

OpenGL® is the only cross-platform graphics API that enables developers 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.

Specifications are available at www.khronos.org/opengl



- See [FunctionName](#) refers to functions on this reference card.
- [\[n.n.n\]](#) and [\[Table n.n\]](#) refer to sections and tables in the OpenGL 4.6 core specification.
- [\[n.n.n\]](#) refers to sections in the OpenGL Shading Language 4.60.1 specification.

Command Execution [2.3]

OpenGL Errors [2.3.1]

enum [GetError](#)(void);

Graphics Reset Recovery [2.3.2]

enum [GetGraphicsResetStatus](#)(void);

Returns: NO_ERROR, GUILTY_CONTEXT_RESET, INNOCENT_UNKNOWN_CONTEXT_RESET

[GetInterverv](#)(RESET_NOTIFICATION_STRATEGY);

Returns: NO_RESET_NOTIFICATION, LOSE_CONTEXT_ON_RESET

Flush and Finish [2.3.3]

void [Flush](#)(void); void [Finish](#)(void);

Floating-Point Numbers [2.3.4]

16-Bit	1-bit sign, 5-bit exp., 10-bit mant.
Unsigned 11-Bit	no sign bit, 5-bit exp., 6-bit mant.
Unsigned 10-Bit	no sign bit, 5-bit exp., 5-bit mant.

Command Letters [Tables 2.1, 2.2]

Where a letter denotes a type in a function name, T within the prototype is the same type.

b - byte (8 bits)	ub - ubyte (8 bits)
s - short (16 bits)	us - ushort (16 bits)
i - int (32 bits)	ui - uint (32 bits)
i64 - int64 (64 bits)	ui64 - uint64 (64 bits)
f - float (32 bits)	d - double (64 bits)

Timer Queries [4.3]

Timer queries track the amount of time needed to fully complete a set of GL commands.

void [QueryCounter](#)(uint id, [TIMESTAMP](#));
void [GetInterverv](#)([TIMESTAMP](#), int *data);
void [GetInteger64v](#)([TIMESTAMP](#), int64 *data);

Synchronization

Sync Objects and Fences [4.1]

void [DeleteSync](#)(sync sync);

sync [FenceSync](#)(enum condition, bitfield flags);

condition: SYNC_GPU_COMMANDS_COMPLETE
flags: must be 0

Waiting for Sync Objects [4.1.1]

enum [ClientWaitSync](#)(sync sync, bitfield flags, uint64 timeout_ns);
flags: SYNC_FLUSH_COMMANDS_BIT, or zero

void [WaitSync](#)(sync sync, bitfield flags, uint64 timeout);

timeout: TIMEOUT_IGNORED

Sync Object Queries [4.1.3]

void [GetSynciv](#)(sync sync, enum pname, sizei bufSize, sizei *length, int *values);
pname: OBJECT_TYPE, SYNC_STATUS, CONDITION, FLAGS

boolean [IsSync](#)(sync sync);

Buffer Objects [6]

void [GenBuffers](#)(sizei n, uint *buffers);
void [CreateBuffers](#)(sizei n, uint *buffers);
void [DeleteBuffers](#)(sizei n, const uint *buffers);

Create and Bind Buffer Objects [6.1]

void [BindBuffer](#)(enum target, uint buffer);
target: [Table 6.1] [ARRAY, UNIFORM, BUFFER, ATOMIC_COUNTER, QUERY, BUFFER, COPY_READ, WRITE, BUFFER, DISPATCH, DRAW, INDIRECT, BUFFER, ELEMENT_ARRAY, TEXTURE, BUFFER, PIXEL_UNPACK, BUFFER, PARAMETER, SHADER_STORAGE, BUFFER, TRANSFORM_FEEDBACK, BUFFER]

void [BindBufferRange](#)(enum target, uint index, uint buffer, intptr offset, sizeptr size);

target: ATOMIC_COUNTER_BUFFER, SHADER_STORAGE, UNIFORM_BUFFER, TRANSFORM_FEEDBACK_BUFFER

void [BindBufferBase](#)(enum target, uint index, uint buffer);

target: See [BindBufferRange](#)

void [BindBuffersRange](#)(enum target, uint first, sizei count, const uint *buffers, const intptr *offsets, const sizeptr *size);
target: See [BindBufferRange](#)

void [BindBuffersBase](#)(enum target, uint first, sizei count, const uint *buffers);
target: See [BindBufferRange](#)

Create/Modify Buffer Object Data [6.2]

void [BufferStorage](#)(enum target, sizeptr size, const void *data, bitfield flags);

target: See [BindBuffer](#)
flags: Bitwise OR of MAP_READ_BIT, [DYNAMIC, CLIENT]_STORAGE_BIT, MAP_COHERENT, PERSISTENT_BIT

void [BufferData](#)(enum target, sizeptr size, const void *data, enum usage);

target: See [BindBuffer](#)
usage: DYNAMIC, [DRAW, READ, COPY], [STATIC, STREAM]_DRAW, READ, COPY

void [NamedBufferData](#)(uint buffer, sizeptr size, const void *data, enum usage);

void [BufferSubData](#)(enum target, intptr offset, sizeptr size, const void *data);
target: See [BindBuffer](#)

void [NamedBufferSubData](#)(uint buffer, intptr offset, sizeptr size, const void *data);

void [ClearBufferSubData](#)(enum target, enum internalFormat, intptr offset, sizeptr size, enum format, enum type, const void *data);
target: See [BindBuffer](#)
internalFormat: See [TexBuffer](#) on pg. 3 of this card
format: RED, GREEN, BLUE, RG, RGB, RGBA, BGR, BGRA, [RED, GREEN, BLUE, RG, RGB]_INTEGER, [RGBA, BGR, BGRA]_INTEGER, STENCIL_INDEX, DEPTH, [COMPONENT, STENCIL]

void [ClearNamedBufferSubData](#)(uint buffer, enum internalFormat, intptr offset, sizeptr size, enum format, enum type, const void *data);
internalFormat, format, type: See [ClearBufferSubData](#)

void [ClearBufferData](#)(enum target, enum internalFormat, enum format, enum type, const void *data);
target, internalFormat, format: See [ClearBufferSubData](#)

void [ClearNamedBufferData](#)(uint buffer, enum internalFormat, enum format, enum type, const void *data);
internalFormat, format, type: See [ClearBufferData](#)

Map/Unmap Buffer Data [6.3]

void [*MapBufferRange](#)(enum target, intptr offset, sizeptr length, bitfield access);
target: See [BindBuffer](#)
access: The Bitwise OR of MAP_X_BIT, where X may be READ, WRITE, PERSISTENT, COHERENT, INVALIDATE, [BUFFER, RANGE], FLUSH_EXPLICIT, UNSYNCHRONIZED

void [*MapNamedBufferRange](#)(uint buffer, intptr offset, sizeptr length, bitfield access);
target: See [BindBuffer](#)
access: See [MapBufferRange](#)

OpenGL Command Syntax [2.2]

