

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

Specifications are available at www.opengl.org/registry



- *See [FunctionName](#)* refers to functions on this reference card.
- [\[n.n.n\]](#) and [\[Table n.n\]](#) refer to sections and tables in the OpenGL 4.5 core specification.
- [\[n.n.n\]](#) refers to sections in the OpenGL Shading Language 4.50 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

GetIntegerv

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 exponent, 10-bit mantissa
Unsigned 11-Bit	no sign bit, 5-bit exponent, 6-bit mantissa
Unsigned 10-Bit	no sign bit, 5-bit exponent, 5-bit mantissa

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)

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_{UN}PACK_BUFFER, 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, WRITE}_BIT, {DYNAMIC, CLIENT}_STORAGE_BIT, MAP_{COHERENT, PERSISTENT}_BIT

void [NamedBufferStorage](#)(uint buffer, sizeptr size, const void *data, bitfield flags);

flags: [See BufferStorage](#)

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

void [CreateQueries](#)(enum target, 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, TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN

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 [GetQueryObjectiiv](#)(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]

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 [GetIntegerv](#)([TIMESTAMP](#), int *data);

void [GetInteger64v](#)([TIMESTAMP](#), int64 *data);

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);

target: [See BindBuffer](#)

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 [GetBufferParameteri](#)[64]v(enum target, enum pname, int[64] *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 [GetNamedBufferParameteri](#)[64]v(uint buffer, enum pname, int[64] *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);

(Continued on next page) ►

◀ Shaders and Programs (cont.)

Program Objects [7.3]

```
uint CreateProgram(void);

void AttachShader(uint program, uint shader);

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 pname,
                          int *params);
pname: PROGRAM_INTERFACE;
value: TRUE, FALSE

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

void GetProgramResourceName(
    uint program, enum pname,
    const char *name);

void GetProgramResourceiv(
    uint program, enum pname,
    const char *name,
    sizei bufSize, sizei *length,
    int *params);
pname: ACTIVE_RESOURCES, MAX_NAME_LENGTH,
MAX_NUM_ACTIVE_VARIABLES,
MAX_NUM_COMPATIBLE_SUBROUTINES
```

```
uint GetProgramResourceLocation(
    uint program, enum pname,
    const char *name);

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

void GetProgramResourceName(
    uint program, enum pname,
    const char *name);

void GetProgramResourceiv(
    uint program, enum pname,
    const char *name,
    sizei bufSize, sizei *length,
    int *params);
pname: ACTIVE_RESOURCES, MAX_NAME_LENGTH,
MAX_NUM_ACTIVE_VARIABLES,
MAX_NUM_COMPATIBLE_SUBROUTINES
```

Program Pipeline Objects [7.4]

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

void DeleteProgramPipelines(sizei n,
                          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 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 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}ui(
    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 UniformSubroutinesiv(
    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);
```

```
void SamplerParameter{i f}(uint sampler,
                        enum pname, T param);
pname: TEXTURE_X where X may be WRAP_{S, T, R},
{MIN, MAG}_FILTER, {MIN, MAX}_LOD,
BORDER_COLOR, LOD_BIAS,
COMPARE_{MODE, FUNC} [Table 23.18]
```

```
void SamplerParameter{i f}v(uint sampler,
                            enum pname, const T *param);
pname: See SamplerParameter{i f}
```

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

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

```
boolean IsSampler(uint sampler);
```

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, COMPILER}_STATUS, COMPUTE_SHADER,
SHADER_SOURCE_LENGTH
```

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

```
void GetUniform{i 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 SamplerParameter{i f}(uint sampler,
                        enum pname, T param);
pname: TEXTURE_X where X may be WRAP_{S, T, R},
{MIN, MAG}_FILTER, {MIN, MAX}_LOD,
BORDER_COLOR, LOD_BIAS,
COMPARE_{MODE, FUNC} [Table 23.18]
```

```
void SamplerParameter{i f}v(uint sampler,
                            enum pname, const T *param);
pname: See SamplerParameter{i f}
```

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

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

```
boolean IsSampler(uint sampler);
```

Sampler Queries [8.3]

```
void GetSamplerParameter{i f}v(
    uint sampler, enum pname, T *params);
pname: See SamplerParameter{i f}
```

```
void GetSamplerParameter{i ui}v(
    uint sampler, enum pname, const T *params);
pname: See SamplerParameter{i f}
```

