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 4.4 core specification.
- [n.n.n] refers to sections in the OpenGL Shading Language 4.40 specification.

OpenGL Errors [2.3.1]

enum **GetError**(void); Returns the numeric error code.

OpenGL Operation

Floating-Point Numbers [2.3.3]

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 from the table below is used to denote 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

Flush and Finish [2.3.2]

void **Flush**(void);

void **Finish**(void);

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

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

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}

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 **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
pname: CURRENT_QUERY, QUERY_COUNTER_BITS

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

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

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

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

pname: QUERY_RESULT{_AVAILABLE}, QUERY_RESULT_NO_WAIT

Timer Queries [4.3]

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

void **QueryCounter**(uint *id*, TIMESTAMP);

void **GetInteger64v**(TIMESTAMP, int64 **data*);

Buffer Objects [6]

void **GenBuffers**(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_BUFFER, COPY_{READ, WRITE}_BUFFER, {DISPATCH, DRAW}_INDIRECT_BUFFER, ELEMENT_ARRAY_BUFFER, PIXEL_[UN]PACK_BUFFER, {QUERY, TEXTURE}_BUFFER, SHADER_STORAGE_BUFFER, TRANSFORM_FEEDBACK_BUFFER

void **BindBufferRange**(enum *target*, uint *index*, uint *buffer*, intptr *offset*, sizeiptr *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 sizeiptr **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*, sizeiptr *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 **BufferData**(enum *target*, sizeiptr *size*, const void **data*, enum *usage*);

target: see BindBuffer
usage: DYNAMIC_{DRAW, READ, COPY}, STATIC_{DRAW, READ, COPY}, STREAM_{DRAW, READ, COPY}

void **BufferSubData**(enum *target*, intptr *offset*, sizeiptr *size*, const void **data*);

target: see BindBuffer

void **ClearBufferSubData**(enum *target*, enum *internalFormat*, intptr *offset*, sizeiptr *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 **ClearBufferData**(enum *target*, enum *internalformat*, enum *format*, enum *type*, const void **data*);

target, internalformat, format: see ClearBufferSubData

Map/Unmap Buffer Data [6.3]

void ***MapBufferRange**(enum *target*, intptr *offset*, sizeiptr *length*, bitfield *access*);

access: The logical OR of MAP_*X*_BIT, where *X* may be READ, WRITE, PERSISTENT, COHERENT, INVALIDATE_{BUFFER, RANGE}, FLUSH_EXPLICIT, UNSYNCHRONIZED
target: see BindBuffer

void ***MapBuffer**(enum *target*, enum *access*);

access: see MapBufferRange

void **FlushMappedBufferRange**(enum *target*, intptr *offset*, sizeiptr *length*);

target: see BindBuffer

boolean **UnmapBuffer**(enum *target*);

target: see BindBuffer

Invalidate Buffer Data [6.5]

void **InvalidateBufferSubData**(uint *buffer*, intptr *offset*, sizeiptr *length*);

void **InvalidateBufferData**(uint *buffer*);

Copy Between Buffers [6.6]

void **CopyBufferSubData**(enum *readtarget*, enum *writetarget*, intptr *readoffset*, intptr *writeoffset*, sizeiptr *size*);

readtarget and *writetarget*: *see BindBuffer*

Buffer Object Queries [6, 6.7]

boolean **IsBuffer**(uint *buffer*);

void **GetBufferParameteriv**(enum *target*, enum *pname*, int **data*);

target: see BindBuffer
pname: [Table 6.2] BUFFER_SIZE, BUFFER_USAGE, BUFFER_{ACCESS[_FLAGS], BUFFER_MAPPED, BUFFER_MAP_{OFFSET, LENGTH}, BUFFER_IMMUTABLE_STORAGE, BUFFER_ACCESS_FLAGS

