RGBA32F, RGBA8I, RGBA16I, RGBA32I, RGBA8UI, RGBA16UI, RGBA32UI

### Texture Parameters [3.8.6] [3.9.6]

void **TexParameter{if}**(enum *target*, enum *pname*, T *param*);

void **TexParameter{if}v**(enum *target*, enum *pname*, T **params*);

void **TexParameterI{i ui}v**(enum *target*, enum *pname*, T **params*);
*target:* TEXTURE_1D*, TEXTURE_2D*, TEXTURE_3D, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP
*pname:* TEXTURE_WRAP_{S, T, R}, TEXTURE_{MIN, MAG}_FILTER, TEXTURE_BORDER_COLOR, TEXTURE_PRIORITY, TEXTURE_{MIN, MAX}_LOD, TEXTURE_{BASE, MAX}_LEVELS, TEXTURE_LOD_BIAS, DEPTH_TEXTURE_MODE, TEXTURE_COMPARE_{MODE, FUNC}, GENERATE_MIPMAP [Table 3.16] [Table 3.22]

### Seamless Cube Map Filtering [3.8.8] [3.9.8]

**Enable/Disable**(TEXTURE_CUBE_MAP_SEAMLESS)

### Manual Mipmap Generation [3.8.9] [3.9.9]

void **GenerateMipmap**(enum *target*);
*target:* TEXTURE_1D*, TEXTURE_2D*, TEXTURE_3D, TEXTURE_CUBE_MAP

### Texture Objects [3.8.14] [3.9.14]

void **BindTexture**(enum *target*, uint *texture*);
*target:* TEXTURE_{1, 2}D{_ARRAY}, TEXTURE_3D, TEXTURE_RECTANGLE, TEXTURE_BUFFER, TEXTURE_CUBE_MAP, TEXTURE_2D_MULTISAMPLE{_ARRAY}

void **DeleteTextures**(sizei *n*, uint **textures*);

void **GenTextures**(sizei *n*, uint **textures*);

boolean **AreTexturesResident**(sizei *n*, uint **textures*, boolean **residences*);

void **PrioritizeTextures**(sizei *n*, uint **textures*, clampf **priorities*);

### Texture Environments & Texture Functions [3.9.15]

void **TexEnv{if}**(enum *target*, enum *pname*, T *param*);

void **TexEnv{if}v**(enum *target*, enum *pname*, T *params*);
*target:* TEXTURE_FILTER_CONTROL, POINT_SPRITE, TEXTURE_ENV
*pname:* TEXTURE_LOD_BIAS, TEXTURE_ENV_MODE, TEXTURE_ENV_COLOR, COMBINE_RGB, COMBINE_ALPHA, RGB_SCALE, ALPHA_SCALE, COORD_REPLACE, SRC*n*_RGB, SRC*n*_ALPHA, OPERAND*n*_RGB, OPERAND*n*_ALPHA (where *n* is [0, 1, 2])

### Texture Application [3.9.19]

**Enable/Disable**(*param*)
*param:* TEXTURE_1D, TEXTURE_2D, TEXTURE_3D, TEXTURE_CUBE_MAP

### Enumerated Queries [6.1.3]

void **GetTexEnv{if}v**(enum *env*, enum *value*, T *data*);
*env:* POINT_SPRITE, TEXTURE_ENV, TEXTURE_FILTER_CONTROL

void **GetTexGen{ifd}v**(enum *coord*, enum *value*, T *data*);
*coord:* S, T, R, Q

void **GetTexParameter{if}v**(enum *target*, enum *value*, T *data*);

void **GetTexParameterI{i ui}v**(enum *target*, enum *value*, T *data*);
*target:* TEXTURE_1D*, TEXTURE_2D*, TEXTURE_3D, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP
*value:* TEXTURE_RESIDENT, TEXTURE_WRAP_{S, T, R}, TEXTURE_{MIN, MAG}_FILTER, TEXTURE_BORDER_COLOR, TEXTURE_PRIORITY, TEXTURE_{MIN, MAX}_LOD, TEXTURE_{BASE, MAX}_LEVEL, TEXTURE_LOD_BIAS, DEPTH_TEXTURE_MODE, TEXTURE_COMPARE_{MODE, FUNC}, GENERATE_MIPMAP