Pixel Storage Modes [8.4.1]

```
void PixelStore{i f}(enum pname, T param);
pname: [Tables 8.1, 18.1] {UNPACK_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.)

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](#)

void **TextureStorage3D**(uint *texture*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*);
internalformat: [See TexStorage1D](#)

void **TexStorage2DMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, boolean *fixedsamplelocations*);
target: TEXTURE_2D_MULTISAMPLE

void **TexStorage3DMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, boolean *fixedsamplelocations*);
target: TEXTURE_2D_MULTISAMPLE_ARRAY

void **TextureStorage2DMultisample**(uint *texture*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, boolean *fixedsamplelocations*);

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), RG(32,16,8), R(32,16,8), 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*);

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

attachment: DEPTH, FRONT_LEFT, RIGHT, STENCIL, BACK_LEFT, RIGHT, COLOR_ATTACHMENT, {DEPTH, STENCIL, DEPTH_STENCIL_ATTACHMENT
pname: FRAMEBUFFER_ATTACHMENT_X where X may be OBJECT_TYPE, NAME, COMPONENT_TYPE, {RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL}_SIZE, COLOR_ENCODING, TEXTURE_LAYER, LEVEL, LAYERED, TEXTURE_CUBE_MAP_FACE

void **GetNamedFramebufferAttachmentParameteriv**(uint *framebuffer*, enum *attachment*, enum *pname*, int **params*);
attachment, *pname*: [See GetFramebufferParameteriv](#)

Renderbuffer Objects [9.2.4]

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

void **{Create, Gen}Renderbuffers**(sizei *n*, uint **renderbuffers*);

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

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

void **RenderbufferStorageMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*);
target: RENDERBUFFER
internalformat: [See TexImage3DMultisample](#)

void **NamedRenderbufferStorageMultisample**(uint *renderbuffer*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*);
internalformat: [See TexImage3DMultisample](#)

void **RenderbufferStorage**(enum *target*, enum *internalformat*, sizei *width*, sizei *height*);
target: RENDERBUFFER
internalformat: [See TexImage3DMultisample](#)

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](#)

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.

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

void **VertexAttribI{1234}{i ui}v**(uint *index*, const T **values*);

void **VertexAttribI4{b s ub us}v**(uint *index*, const T **values*);

void **VertexAttribI4Nub**(uint *index*, const T *values*);

void **VertexAttribI4N{b s i ub us ui}v**(uint *index*, const T **values*);

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

void **VertexAttribP{1234}uiv**(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 VertexAttribP4uiv)]

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*);

Generic Vertex Attribute Arrays [10.3.2]
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 **VertexArrayAttribFormat**(uint *vaobj*, uint *attribindex*, int *size*, enum *type*, boolean *normalized*, unit *relativeoffset*);
type: [See VertexAttribFormat](#)

void **VertexArrayAttribIFormat**(uint *vaobj*, uint *attribindex*, int *size*, enum *type*, unit *relativeoffset*);
type: [See VertexAttribIFormat](#)

void **VertexArrayAttribLFormat**(uint *vaobj*, uint *attribindex*, int *size*, enum *type*, unit *relativeoffset*);
type: [See VertexAttribLFormat](#)

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

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

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

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

void **EnableVertexAttribArray**(uint index);

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

void **DisableVertexAttribArray**(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 **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);

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

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 **GetVertexAttrib(d f i)iv**(uint index, enum pname, T *params);
pname: See [GetVertexArrayIndexdiv](#) plus VERTEX_ATTRIB_ARRAY_BUFFER_BINDING, VERTEX_ATTRIB_BINDING, CURRENT_VERTEX_ATTRIB

void **GetVertexAttrib(i ui)iv**(uint index, enum pname, T *params);
pname: See [GetVertexAttrib\(d f i\)iv](#)

void **GetVertexAttribLdv**(uint index, enum pname, double *params);
pname: See [GetVertexAttrib\(d f i\)iv](#)

void **GetVertexAttribPointerv**(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);

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 **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

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

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

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 **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_DISTANCEi 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

Controlling Viewport [13.6.1]

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

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

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

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

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\(if\)fv](#)

void **PointParameter(i f)fv**(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 **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 **PolygonOffset**(float factor, float units);

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

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

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);

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

buffer: COLOR, DEPTH, STENCIL

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

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

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

buffer: DEPTH_STENCIL

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

buffer: See **ClearBuffer**

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 **GetBooleanyv**(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, 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);

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_ATTACHMENTi (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_ATTACHMENTi (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_ATTACHMENTi (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_UNSIGNED_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);

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 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

(Continued on next page) ►

◀ States (cont.)

```
ubyte *GetStringi(enum name, uint index);
name: EXTENSIONS, SHADING_LANGUAGE_VERSION
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, TEXTURE_2D, TEXTURE_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,
INTERNALFORMAT_{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
```

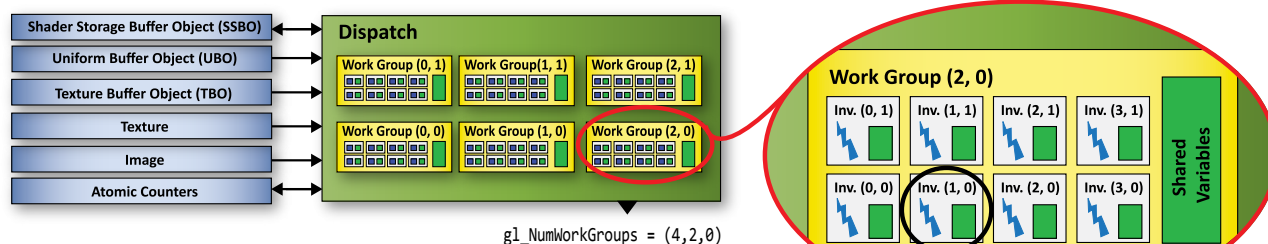
TransformFeedback Queries [22.4]

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

```
void GetTransformFeedbacki_v(uint xfb,
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
```

OpenGL Compute Programming Model and Compute Memory Hierarchy



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