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

target: see BindBuffer
pname: see GetBufferParameteriv

void **GetBufferSubData**(enum *target*, intptr *offset*, sizeiptr *size*, void **data*);

target: see BindBuffer

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

target: see BindBuffer
pname: BUFFER_MAP_POINTER

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*);

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 *programInterface*, enum *pname*, int **params*);

programInterface: ATOMIC_COUNTER_BUFFER, BUFFER_VARIABLE, UNIFORM[_BLOCK], PROGRAM_{INPUT, OUTPUT}, SHADER_STORAGE_BLOCK, {GEOMETRY, VERTEX}_SUBROUTINE, TESS_{CONTROL, EVALUATION}_SUBROUTINE, {FRAGMENT, COMPUTE}_SUBROUTINE, TESS_CONTROL_SUBROUTINE_UNIFORM, TESS_EVALUATION_SUBROUTINE_UNIFORM, {GEOMETRY, VERTEX}_SUBROUTINE_UNIFORM, {FRAGMENT, COMPUTE}_SUBROUTINE_UNIFORM, TRANSFORM_FEEDBACK_{BUFFER, VARYING}
pname: ACTIVE_RESOURCES, MAX_NAME_LENGTH, MAX_NUM_ACTIVE_VARIABLES, MAX_NUM_COMPATIBLE_SUBROUTINES

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

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

void **GetProgramResourceiv**(uint *program*, enum *programInterface*, uint *index*, sizei *propCount*, const enum **props*, sizei *bufSize*, sizei **length*, int **params*);

**props:* [see Table 7.2]

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

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

(Continued on next page >)

Shaders and Programs (cont.)

Program Pipeline Objects [7.4]

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

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

boolean IsProgramPipeline(uint pipeline);

void BindProgramPipeline(uint pipeline);

void 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 **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},
    UNIFORM_{SIZE, BLOCK_INDEX, UNIFORM_OFFSET},
    UNIFORM_{ARRAY, MATRIX}_STRIDE,
    UNIFORM_IS_ROW_MAJOR,
    UNIFORM_ATOMIC_COUNTER_BUFFER_INDEX

uint GetUniformBlockIndex(uint program,
    const char *uniformBlockName);

void GetActiveUniformBlockName(
    uint program, uint uniformBlockIndex,
    sizei bufSize, sizei length,
    char *uniformBlockName);

void GetActiveUniformBlockiv(
    uint program, uint uniformBlockIndex,
    enum pname, int *params);
  pname: UNIFORM_BLOCK_{BINDING, DATA_SIZE},
    UNIFORM_BLOCK_NAME_LENGTH,
    UNIFORM_BLOCK_ACTIVE_UNIFORMS[_INDICES],
    UNIFORM_BLOCK_REFERENCED_BY_X_SHADER,
    where X may be one of VERTEX, FRAGMENT,
    COMPUTE, GEOMETRY, TESS_CONTROL, or
    TESS_EVALUATION [Table 7.7]

void GetActiveAtomicCounterBufferiv(
    uint program, uint bufferIndex,
    enum pname, int *params);
  pname: see GetActiveUniformBlockiv, however
    replace the prefix UNIFORM_BLOCK_ with
    ATOMIC_COUNTER_BUFFER_
```

Load Uniform Vars. In Default Uniform Block

```
void Uniform{1234}{i f d ui}(int location,
    T value);

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

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

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

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

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

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

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

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

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

Uniform Buffer Object Bindings

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

Shader Buffer Variables [7.8]

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

Subroutine Uniform Variables [7.9]

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

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

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

void GetActiveSubroutineName(
    uint program, enum shadertype,
    uint index, sizei bufsize, sizei *length,
    char *name);

void GetActiveSubroutineUniformName(
    uint program, enum shadertype,
    uint index, sizei bufsize, sizei *length,
    char *name);

void GetActiveSubroutineUniformiv(
    uint program, enum shadertype,
    uint index, enum pname, int *values);
  pname: [NUM_]COMPATIBLE_SUBROUTINES

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

Shader Memory Access [7.12.2]