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

```
return-type Name{1234}{b s i i64 f d ub us ui ui64}{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. If "v" is present, an array of N items is 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](#)

Asynchronous Queries [4.2, 4.2.1]

void [EndQueryIndexed](#)(enum target, uint index);

void [CreateQueries](#)(sizei n, uint *ids);

target: See [BeginQuery](#), plus [TIMESTAMP](#)

void [DeleteQueries](#)(sizei n, const uint *ids);

void [BeginQuery](#)(enum target, uint id);

target: ANY_SAMPLES_PASSED_CONSERVATIVE, PRIMITIVES_GENERATED, SAMPLES_PASSED, TIME_ELAPSED, [PRIMITIVES, VERTICES]_SUBMITTED, TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN, TRANSFORM_FEEDBACK_STREAM_OVERFLOW, [COMPUTE, VERTEX]_SHADER_INVOCATIONS, [FRAGMENT, GEOMETRY]_SHADER_INVOCATIONS, TESS_EVALUATION_SHADER_INVOCATIONS, TESS_CONTROL_SHADER_PATCHES, GEOMETRY_SHADER_PRIMITIVES_EMITTED, CLIPPING_INPUT_OUTPUT_PRIMITIVES

void [BeginQueryIndexed](#)(enum target, uint index, uint id);

target: See [BeginQuery](#)

void [EndQuery](#)(enum target);

void [EndQueryIndexed](#)(enum target, uint index);

boolean [IsQuery](#)(uint id);

void [GetQueryiv](#)(enum target, enum pname, int *params);

target: See [BeginQuery](#), plus [TIMESTAMP](#)
pname: CURRENT_QUERY, QUERY_COUNTER_BITS

void [GetQueryIndexediv](#)(enum target, uint index, enum pname, int *params);

target: See [BeginQuery](#), plus [TIMESTAMP](#)
pname: CURRENT_QUERY, QUERY_COUNTER_BITS

void [GetQueryObjectiv](#)(uint id, enum pname, int *params);

void [GetQueryObjectuiv](#)(uint id, enum pname, uint *params);

void [GetQueryObjecti64v](#)(uint id, enum pname, int64 *params);

void [GetQueryObjectui64v](#)(uint id, enum pname, uint64 *params);

pname: QUERY_TARGET, QUERY_RESULT[NO_WAIT_AVAILABLE]

void [*MapBuffer](#)(enum target, enum access);
access: See [MapBufferRange](#)

void [*MapNamedBuffer](#)(uint buffer, enum access);

access: See [MapBufferRange](#)

void [FlushMappedBufferRange](#)(intptr offset, sizeptr length);

void [FlushMappedNamedBufferRange](#)(uint buffer, intptr offset, sizeptr length);

boolean [UnmapBuffer](#)(enum target);

boolean [UnmapNamedBuffer](#)(uint buffer);

Invalidate Buffer Data [6.5]

void [InvalidateBufferSubData](#)(uint buffer, intptr offset, sizeptr length);

void [InvalidateBufferData](#)(uint buffer);

Buffer Object Queries [6, 6.7]

boolean [IsBuffer](#)(uint buffer);

void [GetBufferSubData](#)(enum target, intptr offset, sizeptr size, void *data);
target: See [BindBuffer](#)

void [GetNamedBufferSubData](#)(uint buffer, intptr offset, sizeptr size, void *data);

void [GetBufferParameteri64v](#)(enum target, enum pname, int64 *data);

target: See [BindBuffer](#)
pname: [Table 6.2] BUFFER_SIZE, BUFFER_USAGE, BUFFER_ACCESS_FLAGS, BUFFER_MAPPED, BUFFER_MAP_OFFSET, LENGTH, BUFFER_IMMUTABLE_STORAGE, ACCESS_FLAGS

void [GetNamedBufferParameteri64v](#)(uint buffer, enum pname, int64 *data);

void [GetBufferPointerv](#)(enum target, enum pname, const void **params);

target: See [BindBuffer](#)
pname: BUFFER_MAP_POINTER

void [GetNamedBufferPointerv](#)(uint buffer, enum pname, const void **params);
pname: BUFFER_MAP_POINTER

Copy Between Buffers [6.6]

void [CopyBufferSubData](#)(enum readTarget, enum writeTarget, intptr readOffset, intptr writeOffset, sizeptr size);
readTarget and writeTarget: See [BindBuffer](#)

void [CopyNamedBufferSubData](#)(uint readBuffer, uint writeBuffer, intptr readOffset, intptr writeOffset, sizeptr size);

Shaders and Programs

Shader Objects [7.1-2]

uint [CreateShader](#)(enum type);

type: [COMPUTE, FRAGMENT]_SHADER, [GEOMETRY, VERTEX]_SHADER, TESS_EVALUATION, CONTROL_SHADER

void [ShaderSource](#)(uint shader, sizei count, const char *const *string, const int *length);

void [CompileShader](#)(uint shader);

void [ReleaseShaderCompiler](#)(void);

void [DeleteShader](#)(uint shader);

boolean [IsShader](#)(uint shader);

void [ShaderBinary](#)(sizei count, const uint *shaders, enum binaryFormat, const void *binary, sizei length);

void [SpecializeShader](#)(uint shader, const char *pEntryPoint, uint numSpecializationConstants, const uint *pConstantIndex, const int *pConstantValue);

Program Objects [7.3]

uint [CreateProgram](#)(void);

void [AttachShader](#)(uint program, uint shader);

(Continued on next page) ►

◀ Shaders and Programs (cont.)

void **DetachShader**(uint *program*,
uint *shader*);
void **LinkProgram**(uint *program*);
void **UseProgram**(uint *program*);
uint **CreateShaderProgramv**(enum *type*,
sizei *count*, const char * *const* * *strings*);

void **ProgramParameteri**(uint *program*,
enum *pname*, int *value*);
pname: PROGRAM_SEPARABLE,
PROGRAM_BINARY_RETRIEVABLE_HINT
value: TRUE, FALSE

void **DeleteProgram**(uint *program*);
boolean **IsProgram**(uint *program*);

Program Interfaces [7.3.1]

void **GetProgramInterfaceiv**(uint *program*,
enum *programInterface*, enum *pname*,
int * *params*);
programInterface:
ATOMIC_COUNTER_BUFFER, BUFFER_VARIABLE,
UNIFORM_BLOCK, PROGRAM_INPUT_OUTPUT,
SHADER_STORAGE_BLOCK,
(GEOMETRY, VERTEX)_SUBROUTINE,
TESS_CONTROL_EVALUATION_SUBROUTINE,
(FRAGMENT, COMPUTE)_SUBROUTINE,
TESS_CONTROL_SUBROUTINE_UNIFORM,
TESS_EVALUATION_SUBROUTINE_UNIFORM,
(GEOMETRY, VERTEX)_SUBROUTINE_UNIFORM,
(FRAGMENT, COMPUTE)_SUBROUTINE_UNIFORM,
TRANSFORM_FEEDBACK_BUFFER_VARYING
pname: ACTIVE_RESOURCES, MAX_NAME_LENGTH,
MAX_NUM_ACTIVE_VARIABLES,
MAX_NUM_COMPATIBLE_SUBROUTINES

uint **GetProgramResourceIndex**(
uint *program*, enum *programInterface*,
const char * *name*);

void **GetProgramResourceName**(
uint *program*, enum *programInterface*,
uint *index*, sizei *bufSize*, sizei * *length*,
char * *name*);

void **GetProgramResourceiv**(uint *program*,
enum *programInterface*, uint *index*,
sizei *propCount*, const enum * *props*,
sizei *bufSize*, sizei * *length*, int * *params*);
**props*: [See Table 7.2]

int **GetProgramResourceLocation**(
uint *program*, enum *programInterface*,
const char * *name*);

int **GetProgramResourceLocationIndex**(
uint *program*, enum *programInterface*,
const char * *name*);

Program Pipeline Objects [7.4]

void **GenProgramPipelines**(sizei *n*,
uint * *pipelines*);

void **DeleteProgramPipelines**(sizei *n*,
const uint * *pipelines*);

boolean **IsProgramPipeline**(uint *pipeline*);

void **BindProgramPipeline**(uint *pipeline*);

void **CreateProgramPipelines**(sizei *n*,
uint * *pipelines*);

void **UseProgramStages**(uint *pipeline*,
bitfield *stages*, uint *program*);
stages: ALL_SHADER_BITS or the bitwise OR of
TESS_CONTROL_EVALUATION_SHADER_BIT,
(VERTEX, GEOMETRY, FRAGMENT)_SHADER_BIT,
COMPUTE_SHADER_BIT

void **ActiveShaderProgram**(uint *pipeline*,
uint *program*);

Program Binaries [7.5]

void **GetProgramBinary**(uint *program*,
sizei *bufSize*, sizei * *length*,
enum * *binaryFormat*, void * *binary*);

void **ProgramBinary**(uint *program*,
enum *binaryFormat*, const void * *binary*,
sizei *length*);

Uniform Variables [7.6]

int **GetUniformLocation**(uint *program*,
const char * *name*);

void **GetActiveUniformName**(uint *program*,
uint *uniformIndex*, sizei *bufSize*,
sizei * *length*, char * *uniformName*);

void **GetUniformIndices**(uint *program*,
sizei *uniformCount*,
const char * *const* * *uniformNames*,
uint * *uniformIndices*);

void **GetActiveUniform**(uint *program*,
uint *index*, sizei *bufSize*, sizei * *length*,
int * *size*, enum * *type*, char * *name*);
**type* returns: DOUBLE_{VECn, MATn, MATmXn},
DOUBLE, FLOAT_{VECn, MATn, MATmXn}, FLOAT,
INT, INT_VECn, UNSIGNED_INT_VECn, BOOL,
BOOL_VECn, or any value in [Table 7.3]

void **GetActiveUniformsiv**(uint *program*,
sizei *uniformCount*,
const uint * *uniformIndices*, enum *pname*,
int * *params*);

pname: [Table 7.6]
UNIFORM_{NAME_LENGTH, TYPE, OFFSET},
UNIFORM_{SIZE, BLOCK_INDEX, UNIFORM},
UNIFORM_{ARRAY, MATRIX}_STRIDE,
UNIFORM_IS_ROW_MAJOR,
UNIFORM_ATOMIC_COUNTER_BUFFER_INDEX

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, DATA_SIZE},
UNIFORM_BLOCK_NAME_LENGTH,
UNIFORM_BLOCK_ACTIVE_UNIFORMS_{INDICES},
UNIFORM_BLOCK_REFERENCED_BY_X_SHADER,
where X may be one of VERTEX, FRAGMENT,
COMPUTE, GEOMETRY, TESS_CONTROL, or
TESS_EVALUATION [Table 7.7]

void **GetActiveAtomicCounterBufferiv**(
uint *program*, uint *bufferIndex*,
enum *pname*, int * *params*);

pname: See **GetActiveUniformBlockiv**, however
replace the prefix UNIFORM_BLOCK_ with
ATOMIC_COUNTER_BUFFER_

Load Uniform Vars. in Default Uniform Block
void **Uniform{1234}{i f d ui}(int *location*,
T *value*);**

void **Uniform{1234}{i f d ui}v(int *location*,
sizei *count*, const T * *value*);**

void **UniformMatrix{234}{f d}v(
int *location*, sizei *count*, boolean *transpose*,
const float * *value*);**

void **UniformMatrix{2x3,3x2,2x4,4x2,3x4,4x3}{fd}v(
int *location*, sizei *count*,
boolean *transpose*, const float * *value*);**

void **ProgramUniform{1234}{i f d}(
uint *program*, int *location*, T *value*);**

void **ProgramUniform{1234}{i f d}v(
uint *program*, int *location*, sizei *count*,
const T * *value*);**

void **ProgramUniform{1234}uiv(
uint *program*, int *location*, sizei *count*,
const T * *value*);**

void **ProgramUniform{1234}ui(
uint *program*, int *location*, T *value*);**

void **ProgramUniformMatrix{234}{f d}v(
uint *program*, int *location*, sizei *count*,
boolean *transpose*, const T * *value*);**

void **ProgramUniformMatrixf{2x3,3x2,2x4,
4x2,3x4,4x3}{f d}v(
uint *program*, int *location*, sizei *count*,
boolean *transpose*, const T * *value*);**

Uniform Buffer Object Bindings

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

Shader Buffer Variables [7.8]

void **ShaderStorageBlockBinding**(
uint *program*, uint *storageBlockIndex*,
uint *storageBlockBinding*);

Subroutine Uniform Variables [7.9]

Parameter *shaderType* for the functions in this
section may be {COMPUTE, VERTEX}_SHADER,
TESS_{CONTROL, EVALUATION}_SHADER, or
{FRAGMENT, GEOMETRY}_SHADER

int **GetSubroutineUniformLocation**(
uint *program*, enum *shaderType*,
const char * *name*);

uint **GetSubroutineIndex**(uint *program*,
enum *shaderType*, const char * *name*);

void **GetActiveSubroutineName**(
uint *program*, enum *shaderType*,
uint *index*, sizei *bufSize*, sizei * *length*,
char * *name*);

void **GetActiveSubroutineUniformName**(
uint *program*, enum *shaderType*,
uint *index*, sizei *bufSize*, sizei * *length*,
char * *name*);

void **GetActiveSubroutineUniformiv**(
uint *program*, enum *shaderType*,
uint *index*, enum *pname*, int * *values*);
pname: {NUM}_COMPATIBLE_SUBROUTINES

void **UniformSubroutinesuiv**(
enum *shaderType*, sizei *count*,
const uint * *indices*);

Shader Memory Access [7.12.2]

See diagram on page 6 for more information.

void **MemoryBarrier**(bitfield *barriers*);
barriers: ALL_BARRIER_BITS or the OR of
X_BARRIER_BIT where X may be: QUERY_BUFFER,
VERTEX_ATTRIB_ARRAY_ELEMENT_ARRAY,
UNIFORM, TEXTURE_FETCH, BUFFER_UPDATE,
SHADER_IMAGE_ACCESS, COMMAND,
PIXEL_BUFFER, TEXTURE_UPDATE, FRAMEBUFFER,
TRANSFORM_FEEDBACK, ATOMIC_COUNTER,
SHADER_STORAGE, CLIENT_MAPPED_BUFFER,

void **MemoryBarrierByRegion**(bitfield
barriers);
barriers: ALL_BARRIER_BITS or the OR of
X_BARRIER_BIT where X may be:
ATOMIC_COUNTER, FRAMEBUFFER,
SHADER_IMAGE_ACCESS, SHADER_STORAGE,
TEXTURE_FETCH, UNIFORM

Shader and Program Queries [7.13]

void **GetShaderiv**(uint *shader*, enum *pname*,
int * *params*);
pname: SHADER_TYPE, INFO_LOG_LENGTH,
{DELETE, COMPILE}_STATUS, COMPUTE_SHADER,
SHADER_SOURCE_LENGTH, SPIR_V_BINARY

void **GetProgramiv**(uint *program*,
enum *pname*, int * *params*);
pname: ACTIVE_ATOMIC_COUNTER_BUFFERS,
ACTIVE_ATTRIBUTES,
ACTIVE_ATTRIBUTE_MAX_LENGTH,
ACTIVE_UNIFORMS, ACTIVE_UNIFORM_BLOCKS,
ACTIVE_UNIFORM_BLOCK_MAX_NAME_LENGTH,
ACTIVE_UNIFORM_MAX_LENGTH,
ATTACHED_SHADERS, VALIDATE_STATUS,
COMPUTE_WORK_GROUP_SIZE, DELETE_STATUS,
GEOMETRY_{INPUT, OUTPUT}_TYPE,
GEOMETRY_SHADER_INVOCATIONS,
GEOMETRY_VERTICES_OUT, INFO_LOG_LENGTH,
LINK_STATUS, PROGRAM_SEPARABLE,
PROGRAM_BINARY_RETRIEVABLE_HINT,
TESS_CONTROL_OUTPUT_VERTICES,
TESS_GEN_{MODE, SPACING},
TESS_GEN_{VERTEX_ORDER, POINT_MODE},
TRANSFORM_FEEDBACK_BUFFER_MODE,
TRANSFORM_FEEDBACK_VARYINGS,
TRANSFORM_FEEDBACK_VARYING_MAX_LENGTH

void **GetProgramPipelineiv**(uint *pipeline*,
enum *pname*, int * *params*);

pname: ACTIVE_PROGRAM, VALIDATE_STATUS,
{VERTEX, FRAGMENT, GEOMETRY}_SHADER,
TESS_{CONTROL, EVALUATION}_SHADER,
INFO_LOG_LENGTH, COMPUTE_SHADER

void **GetAttachedShaders**(uint *program*,
sizei *maxCount*, sizei * *count*,
uint * *shaders*);

void **GetShaderInfoLog**(uint *shader*,
sizei *bufSize*, sizei * *length*, char * *infoLog*);

void **GetProgramInfoLog**(uint *program*,
sizei *bufSize*, sizei * *length*, char * *infoLog*);

void **GetProgramPipelineInfoLog**(
uint *pipeline*, sizei *bufSize*,
sizei * *length*, char * *infoLog*);

void **GetShaderSource**(uint *shader*,
sizei *bufSize*, sizei * *length*, char * *source*);

void **GetShaderPrecisionFormat**(
enum *shaderType*, enum *precisionType*,
int * *range*, int * *precision*);
shaderType: {VERTEX, FRAGMENT}_SHADER
precisionType: {LOW, MEDIUM, HIGH}_{FLOAT, INT}

void **GetUniform{f d i ui}v**(uint *program*,
int *location*, T * *params*);

void **GetnUniform{f d i ui}v**(uint *program*,
int *location*, sizei *bufSize*, T * *params*);

void **GetUniformSubroutineiv**(
enum *shaderType*, int *location*,
uint * *params*);

void **GetProgramStageiv**(uint *program*,
enum *shaderType*, enum *pname*,
int * *values*);
pname: ACTIVE_SUBROUTINES,
ACTIVE_SUBROUTINE_X where X may be
UNIFORMS, MAX_LENGTH, UNIFORM_LOCATIONS,
UNIFORM_MAX_LENGTH

Textures and Samplers [8]

void **ActiveTexture**(enum *texture*);
texture: TEXTUREi (where i is
[0, max{MAX_TEXTURE_COORDS,
MAX_COMBINED_TEXTURE_IMAGE_UNITS}-1])

Texture Objects [8.1]

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

void **BindTexture**(enum *target*, uint *texture*);
target: TEXTURE_{1D, 2D}_{ARRAY},
TEXTURE_{3D, RECTANGLE, BUFFER},
TEXTURE_CUBE_MAP_{ARRAY},
TEXTURE_2D_MULTISAMPLE_{ARRAY}

void **BindTextures**(uint *first*, sizei *count*,
const uint * *textures*);
target: See **BindTexture**

void **BindTextureUnit**(uint *unit*, uint *texture*);

void **CreateTextures**(enum *target*, sizei *n*,
uint * *textures*);
target: See **BindTexture**

void **DeleteTextures**(sizei *n*,
const uint * *textures*);
boolean **IsTexture**(uint *texture*);

Sampler Objects [8.2]

void **GenSamplers**(sizei *count*, uint * *samplers*);

void **CreateSamplers**(sizei *n*, uint * *samplers*);

void **BindSampler**(uint *unit*, uint *sampler*);

void **BindSamplers**(uint *first*, sizei *count*,
const uint * *samplers*);

void **SamplerParameteri**(i f}(uint *sampler*,
enum *pname*, T *param*);

void **SamplerParameterf**(i f}v}(uint *sampler*,
enum *pname*, const T * *param*);

void **SamplerParameteri**(i ui}v}(uint *sampler*,
enum *pname*, const T * *params*);
pname: for all **SamplerParameter*** functions:
TEXTURE_X where X may be WRAP_{S, T, R},
{MIN, MAG}_FILTER, {MIN, MAX}_LOD,
BORDER_COLOR, LOD_BIAS, MAX_ANISOTROPY,
COMPARE_{MODE, FUNC} [Table 23.18]

void **DeleteSamplers**(sizei *count*,
const uint * *samplers*);

boolean **IsSampler**(uint *sampler*);

Sampler Queries [8.3]

void **GetSamplerParameteri**(i f}v}(
uint *sampler*, enum *pname*, T * *params*);
pname: See **SamplerParameteri**{f}

void **GetSamplerParameteri**(i ui}v}(
uint *sampler*, enum *pname*, T * *params*);
pname: See **SamplerParameteri**{f}

Pixel Storage Modes [8.4.1]

void **PixelStorei**(i f}(enum *pname*, T *param*);
pname: [Tables 8.1, 18.1] [UN]PACK_X where X may
be SWAP_BYTES, LSB_FIRST, ROW_LENGTH,
SKIP_{IMAGES, PIXELS, ROWS}, ALIGNMENT,
IMAGE_HEIGHT, COMPRESSED_BLOCK_WIDTH,
COMPRESSED_BLOCK_{HEIGHT, DEPTH, SIZE}

(Continued on next page) ▶

◀ Textures and Samplers (cont.)

Texture Image Spec. [8.5]

void TexImage3D(enum target, int level, int internalformat, sizei width, sizei height, sizei depth, int border, enum format, enum type, const void *data);

target: [PROXY_TEXTURE_CUBE_MAP_ARRAY, [PROXY_TEXTURE_2D_ARRAY, [PROXY_TEXTURE_3D
internalformat: STENCIL_INDEX, RED, DEPTH, [COMPONENT, STENCIL], RG, RGB, RGBA, COMPRESSED_{RED, RG, RGB, RGBA, SRGB, SRGB_ALPHA}, a sized internal format from [Tables 8.12 - 8.13], or a COMPRESSED_ format from [Table 8.14]

format: DEPTH_{COMPONENT, STENCIL}, RED, GREEN, BLUE, RG, RGB, RGBA, BGR, BGRA, [BGRA, RED, GREEN, BLUE]_INTEGER, [RG, RGB, RGBA, BGR]_INTEGER, STENCIL_INDEX, [Table 8.3]

type: [UNSIGNED_]BYTE, SHORT, INT, [HALF_]FLOAT, or a value from [Table 8.2]

void TexImage2D(enum target, int level, int internalformat, sizei width, sizei height, int border, enum format, enum type, const void *data);

target: [PROXY_TEXTURE_{2D, RECTANGLE}, [PROXY_TEXTURE_{1D_ARRAY, CUBE_MAP}], TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}

internalformat, format, type: See [TexImage3D](#)

void TexImage1D(enum target, int level, int internalformat, sizei width, int border, enum format, enum type, const void *data);

target: TEXTURE_1D, PROXY_TEXTURE_1D
type, internalformat, format: See [TexImage3D](#)

Alternate Texture Image Spec. [8.6]

void CopyTexImage2D(enum target, int level, enum internalformat, int x, int y, sizei width, sizei height, int border);

target: TEXTURE_{2D, RECTANGLE, 1D_ARRAY}, TEXTURE_CUBE_MAP_{POSITIVE, NEGATIVE}_{X, Y, Z}
internalformat: See [TexImage3D](#)

void CopyTexImage1D(enum target, int level, enum internalformat, int x, int y, sizei width, int border);

target: TEXTURE_1D
internalformat: See [TexImage3D](#)

void TexSubImage3D(enum target, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, enum type, const void *data);

target: TEXTURE_3D, TEXTURE_2D_ARRAY, TEXTURE_CUBE_MAP_ARRAY
format, type: See [TexImage3D](#)

void TexSubImage2D(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, enum type, const void *data);

target: See [CopyTexImage2D](#)
format, type: See [TexImage3D](#)

void TexSubImage1D(enum target, int level, int xoffset, sizei width, enum format, enum type, const void *data);

target, format, type: See [CopyTexImage1D](#)

void CopyTexSubImage3D(enum target, int level, int xoffset, int yoffset, int zoffset, int x, int y, sizei width, sizei height);

target: See [TexSubImage3D](#)

void CopyTexSubImage2D(enum target, int level, int xoffset, int yoffset, int x, int y, sizei width, sizei height);

target: See [TexImage2D](#)

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

target: See [TexSubImage1D](#)

void TextureSubImage3D(uint texture, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, enum type, const void *pixels);

format, type: See [TexImage3D](#)

void TextureSubImage2D(uint texture, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, enum type, const void *pixels);

format, type: See [TexImage3D](#)

void TextureSubImage1D(uint texture, int level, int xoffset, sizei width, enum format, enum type, const void *pixels);
format, type: See [TexImage3D](#)

void CopyTextureSubImage3D(uint texture, int level, int xoffset, int yoffset, int zoffset, int x, int y, sizei width, sizei height);

void CopyTextureSubImage2D(uint texture, int level, int xoffset, int yoffset, int x, int y, sizei width, sizei height);

void CopyTextureSubImage1D(uint texture, int level, int xoffset, int x, int y, sizei width);

Compressed Texture Images [8.7]

void CompressedTexImage3D(enum target, int level, enum internalformat, sizei width, sizei height, sizei depth, int border, sizei imageSize, const void *data);

target: See [TexImage3D](#)

internalformat: A COMPRESSED_ format from [Table 8.14]

void CompressedTexImage2D(enum target, int level, enum internalformat, sizei width, sizei height, int border, sizei imageSize, const void *data);

target: See [TexImage2D](#)

internalformat: May be one of the COMPRESSED_ formats from [Table 8.14]

void CompressedTexImage1D(enum target, int level, enum internalformat, sizei width, int border, sizei imageSize, const void *data);

target: TEXTURE_1D, PROXY_TEXTURE_1D

internalformat: See [TexImage1D](#), omitting compressed rectangular texture formats

void CompressedTexSubImage3D(enum target, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, sizei imageSize, const void *data);

target: See [TexSubImage3D](#)
format: See [internalformat for CompressedTexImage3D](#)

void CompressedTexSubImage2D(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, sizei imageSize, const void *data);

target: See [TexSubImage2D](#)
format: See [internalformat for CompressedTexImage2D](#)

void CompressedTexSubImage1D(enum target, int level, int xoffset, sizei width, enum format, sizei imageSize, const void *data);

target: See [TexSubImage1D](#)
format: See [internalformat for CompressedTexImage1D](#)

void CompressedTextureSubImage3D(uint texture, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, sizei imageSize, const void *data);

format: See [internalformat for CompressedTextureImage3D](#)

void CompressedTextureSubImage2D(uint texture, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, sizei imageSize, const void *data);

format: See [internalformat for CompressedTextureImage2D](#)

void CompressedTextureSubImage1D(uint texture, int level, int xoffset, sizei width, enum format, sizei imageSize, const void *data);

format: See [internalformat for CompressedTextureImage1D](#)

Multisample Textures [8.8]

void TexImage3DMultisample(enum target, sizei samples, int internalformat, sizei width, sizei height, sizei depth, boolean fixedsamplelocations);

target: [PROXY_TEXTURE_2D_MULTISAMPLE_ARRAY
internalformat: RED, RG, RGB, RGBA, RGBA{32, 32UI}, DEPTH_COMPONENT{16, 24, 32, 32F}, DEPTH{24, 32F}, STENCIL8, STENCIL_INDEX{1, 4, 8, 16}

void TexImage2DMultisample(enum target, sizei samples, int internalformat, sizei width, sizei height, boolean fixedsamplelocations);

target: [PROXY_TEXTURE_2D_MULTISAMPLE
internalformat: See [TexImage3DMultisample](#)

Buffer Textures [8.9]

void TexBufferRange(enum target, enum internalFormat, uint buffer, intptr offset, sizei size);

void TextureBufferRange(uint texture, enum internalFormat, uint buffer, intptr offset, sizei size);
internalFormat: See [TexBuffer](#)

void TexBuffer(enum target, enum internalFormat, uint buffer);

target: TEXTURE_BUFFER

internalFormat: [Table 8.16] R8, R8{I, UI}, R16, R16{F, I, UI}, R32{F, I, UI}, RG8, RG8{I, UI}, RG16, RG16{F, I, UI}, RG32{F, I, UI}, RGB32F, RGB32{I, UI}, RGBA8, RGBA8{I, UI}, RGBA16, RGBA16{F, I, UI}, RGBA32{F, I, UI}

void TextureBuffer(uint texture, enum internalFormat, uint buffer);
internalFormat: See [TexBuffer](#)

Texture Parameters [8.10]

void TexParameteri(i f)(enum target, enum pname, T param);

target: See [BindTexture](#)

void TexParameterf(i f)(enum target, enum pname, const T *params);

target: See [BindTexture](#)

void TexParameterI(i ui)(enum target, enum pname, const T *params);

target: See [BindTexture](#)

pname for all TexParameter* functions: DEPTH_STENCIL_TEXTURE_MODE or TEXTURE_X where X may be one of WRAP_{S, T, R}, BORDER_COLOR, {MIN, MAG}_FILTER, LOD_BIAS_{MIN, MAX}_LOD, {BASE, MAX}_LEVEL, SWIZZLE_{R, G, B, A, RGBA}, COMPARE_{MODE, FUNC} [Table 8.17]

void TextureParameteri(i f)(uint texture, enum pname, T param);

pname: See [BindTexture](#)

void TextureParameterf(i f)(uint texture, enum pname, const T *params);

pname: See [BindTexture](#)

void TextureParameterI(i ui)(uint texture, enum pname, const T *params);

pname for all TextureParameter* functions: TEXTURE_{3D, RECTANGLE, MAX_ANISOTROPY}, TEXTURE_{1D, 2D, CUBE_MAP}_{ARRAY, TEXTURE_2D_MULTISAMPLE}_{ARRAY}

Texture Queries [8.11]

void GetTexParameterI(i f)(enum target, enum pname, T *params);

target: See [BindTexture](#)

pname: See [GetTexParameterI\(i ui\)](#)

void GetTexParameterI(i ui)(enum target, enum pname, T *params);

target: See [BindTexture](#)

pname: IMAGE_FORMAT_COMPATIBILITY_TYPE, TEXTURE_IMMUTABLE_{FORMAT, LEVELS}, TEXTURE_VIEW_MIN_{LEVEL, LAYER}, TEXTURE_VIEW_NUM_{LEVELS, LAYERS}, DEPTH_STENCIL_TEXTURE_MODE, or TEXTURE_X where X may be one of WRAP_{S, T, R}, BORDER_COLOR, TARGET, {MIN, MAG}_FILTER, LOD_BIAS_{MIN, MAX}_LOD, {BASE, MAX}_LEVEL, SWIZZLE_{R, G, B, A, RGBA}, COMPARE_{MODE, FUNC} [Table 8.17]

void GetTextureParameterI(i f)(uint texture, enum pname, T *data);

pname: See [GetTexParameterI\(i ui\)](#)

void GetTextureParameterf(i f)(uint texture, enum pname, T *data);

pname: See [GetTexParameterI\(i ui\)](#)

void GetTexLevelParameterI(i f)(enum target, int level, enum pname, T *params);

target: [PROXY_TEXTURE_{1D, 2D, 3D}, TEXTURE_BUFFER, PROXY_TEXTURE_CUBE_MAP, [PROXY_TEXTURE_{1D, 2D, CUBE_MAP}_ARRAY, [PROXY_TEXTURE_RECTANGLE, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, [PROXY_TEXTURE_2D_MULTISAMPLE}_{ARRAY}

pname: TEXTURE_*, where * may be WIDTH, HEIGHT, DEPTH, FIXED_SAMPLE_LOCATIONS, INTERNAL_FORMAT, SHARED_SIZE, COMPRESSED, COMPRESSED_IMAGE_SIZE, SAMPLES, BUFFER_{OFFSET, SIZE}, or X_{SIZE, TYPE} where X can be RED, GREEN, BLUE, ALPHA, DEPTH

void GetTextureLevelParameterf(i f)(uint texture, int level, enum pname, T *params);

pname: See [GetTexLevelParameterI\(i f\)](#)

void GetTexImage(enum target, int level, enum format, enum type, void *pixels);

target: TEXTURE_{1, 2D}_{ARRAY, TEXTURE_{3D, RECTANGLE, CUBE_MAP_ARRAY}, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}

format: See [TexImage3D](#)

type: [UNSIGNED_]BYTE, SHORT, INT, [HALF_]FLOAT, or a value from [Table 8.2]

void GetTextureImage(uint texture, int level, enum format, enum type, sizei bufSize, void *pixels);

level: LOD level

format, type: See [GetTexImage](#)

void GetnTexImage(enum tex, int level, enum format, enum type, sizei bufSize, void *pixels);

tex: TEXTURE_{1D, 2D, 3D}_{ARRAY, TEXTURE_3D, TEXTURE_CUBE_MAP_ARRAY, RECTANGLE}, TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}

level, format, type: See [GetTextureImage](#)

void GetTextureSubImage(uint texture, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, enum type, sizei bufSize, void *pixels);

level, format, type: See [GetTextureImage](#)

void GetCompressedTexImage(enum target, int level, void *pixels);

target: See [GetTextureImage](#)

void GetCompressedTextureImage(uint texture, int level, sizei bufSize, void *pixels);

level: See [GetTextureImage](#)

void GetnCompressedTexImage(enum target, int level, sizei bufSize, void *pixels);

target: See [GetCompressedTexImage](#)

level: LOD level

void GetCompressedTextureSubImage(uint texture, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, sizei bufSize, void *pixels);

level: LOD level

Cube Map Texture Select [8.13.1]

Enable/Disable/IsEnabled(TEXTURE_CUBE_MAP_SEAMLESS);

Manual Mipmap Generation [8.14.4]

void GenerateMipmap(enum target);

target: TEXTURE_{1D, 2D, 3D}, TEXTURE_{1D, 2D}_ARRAY, TEXTURE_CUBE_MAP_ARRAY

void GenerateTextureMipmap(uint texture);

Texture Views [8.18]

void TextureView(uint texture, enum target, uint origtexture, enum internalformat, uint minlevel, uint numlevels, uint minlayer, uint numlayers);

target: TEXTURE_{1D, 2D, CUBE_MAP}_{ARRAY, TEXTURE_3D, TEXTURE_RECTANGLE, TEXTURE_2D_MULTISAMPLE}_{ARRAY}

internalformat:

R8, R8{UI, I}, R8_SNORM, R11F_G11F_B10F, R16{F, UI, I}, R16F_SNORM, R32{F, UI, I}, SRGB8{UI, I}, RG8{F, UI, I}, RG8F_SNORM, RG16{F, UI, I}, RG16F_SNORM, RG32{F, UI, I}, RG8B8_SNORM, RGB9_E5, RGB10_A2{UI, I}, RGBA8{UI, I}, RGBA8F_SNORM, RGB16{F, UI, I}, RGB16F_SNORM, RGB32{F, UI, I}, RGBA16{F, UI, I}, RGBA16F_SNORM, RGBA32{F, UI, I}, SRGB8_ALPHA8, COMPRESSED_X where X may be [SIGNED]_RED, [SIGNED]_RG, [SIGNED]_RGB, [SIGNED]_RGBA, [SIGNED]_BPTC_UNORM, RGB_BPTC_{UN}SIGNED_FLOAT

Immutable-Format Tex. Images [8.19]

void TexStorage1D(enum target, sizei levels, enum internalformat, sizei width);

target: TEXTURE_1D

internalformat: any of the sized internal color, depth, and stencil formats in [Tables 8.18-20]

(Continued on next page) ▶

◀ Textures and Samplers (cont.)

void **TexStorage2D**(enum *target*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*);

target: TEXTURE_{RECTANGLE, CUBE_MAP}, TEXTURE_{1D_ARRAY, 2D}
internalformat: [See TexStorage1D](#)

void **TexStorage3D**(enum *target*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*);

target: TEXTURE_3D, TEXTURE_{CUBE_MAP, 2D}_{ARRAY}
internalformat: [See TexStorage1D](#)

void **TextureStorage1D**(uint *texture*, sizei *levels*, enum *internalformat*, sizei *width*);

internalformat: [See TexStorage1D](#)

void **TextureStorage2D**(uint *texture*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*);

internalformat: [See TexStorage1D](#)

Framebuffer Objects**Binding and Managing [9.2]**

void **BindFramebuffer**(enum *target*, uint *framebuffer*);

target: [DRAW_, READ_]FRAMEBUFFER

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

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

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

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

Framebuffer Object Parameters [9.2.1]

void **FramebufferParameteri**(enum *target*, enum *pname*, int *param*);

target: [DRAW_, READ_]FRAMEBUFFER
pname: FRAMEBUFFER_DEFAULT_X where X may be WIDTH, HEIGHT, FIXED_SAMPLE_LOCATIONS, SAMPLES, LAYERS

void **NamedFramebufferParameteri**(uint *framebuffer*, enum *pname*, int *param*);

pname: [See FramebufferParameteri](#)

Framebuffer Object Queries [9.2.3]

void **GetFramebufferParameteriv**(enum *target*, enum *pname*, int **params*);

target: [See FramebufferParameteri](#)

pname: [See FramebufferParameteri](#) plus DOUBLEBUFFER, SAMPLES, SAMPLE_BUFFERS, IMPLEMENTATION_COLOR_READ_FORMAT, IMPLEMENTATION_COLOR_READ_TYPE, STEREO

void **GetNamedFramebufferParameteriv**(uint *framebuffer*, enum *pname*, int **params*);

pname: [See GetFramebufferParameteri](#)

void **GetFramebufferAttachmentParameteriv**(enum *target*, enum *attachment*, enum *pname*, int **params*);

target: [DRAW_, READ_]FRAMEBUFFER