void **GetTexLevelParameter{if}v**(enum *target*, int *lod*, enum *value*, T *data*);
*target:* TEXTURE_1D*, TEXTURE_2D*, {PROXY_}TEXTURE_3D, {PROXY_}TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_*, PROXY_TEXTURE_1D{_ARRAY}, PROXY_TEXTURE_2D{_ARRAY}, {PROXY_}TEXTURE_2D_MULTISAMPLE*, PROXY_TEXTURE_CUBE_MAP , TEXTURE_BUFFER
*value:* {PROXY_}TEXTURE_{1, 2}D{_ARRAY}, {PROXY_}TEXTURE_3D, {PROXY_}TEXTURE_RECTANGLE, {PROXY_}TEXTURE_2D_MULTISAMPLE{_ARRAY}, TEXTURE_BUFFER, TEXTURE_CUBE_MAP_{POSITIVE|NEGATIVE}_{X, Y, Z}, PROXY_TEXTURE_CUBE_MAP

### Texture Queries [6.1.4]

void **GetTexImage**(enum *target*, int *lod*, enum *format*, enum *type*, void **img*);
*target:* TEXTURE_{1, 2}D{_ARRAY}, TEXTURE_3D, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_{POSITIVE|NEGATIVE}_{X, Y, Z}
*format:* *See TexImage3D*
*type:* BITMAP, {UNSIGNED_}BYTE/SHORT/INT*, {HALF_}FLOAT, FLOAT_32_UNSIGNED_INT_24_8_REV [Table 3.2] [Table 3.5]

void **GetCompressedTexImage**(enum *target*, int *lod*, void **img*);
*target:* *See GetTexImage*

boolean **IsTexture**(uint *texture*);

## Color Sum, Fog, and Hints

### Color Sum [3.10]

**Enable/Disable**(COLOR_SUM)

### Fog [3.11]

**Enable/Disable**(FOG)

void **Fog{if}**(enum *pname*, T *param*);

void **Fog{if}v**(enum *pname*, T *params*);
*pname:* FOG_MODE, FOG_COORD_SRC, FOG_DENSITY, FOG_START, FOG_END, FOG_COLOR, FOG_INDEX

### Hints [5.3] [5.7]

void **Hint**(enum *target*, enum *hint*);
*target:* LINE_SMOOTH_HINT, FRAGMENT_SHADER_DERIVATIVE_HINT, TEXTURE_COMPRESSION_HINT, POLYGON_SMOOTH_HINT, PERSPECTIVE_CORRECTION_HINT, POINT_SMOOTH_HINT, FOG_HINT, GENERATE_MIPMAP_HINT
*hint:* FASTEST, NICEST, DONT_CARE

## Drawing, Reading, and Copying Pixels

### Reading Pixels [4.3.1] [4.3.2]

void **ReadPixels**(int *x*, int *y*, sizei *width*, sizei *height*, enum *format*, enum *type*, void **data*);
*format:* {COLOR, STENCIL}_INDEX, DEPTH_COMPONENT, DEPTH_STENCIL, RED, GREEN, BLUE, ALPHA, RG, RGB, RGBA, BGR, BGRA, LUMINANCE{_ALPHA}, {RED, GREEN, BLUE, ALPHA, RG, RGB, RGBA, BGR, BGRA}_INTEGER [Table 3.6]
type: BITMAP, {UNSIGNED_}BYTE/SHORT/INT*, {HALF_}FLOAT, FLOAT_32_UNSIGNED_INT_24_8_REV [Table 3.2] [Table 3.5]

void **ReadBuffer**(enum *src*);
*src:* NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, AUX*i* (where *i* is [0, AUX_BUFFERS - 1 ]), COLOR_ATTACHMENT*i* (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1])

### Copying Pixels [4.3.2] [4.3.3]

void **CopyPixels**(int *x*, int *y*, sizei *width*, sizei *height*, enum *type*);
*type:* COLOR, STENCIL, DEPTH, DEPTH_STENCIL