See diagram on page 6 for more information.

```
void MemoryBarrier(bitfield barriers);
  barriers: ALL_BARRIER_BITS or the OR of
    X_BARRIER_BIT where X may be:
    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,
    QUERY_BUFFER
```

Shader|Program Queries [7.13]

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

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,
    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,
    VALIDATE_STATUS

void GetProgramPipelineiv(uint pipeline,
    enum pname, int *params);
  pname: ACTIVE_PROGRAM, VALIDATE_STATUS,
    {VERTEX, FRAGMENT, GEOMETRY}_SHADER,
    TESS_{CONTROL, EVALUATION}_SHADER,
    INFO_LOG_LENGTH, COMPUTE_SHADER

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

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

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

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

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

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

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

void GetUniformSubroutineuiv(
    enum shadertype, int location,
    uint *params);

void GetProgramStageiv(uint program,
    enum shadertype, enum pname,
    int *values);
  pname: ACTIVE_SUBROUTINES,
    ACTIVE_SUBROUTINES_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 DeleteTextures(sizei n,
    const uint *textures);

boolean IsTexture(uint texture);
```

Sampler Objects [8.2]

```
void GenSamplers(sizei count, 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{if}

void SamplerParameterI{i ui}v(uint sampler,
    enum pname, const T *params);
  pname: see SamplerParameter{if}

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{if}

void GetSamplerParameterI{i ui}v(
    uint sampler, enum pname, T *params);
  pname: see SamplerParameter{if}
```

Pixel Storage Modes [8.4.1]

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

Texture Image Spec. [8.5]

```
void TexImage3D(enum target, int level,
    int internalformat, sizei width, sizei height,
    sizei depth, int border, enum format,
    enum type, const void *data);
  target: [PROXY_]TEXTURE_CUBE_MAP_ARRAY,
    [PROXY_]TEXTURE_3D, [PROXY_]TEXTURE_2D_ARRAY
  internalformat: STENCIL_INDEX, RED,
    DEPTH_{COMPONENT, STENCIL}, RG, RGB, RGBA,
    COMPRESSED_{RED, RG, RGB, RGBA, SRGB,
    SRGB_ALPHA), a sized internal format from [Tables
    8.12 - 8.13], or a specific compressed format in
    [Table 8.14]
  format: DEPTH_{COMPONENT, STENCIL}, RED,
    GREEN, BLUE, RG, RGB, RGBA, BGR, BGRA,
    {BGRA, RED, GREEN, BLUE}_INTEGER,
    {RG, RGB, RGBA, BGR}_INTEGER,
    STENCIL_INDEX, [Table 8.3]
  type: [UNSIGNED_]{BYTE, SHORT, INT},
    [HALF_]FLOAT, or a value from [Table 8.2]

void TexImage2D(enum target, int level,
    int internalformat, sizei width,
    sizei height, int border, enum format,
    enum type, const void *data);
  target: [PROXY_]TEXTURE_{2D, RECTANGLE},
    [PROXY_]TEXTURE_1D_ARRAY,
    PROXY_TEXTURE_CUBE_MAP
    TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z},
    TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}
  internalformat, format, type: see TexImage3D

void TexImage1D(enum target, int level,
    int internalformat, sizei width, int border,
    enum format, enum type,
    const void *data);
  target: TEXTURE_1D, PROXY_TEXTURE_1D
  type, internalformat, format: see TexImage3D
```

Alternate Texture Image Spec. [8.6]

```
void CopyTexImage2D(enum target,
    int level, enum internalformat, int x,
    int y, sizei width, sizei height, int border);
  target: TEXTURE_{2D, RECTANGLE, 1D_ARRAY},
    TEXTURE_CUBE_MAP_{POSITIVE, NEGATIVE}_{X, Y, Z}
  internalformat: see TexImage3D

void CopyTexImage1D(enum target,
    int level, enum internalformat, int x,
    int y, sizei width, int border);
  target: TEXTURE_1D
  internalformat: see TexImage3D