Vertices**Separate Patches [10.1.15]**

void **PatchParameteri**(enum *pname*, int *value*);

pname: PATCH_VERTICES

Current Vertex Attribute Values [10.2]

Use the commands **VertexAttrib*** for attributes of type float, **VertexAttribI*** for int or uint, or **VertexAttribL*** for double.

Vertex Arrays**Vertex Array Objects [10.3.1]**

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

void **GenVertexArrays**(sizei *n*, uint **arrays*);

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

void **BindVertexArray**(uint *array*);

void **CreateVertexArrays**(sizei *n*, uint **arrays*);

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

void **VertexArrayElementBuffer**(uint *vaobj*, uint *buffer*);

void **VertexAttribFormat**(uint *attribindex*, int *size*, enum *type*, boolean *normalized*, unit *relativeoffset*);

type: [UNSIGNED_BYTE, [UNSIGNED_SHORT, [UNSIGNED_INT, [HALF_FLOAT, DOUBLE, FIXED, [UNSIGNED_INT_2_10_10_10_REV, [UNSIGNED_INT_10F_11F_11F_REV

void **VertexAttribIFormat**(uint *attribindex*, int *size*, enum *type*, unit *relativeoffset*);

type: [UNSIGNED_BYTE, [UNSIGNED_SHORT, [UNSIGNED_INT

void **VertexAttribLFormat**(uint *attribindex*, int *size*, enum *type*, unit *relativeoffset*);

type: DOUBLE

void **VertexAttrib{1234}{s f d}{i ui}(uint *index*, T *values*);**

void **VertexAttrib{123}{s f d}v**(uint *index*, const T **values*);

void **VertexAttrib4{b s i f d}ub us ui**v(uint *index*, const T **values*);

void **VertexAttrib4Nub**(uint *index*, ubyte *x*, ubyte *y*, ubyte *z*, ubyte *w*);

void **VertexAttrib4N{b s i ub us ui}**v(uint *index*, const T **values*);

void **TextureStorage3DMultisample**(uint *texture*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, boolean *fixedsamplelocations*);

Invalidate Texture Image Data [8.20]

void **InvalidateTexSubImage**(uint *texture*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*);

void **InvalidateTexImage**(uint *texture*, int *level*);

Clear Texture Image Data [8.21]

void **ClearTexSubImage**(uint *texture*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, enum *type*, const void **data*);

format, *type*: [See TexImage3D, pg 2 this card](#)

void **ClearTexImage**(uint *texture*, int *level*, enum *format*, enum *type*, const void **data*);

format, *type*: [See TexImage3D, pg 2 this card](#)

Texture Image Loads/Stores [8.26]

void **BindImageTexture**(uint *index*, uint *texture*, int *level*, boolean *layered*, int *layer*, enum *access*, enum *format*);

access: READ_ONLY, WRITE_ONLY, READ_WRITE
format: RGBA(32,16)F, RG(32,16)F, R11F_G11F_B10F, R(32,16)F, RGBA(32,16,8)UI, RGB10_A2UI, RG(32,16,8)UI, R(32,16,8)UI, RGBA(32,16,8)I, RG(32,16,8)I, R(32,16,8)I, RGBA(16,8), RGB10_A2, RG(16,8), R(16,8), RGBA(16,8)_SNORM, RG(16,8)_SNORM, R(16,8)_SNORM [Table 8.26]

void **BindImageTextures**(uint *first*, sizei *count*, const uint **textures*);

void **NamedRenderbufferStorage**(uint *renderbuffer*, enum *internalformat*, sizei *width*, sizei *height*);

internalformat: [See TexImage3DMultisample](#)

Renderbuffer Object Queries [9.2.6]

void **GetRenderbufferParameteriv**(enum *target*, enum *pname*, int **params*);

target: RENDERBUFFER

pname: [Table 23.27]

RENDERBUFFER_X where X may be WIDTH, HEIGHT, INTERNAL_FORMAT, SAMPLES, {RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL}_SIZE

void **GetNamedRenderbufferParameteriv**(uint *renderbuffer*, enum *pname*, int **params*);

pname: [See GetRenderbufferParameteriv](#)

Attaching Renderbuffer Images [9.2.7]

void **FramebufferRenderbuffer**(enum *target*, enum *attachment*, enum *renderbuffertarget*, uint *renderbuffer*);

target: [DRAW_, READ_]FRAMEBUFFER

attachment: [Table 9.1]

{DEPTH, STENCIL, DEPTH_STENCIL}_ATTACHMENT, COLOR_ATTACHMENT*i* where *i* is [0, MAX_COLOR_ATTACHMENTS - 1]

renderbuffertarget: RENDERBUFFER if *renderbuffer* is non-zero, else undefined

void **NamedFramebufferRenderbuffer**(uint *framebuffer*, enum *attachment*, enum *renderbuffertarget*, uint *renderbuffer*);

attachment, *renderbuffertarget*: [See FramebufferRenderbuffer](#)

Attaching Texture Images [9.2.8]

void **FramebufferTexture**(enum *target*, enum *attachment*, uint *texture*, int *level*);

target: [DRAW_, READ_]FRAMEBUFFER

attachment: [See FramebufferRenderbuffer](#)

void **NamedFramebufferTexture**(uint *framebuffer*, enum *attachment*, uint *texture*, int *level*);

attachment: [See FramebufferRenderbuffer](#)

void **FramebufferTexture1D**(enum *target*, enum *attachment*, enum *textarget*, uint *texture*, int *level*);

textarget: TEXTURE_1D
target, *attachment*: [See FramebufferRenderbuffer](#)

void **FramebufferTexture2D**(enum *target*, enum *attachment*, enum *textarget*, uint *texture*, int *level*);

textarget: TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}, TEXTURE_{2D, RECTANGLE, 2D_MULTISAMPLE} (unspecified if *texture* is 0)
target, *attachment*: [See FramebufferRenderbuffer](#)

void **FramebufferTexture3D**(enum *target*, enum *attachment*, enum *textarget*, uint *texture*, int *level*, int *layer*);

textarget: TEXTURE_3D (unspecified if *texture* is 0)
target, *attachment*: [See FramebufferRenderbuffer](#)

void **FramebufferTextureLayer**(enum *target*, enum *attachment*, uint *texture*, int *level*, int *layer*);

target, *attachment*: [See FramebufferRenderbuffer](#)

void **NamedFramebufferTextureLayer**(uint *framebuffer*, enum *attachment*, uint *texture*, int *level*, int *layer*);

attachment: [See FramebufferRenderbuffer](#)

Feedback Loops [9.3.1]

void **TextureBarrier**(void);

Framebuffer Completeness [9.4.2]

enum **CheckFramebufferStatus**(enum *target*);

target: [DRAW_, READ_]FRAMEBUFFER
returns: FRAMEBUFFER_COMPLETE or a constant indicating the violating value

enum **CheckNamedFramebufferStatus**(uint *framebuffer*, enum *target*);

target: [See CheckFramebufferStatus](#)

const T **values*);

void **VertexAttribP{1234}ui**(uint *index*, enum *type*, boolean *normalized*, uint *value*);

void **VertexAttribP{1234}iui**(uint *index*, enum *type*, boolean *normalized*, const uint **value*);

type: [UNSIGNED_INT_2_10_10_10_REV, or UNSIGNED_INT_10F_11F_11F_REV (except for VertexAttribP4ui)]

void **VertexArrayVertexBuffer**(uint *vaobj*, uint *bindingindex*, uint *buffer*, intptr *offset*, sizei *stride*);

void **BindVertexBuffers**(uint *first*, sizei *count*, const uint **buffers*, const intptr **offsets*, const sizei **strides*);

void **VertexArrayVertexBuffers**(uint *vaobj*, uint *first*, sizei *count*, const uint **buffers*, const intptr **offsets*, const sizei **strides*);

void **VertexAttribBinding**(uint *attribindex*, uint *bindingindex*);

(Continued on next page) ▶

◀ Vertex Arrays (cont.)

void **VertexArrayAttribBinding**(uint *vaobj*,
uint *attribindex*, uint *bindingindex*);

void **VertexArrayAttribPointer**(uint *index*, int *size*,
enum *type*, boolean *normalized*,
sizei *stride*, const void **pointer*);
type: See [VertexArrayFormat](#)

void **VertexArrayIPointer**(uint *index*,
int *size*, enum *type*, sizei *stride*,
const void **pointer*);
type: See [VertexArrayFormat](#)
index: [0, MAX_VERTEX_ATTRIBS - 1]

void **VertexArrayLPointer**(uint *index*, int *size*,
enum *type*, sizei *stride*, const void **pointer*);
type: DOUBLE

void **EnableVertexArrayAttrib**(uint *index*);

void **EnableVertexArrayAttrib**(uint *vaobj*,
uint *index*);

void **DisableVertexArrayAttrib**(uint *index*);

void **DisableVertexArrayAttrib**(uint *vaobj*,
uint *index*);

Vertex Attribute Divisors [10.3.4]

void **VertexBindingDivisor**(uint *bindingindex*,
uint *divisor*);

void **VertexArrayBindingDivisor**(uint *vaobj*,
uint *bindingindex*, uint *divisor*);

void **VertexAttribDivisor**(uint *index*,
uint *divisor*);

Primitive Restart [10.3.6]

Enable/Disable/IsEnabled(*target*);
target: PRIMITIVE_RESTART_FIXED_INDEX

void **PrimitiveRestartIndex**(uint *index*);

Drawing Commands [10.4]

For all the functions in this section:

mode: POINTS, PATCHES, LINE_STRIP,
LINE_LOOP, TRIANGLE_STRIP, TRIANGLE_FAN,
LINES, LINES_ADJACENCY, TRIANGLES,
TRIANGLES_ADJACENCY, LINE_STRIP_ADJACENCY,
TRIANGLE_STRIP_ADJACENCY
type: UNSIGNED_BYTE, SHORT, INT

void **DrawArrays**(enum *mode*, int *first*,
sizei *count*);

void **DrawArraysInstancedBaseInstance**(
enum *mode*, int *first*, sizei *count*,
sizei *instancecount*, uint *baseinstance*);

void **DrawArraysInstanced**(enum *mode*,
int *first*, sizei *count*, sizei *instancecount*);

void **DrawArraysIndirect**(enum *mode*,
const void **indirect*);

void **MultiDrawArrays**(enum *mode*,
const int **first*, const sizei **count*,
sizei *drawcount*);

void **MultiDrawArraysIndirect**(enum *mode*,
const void **indirect*, sizei *drawcount*,
sizei *stride*);

void **MultiDrawArraysIndirectCount**(
enum *mode*, const void **indirect*,
intptr *drawcount*, intptr *maxdrawcount*,
sizei *stride*);

void **DrawElements**(enum *mode*, sizei *count*,
enum *type*, const void **indices*);

void **DrawElementsInstancedBaseInstance**(
enum *mode*, sizei *count*, enum *type*,
const void **indices*, sizei *instancecount*,
uint *baseinstance*);

void **DrawElementsInstanced**(enum *mode*,
sizei *count*, enum *type*, const void **indices*,
sizei *instancecount*);

int **GetAttribLocation**(uint *program*,
const char **name*);

Transform Feedback Variables [11.1.2]

void **TransformFeedbackVaryings**(
uint *program*, sizei *count*,
const char *const **varyings*,
enum *bufferMode*);

bufferMode:
INTERLEAVED_ATTRIBS, SEPARATE_ATTRIBS

void **EndTransformFeedback**(void);

void **PauseTransformFeedback**(void);

void **ResumeTransformFeedback**(void);

void **TransformFeedbackBufferRange**(
uint *xfb*, uint *index*, uint *buffer*, intptr *offset*,
sizei *ptr* *size*);

void **TransformFeedbackBufferBase**(
uint *xfb*, uint *index*, uint *buffer*);

Transform Feedback Drawing [13.2.3]

void **DrawTransformFeedback**(
enum *mode*, uint *id*);
mode: See [Drawing Commands \[10.4\]](#) above

void **DrawTransformFeedbackInstanced**(
enum *mode*, uint *id*, sizei *instancecount*);

void **PointParameter**{i f}v(enum *pname*,
const T **params*);

pname: POINT_FADE_THRESHOLD_SIZE,
POINT_SPRITE_COORD_ORIGIN
params: The fade threshold if *pname* is
POINT_FADE_THRESHOLD_SIZE;
{LOWER, UPPER}_LEFT if *pname* is
POINT_SPRITE_COORD_ORIGIN

Enable/Disable/IsEnabled(*target*);
target: PROGRAM_POINT_SIZE

Line Segments [14.5]

Enable/Disable/IsEnabled(*target*);
target: LINE_SMOOTH

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

Polygons [14.6, 14.6.1]

Enable/Disable/IsEnabled(*target*);
target: POLYGON_SMOOTH, CULL_FACE

void **MultiDrawElements**(enum *mode*,
const sizei **count*, enum *type*,
const void *const **indices*,
sizei *drawcount*);

void **DrawRangeElements**(enum *mode*,
uint *start*, uint *end*, sizei *count*,
enum *type*, const void **indices*);

void **DrawElementsBaseVertex**(enum *mode*,
sizei *count*, enum *type*, const void **indices*,
int *basevertex*);

void **DrawRangeElementsBaseVertex**(
enum *mode*, uint *start*, uint *end*,
sizei *count*, enum *type*, const void **indices*,
int *basevertex*);

void **DrawElementsInstancedBaseVertex**(
enum *mode*, sizei *count*, enum *type*,
const void **indices*, sizei *instancecount*,
int *basevertex*);

void **DrawElementsInstancedBase-
VertexBaseInstance**(enum *mode*,
sizei *count*, enum *type*,
const void **indices*, sizei *instancecount*,
int *basevertex*, uint *baseinstance*);

void **DrawElementsIndirect**(enum *mode*,
enum *type*, const void **indirect*);

void **MultiDrawElementsIndirect**(
enum *mode*, enum *type*,
const void **indirect*, sizei *drawcount*,
sizei *stride*);

void **MultiDrawElementsIndirectCount**(
enum *mode*, enum *type*, const void **indirect*,
intptr *drawcount*, sizei *maxdrawcount*,
sizei *stride*);

void **MultiDrawElementsBaseVertex**(
enum *mode*, const sizei **count*,
enum *type*, const void *const **indices*,
sizei *drawcount*, const int **basevertex*);

void **GetTransformFeedbackVarying**(
uint *program*, uint *index*, sizei *bufSize*,
sizei **length*, sizei **size*, enum **type*,
char **name*);

*type returns NONE, FLOAT, FLOAT_VECn,
DOUBLE, DOUBLE_VECn, INT, UNSIGNED_INT,
INT_VECn, UNSIGNED_INT_VECn,
MATnxm, FLOAT_MATnxm, DOUBLE_MATnxm,
FLOAT_MATn, DOUBLE_MATn

void **DrawTransformFeedbackStream**(
enum *mode*, uint *id*, uint *stream*);

void **DrawTransformFeedbackStreamInstanced**(
enum *mode*, uint *id*, uint *stream*,
sizei *instancecount*);

Flatshading [13.4]

void **ProvokingVertex**(enum *provokemode*);
provokemode: {FIRST, LAST}_VERTEX_CONVENTION

Primitive Clipping [13.5]

Enable/Disable/IsEnabled(*target*);
target: DEPTH_CLAMP, CLIP_DISTANCE*i* where
i = [0..MAX_CLIP_DISTANCES - 1]

void **ClipControl**(enum *origin*, enum *depth*);
origin: LOWER_LEFT or UPPER_LEFT
depth: NEGATIVE_ONE_TO_ONE or ZERO_TO_ONE

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

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

Polygon Rast. & Depth Offset [14.6.4-5]

void **PolygonMode**(enum *face*, enum *mode*);
face: FRONT_AND_BACK

mode: POINT, LINE, FILL

void **PolygonOffsetClamp**(float *factor*,
float *units*, float *clamp*);

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

Enable/Disable/IsEnabled(*target*);
target: POLYGON_OFFSET_{POINT, LINE, FILL}

Vertex Array Queries [10.5]

void **GetVertexArrayiv**(uint *vaobj*,
enum *pname*, int **param*);
pname: ELEMENT_ARRAY_BUFFER_BINDING

void **GetVertexArrayIndexdiv**(uint *vaobj*,
uint *index*, enum *pname*, int **param*);
pname: VERTEX_ATTRIB_RELATIVE_OFFSET or
VERTEX_ATTRIB_ARRAY_X where X is one of
ENABLED, SIZE, STRIDE, TYPE, NORMALIZED,
INTEGER, LONG, DIVISOR

void **GetVertexArrayIndexd64iv**(uint *vaobj*,
uint *index*, enum *pname*, int64 **param*);
pname: VERTEX_BINDING_OFFSET

void **GetVertexArrayIdi**{d f}v(uint *index*,
enum *pname*, T **params*);
pname: See [GetVertexArrayIndexdiv](#) plus
VERTEX_ATTRIB_ARRAY_BUFFER_BINDING,
VERTEX_ATTRIB_BINDING,
CURRENT_VERTEX_ATTRIB

void **GetVertexArrayIui**{v}(uint *index*,
enum *pname*, T **params*);
pname: See [GetVertexArrayIdi](#) f i v

void **GetVertexArrayLdv**(uint *index*,
enum *pname*, double **params*);
pname: See [GetVertexArrayIdi](#) d f i v

void **GetVertexArrayPointerv**(uint *index*,
enum *pname*, const void ***pointer*);
pname: VERTEX_ATTRIB_ARRAY_POINTER

Conditional Rendering [10.9]

void **BeginConditionalRender**(uint *id*,
enum *mode*);
mode: QUERY_NO_WAIT, INVERTED,
QUERY_BY_REGION_NO_WAIT, INVERTED

void **EndConditionalRender**(void);

Vertex Attributes [11.1.1]

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

void **BindAttribLocation**(uint *program*,
uint *index*, const char **name*);

void **GetActiveAttrib**(uint *program*,
uint *index*, sizei *bufSize*, sizei **length*,
int **size*, enum **type*, char **name*);

Vertex Post-Processing [13]**Transform Feedback [13.2]**

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

void **DeleteTransformFeedbacks**(sizei *n*,
const uint **ids*);

boolean **IsTransformFeedback**(uint *id*);

void **BindTransformFeedback**(
enum *target*, uint *id*);
target: TRANSFORM_FEEDBACK

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

void **BeginTransformFeedback**(
enum *primitiveMode*);
primitiveMode: TRIANGLES, LINES, POINTS

Rasterization [13.4, 14]

Enable/Disable/IsEnabled(*target*);
target: RASTERIZER_DISCARD

Multisampling [14.3.1]

Use to antialias points, and lines.

Enable/Disable/IsEnabled(*target*);
target: MULTISAMPLE, SAMPLE_SHADING

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

void **MinSampleShading**(float *value*);

Points [14.4]

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

void **PointParameter**{i f}(enum *pname*,
T *param*);
pname, *param*: See [PointParameter](#){i f}v

Shader Execution [11.1.3]

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

void **ValidateProgramPipeline**(uint *pipeline*);

Tessellation Prim. Generation [11.2.2]

void **PatchParameterfv**(enum *pname*,
const float **values*);
pname: PATCH_DEFAULT_INNER_LEVEL,
PATCH_DEFAULT_OUTER_LEVEL

Controlling Viewport [13.6.1]

void **DepthRangeArrayv**(uint *first*,
sizei *count*, const double **v*);

void **DepthRangeIndexd**(uint *index*,
double *n*, double *f*);

void **DepthRange**(double *n*, double *f*);

void **DepthRangef**(float *n*, float *f*);

void **ViewportArrayv**(uint *first*, sizei *count*,
const float **v*);

void **ViewportIndexdf**(uint *index*, float *x*,
float *y*, float *w*, float *h*);

void **ViewportIndexdfv**(uint *index*,
const float **v*);

void **Viewport**(int *x*, int *y*, sizei *w*, sizei *h*);

Fragment Shaders [15.2]

void **BindFragDataLocationIndexed**(
uint *program*, uint *colorNumber*,
uint *index*, const char **name*);

void **BindFragDataLocation**(uint *program*,
uint *colorNumber*, const char **name*);

int **GetFragDataLocation**(uint *program*,
const char **name*);

int **GetFragDataIndex**(uint *program*,
const char **name*);

Compute Shaders [19]

void **DispatchCompute**(uint *num_groups_x*,
uint *num_groups_y*, uint *num_groups_z*);

void **DispatchComputeIndirect**(
intptr *indirect*);

Per-Fragment Operations

Scissor Test [17.3.2]

Enable/Disable/IsEnabled(SCISSOR_TEST);

Enablei/Disablei/IsEnabledi(SCISSOR_TEST, uint index);

void **ScissorArrayv**(uint first, sizei count, const int *v);

void **ScissorIndexed**(uint index, int left, int bottom, sizei width, sizei height);

void **ScissorIndexedv**(uint index, int *v);

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

Multisample Fragment Ops. [17.3.3]

Enable/Disable/IsEnabled(target);

target: SAMPLE_ALPHA_TO_COVERAGE, ONE, SAMPLE_COVERAGE, SAMPLE_MASK

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

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

Stencil Test [17.3.5]

Enable/Disable/IsEnabled(STENCIL_TEST);

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

func: NEVER, ALWAYS, LESS, GREATER, EQUAL, LEQUAL, GEQUAL, NOTEQUAL

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

func: *See StencilFunc*

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, dppass: KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECR_WRAP

Depth Buffer Test [17.3.6]

Enable/Disable/IsEnabled(DEPTH_TEST);

void **DepthFunc**(enum func);

func: *See StencilFunc*

Occlusion Queries [17.3.7]

BeginQuery(enum target, uint id);

EndQuery(enum target);

target: SAMPLES_PASSED, ANY_SAMPLES_PASSED, ANY_SAMPLES_PASSED_CONSERVATIVE

Whole Framebuffer

Selecting Buffers for Writing [17.4.1]

void **DrawBuffer**(enum buf);

buf: [Tables 17.4-5] NONE, {FRONT, BACK}_LEFT, RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, COLOR_ATTACHMENT*i* (*i* = [0, MAX_COLOR_ATTACHMENTS - 1])

void **NamedFramebufferDrawBuffer**(uint framebuffer, enum buf);

buf: *See DrawBuffer*

void **DrawBuffers**(sizei *n*, const enum *bufs);

*bufs: [Tables 17.5-6] {FRONT, BACK}_LEFT, RIGHT, NONE, COLOR_ATTACHMENT*i* (*i* = [0, MAX_COLOR_ATTACHMENTS - 1])

void **NamedFramebufferDrawBuffers**(uint framebuffer, sizei *n*, const enum *bufs);

*bufs: *See DrawBuffers*

Reading and Copying Pixels

Reading Pixels [18.2]

void **ReadBuffer**(enum src);

src: NONE, {FRONT, BACK}_LEFT, RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, COLOR_ATTACHMENT*i* (*i* = [0, MAX_COLOR_ATTACHMENTS - 1])

void **NamedFramebufferReadBuffer**(uint framebuffer, enum src);

src: *See ReadBuffer*

void **ReadPixels**(int x, int y, sizei width, sizei height, enum format, enum type, void *data);

format: STENCIL_INDEX, RED, GREEN, BLUE, RG, RGB, RGBA, BGR, DEPTH_COMPONENT, STENCIL, {RED, GREEN, BLUE, RG, RGB}_INTEGER, {RGBA, BGR, BGRA}_INTEGER, BGRA [Table 8.3]

type: HALF_FLOAT, UNSIGNED_BYTE, [UNSIGNED_SHORT, UNSIGNED_INT, FLOAT_32_UNSIGN_INT_24_8_REV, UNSIGNED_BYTE, SHORT, INT]* values in [Table 8.2]

void **ReadnPixels**(int x, int y, sizei width, sizei height, enum format, enum type, sizei bufSize, void *data);

format, type: *See ReadPixels*

Final Conversion [18.2.8]

void **ClampColor**(enum target, enum clamp);

target: CLAMP_READ_COLOR
clamp: TRUE, FALSE, FIXED_ONLY

Copying Pixels [18.3]

void **BlitFramebuffer**(int srcX0, int srcY0, int srcX1, int srcY1, int dstX0, int dstY0, int dstX1, int dstY1, bitfield mask, enum filter);

Fine Control of Buffer Updates [17.4.2]

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 [17.4.3]

void **Clear**(bitfield buf);

buf: 0 or the OR of {COLOR, DEPTH, STENCIL}_BUFFER_BIT

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

void **ClearDepth**(double d);

void **ClearDepthf**(float d);

void **ClearStencil**(int s);

Debug Output [20]

Enable/Disable/IsEnabled(DEBUG_OUTPUT);

Debug Message Callback [20.2]

void **DebugMessageCallback**(DEBUGPROC callback, const void *userParam);

callback: has the following prototype:

void callback(enum source, enum type, uint id, enum severity, sizei length, const char *message, const void *userParam);

source: DEBUG_SOURCE_X where X may be API, SHADER_COMPILER, WINDOW_SYSTEM, THIRD_PARTY, APPLICATION, OTHER

type: DEBUG_TYPE_X where X may be ERROR, MARKER, OTHER, DEPRECATED_BEHAVIOR, UNDEFINED_BEHAVIOR, PERFORMANCE, PORTABILITY, {PUSH, POP}_GROUP

mask: Bitwise 0 of the bitwise OR of {COLOR, DEPTH, STENCIL}_BUFFER_BIT
filter: LINEAR, NEAREST

void **BlitNamedFramebuffer**(uint readFramebuffer, uint drawFramebuffer, int srcX0, int srcY0, int srcX1, int srcY1, int dstX0, int dstY0, int dstX1, int dstY1, bitfield mask, enum filter);

mask, filter: *See BlitFramebuffer*

void **CopyImageSubData**(uint srcName, enum srcTarget, int srcLevel, int srcX, int srcY, int srcZ, uint dstName, enum dstTarget, int dstLevel, int dstX, int dstY, int dstZ, sizei srcWidth, sizei srcHeight, sizei srcDepth);

srcTarget, dstTarget: *See target for BindTexture* in section [8.1] on this card, plus GL_RENDERTARGET

Blending [17.3.8]

Enable/Disable/IsEnabled(BLEND);

Enablei/Disablei/IsEnabledi(BLEND, uint index);

void **BlendEquation**(enum mode);

void **BlendEquationSeparate**(enum modeRGB, enum modeAlpha);

modeRGB, modeAlpha: MIN, MAX, FUNC_{ADD, SUBTRACT, REVERSE_SUBTRACT}

void **BlendEquationi**(uint buf, enum mode);

void **BlendEquationSeparatei**(uint buf, enum modeRGB, enum modeAlpha);

modeRGB, modeAlpha:

See BlendEquationSeparate

void **BlendFunc**(enum src, enum dst);

src, dst: *See BlendFuncSeparate*

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

srcRGB, dstRGB, srcAlpha, dstAlpha: ZERO, ONE, SRC_ALPHA_SATURATE, {SRC, SRC1, DST, CONSTANT}_{COLOR, ALPHA}, ONE_MINUS_{SRC, SRC1}_{COLOR, ALPHA}, ONE_MINUS_{DST, CONSTANT}_{COLOR, ALPHA}

void **ClearBufferf**(i f ui)v(enum buffer, int drawbuffer, const T *value);

buffer: COLOR, DEPTH, STENCIL

void **ClearNamedFramebufferf**(i f ui)v(uint framebuffer, enum buffer, int drawbuffer, const T *value);

buffer: *See ClearBufferf*(i f ui)v

void **ClearBufferfv**(enum buffer, int drawbuffer, float depth, int stencil);

buffer: DEPTH, STENCIL

void **ClearNamedFramebufferfv**(uint framebuffer, enum buffer, int drawbuffer, float depth, int stencil);

buffer: *See ClearBufferf*

Invalidating Framebuffers [17.4.4]

void **InvalidateSubFramebuffer**(enum target, sizei numAttachments, const enum *attachments, int x, int y, sizei width, sizei height);

target: [DRAW_, READ_]FRAMEBUFFER

severity: DEBUG_SEVERITY_{HIGH, MEDIUM}, DEBUG_SEVERITY_{LOW, NOTIFICATION}

Controlling Debug Messages [20.4]

void **DebugMessageControl**(enum source, enum type, enum severity, sizei count, const uint *ids, boolean enabled);

source, type, severity: *See DebugMessageCallback* (above), plus DONT_CARE

Externally Generated Messages [20.5]

void **DebugMessageInsert**(enum source, enum type, uint id, enum severity, int length, const char *buf);

source: DEBUG_SOURCE_{APPLICATION, THIRD_PARTY}

type, severity: *See DebugMessageCallback*

Debug Groups [20.6]

void **PushDebugGroup**(enum source, uint id, sizei length, const char *message);

source: *See DebugMessageInsert*

void **PopDebugGroup**(void);

State and State Requests

A complete list of symbolic constants for states is shown in the tables in [23].

Simple Queries [22.1]

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

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

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

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

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

void **GetDoublei_v**(enum target, uint index, double *data);

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

void **BlendFunci**(uint buf, enum src, enum dst);

src, dst: *See BlendFuncSeparate*

void **BlendFuncSeparatei**(uint buf, enum srcRGB, enum dstRGB, enum srcAlpha, enum dstAlpha);

dstRGB, dstAlpha, srcRGB, srcAlpha:

See BlendFuncSeparate

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

Dithering [17.3.10]

Enable/Disable/IsEnabled(DITHER);

Logical Operation [17.3.11]

Enable/Disable/IsEnabled(COLOR_LOGIC_OP);

void **LogicOp**(enum op);

op: CLEAR, AND, AND_REVERSE, COPY, AND_INVERTED, NOOP, XOR, OR, NOR, EQUIV, INVERT, OR_REVERSE, COPY_INVERTED, OR_INVERTED, NAND, SET

Hints [21.5]

void **Hint**(enum target, enum hint);

target: FRAGMENT_SHADER_DERIVATIVE_HINT, TEXTURE_COMPRESSION_HINT, {LINE, POLYGON}_SMOOTH_HINT
hint: FASTEST, NICEST, DONT_CARE

attachments: COLOR_ATTACHMENT*i*, DEPTH, COLOR, {DEPTH, STENCIL, DEPTH_STENCIL}_ATTACHMENT, {FRONT, BACK}_{LEFT, RIGHT}, STENCIL

void **InvalidateNamedFramebufferSubData**(uint framebuffer, sizei numAttachments, const enum *attachments, int x, int y, sizei width, sizei height);

attachments: *See InvalidateSubFramebuffer*

void **InvalidateFramebuffer**(enum target, sizei numAttachments, const enum *attachments);

target, *attachments: *See InvalidateSubFramebuffer*

void **InvalidateNamedFramebufferData**(uint framebuffer, sizei numAttachments, const enum *attachments);

*attachments: *See InvalidateSubFramebuffer*

Debug Labels [20.7]

void **ObjectLabel**(enum identifier, uint name, sizei length, const char *label);

identifier: BUFFER, FRAMEBUFFER, RENDERBUFFER, PROGRAM_PIPELINE, PROGRAM, QUERY, SAMPLER, SHADER, TEXTURE, TRANSFORM_FEEDBACK, VERTEX_ARRAY

void **ObjectPtrLabel**(void* ptr, sizei length, const char *label);

Synchronous Debug Output [20.8]

Enable/Disable/IsEnabled(DEBUG_OUTPUT_SYNCHRONOUS);

Debug Output Queries [20.9]

uint **GetDebugMessageLog**(uint count, sizei bufSize, enum *sources, enum *types, uint *ids, enum *severities, sizei *lengths, char *messageLog);

void **GetObjectLabel**(enum identifier, uint name, sizei bufSize, sizei *length, char *label);

void **GetObjectPtrLabel**(void* ptr, sizei bufSize, sizei *length, char *label);

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

void **GetFloati_v**(enum target, uint index, float *data);

void **GetInteger64i_v**(enum target, uint index, int64 *data);

boolean **IsEnabled**(enum cap);

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

String Queries [22.2]

void **GetPointerv**(enum pname, void **params);

ubyte **GetString**(enum name);

name: RENDERER, VENDOR, VERSION, SHADING_LANGUAGE_VERSION

(Continued on next page) ►

States, State Requests (cont.)

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

Internal Format Queries [22.3]

void **GetInternalformativ**(enum *target*,
 enum *internalformat*, enum *pname*,
 sizei *bufSize*, int **params*);

target, *pname*, *internalformat*:

See **GetInternalformati64v**

void **GetInternalformati64v**(enum *target*,
 enum *internalformat*, enum *pname*,
 sizei *bufSize*, int64 **params*);

target: [Table 22.2]

TEXTURE_{1D, 2D, 3D, CUBE_MAP}[_ARRAY],
 TEXTURE_2D_MULTISAMPLE[_ARRAY],
 TEXTURE_{BUFFER, RECTANGLE}, RENDERBUFFER
internalformat: any value

pname:
 CLEAR_{BUFFER, TEXTURE},
 COLOR_ENCODING,
 COLOR_{COMPONENTS, RENDERABLE},
 COMPUTE_TEXTURE,
 DEPTH_{COMPONENTS, RENDERABLE},
 FILTER, FRAMEBUFFER_BLEND,
 FRAMEBUFFER_RENDERABLE[_LAYERED],
 {FRAGMENT, GEOMETRY}_TEXTURE,
 GET_TEXTURE_IMAGE_FORMAT,
 GET_TEXTURE_IMAGE_TYPE,
 IMAGE_COMPATIBILITY_CLASS,
 IMAGE_PIXEL_{FORMAT, TYPE},
 IMAGE_FORMAT_COMPATIBILITY_TYPE,
 IMAGE_TEXEL_SIZE,
 INTERNALFORMAT_{PREFERRED, SUPPORTED},
 INTERNALFORMAT_{(RED, GREEN, BLUE)_SIZE,
 (DEPTH, STENCIL)_SIZE},
 INTERNALFORMAT_{(ALPHA, SHARED)_SIZE},
 INTERNALFORMAT_{(RED, GREEN)_TYPE},
 INTERNALFORMAT_{(BLUE, ALPHA)_TYPE},
 INTERNALFORMAT_{DEPTH, STENCIL}_TYPE,
 [MANUAL_GENERATE]MIPMAP,
 MAX_COMBINED_DIMENSIONS,
 MAX_{WIDTH, HEIGHT, DEPTH, LAYERS},

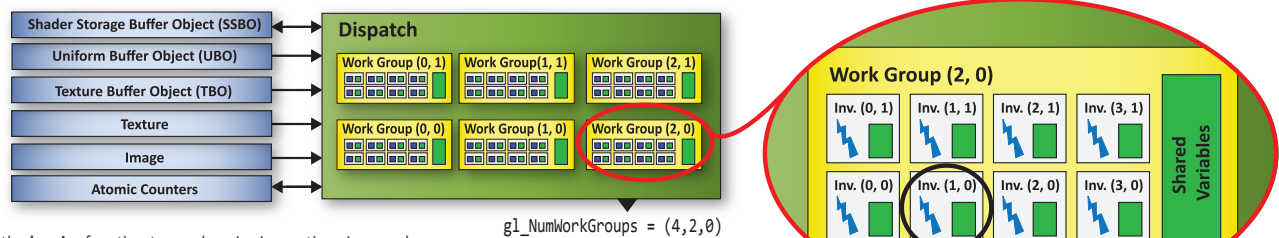
NUM_SAMPLE_COUNTS,
 READ_PIXELS_{FORMAT, _TYPE},
 SAMPLES, SHADER_IMAGE_ATOMIC,
 SHADER_IMAGE_{LOAD, STORE},
 SIMULTANEOUS_TEXTURE_AND_DEPTH_TEST,
 SIMULTANEOUS_TEXTURE_AND_DEPTH_WRITE,
 SIMULTANEOUS_TEXTURE_AND_STENCIL_TEST,
 SIMULTANEOUS_TEXTURE_AND_STENCIL_WRITE,
 SRGB_{READ, WRITE},
 STENCIL_{COMPONENTS, RENDERABLE},
 TESS_{CONTROL, EVALUATION}_TEXTURE,
 TEXTURE_COMPRESSED_BLOCK_SIZE,
 TEXTURE_COMPRESSED_BLOCK_{HEIGHT, WIDTH},
 TEXTURE_GATHER[_SHADOW],
 TEXTURE_IMAGE_FORMAT,
 TEXTURE_IMAGE_TYPE,
 TEXTURE_{SHADOW, VIEW},
 VERTEX_TEXTURE,
 VIEW_COMPATIBILITY_CLASS

enum *pname*, uint *index*, int **param*);
pname: TRANSFORM_FEEDBACK_BUFFER_BINDING

void **GetTransformFeedbacki64_v**(uint *xfb*,
 enum *pname*, uint *index*, int64 **param*);
pname: TRANSFORM_FEEDBACK_BUFFER_START,
 TRANSFORM_FEEDBACK_BUFFER_SIZE

TransformFeedback Queries [22.4]

void **GetTransformFeedbackiv**(uint *xfb*,
 enum *pname*, int **param*);
pname: TRANSFORM_FEEDBACK_{PAUSED, ACTIVE}
 void **GetTransformFeedbacki_v**(uint *xfb*,

OpenGL Compute Programming Model and Compute Memory Hierarchy

gl_NumWorkGroups = (4, 2, 0)

Use the **barrier** function to synchronize invocations in a work group:

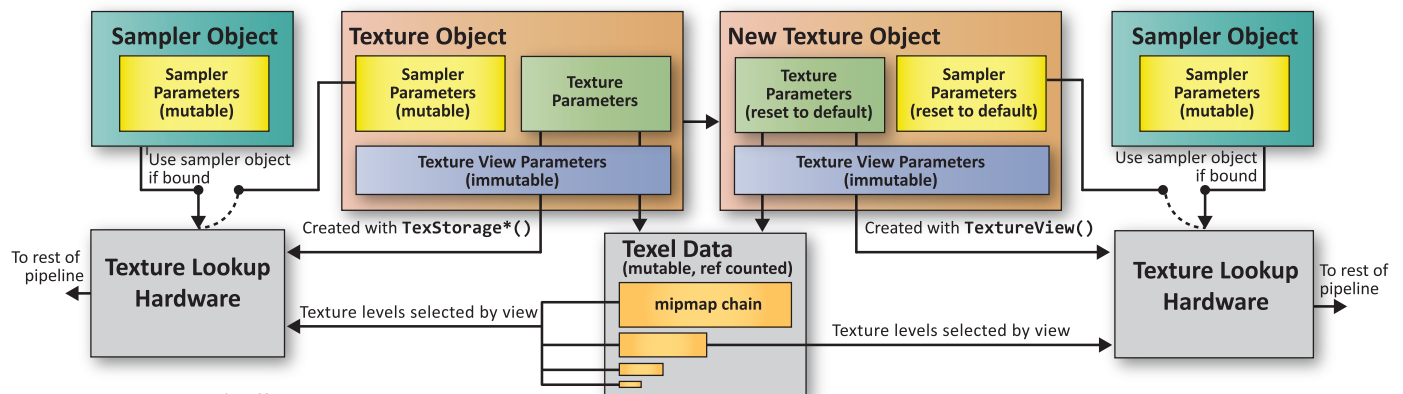
```
void barrier();
```

Use the **memoryBarrier*** or **groupMemoryBarrier** functions to order reads/writes accessible to other invocations:

```
void memoryBarrier();  
void memoryBarrierAtomicCounter();  
void memoryBarrierBuffer();  
void memoryBarrierImage();  
void memoryBarrierShared(); // Only for compute shaders  
void groupMemoryBarrier(); // Only for compute shaders
```

Use the compute shader built-in variables to specify work groups and invocations:

```
in vec3 gl_NumWorkGroups; // Number of workgroups dispatched  
const vec3 gl_WorkGroupSize; // Size of each work group for current shader  
in vec3 gl_WorkGroupID; // Index of current work group being executed  
in vec3 gl_LocalInvocationID; // index of current invocation in a work group  
in vec3 gl_GlobalInvocationID; // Unique ID across all work groups and threads. (gl_GlobalInvocationID = gl_WorkGroupID * gl_WorkGroupSize + gl_LocalInvocationID)
```

OpenGL Texture Views and Texture Object State**Texture state set with TextureView()**

enum *internalformat* // base internal format
 enum *target* // texture target
 uint *minlevel* // first level of mipmap
 uint *numlevels* // number of mipmap levels
 uint *minlayer* // first layer of array texture
 uint *numlayers* // number of layers in array

Sampler Parameters (mutable)

TEXTURE_BORDER_COLOR
 TEXTURE_COMPARE_{FUNC, MODE}
 TEXTURE_LOD_BIAS
 TEXTURE_{MAX, MIN}_LOD
 TEXTURE_{MAG, MIN}_FILTER
 TEXTURE_MAX_ANISOTROPY
 TEXTURE_WRAP_{S, T, R}

Texture Parameters (immutable)

TEXTURE_WIDTH TEXTURE_HEIGHT
 TEXTURE_DEPTH TEXTURE_FIXED_SAMPLE_LOCATIONS
 TEXTURE_COMPRESSED TEXTURE_COMPRESSED_IMAGE_SIZE
 TEXTURE_IMMUTABLE_FORMAT TEXTURE_SAMPLES
Texture Parameters (mutable)
 TEXTURE_SWIZZLE_{R, G, B, A} TEXTURE_MAX_LEVEL
 TEXTURE_BASE_LEVEL DEPTH_STENCIL_TEXTURE_MODE

Texture View Parameters (immutable)

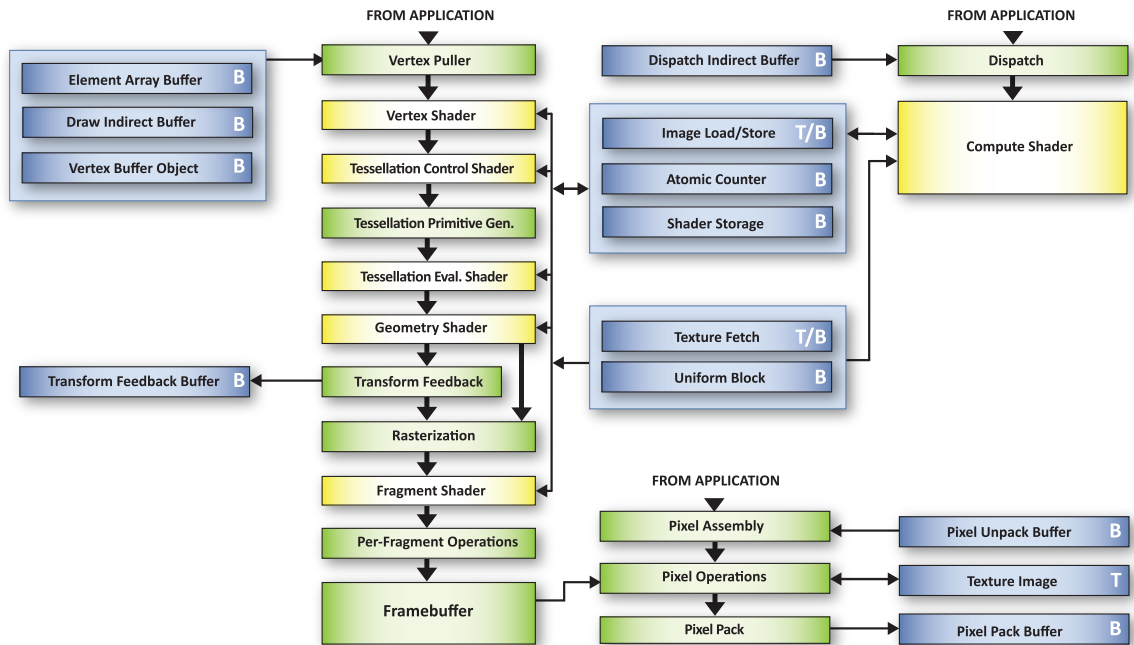
<target>
 TEXTURE_INTERNAL_FORMAT TEXTURE_SHARED_SIZE
 TEXTURE_VIEW_{MIN, NUM}_LEVEL TEXTURE_VIEW_{MIN, NUM}_LAYER
 TEXTURE_IMMUTABLE_LEVELS IMAGE_FORMAT_COMPATIBILITY_TYPE
 TEXTURE_{RED, GREEN, BLUE, ALPHA, DEPTH}_TYPE
 TEXTURE_{RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL}_SIZE

OpenGL Pipeline

A typical program that uses OpenGL begins with calls to open a window into the framebuffer into which the program will draw. Calls are made to allocate a GL context which is then associated with the window, then OpenGL commands can be issued.

The heavy black arrows in this illustration show the OpenGL pipeline and indicate data flow.

- B Blue blocks indicate various buffers that feed or get fed by the OpenGL pipeline.
- F Green blocks indicate fixed function stages.
- S Yellow blocks indicate programmable stages.
- T Texture binding
- B Buffer binding

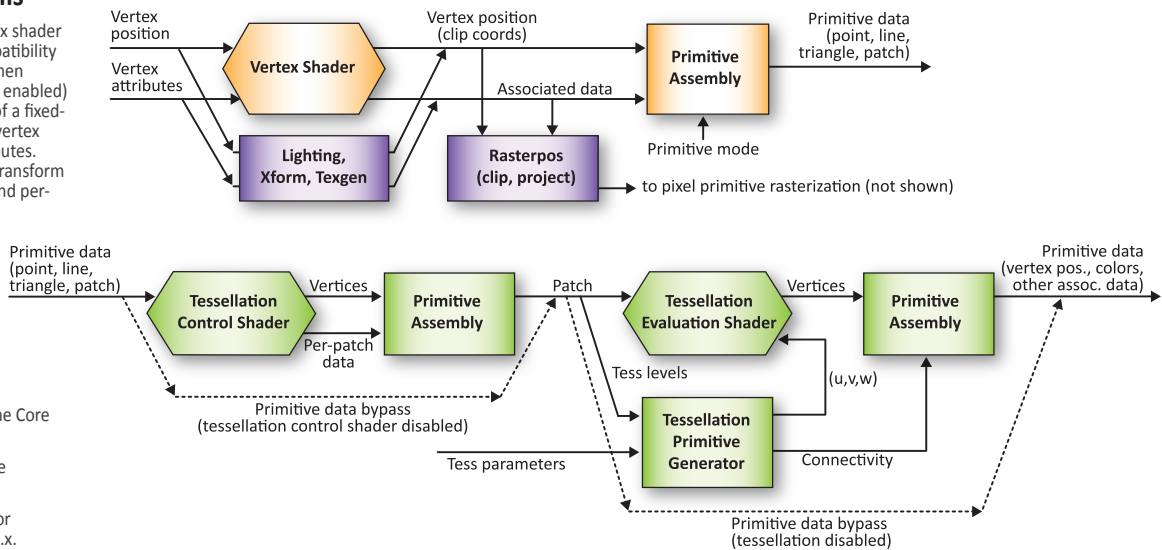


Vertex & Tessellation Details

Each vertex is processed either by a vertex shader or fixed-function vertex processing (compatibility only) to generate a transformed vertex, then assembled into primitives. Tessellation (if enabled) operates on patch primitives, consisting of a fixed-size collection of vertices, each with per-vertex attributes and associated per-patch attributes. Tessellation control shaders (if enabled) transform an input patch and compute per-vertex and per-patch attributes for a new output patch.

A fixed-function primitive generator subdivides the patch according to tessellation levels computed in the tessellation control shaders or specified as fixed values in the API (TCS disabled). The tessellation evaluation shader computes the position and attributes of each vertex produced by the tessellator.

- F Orange blocks indicate features of the Core specification.
- F Purple blocks indicate features of the Compatibility specification.
- F Green blocks indicate features new or significantly changed with OpenGL 4.x.



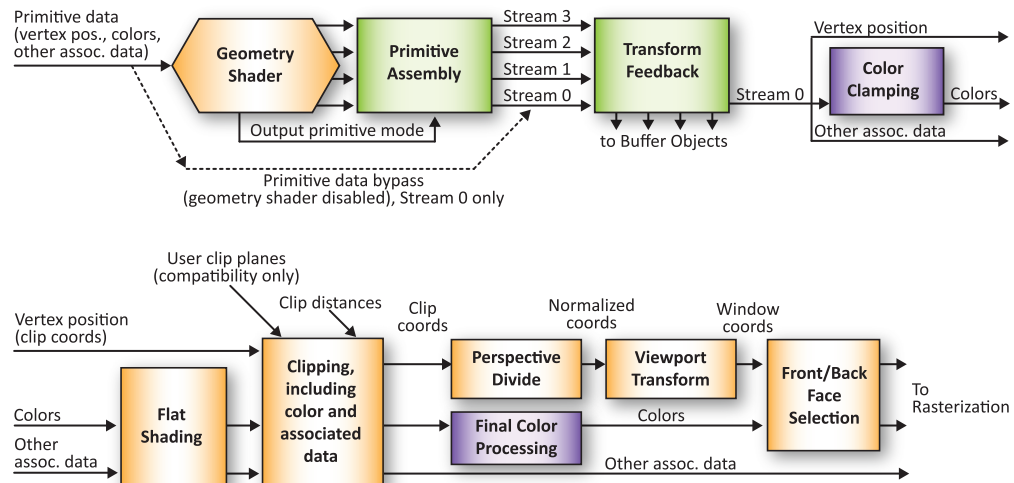
Geometry & Follow-on Details

Geometry shaders (if enabled) consume individual primitives built in previous primitive assembly stages. For each input primitive, the geometry shader can output zero or more vertices, with each vertex directed at a specific vertex stream. The vertices emitted to each stream are assembled into primitives according to the geometry shader's output primitive type.

Transform feedback (if active) writes selected vertex attributes of the primitives of all vertex streams into buffer objects attached to one or more binding points.

Primitives on vertex stream zero are then processed by fixed-function stages, where they are clipped and prepared for rasterization.

- F Orange blocks indicate features of the Core specification.
- F Purple blocks indicate features of the Compatibility specification.
- F Green blocks indicate features new or significantly changed with OpenGL 4.x.



The OpenGL® Shading Language is used to create shaders for each of the programmable processors contained in the OpenGL processing pipeline. The OpenGL Shading Language is actually several closely related languages. Currently, these processors are the vertex, tessellation control, tessellation evaluation, geometry, fragment, and compute shaders.

[n.n.n] and [Table n.n] refer to sections and tables in the OpenGL Shading Language 4.60.1 specification at www.khronos.org/opengl

Preprocessor [3.3]

Preprocessor Operators

#version 450	Required when using version 4.50.
#version 450 profile	profile is core, compatibility, or es (for ES versions 1.00, 3.00, or 3.10).
#extension extension_name : behavior	- behavior: require, enable, warn, disable - extension_name: extension supported by compiler, or "all"
#extension all : behavior	

Preprocessor Directives

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

Predefined Macros

__LINE__	__FILE__	Decimal integer constants. __FILE__ says which source string is being processed.
VERSION		Decimal integer, e.g.: 450
GL_core_profile		Defined as 1
GL_es_profile		1 if the ES profile is supported
GL_compatibility_profile		Defined as 1 if the implementation supports the compatibility profile.
GL_SPIRV		Defined and equals 100 when shaders are compiled for OpenGL SPIR-V.

Operators and Expressions [5.1]

The following operators are numbered in order of precedence. Relational and equality operators evaluate to Boolean. Also See lessThan(), equal().

1.	()	parenthetical grouping
2.	[], (), ., ++, --	array subscript, function call, constructor, structure field, selector, swizzle, 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.	?:	selects an entire operand
16.	=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, =	assignment, arithmetic assignments
17.	,	sequence

Vector & Scalar Components [5.5]

In addition to array numeric subscript syntax, names of vector and scalar components are denoted by a single letter. Components can be swizzled and replicated. Scalars have only an *x*, *y*, or *z* component.

{ <i>x</i> , <i>y</i> , <i>z</i> , <i>w</i> }	Points or normals
{ <i>r</i> , <i>g</i> , <i>b</i> , <i>a</i> }	Colors
{ <i>s</i> , <i>t</i> , <i>p</i> , <i>q</i> }	Texture coordinates

Types [4.1]

Transparent Types

void	no function return value
bool	Boolean
int, uint	signed/unsigned integers
float	single-precision floating-point scalar
double	double-precision floating scalar
vec2, vec3, vec4	floating point vector
dvec2, dvec3, dvec4	double precision floating-point vectors
bvec2, bvec3, bvec4	Boolean vectors
ivec2, ivec3, ivec4, uvec2, uvec3, uvec4	signed and unsigned integer vectors
mat2, mat3, mat4	2x2, 3x3, 4x4 float matrix
mat2x2, mat2x3, mat2x4	2-column float matrix of 2, 3, or 4 rows
mat3x2, mat3x3, mat3x4	3-column float matrix of 2, 3, or 4 rows
mat4x2, mat4x3, mat4x4	4-column float matrix of 2, 3, or 4 rows
dmat2, dmat3, dmat4	2x2, 3x3, 4x4 double-precision float matrix
dmat2x2, dmat2x3, dmat2x4	2-col. double-precision float matrix of 2, 3, 4 rows
dmat3x2, dmat3x3, dmat3x4	3-col. double-precision float matrix of 2, 3, 4 rows
dmat4x2, dmat4x3, dmat4x4	4-column double-precision float matrix of 2, 3, 4 rows

Floating-Point Opaque Types

sampler1D, 2D, 3D	1D, 2D, or 3D texture
image1D, 2D, 3D	cube mapped texture
samplerCube	cube mapped texture
sampler2DRect	rectangular texture
sampler1D, 2D Array	1D or 2D array texture
image1D, 2D Array	buffer texture
samplerBuffer	buffer texture
imageBuffer	buffer texture
sampler2DMS	2D multi-sample texture
sampler2DMSArray	2D multi-sample array texture
image2DMSArray	2D multi-sample array texture
samplerCubeArray	cube map array texture
sampler1DShadow	1D or 2D depth texture with comparison
sampler2DRectShadow	rectangular tex. / compare
sampler1DArrayShadow	1D or 2D array depth texture with comparison
sampler2DArrayShadow	2D array depth texture with comparison
samplerCubeShadow	cube map depth texture with comparison
samplerCubeArrayShadow	cube map array depth texture with comparison

Signed Integer Opaque Types

isampler1, 2, 3, 4	integer 1D, 2D, or 3D texture
iimage1, 2, 3, 4	integer 1D, 2D, or 3D image
isamplerCube	integer cube mapped texture
iimageCube	integer cube mapped image
isampler2DRect	int. 2D rectangular texture

Signed Integer Opaque Types (cont'd)

iimage2DRect	int. 2D rectangular image
isampler1, 2, 3, 4 Array	integer 1D, 2D array texture
iimage1, 2, 3, 4 Array	integer 1D, 2D array image
isamplerBuffer	integer buffer texture
iimageBuffer	integer buffer image
isampler2DMS	int. 2D multi-sample texture
iimage2DMS	int. 2D multi-sample image
isampler2DMSArray	int. 2D multi-sample array tex.
iimage2DMSArray	int. 2D multi-sample array image
isamplerCubeArray	int. cube map array texture
iimageCubeArray	int. cube map array image

Unsigned Integer Opaque Types

atomic_uint	uint atomic counter
usampler1, 2, 3, 4	uint 1D, 2D, or 3D texture
uimage1, 2, 3, 4	uint 1D, 2D, or 3D image
usamplerCube	uint cube mapped texture
uimageCube	uint cube mapped image
usampler2DRect	uint rectangular texture
uimage2DRect	uint rectangular image
usampler1, 2, 3, 4 Array	1D or 2D array texture
uimage1, 2, 3, 4 Array	1D or 2D array image
usamplerBuffer	uint buffer texture
uimageBuffer	uint buffer image
usampler2DMS	uint 2D multi-sample texture
uimage2DMS	uint 2D multi-sample image
usampler2DMSArray	uint 2D multi-sample array tex.

Unsigned Integer Opaque Types (cont'd)

uimage2DMSArray	uint 2D multi-sample array image
usamplerCubeArray	uint cube map array texture
uimageCubeArray	uint cube map array image

Implicit Conversions

int	-> uint	ivec2	-> dvec2
int, uint	-> float	ivec3	-> dvec3
int, uint, float	-> double	ivec4	-> dvec4
ivec2	-> uvec2	vec2	-> dvec2
ivec3	-> uvec3	vec3	-> dvec3
ivec4	-> uvec4	vec4	-> dvec4
ivec2	-> vec2	mat2	-> dmat2
ivec3	-> vec3	mat3	-> dmat3
ivec4	-> vec4	mat4	-> dmat4
uvec2	-> vec2	mat2x3	-> dmat2x3
uvec3	-> vec3	mat2x4	-> dmat2x4
uvec4	-> vec4	mat3x2	-> dmat3x2
ivec2	-> dvec2	mat3x4	-> dmat3x4
ivec3	-> dvec3	mat4x2	-> dmat4x2
ivec4	-> dvec4	mat4x3	-> dmat4x3

Aggregation of Basic Types

Arrays	float[3] foo; float foo[3]; int a [3][2]; // Structures, blocks, and structure members // can be arrays. Arrays of arrays supported.
Structures	struct type-name { members }; struct-name[]; // optional variable declaration
Blocks	in/out/uniform block-name { // interface matching by block name optionally-qualified members }; instance-name[]; // optional instance name, optionally an array

Qualifiers

Storage Qualifiers [4.3]

Declarations may have one storage qualifier.

none	(default) local read/write memory, or input parameter
const	read-only variable
in	linkage into shader from previous stage
out	linkage out of a shader to next stage
uniform	linkage between a shader, OpenGL, and the application
buffer	accessible by shaders and OpenGL API
shared	compute shader only, shared among work items in a local work group

Auxiliary Storage Qualifiers

Use to qualify some input and output variables:

centroid	centroid-based interpolation
sampler	per-sample interpolation
patch	per-tessellation-patch attributes

Interface Blocks [4.3.9]

in, out, uniform, and buffer variable declarations can be grouped. For example:

```
uniform Transform {
    // allowed restatement qualifier:
    mat4 ModelViewMatrix;
    uniform mat3 NormalMatrix;
};
```

Layout Qualifiers [4.4]

The following table summarizes the use of layout qualifiers applied to non-opaque types and the kinds of declarations they may be applied to. Op = Opaque types only, FC = gl_FragCoord only, FD = gl_FragDepth only.

Layout Qualifier	Qualif. Only	Indiv. Var.	Block	Block Mem.	Allowed Interfaces
shared, packed, std[140, 430]	X		X		
(row, column)_major	X		X	X	
binding =		Op	X		uniform/buffer
offset =				X	
align =			X	X	
location =		X			uniform/buffer and subroutine variables
location =		X	X	X	all in/out, except for compute
component =		X		X	
index =		X			fragment out and subroutine functions
triangles, quads, isolines	X				
equal_spacing,					
fractional_even_spacing,	X				tessellation evaluation in
fractional_odd_spacing					
cw, ccw	X				
point_mode	X				
points	X				geometry in/out
[points], lines, triangles,					
(triangles, lines)_adjacency	X				geometry in
invocations =	X				geometry in

Layout Qualifier	Qualif. Only	Indiv. Var.	Block	Block Mem.	Allowed Interfaces
origin_upper_left					
pixel_center_integer					
early_fragment_tests	X				fragment in
local_size {x, y, z} =	X				compute in
local_size {x,y,z}_id =	X				compute in
xfb {buffer, stride} =	X	X	X	X	vertex, tessellation, and geometry out
xfb_offset =		X	X	X	tessellation control out
vertices =	X				
[points], line_strip,		X			geometry out
triangle_strip		X			
max_vertices =	X				
stream =	X	X	X	X	
depth {any, greater, less, unchanged}				FD	fragment out
constant_id		scalar only			const

Opaque Uniform Layout Qualifiers [4.4.6]

Used to bind opaque uniform variables to specific buffers or units.

binding = integer-constant-expression

Atomic Counter Layout Qualifiers

binding = integer-constant-expression

offset = integer-constant-expression

(Continued on next page) ►

Qualifiers (continued)

Format Layout Qualifiers

One qualifier may be used with variables declared as “image” to specify the image format.