```
void barrier();
```

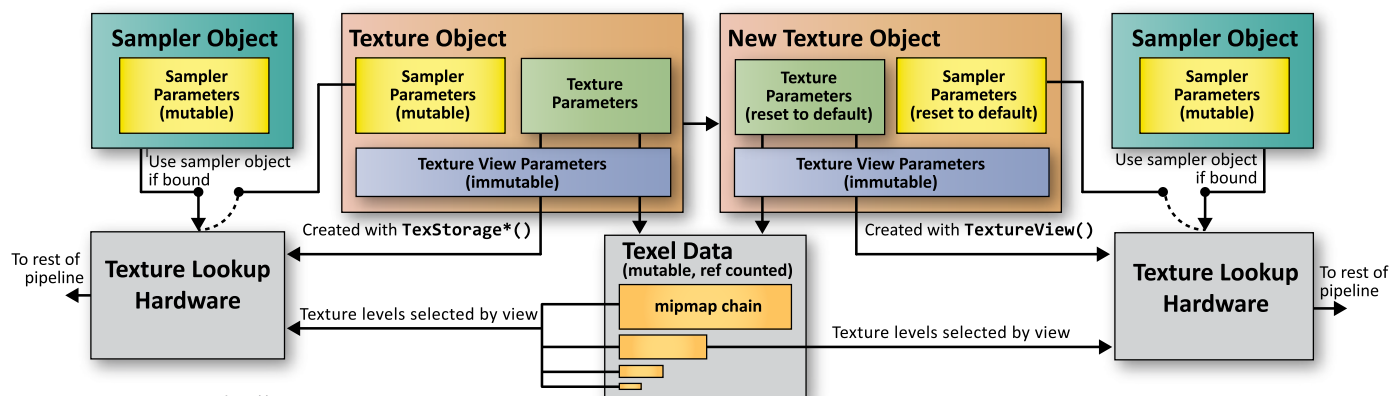
Use the `memoryBarrier*` or `groupMemoryBarrier` functions to order reads/writes accessible to other invocations:

```
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_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)

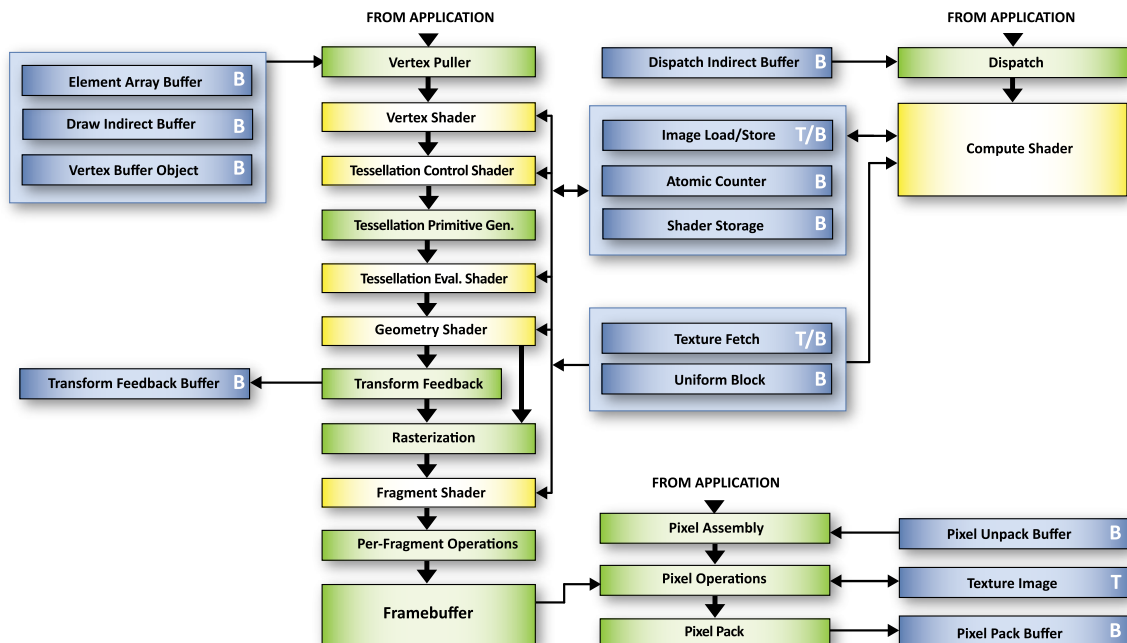
```
<target>
TEXTURE_INTERNAL_FORMAT TEXTURE_SHARED_SIZE
TEXTURE_VIEW_{MIN, NUM}_LEVEL TEXTURE_VIEW_{MIN, NUM}_LAYER
TEXTURE_IMMUTABLE_LEVELS IMAGE_FORMAT_COMPATIBILITY_TYPE
TEXTURE_{RED, GREEN, BLUE, ALPHA, DEPTH}_TYPE
TEXTURE_{RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL}_SIZE
```

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.