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

void TexSubImage2D(enum target, int level,
    int xoffset, int yoffset, sizei width,
    sizei height, enum format, enum type,
    const void *data);
  target: see CopyTexImage2D
  format, type: see TexImage3D

void TexSubImage1D(enum target, int level,
    int xoffset, sizei width, enum format,
    enum type, const void *data);
  target: TEXTURE_1D
  format, type: see TexImage3D

void CopyTexSubImage3D(enum target,
    int level, int xoffset, int yoffset, int zoffset,
    int x, int y, sizei width, sizei height);
  target: see TexSubImage3D

void CopyTexSubImage2D(enum target,
    int level, int xoffset, int yoffset, int x,
    int y, sizei width, sizei height);
  target: see TexImage2D
```

(Continued on next page >)

Framebuffer Objects

Binding and Managing [9.2]

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

target: [DRAW_, READ_]FRAMEBUFFER

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

Framebuffer Object Queries [9.2.3]

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

target, pname: ***see FramebufferParameteri***

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*i*, {DEPTH, STENCIL, DEPTH_STENCIL}_ATTACHMENT

pname: FRAMEBUFFER_ATTACHMENT_ *X*

Vertices

Separate Patches [10.1.15]

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

pname: PATCH_VERTICES

Current Vertex Attribute Values [10.2]

Specify generic attributes with components of type float (**VertexAttrib***), int or uint (**VertexAttribI***), or double (**VertexAttribL***).

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, T values);

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 **VertexAttribL{1234}d**(uint index,
    T values);

void **VertexAttribL{1234}dv**(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,
UNSIGNED_INT_10F_11F_11F_REV

Vertex Arrays

Generic Vertex Attribute Arrays [10.3.1]

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

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

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

void **VertexAttribPointer**(uint index, int size,
    enum type, boolean normalized,
    sizei stride, const void *pointer);

type: see VertexAttribFormat

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

type: see VertexAttribIFormat

index: [0, MAX_VERTEX_ATTRIBS - 1]

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

type: DOUBLE

index: [0, MAX_VERTEX_ATTRIBS - 1]

void **EnableVertexAttribArray**(uint index);

void **DisableVertexAttribArray**(uint index);

index: [0, MAX_VERTEX_ATTRIBS - 1]

Vertex Attribute Divisors [10.3.2]

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

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

Primitive Restart [10.3.5]

Enable/Disable/IsEnabled(target);

target: PRIMITIVE_RESTART{_FIXED_INDEX}

void **PrimitiveRestartIndex**(uint index);

Vertex Array Objects [10.4]

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);

boolean **IsVertexArray**(uint array);

Drawing Commands [10.5]

For all the functions in this section:

mode: POINTS, LINE_STRIP, LINE_LOOP, LINES,
TRIANGLE_{STRIP, FAN}, TRIANGLES, PATCHES,
LINES_ADJACENCY, TRIANGLES_ADJACENCY, {LINE,
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 **DrawElementsInstancedBase-
VertexBaseInstance**(enum mode,
    sizei count, enum type,
    const void *indices, sizei instancecount,