### Blitting Pixel Rectangles [4.3.2] [4.3.3]

void **BlitFramebuffer**(int *srcX0*, int *srcY0*, int *srcX1*, int *srcY1*, int *dstX0*, int *dstY0*, int *dstX1*, int *dstY1*, bitfield *mask*, enum *filter*);
*mask:* Bitwise OR of COLOR_BUFFER_BIT, DEPTH_BUFFER_BIT, STENCIL_BUFFER_BIT
*filter:* LINEAR, NEAREST

*Also see DrawPixels, ClampColor, and PixelZoom in the Rasterization section of this reference card.*

## Per-Fragment Operations

### Scissor Test [4.1.2]

**Enable/Disable**(SCISSOR_TEST)

void **Scissor**(int *left*, int *bottom*, sizei *width*, sizei *height* );

### Multisample Fragment Operations [4.1.3]

**Enable/Disable**(*cap*)
*cap:* SAMPLE_ALPHA_TO_COVERAGE, SAMPLE_ALPHA_TO_ONE, SAMPLE_COVERAGE

void **SampleCoverage**(clampf *value*, boolean *invert*);

void **SampleMaski**(uint *maskNumber*, bitfield *mask*);

### Alpha Test [4.1.4]

**Enable/Disable**(ALPHA_TEST)

void **AlphaFunc**(enum *func*, clampf *ref*);
*func:* NEVER, ALWAYS, LESS,LEQUAL, EQUAL, GEQUAL, GREATER, NOTEQUAL

### Stencil Test [4.1.4] [4.1.5]

**Enable/Disable**(STENCIL_TEST)

void **StencilFunc**(enum *func*, int *ref*, uint *mask*);

void **StencilFuncSeparate**(enum *face*, enum *func*, int *ref*, uint *mask*);

void **StencilOp**(enum *sfail*, enum *dpfail*, enum *dppass*);

void **StencilOpSeparate**(enum *face*, enum *sfail*, enum *dpfail*, enum *dppass*);
*face:* FRONT, BACK, FRONT_AND_BACK
*sfail, dpfail,* and *dppass:* KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECR_WRAP
*func:* NEVER, ALWAYS, LESS, LEQUAL, EQUAL, GREATER, GEQUAL, NOTEQUAL

### Depth Buffer Test [4.1.5] [4.1.6]

**Enable/Disable**(DEPTH_TEST)

void **DepthFunc**(enum *func*);
*func:* *See StencilOpSeparate*

### Occlusion Queries [4.1.6] [4.1.7]

**BeginQuery**(SAMPLES_PASSED, uint *id*);

**EndQuery**(SAMPLES_PASSED);

### Blending [4.1.7] [4.1.8]

**Enablei/Disablei**(BLEND, uint *index*) *(individual buffers)*

**Enable/Disable**(BLEND) *(all draw buffers)*

void **BlendEquation**(enum *mode*);

void **BlendEquationSeparate**(enum *modeRGB*, enum *modeAlpha*);
*mode, modeRGB,* and *modeAlpha*: FUNC_ADD, FUNC_SUBTRACT, FUNC_REVERSE_SUBTRACT, MIN, MAX

*(Continued >)*

# OpenGL 3.2 API Quick Reference Card

## Per-Fragment Operations (cont.)

void **BlendFuncSeparate**(enum *srcRGB*, enum *dstRGB*, enum *srcAlpha*, enum *dstAlpha*);

void **BlendFunc**(enum *src*, enum *dst*);
*dst, dstRGB,* and *dstAlpha*: ZERO, ONE, {ONE_MINUS_}SRC_COLOR, {ONE_MINUS_}DST_COLOR, {ONE_MINUS_}SRC_ALPHA, {ONE_MINUS_}DST_ALPHA, {ONE_MINUS_}CONSTANT_COLOR, {ONE_MINUS_}CONSTANT_ALPHA
*src, srcRGB, srcAlpha*: same for *dst*, plus SRC_ALPHA_SATURATE

void **BlendColor**(clampf *red*, clampf *green*, clampf *blue*, clampf *alpha*);

### Dithering [4.1.9] [4.1.10]

**Enable/Disable**(DITHER)

### Logical Operation [4.1.10] [4.1.11]

**Enable/Disable**(COLOR_LOGIC_OP)

void **LogicOp**(enum *op*);
*op:* CLEAR, AND, AND_REVERSE, COPY, AND_INVERTED, NOOP, OR, OR, NOR, EQUIV, INVERT, OR_REVERSE, COPY_INVERTED, OR_INVERTED, NAND, SET