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



Vertex & Tessellation Details

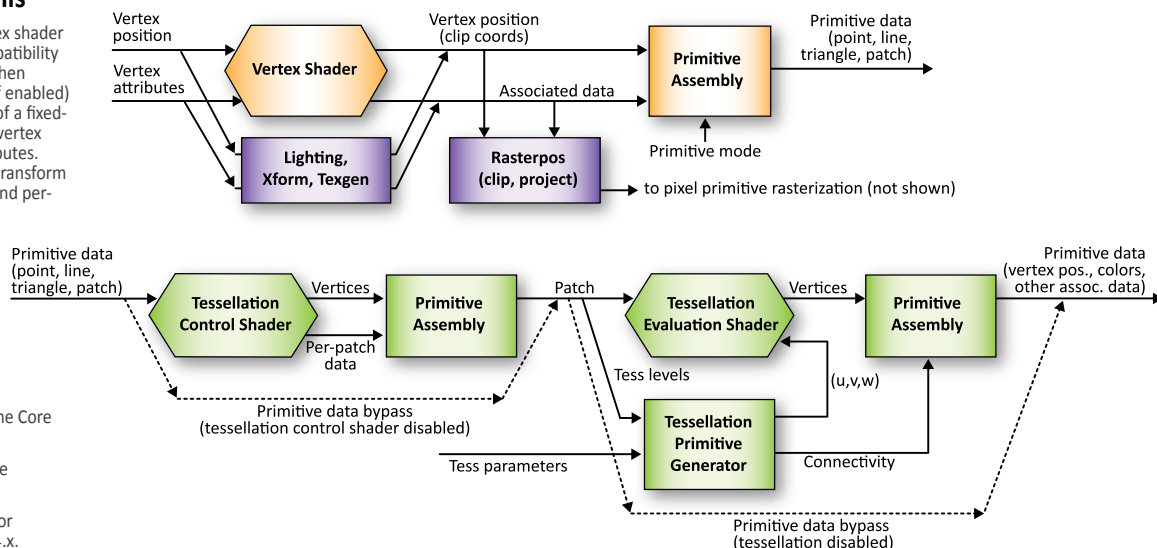
Each vertex is processed either by a vertex shader or fixed-function vertex processing (compatibility only) to generate a transformed vertex, then assembled into primitives. Tessellation (if enabled) operates on patch primitives, consisting of a fixed-size collection of vertices, each with per-vertex 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.

Orange blocks indicate features of the Core specification.

Purple blocks indicate features of the Compatibility specification.

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.

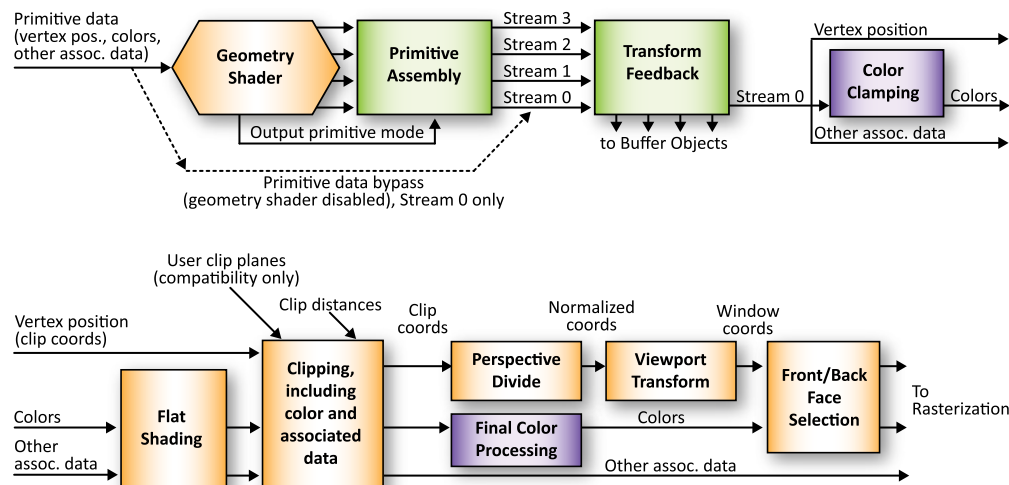
Transform feedback (if active) writes selected vertex attributes of the primitives of all vertex streams into buffer objects attached to one or more binding points.

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

Orange blocks indicate features of the Core specification.

Purple blocks indicate features of the Compatibility specification.

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 4.50 specification at www.khronos.org/registry

Preprocessor [3.3]

Preprocessor Operators

#version 450	Required when using version 4.50.
#version 450 <i>profile</i>	<i>profile</i> is core, compatibility, or es (for ES versions 1.00, 3.00, or 3.10).
#extension <i>extension_name</i> : <i>behavior</i>	<ul style="list-style-type: none"> <i>behavior</i>: require, enable, warn, disable <i>extension_name</i>: extension supported by compiler, or "all"
#extension all : <i>behavior</i>	

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.

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.