    int basevertex, uint baseinstance);

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

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

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

Vertex Array Queries [10.6]

void **GetVertexAttrib{d f i}v**(uint index,
    enum pname, T *params);

pname: CURRENT_VERTEX_ATTRIB or
VERTEX_ATTRIB_ARRAY_X where X is one of
BUFFER_BINDING, DIVISOR, ENABLED, INTEGER,
LONG, NORMALIZED, SIZE, STRIDE, or TYPE

void **GetVertexAttribI{i ui}v**(uint index,
    enum pname, T *params);

pname: see GetVertexAttrib{d f i}v

void **GetVertexAttribLdv**(uint index,
    enum pname, double *params);

pname: see GetVertexAttrib{d f i}v

void **GetVertexAttribPointerv**(uint index,
    enum pname, const void **pointer);

pname: VERTEX_ATTRIB_ARRAY_POINTER

Conditional Rendering [10.10]

void **BeginConditionalRender**(uint id,
    enum mode);

mode: {QUERY_BY_REGION, QUERY}_{WAIT,
NO_WAIT}

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, SEPARATE}_ATTRIBS

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

*type returns NONE, FLOAT[_VECn],
DOUBLE[_VECn], [UNSIGNED_]INT,
[UNSIGNED_]INT_VECn, MATnxm,
{FLOAT, DOUBLE}_{MATn, MATnxm}

Shader Execution [11.1.3]

void **ValidateProgram**(uint program);

void **ValidateProgramPipeline**(uint pipeline);

Tessellation Control Shaders [11.2.2]

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

pname: PATCH_DEFAULT_{INNER, 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 **BeginTransformFeedback**(
    enum primitiveMode);

primitiveMode: TRIANGLES, LINES, POINTS

void **EndTransformFeedback**(void);

void **PauseTransformFeedback**(void);

void **ResumeTransformFeedback**(void);

Transform Feedback Drawing [13.2.3]

void **DrawTransformFeedback**(
    enum mode, uint id);

mode: see Drawing Commands [10.5] 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]

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}v

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

pname: POINT_FADE_THRESHOLD_SIZE,
POINT_SPRITE_COORD_ORIGIN

param, 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

(Continued on next page >)

Rasterization (cont.)

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}

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

Blending [17.3.8]

Enable/Disable/IsEnabled(BLEND);

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

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

void **BlendEquationSeparate**(enum *modeRGB*, enum *modeAlpha*);
mode, modeRGB, modeAlpha: MIN, MAX , FUNC_{ADD, SUBTRACT, REVERSE_SUBTRACT}

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

void **BlendEquationSeparatei**(uint *buf*, enum *modeRGB*, enum *modeAlpha*);
mode, modeRGB, modeAlpha: *see BlendEquationSeparate*

void **BlendFunc**(enum *src*, enum *dst*);
src, dst: see BlendFuncSeparate

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

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*);

Whole Framebuffer

Selecting a Buffer for Writing [17.4.1]

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

void **DrawBuffers**(sizei *n*, const enum **bufs*);
bufs: [Tables 17.5-6] {FRONT, BACK}_{LEFT, RIGHT}, NONE, COLOR_ATTACHMENT*i* (*i* = [0, MAX_COLOR_ATTACHMENTS - 1])

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*);

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

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