## Whole Framebuffer Operations

### Selecting a Buffer for Writing [4.2.1]

void **DrawBuffer**(enum *buf*);
*buf:* NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, COLOR_ATTACHMENT*i* (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1 ]), AUX*i* (where *i* is [0, AUX_BUFFERS - 1 ])

void **DrawBuffers**(sizei *n*, const enum **bufs*);
*bufs:* NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, COLOR_ATTACHMENT*i* (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1 ]), AUX*i* (where *i* is [0, AUX_BUFFERS - 1 ])

### Fine Control of Buffer Updates [4.2.2]

void **IndexMask**(uint *mask*);

void **ColorMask**(boolean *r*, boolean *g*, boolean *b*, boolean *a*);

void **ColorMaski**(uint *buf*, boolean *r*, boolean *g*, boolean *b*, boolean *a*);

void **DepthMask**(boolean *mask*);

void **StencilMask**(uint *mask*);

void **StencilMaskSeparate**(enum *face*, uint *mask*);
*face:* FRONT, BACK, FRONT_AND_BACK

### Clearing the Buffers [4.2.3]

void **Clear**(bitfield *buf*);
*buf:* Bitwise OR of COLOR_BUFFER_BIT, DEPTH_BUFFER_BIT, STENCIL_BUFFER_BIT, ACCUM_BUFFER_BIT

void **ClearColor**(clampf *r*, clampf *g*, clampf *b*, clampf *a*);

void **ClearIndex**(float *index*);

void **ClearDepth**(clampd *d*);

void **ClearStencil**(int *s*);

void **ClearAccum**(float *r*, float *g*, float *b*, float *a*);

void **ClearBuffer{if ui}v**(enum *buffer*, int *drawbuffer*, const T **value*)
*buffer:* COLOR, DEPTH, STENCIL

void **ClearBufferfi**(enum *buffer*, int *drawbuffer*, float *depth*, int *stencil*);
*buffer:* DEPTH_STENCIL
*drawbuffer:* 0

### Accumulation Buffer [4.2.4]

void **Accum**(enum *op*, float *value*);
*op:* ACCUM, LOAD, RETURN, MULT, ADD.

## Framebuffer Objects

### Binding & Managing Framebuffer Objects [4.4.1]

void **BindFramebuffer**(enum *target*, uint *framebuffer*);
*target:* DRAW_FRAMEBUFFER, READ_FRAMEBUFFER, FRAMEBUFFER

void **DeleteFramebuffers**(sizei *n*, uint **framebuffers*);

void **GenFramebuffers**(sizei *n*, uint **ids*);

### Attaching Images to Framebuffer Objects [4.4.2]

#### Renderbuffer Objects

void **BindRenderbuffer**(enum *target*, uint *renderbuffer*);
*target:* RENDERBUFFER

void **DeleteRenderbuffers**(sizei *n*, const uint **renderbuffers*);

void **GenRenderbuffers**(sizei *n*, uint **renderbuffers*);

void **RenderbufferStorageMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*);
*target:* RENDERBUFFER
*internalformat:* ***See TexImage2DMultisample***

void **RenderbufferStorage**(enum *target*, enum *internalformat*, sizei *width*, sizei *height*);
*target* and *internalformat:* ***See RenderbufferStorageMultisample***

#### Attaching Renderbuffer Images to Framebuffer

void **FramebufferRenderbuffer**(enum *target*, enum *attachment*, enum *renderbuffertarget*, uint *renderbuffer*);
*target:* DRAW_FRAMEBUFFER, READ_FRAMEBUFFER, FRAMEBUFFER
*attachment:* DEPTH_ATTACHMENT, STENCIL_ATTACHMENT, DEPTH_STENCIL_ATTACHMENT, COLOR_ATTACHMENT*i* (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1])
*renderbuffertarget:* RENDERBUFFER