{x, y, z, w}	Points or normals
{r, g, b, a}	Colors
{s, t, p, q}	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	signed and unsigned integer vectors
uvec2, uvec3, uvec4	
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

sampler{1D,2D,3D}	1D, 2D, or 3D texture
image{1D,2D,3D}	cube mapped texture
samplerCube	cube mapped texture
sampler2DRect	rectangular texture
image2DRect	
sampler{1D,2D}Array	1D or 2D array texture
image{1D,2D}Array	
samplerBuffer	buffer texture
imageBuffer	
sampler2DMS	2D multi-sample texture
image2DMS	
sampler2DMSArray	2D multi-sample array texture
image2DMSArray	
samplerCubeArray	cube map array texture
imageCubeArray	
sampler1DShadow	1D or 2D depth texture with comparison
sampler2DShadow	
sampler2DRectShadow	rectangular tex. / compare
sampler1DArrayShadow	1D or 2D array depth texture with comparison
sampler2DArrayShadow	
samplerCubeShadow	cube map depth texture with comparison
samplerCubeArrayShadow	cube map array depth texture with comparison

Signed Integer Opaque Types

isampler{1,2,3}D	integer 1D, 2D, or 3D texture
iimage{1,2,3}D	integer 1D, 2D, or 3D image
isamplerCube	integer cube mapped texture
iimageCube	integer cube mapped image
isampler2DRect	int. 2D rectangular texture

Continue ↴

Signed Integer Opaque Types (cont'd)

iimage2DRect	int. 2D rectangular image
isampler{1,2}DArray	integer 1D, 2D array texture
iimage{1,2}DArray	integer 1D, 2D array image
isamplerBuffer	integer buffer texture
imageBuffer	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
usampler{1,2,3}D	uint 1D, 2D, or 3D texture
uimage{1,2,3}D	uint 1D, 2D, or 3D image
usamplerCube	uint cube mapped texture
uimageCube	uint cube mapped image
usampler2DRect	uint rectangular texture
uimage2DRect	uint rectangular image
usampler{1,2}DArray	uint 1D, 2D, or 3D array texture
uimage{1,2}DArray	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.

Continue ↴

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	uvec2	-> dvec2
int, uint	-> float	uvec3	-> dvec3
int, uint, float	-> double	uvec4	-> 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	-> dmat4x4

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	
component =		X		X	all in/out, except for compute
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				geometry in/out
points	X				
[points], lines, triangles,	X				geometry in
{triangles, lines}_adjacency					
invocations =	X				geometry in

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

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) ►

Examples of operations on matrices and vectors:

Examples of operations on matrices and vectors:

```
m = * 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
```

```
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:

<code>==</code> <code>!=</code>	equality
<code>=</code>	assignment
<code>[]</code>	indexing (arrays only)

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

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

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

```
subroutine uniform subroutineTypeName
    subroutineVarName;
```

Declare types with the **subroutine** keyword:

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

```

```
const int gl_MaxTessControlOutputComponents = 128;
```

```
const int gl_MaxTessControlOutputComponents = 128;
const int gl_MaxTessControlTextureImageUnits = 16;
const int gl_MaxTessControlUniformComponents = 1024;
const int gl_MaxTessControlTotalOutputComponents = 4096;
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;
```

```
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_MaxComputeImageUnits = 8;
const int gl_MaxComputeAtomicCounters = 8;
const int gl_MaxComputeAtomicCounterBuffers = 1;
const int gl_MaxVertexAttribs = 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_MaxTexelControlImageUniforms = 0;
const int gl_MaxTexelEvaluationImageUniforms = 0;
const int gl_MaxGeometryImageUniforms = 0;
const int gl_MaxFragmentImageUniforms = 8;
const int gl_MaxCombinedImageUniforms = 8;
const int gl_MaxFragmentUniformComponents = 1024;
const int gl_MaxDrawBuffers = 8;
const int gl_MaxClipDistances = 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_MaxTexelControlInputComponents = 128;
```

```
const int gl_MaxComputeUniformComponents = 1024;
const int gl_MaxComputeTextureImageUnits = 16;
const int gl_MaxComputeImageUnits = 8;
const int gl_MaxComputeAtomicCounters = 8;
const int gl_MaxComputeAtomicCounterBuffers = 1;
const int gl_MaxVertexAttribs = 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_MaxTexelControlImageUniforms = 0;
const int gl_MaxTexelEvaluationImageUniforms = 0;
const int gl_MaxGeometryImageUniforms = 0;
const int gl_MaxFragmentImageUniforms = 8;
const int gl_MaxCombinedImageUniforms = 8;
const int gl_MaxFragmentUniformComponents = 1024;
const int gl_MaxDrawBuffers = 8;
const int gl_MaxClipDistances = 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_MaxTexelControlInputComponents = 128;
```

```
const int gl_MaxVertexImageUniforms = 0;
const int gl_MaxTexControlImageUniforms = 0;
const int gl_MaxTexEvaluationImageUniforms = 0;
const int gl_MaxGeometryImageUniforms = 0;
const int gl_MaxFragmentImageUniforms = 8;
const int gl_MaxCombinedImageUniforms = 8;
const int gl_MaxFragmentUniformComponents = 1024;
const int gl_MaxDrawBuffers = 8;
const int gl_MaxClipDistances = 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_MaxTexControlInputComponents = 128;
```

```
const int gl_MaxGeometryImageUniforms = 0;
const int gl_MaxFragmentImageUniforms = 8;
const int gl_MaxCombinedImageUniforms = 8;
const int gl_MaxFragmentUniformComponents = 1024;
const int gl_MaxDrawBuffers = 8;
const int gl_MaxClipDistances = 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_Max TessControlInputComponents = 128;
```

```
const int gl_MaxCombinedImageUniforms = 8;
const int gl_MaxFragmentUniformComponents = 1024;
const int gl_MaxDrawBuffers = 8;
const int gl_MaxClipDistances = 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_Max TessControlInputComponents = 128;
```


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</i> , Tf <i>over_x</i>)	arc tangent
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, ivecn. 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> , Ti <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>)	
---	--

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, ivecn. Tiu=int, ivecn, uint, uvecn. Tvec=vecn, uvecn, ivecn.