```
binding = integer-constant-expression,
rgba{32,16}f, rg{32,16}f, r{32,16}f,
rgba{16,8}, r11f_g11f_b10f, rgb10_a2{ui},
rg{16,8}, r{16,8}, rgba{32,16,8}i, rg{32,16,8}i,
r{32,16,8}i, rgba{32,16,8}ui, rg{32,16,8}ui,
r{32,16,8}ui, rgba{16,8}_snorm,
rg{16,8}_snorm, r{16,8}_snorm
```

Interpolation Qualifiers [4.5]

Qualify outputs from vertex shader and inputs to fragment shader.

smooth	perspective correct interpolation
flat	no interpolation
noperspective	linear interpolation

Parameter Qualifiers [4.6]

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

none	(default) same as in
in	for parameters passed into function
const	for parameters that cannot be written to
out	for parameters passed back out of function, but not initialized when passed in
inout	for parameters passed both into and out of a function

Precision Qualifiers [4.7]

Qualify individual variables:

{highp, mediump, lowp} *variable-declaration*;

Establish a default precision qualifier:

```
precision {highp, mediump, lowp}
{int, float};
```

Invariant Qualifiers Examples [4.8]

These are for vertex, tessellation, geometry, and fragment languages.

#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

Precise Qualifier [4.9]

Ensures that operations are executed in stated order with operator consistency. For example:

```
precise out vec4 Position = a * b + c * d;
```

Memory Qualifiers [4.10]

Variables qualified as “image” can have one or more memory qualifiers.

coherent	reads and writes are coherent with other shader invocations
volatile	underlying values may be changed by other sources
restrict	won't be accessed by other code
readonly	read only
writeonly	write only

Specialization-Constant Qualifier [4.11]

SPIR-V specialization constants are expressed in GLSL as **const** with the layout qualifier **constant_id**. Function calls to user-defined functions cannot be used to form constant expressions. [\[also see 4.3.3\]](#)

Order of Qualification [4.12]

Multiple qualifiers may appear in a declaration in any order, but must all appear before the type. Only the layout qualifier may appear more than once. A declaration may have at most one storage qualifier, at most one auxiliary storage qualifier, and at most one interpolation qualifier.

Multiple memory qualifiers may be used. Any rule violation will cause a compile-time error.

Built-In Variables [7]

Vertex Language

Inputs	in int gl_VertexID; in int gl_InstanceID; in int gl_BaseInstance in int gl_BaseVertex in int gl_DrawID
Outputs	out gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; };

Tessellation Control Language

Inputs	in gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; } gl_in[gl_MaxPatchVertices];
Outputs	in int gl_PatchVerticesIn; in int gl_PrimitiveID; in int gl_InvocationID; out gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; } gl_out[]; patch out float gl_TessLevelOuter[4]; patch out float gl_TessLevelInner[2];

Tessellation Evaluation Language

Inputs	in gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; } gl_in[gl_MaxPatchVertices]; in int gl_PatchVerticesIn; in int gl_PrimitiveID; in vec3 gl_TessCoord; patch in float gl_TessLevelOuter[4]; patch in float gl_TessLevelInner[2];
Outputs	out gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; };

Geometry Language

Inputs	in gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; } gl_in[]; in int gl_PrimitiveIDIn; in int gl_InvocationID;
Outputs	out gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; float gl_CullDistance[]; }; out int gl_PrimitiveID; out int gl_Layer; out int gl_ViewportIndex;

Fragment Language

Inputs	in vec4 gl_FragCoord; in bool gl_FrontFacing; in float gl_ClipDistance[]; in float gl_CullDistance[]; in vec2 gl_PointCoord; in int gl_PrimitiveID; in int gl_SampleID; in vec2 gl_SamplePosition; in int gl_SampleMaskIn[]; in int gl_Layer; in int gl_ViewportIndex; in bool gl_HelperInvocation;
Outputs	out float gl_FragDepth; out int gl_SampleMask[];

Compute Language

More information in diagram on page 6.

Inputs	Work group dimensions in uvec3 gl_NumWorkGroups; const uvec3 gl_WorkGroupSize; in uvec3 gl_LocalGroupSize; Work group and invocation IDs in uvec3 gl_WorkGroupID; in uvec3 gl_LocalInvocationID; Derived variables in uvec3 gl_GlobalInvocationID; in uint gl_LocalInvocationIndex;
---------------	---

Operations and Constructors

Vector & Matrix [5.4.2]

length() for matrices returns number of columns
length() for vectors returns number of components

```
mat2(vec2, vec2); // 1 col./arg.
mat2x3(vec2, float, vec2, float); // col. 2
dmat2(dvec2, dvec2); // 1 col./arg.
dmat3(dvec3, dvec3, dvec3); // 1 col./arg.
```

Structure Example [5.4.3]

length() for structures returns number of members

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

Matrix Examples [5.6]

Examples of access components of a matrix with array subscripting syntax:

```
mat4 m; // m is a matrix
m[1] = vec4(2.0); // sets 2nd col. 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 col. 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 +/- m; // matrix +/- matrix comp.-wise
m = m * m; // linear algebraic multiply
f = dot(v, v); // vector dot product
v = cross(v, v); // vector cross product
```

Array Example [5.4.4]

```
const float c[3];
c.length() // will return the integer 3
```

Structure & Array Operations [5.7]

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

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

Array elements are accessed using the array subscript operator (**[]**), e.g.:

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

Statements and Structure

Subroutines [6.1.2]

Subroutine type variables are assigned to functions through the **UniformSubroutinesuiv** command in the OpenGL API.

Declare types with the **subroutine** keyword:

```
subroutine returnType subroutineTypeName(type0
arg0,
type1 arg1, ..., typen argn);
```

Associate functions with subroutine types of matching declarations by defining the functions with the **subroutine** keyword and a list of subroutine types the function matches:

```
subroutine(subroutineTypeName0, ...,
subroutineTypeNameN)
returnType functionName(type0 arg0,
type1 arg1, ..., typen argn){ ... }
// function body
```

Declare subroutine type variables with a specific subroutine type in a subroutine uniform variable declaration:

```
subroutine uniform subroutineTypeName
subroutineVarName;
```

Iteration and Jumps [6.3-4]

Function	call by value-return
Iteration	for (;;) { break, continue } while () { break, continue } do { break, continue } while ();
Selection	if () { } if () { } else { } switch () { case integer: ... break; ... default: ... }
Entry	void main()
Jump	break, continue, return (There is no 'goto')
Exit	return in main() discard // Fragment shader only

Built-In Constants [7.3]

The following are provided to all shaders. The actual values are implementation-dependent, but must be at least the value shown.