#### Attaching Texture Images to a Framebuffer

void **FramebufferTexture**(enum *target*, enum *attachment*, uint *texture*, int *level*);
*target:* DRAW_FRAMEBUFFER, READ_FRAMEBUFFER, FRAMEBUFFER
*attachment:* ***See FramebufferRenderbuffer***

void **FramebufferTexture3D**(enum *target*, enum *attachment*, enum *textarget*, uint *texture*, int *level*, int *layer*);
*textarget:* TEXTURE_3D
*target* and *attachment:* ***See FramebufferRenderbuffer***

void **FramebufferTexture2D**(enum *target*, enum *attachment*, enum *textarget*, uint *texture*, int *level*);
*textarget:* TEXTURE_2D{_MULTISAMPLE}, TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_*
*target* and *attachment:* ***See FramebufferRenderbuffer***

void **FramebufferTexture1D**(enum *target*, enum *attachment*, enum *textarget*, uint *texture*, int *level*);
*textarget:* TEXTURE_1D
*target* and *attachment:* ***See FramebufferRenderbuffer***

void **FramebufferTextureLayer**(enum *target*, enum *attachment*, uint *texture*, int *level*, int *layer*);
*target* and *attachment:* ***See FramebufferTexture3D***

### Framebuffer Completeness [4.4.4]

enum **CheckFramebufferStatus**(enum *target*);
*target:* DRAW_FRAMEBUFFER, READ_FRAMEBUFFER, FRAMEBUFFER
returns: FRAMEBUFFER_COMPLETE or a constant indicating which value violates framebuffer completeness

### Framebuffer Object Queries [6.1.11] [6.1.17]

boolean **IsFramebuffer**(uint *framebuffer*);

void **GetFramebufferAttachmentParameteriv**(enum *target*, enum *attachment*, enum *pname*, int **params*);
*target:* DRAW_FRAMEBUFFER, READ_FRAMEBUFFER, FRAMEBUFFER
*attachment:* FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, COLOR_ATTACHMENT*i*, AUX*i*, DEPTH, STENCIL, DEPTH_ATTACHMENT, STENCIL_ATTACHMENT, DEPTH_STENCIL_ATTACHMENT
*pname:* FRAMEBUFFER_ATTACHMENT_*x* (where *x* may be OBJECT_TYPE, OBJECT_NAME, RED_SIZE, GREEN_SIZE, BLUE_SIZE, ALPHA_SIZE, DEPTH_SIZE, STENCIL_SIZE, COMPONENT_TYPE, COLOR_ENCODING, TEXTURE_LEVEL, LAYERED, TEXTURE_CUBE_MAP_FACE, TEXTURE_LAYER)

### Renderbuffer Object Queries [6.1.12] [6.1.18]

boolean **IsRenderbuffer**(uint *renderbuffer*);

void **GetRenderbufferParameteriv**(enum *target*, enum *pname*, int **params*);
*target:* RENDERBUFFER
*pname:* RENDERBUFFER_*x* (where *x* may be WIDTH, HEIGHT, RED_SIZE, GREEN_SIZE, BLUE_SIZE, ALPHA_SIZE, DEPTH_SIZE, STENCIL_SIZE, INTERNAL_FORMAT, SAMPLES)

## Special Functions

### Evaluators [5.1]

void **Map1{fd}**(enum *target*, T *u1*, T *u2*, int *stride*, int *order*, T *points*);
*target:* MAP1_VERTEX_3, MAP1_VERTEX_4, MAP1_INDEX 1, MAP1_COLOR_4, MAP1_NORMA, MAP1_TEXTURE_COORD_1, MAP1_TEXTURE_COORD_2, MAP1_TEXTURE_COORD_3, MAP1_TEXTURE_COORD_4

void **Map2{fd}**(enum *target*, T *u1*, T *u2*, int *ustride*, int *uorder*, T *v1*, T *v2*, int *vstride*, int *vorder*, T *points*);
*target:* ***See Map1***, except replace MAP1 with MAP2

void **EvalCoord{12}{fd}**(T *arg*);

void **EvalCoord{12}{fdg}**(T *arg*);

void **MapGrid1{fd}**(int *n*, T *u1*, T *u2*);

void **MapGrid2{fd}**(int *nu*, T *u1*, T *u2*, int *nv*, T *v1*, T *v2*);

void **EvalMesh1**(enum *mode*, int *p1*, int *p2*);
*mode:* POINT, LINE

void **EvalMesh2**(enum *mode*, int *p1*, int *p2*, int *q1*, int *q2*);
*mode:* FILL, POINT, LINE

void **EvalPoint1**(int *p*);

void **EvalPoint2**(int *p*, int *q*);