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, ivecn.

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, ivecn. Tiu=int, ivecn, 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>)	
--	--

(Continue ↓)

Integer Functions (cont.)

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>)	
--	--

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 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:

gimage1D *image*, int *P*
 gimage2D *image*, ivec2 *P*
 gimage3D *image*, ivec3 *P*
 gimage2DRect *image*, ivec2 *P*
 gimageCube *image*, ivec3 *P*
 gimageBuffer *image*, int *P*
 gimage1DArray *image*, ivec2 *P*
 gimage2DArray *image*, ivec3 *P*
 gimageCubeArray *image*, ivec3 *P*
 gimage2DMS *image*, ivec2 *P*, int sample
 gimage2DMSArray *image*, ivec3 *P*, int sample

Returns the dimensions of the images or images:

int imageSize (gimage1D, Buffer) <i>image</i>)	
ivec2 imageSize (gimage2D, Cube, Rect, 1DArray, 2DMS) <i>image</i>)	
ivec3 imageSize (gimageCube, 2D, 2DMS)Array <i>image</i>)	
vec3 imageSize (gimage3D <i>image</i>)	

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

int imageSamples (gimage2DMS <i>image</i>)	
int imageSamples (gimage2DMSArray <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 <i>p</i>)	derivative in <i>x</i> and <i>y</i> , either fine or coarse derivatives
Tf dFdy (Tf <i>p</i>)	
Tf dFdxFine (Tf <i>p</i>)	fine derivative in <i>x</i> and <i>y</i> per pixel-row/column derivative
Tf dFdyFine (Tf <i>p</i>)	
Tf dFdxCoarse (Tf <i>p</i>)	coarse derivative in <i>x</i> and <i>y</i> per 2x2-pixel derivative
Tf dFdyCoarse (Tf <i>p</i>)	
Tf fwidth (Tf <i>p</i>)	sum of absolute values of <i>x</i> and <i>y</i> derivatives
Tf fwidthFine (Tf <i>p</i>)	
Tf fwidthCoarse (Tf <i>p</i>)	

Interpolation fragment-processing functionsReturn 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*)

(Continue ↓)

Geometry Shader Functions (cont'd)

Emits values of output variables to the current output primitive:

void **EmitVertex**()

Completes output primitive and starts a new one:

void **EndPrimitive**()**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**()**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(
gsampler{1D[Array],2D[Rect,Array],Cube}sampler[,
int lod])
{int,ivec2,ivec3} textureSize(
gsampler[Buffer,2DMS[Array]]sampler)
{int,ivec2,ivec3} textureSize(
sampler{1D, 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(
gsampler{1D[Array],2D[Array],3D,Cube[Array]} sampler,
{float,vec2,vec3} P)
vec2 textureQueryLod(
sampler{1D[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(
gsampler{1D[Array],2D[Array],3D,Cube[Array]} sampler)
int textureQueryLevels(
sampler{1D[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(
gsampler{1D[Array],2D[Array,Rect],3D,Cube[Array]} sampler,
{float,vec2,vec3,vec4} P [, float bias])
float texture(
sampler{1D[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_sampler{1D,2D[Rect],3D}sampler,
vec{2,3,4} P [, float bias])
float textureProj(sampler{1D,2D[Rect]}Shadow sampler,
vec4 P [, float bias])
```

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

```
vec4 textureLod(
gsampler{1D[Array],2D[Array],3D,Cube[Array]} sampler,
{float,vec2,vec3} P, float lod)
float textureLod(sampler{1D[Array],2D}Shadow sampler,
vec3 P, float lod)
```

Offset added before texture lookup.

```
vec4 textureOffset(
gsampler{1D[Array],2D[Array,Rect],3D}sampler,
{float,vec2,vec3} P, {int,ivec2,ivec3} offset [, float bias])
float textureOffset(
sampler{1D[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(
gsampler{1D[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(
gsampler{1D[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_sampler{1D,2D[Rect],3D}sampler,
vec{2,3,4} P, {int,ivec2,ivec3} offset [, float bias])
float textureProjOffset(
sampler{1D,2D[Rect]}Shadow sampler, vec4 P,
{int,ivec2} offset [, float bias])
```

Offset texture lookup with explicit LOD.

```
vec4 textureLodOffset(
gsampler{1D[Array],2D[Array],3D}sampler,
{float,vec2,vec3} P, float lod, {int,ivec2,ivec3} offset)
float textureLodOffset(
sampler{1D[Array],2D}Shadow sampler, vec3 P, float lod,
{int,ivec2} offset)
```

Projective texture lookup with explicit LOD.

```
vec4 textureProjLod(g_sampler{1D,2D,3D}sampler,
vec{2,3,4} P, float lod)
float textureProjLod(sampler{1D,2D}Shadow sampler,
vec4 P, float lod)
```

Offset projective texture lookup with explicit LOD.