```
const ivec3 gl_MaxComputeWorkGroupCount =
{65535, 65535, 65535};
const ivec3 gl_MaxComputeWorkGroupSize[] =
{1024, 1024, 64};
const int gl_MaxComputeUniformComponents = 1024;
const int gl_MaxComputeTextureImageUnits = 16;
const int gl_MaxComputeImageUniforms = 8;
const int gl_MaxComputeAtomicCounters = 8;
const int gl_MaxComputeAtomicCounterBuffers = 1;
const int gl_MaxVertexAttributes = 16;
const int gl_MaxVertexUniformComponents = 1024;
const int gl_MaxVaryingComponents = 60;
const int gl_MaxVertexOutputComponents = 64;
const int gl_MaxGeometryInputComponents = 64;
const int gl_MaxGeometryOutputComponents = 128;
const int gl_MaxFragmentInputComponents = 128;
const int gl_MaxVertexTextureImageUnits = 16;
const int gl_MaxCombinedTextureImageUnits = 80;
const int gl_MaxTextureImageUnits = 16;
const int gl_MaxImageUnits = 8;
const int gl_MaxCombinedImageUnitsAndFragmentOutputs = 8;
const int gl_MaxImageSamples = 0;
const int gl_MaxVertexImageUniforms = 0;
const int gl_MaxTessControlImageUniforms = 0;
const int gl_MaxGeometryImageUniforms = 0;
const int gl_MaxFragmentImageUniforms = 8;
const int gl_MaxCombinedImageUniforms = 8;
const int gl_MaxGeometryTextureImageUnits = 16;
const int gl_MaxGeometryOutputVertices = 256;
const int gl_MaxGeometryTotalOutputComponents = 1024;
const int gl_MaxGeometryUniformComponents = 1024;
const int gl_MaxGeometryVaryingComponents = 64;
const int gl_MaxTessControlInputComponents = 128;
const int gl_MaxTessControlOutputComponents = 128;
const int gl_MaxTessControlTextureImageUnits = 16;
const int gl_MaxTessControlUniformComponents = 1024;
const int gl_MaxTessEvaluationInputComponents = 128;
const int gl_MaxTessEvaluationOutputComponents = 128;
const int gl_MaxTessEvaluationTextureImageUnits = 16;
const int gl_MaxTessEvaluationUniformComponents = 1024;
const int gl_MaxTessPatchComponents = 120;
const int gl_MaxPatchVertices = 32;
const int gl_MaxTessGenLevel = 64;
const int gl_MaxViewports = 16;
const int gl_MaxVertexUniformVectors = 256;
const int gl_MaxFragmentUniformVectors = 256;
const int gl_MaxVaryingVectors = 15;
const int gl_MaxVertexAtomicCounters = 0;
const int gl_MaxTessControlAtomicCounters = 0;
const int gl_MaxTessEvaluationAtomicCounters = 0;
const int gl_MaxGeometryAtomicCounters = 0;
const int gl_MaxFragmentAtomicCounters = 8;
const int gl_MaxCombinedAtomicCounters = 8;
const int gl_MaxAtomicCounterBindings = 1;
const int gl_MaxVertexAtomicCounterBuffers = 0;
const int gl_MaxTessControlAtomicCounterBuffers = 0;
const int gl_MaxTessEvaluationAtomicCounterBuffers = 0;
const int gl_MaxGeometryAtomicCounterBuffers = 0;
const int gl_MaxFragmentAtomicCounterBuffers = 1;
const int gl_MaxCombinedAtomicCounterBuffers = 1;
const int gl_MaxAtomicCounterBufferSize = 32;
const int gl_MinProgramTexelOffset = -8;
const int gl_MaxProgramTexelOffset = 7;
const int gl_MaxTransformFeedbackBuffers = 4;
const int gl_MaxTransformFeedbackInterleavedComponents = 64;
const int gl_MaxCullDistances = 8;
const int gl_MaxCombinedClipAndCullDistances = 8;
const int gl_MaxSamples = 4;
const int gl_MaxVertexImageUniforms = 0;
const int gl_MaxFragmentImageUniforms = 8;
const int gl_MaxComputeImageUniforms = 8;
const int gl_MaxCombinedImageUniforms = 48;
const int gl_MaxCombinedShaderOutputResources = 16;
```

Built-In Functions

Angle & Trig. Functions [8.1]

Functions will not result in a divide-by-zero error. If the divisor of a ratio is 0, then results will be undefined. Component-wise operation. Parameters specified as *angle* are in units of radians. Tf=float, vecn.

Tf radians (Tf <i>degrees</i>)	degrees to radians
Tf degrees (Tf <i>radians</i>)	radians to degrees
Tf sin (Tf <i>angle</i>)	sine
Tf cos (Tf <i>angle</i>)	cosine
Tf tan (Tf <i>angle</i>)	tangent
Tf asin (Tf <i>x</i>)	arc sine
Tf acos (Tf <i>x</i>)	arc cosine
Tf atan (Tf <i>y</i> , Tf <i>x</i>)	arc tangent
Tf atan (Tf <i>y_over_x</i>)	
Tf sinh (Tf <i>x</i>)	hyperbolic sine
Tf cosh (Tf <i>x</i>)	hyperbolic cosine
Tf tanh (Tf <i>x</i>)	hyperbolic tangent
Tf asinh (Tf <i>x</i>)	hyperbolic sine
Tf acosh (Tf <i>x</i>)	hyperbolic cosine
Tf atanh (Tf <i>x</i>)	hyperbolic tangent

Exponential Functions [8.2]

Component-wise operation. Tf=float, vecn. Td= double, dvecn. Tfd= Tf, Td

Tf pow (Tf <i>x</i> , Tf <i>y</i>)	x^y
Tf exp (Tf <i>x</i>)	e^x
Tf log (Tf <i>x</i>)	ln
Tf exp2 (Tf <i>x</i>)	2^x
Tf log2 (Tf <i>x</i>)	$\log_2$
Tfd sqrt (Tfd <i>x</i>)	square root
Tfd inversesqrt (Tfd <i>x</i>)	inverse square root

Common Functions [8.3]

Component-wise operation. Tf=float, vecn. Tb=bool, bvecn. Ti=int, ivecni. Tu=uint, uvecn. Td= double, dvecn. Tfd= Tf, Td. Tiu= Ti, Tu.

Returns absolute value:	
Tfd abs (Tfd <i>x</i>)	Ti abs (Ti <i>x</i>)
Returns -1.0, 0.0, or 1.0:	
Tfd sign (Tfd <i>x</i>)	Ti sign (Ti <i>x</i>)
Returns nearest integer <= x:	
Tfd floor (Tfd <i>x</i>)	
Returns nearest integer with absolute value <= absolute value of x:	
Tfd trunc (Tfd <i>x</i>)	
Returns nearest integer, implementation-dependent rounding mode:	
Tfd round (Tfd <i>x</i>)	
Returns nearest integer, 0.5 rounds to nearest even integer:	
Tfd roundEven (Tfd <i>x</i>)	
Returns nearest integer >= x:	
Tfd ceil (Tfd <i>x</i>)	
Returns x - floor(x):	
Tfd fract (Tfd <i>x</i>)	
Returns modulus:	
Tfd mod (Tfd <i>x</i> , Tfd <i>y</i>)	Td mod (Td <i>x</i> , double <i>y</i>)
Tf mod (Tf <i>x</i> , float <i>y</i>)	
Returns separate integer and fractional parts:	
Tfd modf (Tfd <i>x</i> , out Tf <i>i</i>)	
Returns minimum value:	
Tfd min (Tfd <i>x</i> , Tfd <i>y</i>)	Tiu min (Tiu <i>x</i> , Tiu <i>y</i>)
Tf min (Tf <i>x</i> , float <i>y</i>)	Ti min (Ti <i>x</i> , int <i>y</i>)
Td min (Td <i>x</i> , double <i>y</i>)	Tu min (Tu <i>x</i> , uint <i>y</i>)

(Continue ↓)

Common Functions (cont.)

Returns maximum value:

Tfd max (Tfd <i>x</i> , Tfd <i>y</i>)	Tiu max (Tiu <i>x</i> , Tiu <i>y</i>)
Tf max (Tf <i>x</i> , float <i>y</i>)	Ti max (Ti <i>x</i> , int <i>y</i>)
Td max (Td <i>x</i> , double <i>y</i>)	Tu max (Tu <i>x</i> , uint <i>y</i>)

Returns min(max(x, minVal), maxVal):

Tfd clamp (Tfd <i>x</i> , Tfd <i>minVal</i> , Tfd <i>maxVal</i>)	
Tf clamp (Tf <i>x</i> , float <i>minVal</i> , float <i>maxVal</i>)	
Td clamp (Td <i>x</i> , double <i>minVal</i> , double <i>maxVal</i>)	
Tiu clamp (Tiu <i>x</i> , Tiu <i>minVal</i> , Tiu <i>maxVal</i>)	
Ti clamp (Ti <i>x</i> , int <i>minVal</i> , int <i>maxVal</i>)	
Tu clamp (Tu <i>x</i> , uint <i>minVal</i> , uint <i>maxVal</i>)	

Returns linear blend of x and y:

Tfd mix (Tfd <i>x</i> , Tfd <i>y</i> , Tfd <i>a</i>)	Ti mix (Ti <i>x</i> , Ti <i>y</i> , Ti <i>a</i>)
Tf mix (Tf <i>x</i> , Tf <i>y</i> , float <i>a</i>)	Tu mix (Tu <i>x</i> , Tu <i>y</i> , Tu <i>a</i>)
Td mix (Td <i>x</i> , Td <i>y</i> , double <i>a</i>)	

Components returned come from x when a components are true, from y when a components are false:

Tfd mix (Tfd <i>x</i> , Tfd <i>y</i> , Tb <i>a</i>)	Tb mix (Tb <i>x</i> , Tb <i>y</i> , Tb <i>a</i>)
Tiu mix (Tiu <i>x</i> , Tiu <i>y</i> , Tb <i>a</i>)	

Returns 0.0 if x < edge, else 1.0:

Tfd step (Tfd <i>edge</i> , Tfd <i>x</i>)	Td step (double <i>edge</i> , Td <i>x</i>)
Tf step (float <i>edge</i> , Tf <i>x</i>)	

Clamps and smooths:

Tfd smoothstep (Tfd <i>edge0</i> , Tfd <i>edge1</i> , Tfd <i>x</i>)	
Tf smoothstep (float <i>edge0</i> , float <i>edge1</i> , Tf <i>x</i>)	
Td smoothstep (double <i>edge0</i> , double <i>edge1</i> , Td <i>x</i>)	

Returns true if x is NaN:

Tb isnan (Tfd <i>x</i>)	
---------------------------------	--

Returns true if x is positive or negative infinity:

Tb isinf (Tfd <i>x</i>)	
---------------------------------	--

Returns signed int or uint value of the encoding of a float:

Ti floatBitsToInt (Tf <i>value</i>)	
Tu floatBitsToUint (Tf <i>value</i>)	

Returns float value of a signed int or uint encoding of a float:

Tf intBitsToFloat (Ti <i>value</i>)	Tf uintBitsToFloat (Tu <i>value</i>)
---	--

Computes and returns a*b + c. Treated as a single operation when using **precise**:

Tfd fma (Tfd <i>a</i> , Tfd <i>b</i> , Tfd <i>c</i>)	
--	--

Splits x into a floating-point significand in the range [0.5, 1.0) and an integer exponent of 2:

Tfd frexp (Tfd <i>x</i> , out Ti <i>exp</i>)	
--	--

Builds a floating-point number from x and the corresponding integral exponent of 2 in exp:

Tfd ldexp (Tfd <i>x</i> , in Ti <i>exp</i>)	
---	--

Floating-Point Pack/Unpack [8.4]

These do not operate component-wise.

Converts each component of v into 8- or 16-bit ints, packs results into the returned 32-bit unsigned integer:

uint packUnorm2x16 (vec2 <i>v</i>)	uint packUnorm4x8 (vec4 <i>v</i>)
uint packSnorm2x16 (vec2 <i>v</i>)	uint packSnorm4x8 (vec4 <i>v</i>)

Unpacks 32-bit p into two 16-bit uints, four 8-bit uints, or signed ints. Then converts each component to a normalized float to generate a 2- or 4-component vector:

vec2 unpackUnorm2x16 (uint <i>p</i>)	
vec2 unpackSnorm2x16 (uint <i>p</i>)	
vec4 unpackUnorm4x8 (uint <i>p</i>)	
vec4 unpackSnorm4x8 (uint <i>p</i>)	

Packs components of v into a 64-bit value and returns a double-precision value:

double packDouble2x32 (uvec2 <i>v</i>)	
--	--

Returns a 2-component vector representation of v:

uvec2 unpackDouble2x32 (double <i>v</i>)	
--	--

Returns a uint by converting the components of a two-component floating-point vector:

uint packHalf2x16 (vec2 <i>v</i>)	
---	--

Returns a two-component floating-point vector:

vec2 unpackHalf2x16 (uint <i>v</i>)	
---	--

(Continue ↓)

Type Abbreviations for Built-in Functions:

Tf=float, vecn. Td=double, dvecn. Tfd= float, vecn, double, dvecn. Tb=bool, bvecn. Tu=uint, uvecn. Ti=int, ivecni. Tiu=int, ivecni, uint, uvecn. Tvec=vecn, uvecn, ivecni.

Within any one function, type sizes and dimensionality must correspond after implicit type conversions. For example, float **round**(float) is supported, but float **round**(vec4) is not.

Geometric Functions [8.5]

These functions operate on vectors as vectors, not component-wise. Tf=float, vecn. Td=double, dvecn. Tfd= float, vecn, double, dvecn.

float length (Tf <i>x</i>)	length of vector
double length (Td <i>x</i>)	
float distance (Tf <i>p0</i> , Tf <i>p1</i>)	distance between points
double distance (Td <i>p0</i> , Td <i>p1</i>)	
float dot (Tf <i>x</i> , Tf <i>y</i>)	dot product
double dot (Td <i>x</i> , Td <i>y</i>)	
vec3 cross (vec3 <i>x</i> , vec3 <i>y</i>)	cross product
dvec3 cross (dvec3 <i>x</i> , dvec3 <i>y</i>)	
Tfd normalize (Tfd <i>x</i>)	normalize vector to length 1
Tfd faceforward (Tfd <i>N</i> , Tfd <i>I</i> , Tfd <i>Nref</i>)	returns <i>N</i> if dot(<i>Nref</i> , <i>I</i>) < 0, else - <i>N</i>
Tfd reflect (Tfd <i>I</i> , Tfd <i>N</i>)	reflection direction $I - 2 * \text{dot}(N, I) * N$
Tfd refract (Tfd <i>I</i> , Tfd <i>N</i> , float <i>eta</i>)	refraction vector

Matrix Functions [8.6]

N and M are 1, 2, 3, 4.

mat matrixCompMult (mat <i>x</i> , mat <i>y</i>)	component-wise multiply
dmat matrixCompMult (dmat <i>x</i> , dmat <i>y</i>)	
matN outerProduct (vecN <i>c</i> , vecN <i>r</i>)	outer product (where $N \neq M$)
dmatN outerProduct (dvecN <i>c</i> , dvecN <i>r</i>)	
matNxM outerProduct (vecM <i>c</i> , vecN <i>r</i>)	outer product
dmatNxM outerProduct (dvecM <i>c</i> , dvecN <i>r</i>)	
matN transpose (matN <i>m</i>)	transpose
dmatN transpose (dmatN <i>m</i>)	
matNxM transpose (matMxN <i>m</i>)	transpose (where $N \neq M$)
dmatNxM transpose (dmatMxN <i>m</i>)	
float determinant (matN <i>m</i>)	determinant
double determinant (dmatN <i>m</i>)	
matN inverse (matN <i>m</i>)	inverse
dmatN inverse (dmatN <i>m</i>)	

Vector Relational Functions [8.7]

Compare x and y component-wise. Sizes of the input and return vectors for any particular call must match. Tvec=vecn, uvecn, ivecni.

bvecn lessThan (Tvec <i>x</i> , Tvec <i>y</i>)	<
bvecn lessThanEqual (Tvec <i>x</i> , Tvec <i>y</i>)	<=
bvecn greaterThan (Tvec <i>x</i> , Tvec <i>y</i>)	>
bvecn greaterThanEqual (Tvec <i>x</i> , Tvec <i>y</i>)	>=
bvecn equal (Tvec <i>x</i> , Tvec <i>y</i>)	==
bvecn equal (bvecn <i>x</i> , bvecn <i>y</i>)	
bvecn notEqual (Tvec <i>x</i> , Tvec <i>y</i>)	!=
bvecn notEqual (bvecn <i>x</i> , bvecn <i>y</i>)	
bool any (bvecn <i>x</i>)	true if any component of x is true
bool all (bvecn <i>x</i>)	true if all comps. of x are true
bvecn not (bvecn <i>x</i>)	logical complement of x

Integer Functions [8.8]

Component-wise operation. Tu=uint, uvecn. Ti=int, ivecni. Tiu=int, ivecni, uint, uvecn.

Adds 32-bit uint x and y, returning the sum modulo 2^{32} :	
Tu uaddCarry (Tu <i>x</i> , Tu <i>y</i> , out Tu <i>carry</i>)	
Subtracts y from x, returning the difference if non-negative, otherwise 2^{32} plus the difference:	
Tu usubBorrow (Tu <i>x</i> , Tu <i>y</i> , out Tu <i>borrow</i>)	
Multiplies 32-bit integers x and y, producing a 64-bit result:	
void umulExtended (Tu <i>x</i> , Tu <i>y</i> , out Tu <i>msb</i> , out Tu <i>lsb</i>)	
void imulExtended (Ti <i>x</i> , Ti <i>y</i> , out Ti <i>msb</i> , out Ti <i>lsb</i>)	
Extracts bits [offset, offset + bits - 1] from value, returns them in the least significant bits of the result:	
Tiu bitfieldExtract (Tiu <i>value</i> , int <i>offset</i> , int <i>bits</i>)	

(Continue ↓)

Integer Functions (cont.)

Returns the reversal of the bits of value:

Tiu bitfieldReverse (Tiu <i>value</i>)	
--	--

Inserts the bits least-significant bits of insert into base:

Tiu bitfieldInsert (Tiu <i>base</i> , Tiu <i>insert</i> , int <i>offset</i> , int <i>bits</i>)	
--	--

Returns the number of bits set to 1:

Ti bitCount (Tiu <i>value</i>)	
--	--

Returns the bit number of the least significant bit:

Ti findLSB (Tiu <i>value</i>)	
---------------------------------------	--

Returns the bit number of the most significant bit:

Ti findMSB (Tiu <i>value</i>)	
---------------------------------------	--

Texture Lookup Functions [8.9]

Available to vertex, geometry, and fragment shaders. See tables on next page.

Atomic-Counter Functions [8.10]

Returns the value of an atomic counter.

Atomically increments c then returns its prior value:	uint atomicCounterIncrement (atomic_uint <i>c</i>)
Atomically decrements c then returns its prior value:	uint atomicCounterDecrement (atomic_uint <i>c</i>)
Atomically returns the counter for c:	uint atomicCounter (atomic_uint <i>c</i>)
Atomic operations performed on c, where Op may be Add, Subtract, Min, Max, And, Or, Xor:	uint atomicCounterOp (atomic_uint <i>c</i> , uint <i>data</i>)
Atomically swap values of c and data; returns its prior value:	uint atomicCounterCompSwap (atomic_uint <i>c</i> , uint <i>data</i>)
Atomically compare values of c and compare; performs atomic swap if equal:	uint atomicCounterCompSwap (atomic_uint <i>c</i> , uint <i>compare</i> , uint <i>data</i>)

Atomic Memory Functions [8.11]

Operates on individual integers in buffer-object or shared-variable storage. OP is Add, Min, Max, And, Or, Xor, Exchange, or CompSwap.

uint atomicOP (coherent inout uint <i>mem</i> , uint <i>data</i>)	
int atomicOP (coherent inout int <i>mem</i> , int <i>data</i>)	

Image Functions [8.12]

In the image functions below, IMAGE_PARAMS may be one of the following:

image1D *image*, int *P*
 image2D *image*, ivec2 *P*
 image3D *image*, ivec3 *P*
 image2DRect *image*, ivec2 *P*
 imageCube *image*, ivec3 *P*
 imageBuffer *image*, int *P*
 image1DArray *image*, ivec2 *P*
 image2DArray *image*, ivec3 *P*
 imageCubeArray *image*, ivec3 *P*
 image2DMS *image*, ivec2 *P*, int *sample*
 image2DMSArray *image*, ivec3 *P*, int *sample*

Returns the dimensions of the images or images:

int imageSize (image(1D, Buffer) <i>image</i>)	
ivec2 imageSize (image(2D, Cube, Rect, 1DArray, 2DMS) <i>image</i>)	
ivec3 imageSize (image(Cube, 2D, 2DMS)Array <i>image</i>)	
vec3 imageSize (image(3D <i>image</i>)	

Returns the number of samples of the image or images bound to image:

int imageSamples (image(2DMS <i>image</i>)	
int imageSamples (image(2DMSArray <i>image</i>)	

Loads texel at the coordinate P from the image unit image:

gvec4 imageLoad (readonly IMAGE_PARAMS)	
--	--

Stores data into the texel at the coordinate P from the image specified by image:

void imageStore (writeonly IMAGE_PARAMS, gvec4 <i>data</i>)	
---	--

(Continued on next page) ►

Built-In Functions (cont.)**Image Functions (cont.)**

Adds the value of *data* to the contents of the selected texel:
 uint **imageAtomicAdd**(coherent *IMAGE_PARAMS*, uint *data*)
 int **imageAtomicAdd**(coherent *IMAGE_PARAMS*, int *data*)

Takes the minimum of the value of *data* and the contents of the selected texel:

```
uint imageAtomicMin(coherent IMAGE_PARAMS, uint data)
int imageAtomicMin(coherent IMAGE_PARAMS, int data)
```

Takes the maximum of the value *data* and the contents of the selected texel:

```
uint imageAtomicMax(coherent IMAGE_PARAMS, uint data)
int imageAtomicMax(coherent IMAGE_PARAMS, int data)
```

Performs a bit-wise AND of the value of *data* and the contents of the selected texel:

```
uint imageAtomicAnd(coherent IMAGE_PARAMS, uint data)
int imageAtomicAnd(coherent IMAGE_PARAMS, int data)
```

Performs a bit-wise OR of the value of *data* and the contents of the selected texel:

```
uint imageAtomicOr(coherent IMAGE_PARAMS, uint data)
int imageAtomicOr(coherent IMAGE_PARAMS, int data)
```

Performs a bit-wise exclusive OR of the value of *data* and the contents of the selected texel:

```
uint imageAtomicXor(coherent IMAGE_PARAMS, uint data)
int imageAtomicXor(coherent IMAGE_PARAMS, int data)
```

(Continue ↓)

Image Functions (cont.)

Copies the value of *data*:

```
uint imageAtomicExchange(coherent IMAGE_PARAMS,
    uint data)
int imageAtomicExchange(coherent IMAGE_PARAMS,
    int data)
int imageAtomicExchange(coherent IMAGE_PARAMS,
    float data)
```

Compares the value of *compare* and contents of selected texel. If equal, the new value is given by *data*; otherwise, it is taken from the original value loaded from texel:

```
uint imageAtomicCompSwap(coherent IMAGE_PARAMS,
    uint compare, uint data)
int imageAtomicCompSwap(coherent IMAGE_PARAMS,
    int compare, int data)
```

Fragment Processing Functions [8.13]

Available only in fragment shaders.

Tf=float, vecn.

Derivative fragment-processing functions

Tf dFdx (Tf p)	derivative in x and y, either fine or coarse derivatives
Tf dFdy (Tf p)	
Tf dFdxFine (Tf p)	fine derivative in x and y per pixel-row/column derivative
Tf dFdyFine (Tf p)	
Tf dFdxCoarse (Tf p)	coarse derivative in x and y per 2x2-pixel derivative
Tf dFdyCoarse (Tf p)	
Tf fwidth (Tf p)	sum of absolute values of x and y derivatives
Tf fwidthFine (Tf p)	
Tf fwidthCoarse (Tf p)	

Interpolation fragment-processing functions

Return value of *interpolant* sampled inside pixel and the primitive:

```
Tf interpolateAtCentroid(Tf interpolant)
```

Return value of *interpolant* at location of sample # *sample*:

```
Tf interpolateAtSample(Tf interpolant, int sample)
```

Return value of *interpolant* sampled at fixed offset *offset* from pixel center:

```
Tf interpolateAtOffset(Tf interpolant, vec2 offset)
```

Noise Functions [8.14]

Returns noise value. Available to fragment, geometry, and vertex shaders. *n* is 2, 3, or 4:

```
float noise1(Tf x)          vecn noisen(Tf x)
```

Geometry Shader Functions [8.15]

Only available in geometry shaders.

Emits values of output variables to current output primitive stream *stream*:

```
void EmitStreamVertex(int stream)
```

Completes current output primitive stream *stream* and starts a new one:

```
void EndStreamPrimitive(int stream)
```

Completes output primitive and starts a new one:

```
void EndPrimitive()
```

Emits values of output variables to the current output primitive:

```
void EmitVertex()
```

Other Shader Functions [8.16-17]

See diagram on page 11 for more information.

Synchronizes across shader invocations:

```
void barrier()
```

Controls ordering of memory transactions issued by a single shader invocation:

```
void memoryBarrier()
```

Controls ordering of memory transactions as viewed by other invocations in a compute work group:

```
void groupMemoryBarrier()
```

Order reads and writes accessible to other invocations:

```
void memoryBarrierAtomicCounter()
```

```
void memoryBarrierShared()
```

```
void memoryBarrierBuffer()
```

```
void memoryBarrierImage()
```

Shader Invocation Group Functions**[8.18]**

Available for multiple shader invocations grouped into a single SIMD invocation group.

Returns true if *value* is true for (any active invocation, all active invocations) in the group:

```
bool allInvocationsEqual(bool value)
```

```
bool allInvocation(bool value)
```

Returns true if *value* is the same for all active invocations in the group:

```
bool allInvocationsEqual(bool value)
```

Texture Functions [8.9]

Available to vertex, geometry, and fragment shaders. *ivec4*=*vec4*, *ivec4*, *uvec4*.

*gsampler**=*sampler**, *isampler**, *usampler**.

The *P* argument needs to have enough components to specify each dimension, array layer, or comparison for the selected sampler. The *dPdx* and *dPdy* arguments need enough components to specify the derivative for each dimension of the sampler.

Texture Query Functions [8.9.1]

textureSize functions return dimensions of *lod* (if present) for the texture bound to sampler. Components in return value are filled in with the width, height, depth of the texture. For array forms, the last component of the return value is the number of layers in the texture array.