### Enumerated Query [6.1.3]

void **GetMap{ifd}v**(enum *map*, enum *value*, T *data*);
*map:* a map type described in section [5.1]
*value:* ORDER, COEFF, DOMAIN

### Selection [5.2]

void **InitNames**(void);

void **PopName**(void);

void **PushName**(uint *name*);

void **LoadName**(uint *name*);

int **RenderMode**(enum *mode*);
*mode:* RENDER, SELECT, FEEDBACK

void **SelectBuffer**(sizei *n*, uint **buffer*);

### Feedback [5.3]

void **FeedbackBuffer**(sizei *n*, enum *type*, float **buffer*);
*type:* 2D, 3D, 3D_COLOR, 3D_COLOR_TEXTURE, 4D_COLOR_TEXTURE

void **PassThrough**(float *token*);

### Display Lists [5.4]

void **NewList**(uint *n*, enum *mode*);
*mode:* COMPILE, COMPILE_AND_EXECUTE

void **EndList**(void);

void **CallList**(uint *n*);

void **CallLists**(sizei *n*, enum *type*, void **lists*);
*type:* BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, INT, UNSIGNED_INT, FLOAT

void **ListBase**(uint *base*);

uint **GenLists**(sizei *s*);

boolean **IsList**(uint *list*);

void **DeleteLists**(uint *list*, sizei *range*);

## Synchronization

### Flush and Finish [5.1] [5.5]

void **Flush**(void);

void **Finish**(void);

### Sync Objects and Fences [5.2] [5.6]

sync **FenceSync**(enum *condition*, bitfield *flags*)
*condition:* SYNC_GPU_COMMANDS_COMPLETE
*flags:* must be 0

void **DeleteSync**(sync *sync*);

### Waiting for Sync Objects [5.2.1] [5.6.1]

enum **ClientWaitSync**(sync *sync*, bitfield *flags*, uint64 *timeout_ns*);
*flags:* SYNC_FLUSH_COMMANDS_BIT, or zero

void **WaitSync**(sync *sync*, bitfield *flags*, uint64 *timeout_ns*);
*timeout_ns:* TIMEOUT_IGNORED

### Sync Object Queries [6.1.7] [6.1.13]

void **GetSynciv**(sync *sync*, enum *pname*, sizei *bufSize*, sizei **length*, int **values*);
*pname:* OBJECT_TYPE, SYNC_STATUS, SYNC_CONDITION, SYNC_FLAGS

boolean **IsSync**(sync *sync*);

## State and State Requests

A complete list of symbolic constants for states is shown in the tables in [6.2].

### Simple Queries [6.1.1]

void **GetBooleanv**(enum *value*, boolean **data*);

void **GetIntegerv**(enum *value*, int **data*);

void **GetInteger64v**(enum *value*, int64 **data*);

void **GetFloatv**(enum *value*, float **data*);

void **GetDoublev**(enum *value*, double **data*);

void **GetBooleani_v**(enum *target*, uint *index*, boolean **data*);

void **GetIntegeri_v**(enum *target*, uint *index*, int **data*);

boolean **IsEnabled**(enum *value*);

boolean **IsEnabledi**(enum *target*, uint *index*);

### Pointer and String Queries [6.1.5] [6.1.11]

void **GetPointerv**(enum *pname*, void ***params*);
*pname:* SELECTION_BUFFER_POINTER, FEEDBACK_BUFFER_POINTER, VERTEX_ARRAY_POINTER, NORMAL_ARRAY_POINTER, COLOR_ARRAY_POINTER, SECONDARY_COLOR_ARRAY_POINTER, INDEX_ARRAY_POINTER, TEXTURE_COORD_ARRAY_POINTER, FOG_COORD_ARRAY_POINTER, EDGE_FLAG_ARRAY_POINTER

ubyte ***GetString**(enum *name*);
*name:* RENDERER, VENDOR, VERSION, SHADING_LANGUAGE_VERSION, EXTENSIONS

ubyte ***GetStringi**(enum *name*, uint *index*);
*name:* EXTENSIONS
*index:* range is [0, NUM_EXTENSIONS - 1]