```
vec4 textureProjLodOffset(g_sampler{1D,2D,3D}sampler,
vec{2,3,4} P, float lod, {int, ivec2, ivec3} offset)
float textureProjLodOffset(sampler{1D,2D}Shadow sampler,
vec4 P, float lod, {int, ivec2} offset)
```

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

```
vec4 textureGrad(
gsampler{1D[Array],2D[Rect,Array],3D,Cube[Array]} sampler,
{float,vec2,vec3,vec4} P, {float,vec2,vec3} dPdx,
{float,vec2,vec3} dPdy)
float textureGrad(
sampler{1D[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(
gsampler{1D[Array],2D[Rect,Array],3D}sampler,
{float,vec2,vec3} P, {float,vec2,vec3} dPdx,
{float,vec2,vec3} dPdy, {int,ivec2,ivec3} offset)
float textureGradOffset(
sampler{1D[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_sampler{1D,2D[Rect],3D}sampler,
{vec2,vec3,vec4} P, {float,vec2,vec3} dPdx,
{float,vec2,vec3} dPdy)
float textureProjGrad(sampler{1D,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(
gsampler{1D,2D[Rect],3D}sampler, vec{2,3,4} P,
{float,vec2,vec3} dPdx, {float,vec2,vec3} dPdy,
{int,ivec2,ivec3} offset)
float textureProjGradOffset(
sampler{1D,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(
gsampler{2D[Array,Rect],Cube[Array]} sampler,
{vec2,vec3,vec4} P [, int comp])
vec4 textureGather(
sampler{2D[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.