```
{int,ivec2,ivec3} textureSize(
    gsampler1D(Array),2D(Rect,Array),Cube)sampler[,
    int lod])
```

```
{int,ivec2,ivec3} textureSize(
    gsampler(Buffer,2DMS(Array))sampler)
```

```
{int,ivec2,ivec3} textureSize(
    sampler1D, 2D, 2DRect,Cube[Array])Shadow sampler[,
    int lod])
```

```
ivec3 textureSize(samplerCubeArray sampler, int lod)
```

textureQueryLod functions return the mipmap array(s) that would be accessed in the *x* component of the return value. Returns the computed level of detail relative to the base level in the *y* component of the return value.

```
vec2 textureQueryLod(
    gsampler1D(Array),2D(Array),3D,Cube[Array]) sampler,
    {float,vec2,vec3} P)
```

```
vec2 textureQueryLod(
    sampler1D(Array),2D(Array),Cube[Array])Shadow sampler,
    {float,vec2,vec3} P)
```

textureQueryLevels functions return the number of mipmap levels accessible in the texture associated with *sampler*.

```
int textureQueryLevels(
    gsampler1D(Array),2D(Array),3D,Cube[Array]) sampler)
```

```
int textureQueryLevels(
    sampler1D(Array),2D(Array),Cube[Array])Shadow sampler)
```

textureSamples returns the number of samples of the texture.

```
int textureSamples(g_sampler2DMS sampler)
```

```
int textureSamples(g_sampler2DMSArray sampler)
```

Texel Lookup Functions [8.9.2]

Use texture coordinate *P* to do a lookup in the texture bound to *sampler*. For shadow forms, *compare* is used as *D_{ref}* and the array layer comes from *P.w*. For non-shadow forms, the array layer comes from the last component of *P*.

```
vec4 texture(
    gsampler1D(Array),2D(Array,Rect),3D,Cube[Array]) sampler,
    {float,vec2,vec3,vec4} P [, float bias])
```

```
float texture(
    sampler1D(Array),2D(Array,Rect),Cube)Shadow sampler,
    {vec3,vec4} P [, float bias])
```

```
float texture(g_samplerCubeArrayShadow sampler, vec4 P,
    float compare)
```

Texture lookup with projection.

```
vec4 textureProj(g_sampler1D,2D[Rect],3D) sampler,
    vec2,3,4) P [, float bias])
```

```
float textureProj(sampler1D,2D[Rect])Shadow sampler,
    vec4 P [, float bias])
```

Texture lookup as in **texture** but with explicit LOD.

```
vec4 textureLod(
    gsampler1D(Array),2D(Array),3D,Cube[Array]) sampler,
    {float,vec2,vec3} P, float lod)
```

```
float textureLod(sampler1D(Array),2D)Shadow sampler,
    vec3 P, float lod)
```

Offset added before texture lookup.

```
vec4 textureOffset(
    gsampler1D(Array),2D(Array,Rect),3D) sampler,
    {float,vec2,vec3} P, {int,ivec2,ivec3} offset [, float bias])
```

```
float textureOffset(
    sampler1D(Array),2D[Rect,Array])Shadow sampler,
    {vec3, vec4} P, {int,ivec2} offset [, float bias])
```

Use integer texture coordinate *P* to lookup a single texel from *sampler*.

```
vec4 texelFetch(
    gsampler1D(Array),2D(Array,Rect),3D) sampler,
    {int,ivec2,ivec3} P[, {int,ivec2} lod])
```

```
vec4 texelFetch(g_sampler(Buffer, 2DMS[Array]) sampler,
    {int,ivec2,ivec3} P[, int sample])
```

Fetch single texel with *offset* added before texture lookup.

```
vec4 texelFetchOffset(
    gsampler1D(Array),2D(Array),3D) sampler,
    {int,ivec2,ivec3} P, int lod, {int,ivec2,ivec3} offset)
```

```
vec4 texelFetchOffset(
    gsampler2DRect sampler, ivec2 P, ivec2 offset)
```

Projective texture lookup with *offset* added before texture lookup.

```
vec4 textureProjOffset(g_sampler1D,2D[Rect],3D) sampler,
    vec2,3,4) P, {int,ivec2,ivec3} offset [, float bias])
```

```
float textureProjOffset(
    sampler1D,2D[Rect])Shadow sampler, vec4 P,
    {int,ivec2} offset [, float bias])
```

Offset texture lookup with explicit LOD.

```
vec4 textureLodOffset(
    gsampler1D(Array),2D(Array),3D) sampler,
    {float,vec2,vec3} P, float lod, {int,ivec2,ivec3} offset)
```

```
float textureLodOffset(
    sampler1D(Array),2D)Shadow sampler, vec3 P, float lod,
    {int,ivec2} offset)
```

Projective texture lookup with explicit LOD.

```
vec4 textureProjLod(g_sampler1D,2D,3D) sampler,
    vec2,3,4) P, float lod)
```

```
float textureProjLod(sampler1D,2D)Shadow sampler,
    vec4 P, float lod)
```

Offset projective texture lookup with explicit LOD.

```
vec4 textureProjLodOffset(g_sampler1D,2D,3D) sampler,
    vec2,3,4) P, float lod, {int, ivec2, ivec3} offset)
```

```
float textureProjLodOffset(sampler1D,2D)Shadow sampler,
    vec4 P, float lod, {int, ivec2} offset)
```

Texture lookup as in **texture** but with explicit gradients.

```
vec4 textureGrad(
    gsampler1D(Array),2D[Rect,Array],3D,Cube[Array]) sampler,
    {float, vec2, vec3,vec4} P, {float, vec2, vec3} dPdx,
    {float, vec2, vec3} dPdy)
```

```
float textureGrad(
    sampler1D(Array),2D[Rect,Array], Cube)Shadow sampler,
    {vec3,vec4} P, {float,vec2} dPdx, {float,vec2, vec3} dPdy)
```

Texture lookup with both explicit gradient and offset.

```
vec4 textureGradOffset(
    gsampler1D(Array),2D[Rect,Array],3D) sampler,
    {float,vec2,vec3} P, {float,vec2,vec3} dPdx,
    {float,vec2,vec3} dPdy, {int,ivec2,ivec3} offset)
```

```
float textureGradOffset(
    sampler1D(Array),2D[Rect,Array])Shadow sampler,
    {vec3,vec4} P, {float,vec2} dPdx, {float,vec2}dPdy,
    {int,ivec2} offset)
```

Texture lookup both projectively as in **textureProj**, and with explicit gradient as in **textureGrad**.

```
vec4 textureProjGrad(g_sampler1D,2D[Rect],3D) sampler,
    {vec2,vec3,vec4} P, {float,vec2,vec3} dPdx,
    {float,vec2,vec3} dPdy)
```

```
float textureProjGrad(sampler1D,2D[Rect])Shadow sampler,
    vec4 P, {float,vec2} dPdx, {float,vec2} dPdy)
```

Texture lookup projectively and with explicit gradient as in **textureProjGrad**, as well as with offset as in **textureOffset**.

```
vec4 textureProjGradOffset(
    gsampler1D,2D[Rect],3D) sampler, vec2,3,4) P,
    {float,vec2,vec3} dPdx, {float,vec2,vec3} dPdy,
    {int,ivec2,ivec3} offset)
```

```
float textureProjGradOffset(
    sampler1D,2D[Rect]Shadow) sampler, vec4 P,
    {float,vec2} dPdx, {float,vec2} dPdy, {ivec2,int,vec2} offset)
```

Texture Gather Instructions [8.9.3]

These functions take components of a floating-point vector operand as a texture coordinate, determine a set of four texels to sample from the base level of detail of the specified texture image, and return one component from each texel in a four-component result vector.

```
vec4 textureGather(
    gsampler2D(Array,Rect),Cube[Array]) sampler,
    {vec2,vec3,vec4} P [, int comp])
```

```
vec4 textureGather(
    sampler2D(Array,Rect),Cube[Array])Shadow sampler,
    {vec2,vec3,vec4} P, float refZ)
```

Texture gather as in **textureGather** by offset as described in **textureOffset** except minimum and maximum offset values are given by {MIN, MAX}_PROGRAM_TEXTURE_GATHER_OFFSET.

```
vec4 textureGatherOffset(g_sampler2D[Array,Rect] sampler,
    {vec2,vec3} P, ivec2 offset [, int comp])
```

```
vec4 textureGatherOffset(
    sampler2D[Array,Rect]Shadow sampler,
    {vec2,vec3} P, float refZ, ivec2 offset)
```

Texture gather as in **textureGatherOffset** except *offsets* determines location of the four texels to sample.

```
vec4 textureGatherOffsets(g_sampler2D[Array,Rect] sampler,
    {vec2,vec3} P, ivec2 offsets[4] [, int comp])
```

```
vec4 textureGatherOffsets(
    sampler2D[Array,Rect]Shadow sampler,
    {vec2,vec3} P, float refZ, ivec2 offsets[4])
```

OpenGL API and OpenGL Shading Language Reference Card Index

The following index shows each item included on this card along with the page on which it is described. The color of the row in the table below is the color of the pane to which you should refer..

		DepthFunc	6	GetProgramInterfaceiv	2	PauseTransformFeedback	5
ActiveShaderProgram	2	DepthMask	6	GetProgramiv	2	PixelStore(i f)	2
ActiveTexture	2	DepthRange*	5	GetProgramPipeline*	2	PointParameter*	5
AttachShader	1	DetachShader	2	GetProgramResource*	2	PointSize	5
Android Platform	9	DisableVertexArrayAttrib	5	GetProgramStageiv	2	PolygonMode	5
B		DisableVertexAttribArray	5	GetQuery*	1	PolygonOffset	5
BeginConditionalRender	5	DispatchCompute*	5	GetRenderbufferParameteriv	4	PolygonOffsetClamp	5
BeginQuery	1, 6	DrawArrays[Indirect]	5	GetSamplerParameter*	2	PopDebugGroup	6
BeginQueryIndexed	1	DrawArraysInstanced[BaseInstance]	5	GetShaderInfoLog	2	Preprocessor	9
BeginTransformFeedback	5	DrawBuffer(s)	6	GetShaderiv	2	PrimitiveRestartIndex	5
BindAttribLocation	5	DrawElements*	5	GetShaderPrecisionFormat	2	ProgramBinary	2
BindBuffer*	1	DrawRangeElements[BaseVertex]	5	GetShaderSource	2	ProgramParameteri	2
BindFragDataLocation*	5	DrawTransformFeedback*	5	GetString	6-7	ProgramUniform*	2
BindFramebuffer	4	E		GetSubroutineIndex	2	ProvokingVertex	5
BindImageTexture(s)	4	EnableVertexArrayAttrib	5	GetSubroutineUniformLocation	2	PushDebugGroup	6
BindProgramPipeline	2	EnableVertexAttribArray	5	GetSynciv	1	Q-R	
BindRenderbuffer	4	EndConditionalRender	5	GetTexImage	3	Qualifiers	9-10
BindSampler(s)	2	EndQuery	6	GetTex[Level]Parameter*	3	QueryCounter	1
BindTexture*	2	EndQuery[Indexed]	1	GetTextureImage	3	ReadBuffer	6
BindTransformFeedback	5	EndTransformFeedback	5	GetTextureLevelParameter(i f)v	3	ReadnPixels	6
BindVertex*	4	Errors	1	GetTextureParameter*	3	ReadPixels	6
BlendColor	6	F		GetTextureSubImage	3	ReleaseShaderCompiler	1
BlendEquation*	6	Fences	1	GetTransformFeedback*	7	RenderbufferStorage	4
BlendFunc*	6	FenceSync	1	GetTransformFeedbackVarying	5	RenderbufferStorageMultisample	4
Blit[Named]Framebuffer	6	Finish	1	GetUniform*	2	ResumeTransformFeedback	5
BufferStorage	1	Flatshading	5	GetVertexArray*	5	S	
Buffer[Sub]Data	1	Flush	1	GetVertexAttrib*	5	SampleCoverage	6
C		FlushMapped[Named]BufferRange	1	H-I		SampleMaski	6
Check[Named]FramebufferStatus	4	FramebufferParameteri	4	Hint	6	SamplerParameter*	2
ClampColor	6	FramebufferRenderbuffer	4	InvalidateBufferData	1	Scissor	6
Clear	6	FramebufferTexture*	4	InvalidateBufferSubData	1	ShaderBinary	1
ClearBuffer[Sub]Data	1	FrontFace	5	InvalidateFramebuffer	6	ShaderSource	1
ClearBuffer*	6	G		InvalidateNamedFramebuffer*	6	ShaderStorageBlockBinding	2
ClearColor	6	GenBuffers	1	InvalidateSubFramebuffer	6	Shading Language	9-12
ClearDepth*	6	Generate[Texture]Mipmap	3	InvalidateTexImage	4	SpecializeShader	1
ClearNamedBuffer[Sub]Data	1	GenFramebuffers	4	InvalidateTexSubImage	4	State and State Requests	6-7
ClearNamedFramebuffer*	6	GenProgramPipelines	2	IsBuffer	1	StencilFunc*	6
ClearStencil	6	GenQueries	1	IsFramebuffer	4	StencilMask*	6
ClearTexImage	4	GenRenderbuffers	4	IsProgram[Pipeline]	2	StencilOp*	6
ClearTexSubImage	4	GenSamplers	2	IsQuery	1	Synchronization	1
ClientWaitSync	1	GenTextures	2	IsRenderbuffer	4	Synchronous Debug Output	6
ClipControl	5	GenTransformFeedbacks	5	IsSampler	2	T	
ColorMask*	6	GenVertexArrays	4	IsShader	1	TexBuffer[Range]	3
Command Syntax	1	GetActiveAtomicCounterBufferiv	2	IsSync	1	TexImage*[Multisample]	3
CompileShader	1	GetActiveAttrib	5	IsTexture	2	TexParameter*	3
CompressedTex[Sub]Image*	3	GetActiveSubroutine*	2	IsTransformFeedback	5	TexStorage*[Multisample]	3-4
CompressedTextureSubImage*	3	GetActiveUniform*	2	IsVertexArray	4	TexSubImage*	3
CopyBufferSubData	1	GetAttachedShaders	2	L		TextureBarrier	4
CopyImageSubData	6	GetAttribLocation	5	LineWidth	5	TextureBuffer[Range]	3
CopyNamedBufferSubData	1	GetBoolean*	6	LinkProgram	2	TextureParameter*	3
CopyTex[Sub]Image*	3	GetBufferParameteri[64]v	1	M		TextureStorage*[Multisample]	4
CopyTextureSubImage*	3	GetBufferPointerv	1	Macros	9	TextureSubImage*	3
CreateBuffers	1	GetBufferSubData	1	MapBuffer[Range]	1	TextureView	3
CreateFramebuffers	4	GetCompressedTex*	3	MapNamedBuffer*	1	Timer Queries	1
CreateProgram	1	GetDebugMessageLog	6	MemoryBarrier	2	TransformFeedbackBufferBase	5
CreateProgramPipelines	2	GetDouble*	6	MemoryBarrierByRegion	2	TransformFeedbackBufferRange	5
CreateQueries	1	GetError	1	MinSampleShading	5	TransformFeedbackVarying(s)	5
CreateRenderbuffers	4	GetFloat*	6	MultiDrawArrays*	5	Types	9
CreateSamplers	2	GetFragData*	5	MultiDrawElements*	5	U	
CreateShader	1	GetFramebuffer[Attachment]Parameter*	4	Multisampling	5	Uniform*	2
CreateShaderProgramv	2	GetGraphicsResetStatus	1	N		UniformBlockBinding	2
CreateTextures	2	GetInteger[64]v	1	NamedBuffer[Sub]Data	1	UniformMatrix*	2
CreateTransformFeedbacks	5	GetInteger64i_v	6	NamedBufferStorage	1	UniformSubroutinesuiv	2
CreateVertexArrays	4	GetInteger64v	6	NamedFramebufferDrawBuffer(s)	6	Unmap[Named]Buffer	1
CullFace	5	GetIntegeri_v	6	NamedFramebufferParameteri	4	UseProgram[Stages]	2
D		GetIntegerv	1	NamedFramebufferReadBuffer	6	V-W	
DebugMessage*	6	GetIntegerv	6	NamedFramebufferRenderbuffer	4	ValidateProgram[Pipeline]	5
DeleteBuffers	1	GetInternalformativ*	7	NamedFramebufferTexture*	4	Variables, built-in	10
DeleteFramebuffers	4	GetMultisamplefv	5	NamedRenderbufferStorage*	4	VertexArrayAttrib*Format	4
DeleteProgram*	2	GetNamedBuffer*	1	O		VertexArrayAttribBindin	5
DeleteProgramPipelines	2	GetNamedFramebuffer*	4	ObjectLabel	6	VertexArrayBindingDivisor	5
DeleteQueries	1	GetNamedRenderbufferParameteriv	4	ObjectPtrLabel	6	VertexArrayElementBuffer	4
DeleteRenderbuffers	4	GetnCompressedTexImage	3	OpenGL Pipeline	8	VertexArrayVertexBuffers(s)	4
DeleteSamplers	2	GetnTexImage	3	Operations and Constructors	10	VertexAttrib*	4-5
DeleteShader	1	GetnUniform(f d i u)v	2	Operators and Expressions	9	VertexBindingDivisor	5
DeleteSync	1	GetObject[Ptr]Label	6	P		Viewpoint*	5
DeleteTextures	2	GetPointerv	6	PatchParameterfv	5	WaitSync	1
DeleteTransformFeedbacks	5	GetProgramBinary	2	PatchParameteri	4		
DeleteVertexArrays	4	GetProgramInfoLog	2				



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.