### Saving and Restoring State [6.1.19]

void **PushAttrib**(bitfield *mask*);
*mask:* ALL_ATTRIB_BITS, or the bitwise OR of the attribute groups in [Table 6.2]

void **PushClientAttrib**(bitfield *mask*);
*mask:* CLIENT_ALL_ATTRIB_BITS, or the bitwise OR of the attribute groups in [Table 6.2]

void **PopAttrib**(void);

void **PopClientAttrib**(void);

# 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**<br>**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;<br>**float** foo[3];<br>* structures and blocks can be arrays<br>* only 1-dimensional arrays supported<br>* structure members can be arrays |
| **Structures** | **struct** *type-name* {<br>*members*<br>} *struct-name*[]; // optional variable declaration, // optionally an array |
| **Blocks** | **in/out/uniform** *block-name* { // interface matching by // block name<br>*optionally-qualified members*<br>} *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<br>#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*<br>#extension all : *behavior* | • *behavior*: require, enable, warn, disable<br>• *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**<br>**centroid in** | linkage into a shader from previous stage (copied in)<br>linkage with centroid based interpolation |
| **out**<br>**centroid out** | linkage out of a shader to subsequent stage (copied out)<br>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<br>gl_BackColor<br>gl_FrontSecondaryColor<br>gl_BackSecondaryColor |
| **Fragment language:** | gl_Color<br>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. | **[ ]**<br>**( )**<br>**.**<br>**++ --** | array subscript<br>function call & constructor structure<br>field or method selector, swizzler<br>postfix increment and decrement |
| 3. | **++ --**<br>**+ - ~ !** | prefix increment and decrement<br>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. | **=+**<br>**= -=**<br>***= /=**<br>**%= <<= >>=**<br>**&= ^= \|=** | assignment<br>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[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 }<br>while ( ) { break, continue }<br>do { break, continue } while ( ); |
| **Selection** | if ( ) { }<br>if ( ) { } else { }<br>switch ( ) { case integer: ... break; ... default: ... } |
| **Jump** | break, continue, return<br>(There is no 'goto') |
| **Entry** | void main() |
| **Exit** | return in main()<br>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*)<br>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*)<br>Ti **abs**(Ti *x*) | absolute value |
| T **sign**(T *x*)<br>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*)<br>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*)<br>T **min**(T *x*, float *y*)<br>Ti **min**(Ti *x*, Ti *y*)<br>Ti **min**(Ti *x*, int *y*)<br>Tu **min**(Tu *x*, Tu *y*)<br>Tu **min**(Tu *x*, uint *y*) | minimum value |
| T **max**(T *x*, T *y*)<br>T **max**(T *x*, float *y*)<br>Ti **max**(Ti *x*, Ti *y*)<br>Ti **max**(Ti *x*, int *y*)<br>Tu **max**(Tu *x*, Tu *y*)<br>Tu **max**(Tu *x*, uint *y*) | maximum value |
| T **clamp**(T *x*, T *minVal*, T *maxVal*)<br>T **clamp**(T *x*, float *minVal*, float *maxVal*)<br>Ti **clamp**(Ti *x*, Ti *minVal*, Ti *maxVal*)<br>Ti **clamp**(Ti *x*, int *minVal*, int *maxVal*)<br>Tu **clamp**(Tu *x*, Tu *minVal*, Tu *maxVal*)<br>Tu **clamp**(Tu *x*, uint *minVal*, uint *maxVal*) | min(max(*x*, *minVal*), *maxVal*) |
| T **mix**(T *x*, T *y*, T *a*)<br>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*)<br>T **step**(float *edge*, T *x*) | 0.0 if *x* < *edge*, else 1.0 |
| T **smoothstep**(T *edge0*, T *edge1*, T *x*)<br>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*)<br>bvec **equal**(bvec *x*, bvec *y*) | == |
| bvec **notEqual**(T *x*, T *y*)<br>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*) | |
| vec2 **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*)