A	CreateTransformFeedbacks	5	GetActiveAtomicCounterBuffer*	2	L	SamplerParameter*	2
ActiveShaderProgram	2	CreateVertexArrays	4	GetActiveAttrib	5	Layout Qualifiers	9
ActiveTexture	2	CullFace	5	GetActiveSubroutine*	2	LineWidth	5
Angle Functions	11			GetActiveUniform*	2	LinkProgram	2
Asynchronous Queries	1	D		GetAttachedShaders	2	LogicOp	6
Atomic Functions	11	DebugMessage*	6	GetAttribLocation	5		
AttachShader	2	DeleteBuffers	1	GetBoolean*	6	M	
		DeleteFramebuffers	4	GetBufferParameter*	1	Macros	9
		DeleteProgram[Pipelines]	2	GetBuffer[Pointerv, SubData]	1	Map[Named]Buffer*	1
B		DeleteQueries	1	Get[n]CompressedTexImage	3	Matrix Operations	10
BeginConditionalRender	5	DeleteRenderbuffers	4	GetCompressedTexSubImage	3	Matrix Functions	11
BeginQuery[Indexed]	1	DeleteSamplers	2	GetCompressedTextureImage	3	MemoryBarrier	2
BeginTransformFeedback	5	DeleteShader	1	GetDebugMessageLog	6	MemoryBarrier	12
BindAttribLocation	5	DeleteSync	1	GetDouble*	6	Memory Qualifiers	10
BindBuffer*	1	DeleteTextures	2	GetError	1	MinSampleShading	5
BindBuffer[s]{Base, Range}	1	DeleteTransformFeedbacks	5	GetFloat*	6	MultiDraw[Arrays, Elements]*	5
BindFragDataLocation[Indexed]	5	DeleteVertexArrays	4	GetFragData*	5	Multisample Fragment Ops	6
BindFramebuffer	4	DepthFunc	6	GetFramebuffer*	4	Multisample Textures	3
BindImageTexture[s]	4	DepthMask	6	GetGraphicsResetStatus	1	Multisampling	5
BindProgramPipeline	2	DepthRange*	5	GetInteger*	7		
BindRenderbuffer	4	Derivative Functions	12	GetInteger64*	7	N	
Bind[Sampler, Texture][s]	2	DetachShader	2	GetInternalformat*	7	NamedBuffer	1
BindTexture[s]	2	DisableVertexArrayAttrib	5	GetMultisamplefv	5	NamedFramebufferDraw	6
BindTextureUnit	2	DisableVertexAttribArray	5	GetNamedBuffer[Indexed]	1	NamedFramebuffer	4
BindTransformFeedback	5	DispatchCompute*	5	GetNamedFramebuffer	4	NamedFramebufferReadBuffer	6
BindVertexArray	4	Dithering	6	GetNamedRenderbuffer	4	NamedRenderbufferStorage	4
BindVertexBuffer[s]	4	DrawArrays*	5	GetObject[Ptr]Label	6	Noise Functions	12
BlendColor	6	DrawBuffer[s]	6	GetPointerv	6-7		
BlendEquation[Separate]*	6	Draw[Range]Elements*	5	GetProgram*	2	O	
BlendFunc[Separate]*	6	DrawTransformFeedback*	5	GetQuery*	1	Object[Ptr]Label	5
Blit[Named]Framebuffer	6			GetRenderbufferParameteriv	4	Occlusion Queries	6
Buffer[Sub]Data	1	E		GetSamplerParameter*	2	Operators	9
BufferStorage	1	EnableVertexArrayAttrib	5	GetShader*	2		
BufferTextures	3	EnableVertexAttribArray	5	GetString*	6	P	
		EndConditionalRender	5	GetSubroutine*	2	Pack/Unpack Functions	11
C		EndQuery[Indexed]	1	GetSynciv	1	Parameter Qualifiers	10
Callback	6	EndQuery	6	Get[n]TexImage	3	PatchParameter	5
Check[Named]FramebufferStatus	4	EndTransformFeedback	5	GetTex[Level]Parameter*	3	PauseTransformFeedback	5
ClampColor	6	Errors	1	GetTexture*	3	Per-Fragment Operations	6
Clear	6	Exponential Functions	11	GetTransformFeedback	7	Pipeline Diagram	8
Clear[Named]Buffer[Sub]Data	1			GetTransformFeedbackVarying	5	PixelStore{if}	2
ClearBuffer*	6	F		Get[n]Uniform	2	PointParameter*	5
ClearColor	6	Feedback Loops	4	GetUniform*	2	PointSize	5
ClearDepth*	6	FenceSync	1	GetVertex[Array, Attrib]*	5	Polygon{Mode, Offset}	5
ClearStencil	6	Finish	1	GL Command Syntax	1	{Pop, Push}DebugGroup	6
ClearTex[Sub]Image	4	Flatshading	5			Precise & Precision Qualifiers	10
ClientWaitSync	1	Floating-Point Pack/Unpack Func.	11	H		Preprocessor	9
Clip Control	5	Flush	1	Hint	6	Primitive Clipping	5
ColorMask*	6	FlushMapped*	1			PrimitiveRestartIndex	5
Command Letters	1	Fragment Language Variables	10	I		ProgramBinary	2
Common Functions	11	Fragment Processing Functions	12	Image Functions	11	Program Objects	2
CompileShader	1	Fragment Shaders	5	Integer Functions	11	ProgramParameteri	2
CompressedTex[Sub]Image*	3	Framebuffer Objects	4	Interpolation Functions	12	ProgramUniform[Matrix]*	2
CompressedTextureSubImage*	3	FramebufferParameteri	4	Interpolation Qualifiers	10	ProvokingVertex	5
Compute Language Variables	10	FramebufferRenderbuffer	4	InvalidateBuffer*	1		
Compute Programming Diagram	7	FramebufferTexture*	4	Invalidate[Sub]Framebuffer	6	Q	
Compute Shaders	5	FrontFace	5	InvalidateNamedFramebuffer	6	Qualifiers	9
Constants	10			InvalidateTex[Sub]Image	4	QueryCounter	1
Conversions	6	G		Invariant Qualifiers	10		
Copy[Named]BufferSubData	1	GenBuffers	1	IsBuffer	1	R	
CopyImageSubData	6	Generate[Texture]Mipmap	3	IsFramebuffer	4	Rasterization	5
CopyTex[Sub]Image*	3	GenFramebuffers	4	IsProgram[Pipeline]	5	ReadBuffer	6
CopyTextureSubImage*	3	GenProgramPipelines	2	IsQuery	1	Read[n]Pixels	6
CreateBuffers	1	GenQueries	1	IsRenderbuffer	4	ReleaseShaderCompiler	1
CreateFrameBuffers	4	GenRenderbuffers	4	IsSampler	2	Renderbuffer Object Queries	4
CreateProgram[Pipelines]	2	GenSamplers	2	IsShader	1	RenderbufferStorage*	4
CreateQueries*	1	GenTextures	2	IsSync	1	ResumeTransformFeedback	5
CreateRenderBuffers	4	GenTransformFeedbacks	4	IsTexture	2		
CreateSamplers*	2	GenVertexArrays	4	IsTransformFeedback	5	S	
CreateShader	1	Geometric Functions	11	IsVertexArray	4	SampleCoverage	5
CreateShaderProgramv	2	Geometry & Follow-on Diagram	8	Iteration and Jumps	10	SampleMaski	5
Createtextures*	2	Geometry Shader Functions	12				
						T	
						Tessellation Diagram	7
						Tessellation Variables	10
						TexBuffer*	3
						Texel Lookup Functions	12
						TexImage*[Multisample]	3
						TexParameter*	3
						TexStorage*	3
						TexStorage*[Multisample]	4
						TexSubImage*	3
						TextureBarrier	4
						TextureBuffer[Range]	3
						Texture Functions	12
						Texture Queries	3
						TextureStorage*[Multisample]	4
						TextureSubImage	3
						TextureView	3
						Texture View/State Diagram	7
						Timer Queries	1
						Transform Feedback	5
						TransformFeedbackVaryings	5
						Types	9
						U	
						Uniform Qualifiers	9-10
						Uniform Variables	2
						Uniform*	2
						UniformBlockBinding	2
						Uniform[Matrix]*	2
						UniformSubroutinesuiv	2
						Unmap[Named]Buffer	1
						UseProgram[Stages]	2
						V	
						ValidateProgram[Pipeline]	5
						Vector & Matrix	10
						Vector Relational Functions	11
						Vertex & Tessellation Diagram	8
						Vertex Arrays	4-5
						VertexAttrib*	4
						Vertex[Array]Attrib*	4
						VertexArrayBindingDivisor	5
						VertexArray*Buffer	4
						VertexAttrib*Format	4
						VertexAttrib*Pointer	5
						VertexAttrib[Binding, Divisor]	4-5
						VertexBindingDivisor	4
						Vertex Language Variables	10
						Viewport*	5
						W	
						WaitSync	1



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.