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

void **InvalidateFramebuffer**(enum *target*, sizei *numAttachments*, const enum **attachments*);
target, attachment: *see InvalidateSubFramebuffer*

Reading and Copying Pixels

Reading Pixels [18.2]

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

Final Conversion [18.2.6]

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 OR of {COLOR, DEPTH, STENCIL}_BUFFER_BIT or 0
filter: LINEAR, NEAREST

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

Debug Output [20]

Enable/Disable/IsEnabled(DEBUG_OUTPUT);

Debug Message Callback [20.2]

void **DebugMessageCallback**(DEBUGPROC *callback*, void **userParam*);
callback: has the prototype:

void **callback**(enum *source*, enum *type*, uint *id*, enum *severity*, sizei *length*, const char **message*, void **userParam*);
source: DEBUG_SOURCE_*X* where *X* may be API, SHADER_COMPILER, WINDOW_SYSTEM, THIRD_PARTY, APPLICATION, OTHER
type: DEBUG_TYPE_*X* where *X* may be ERROR, MARKER, OTHER, DEPRECATED_BEHAVIOR, UNDEFINED_BEHAVIOR, PERFORMANCE, PORTABILITY, {PUSH, POP}_GROUP
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 callback (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);

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*);

Compute Shaders [19]

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

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

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

State and State Requests

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

Simple Queries [22.1]

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

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

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

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

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

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

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

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

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

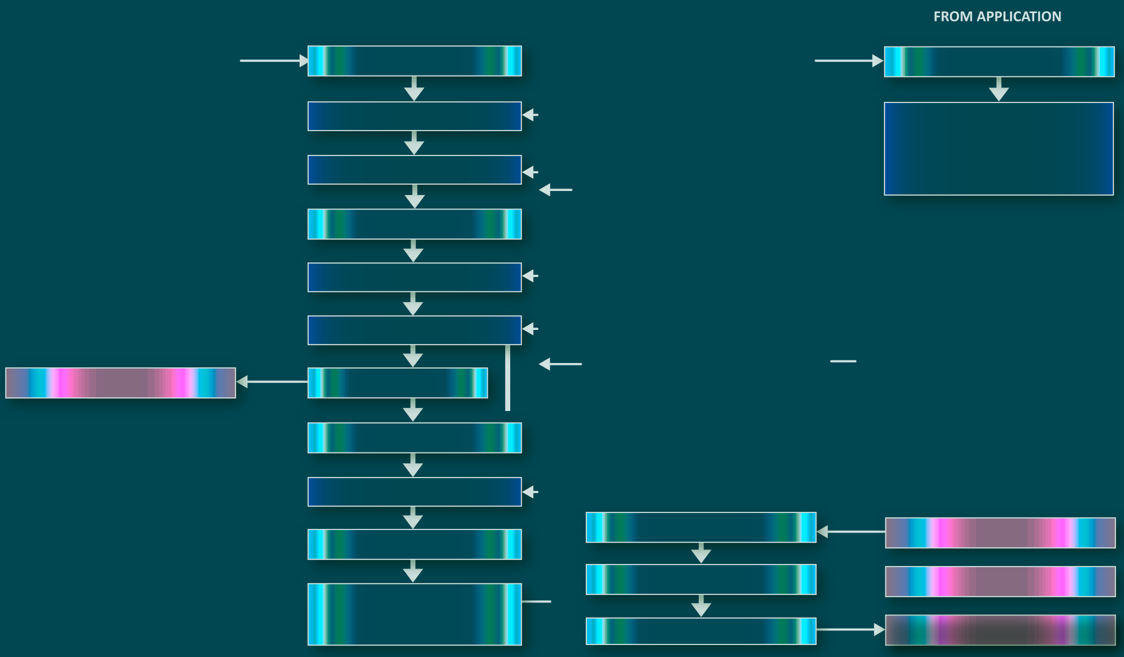
(Continued on next page >)

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



Types [4.1]

Transparent Types

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

Floating-Point Opaque Types

sampler{1D,2D,3D}
image{1D,2D,3D}
	1D, 2D, or 3D texture

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

Signed Integer Opaque Types

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
iimageBuffer	integer buffer image
isampler2DMS	int. 2D multi-sample texture
iimage2DMS	int. 2D multi-sample image
isampler2DMSArray	int. 2D multi-sample array tex.
iimage2DMSArray	int. 2D multi-sample array image
isamplerCubeArray	int. cube map array texture
iimageCubeArray	int. cube map array image

Unsigned Integer Opaque Types

atomic_uint	uint atomic counter
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	1D or 2D 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
int, uint	->	float
int, uint, float	->	double
ivec2	->	uvec2
ivec3	->	uvec3
ivec4	->	uvec4
ivec2	->	vec2
ivec3	->	vec3
ivec4	->	vec4
uvec2	->	vec2
uvec3	->	vec3
uvec4	->	vec4
ivec2	->	dvec2
ivec3	->	dvec3
ivec4	->	dvec4

uvec2	->	dvec2
uvec3	->	dvec3
uvec4	->	dvec4
vec2	->	dvec2
vec3	->	dvec3
vec4	->	dvec4
mat2	->	dmat2
mat3	->	dmat3
mat4	->	dmat4
mat2x3	->	dmat2x3
mat2x4	->	dmat2x4
mat3x2	->	dmat3x2
mat3x4	->	dmat3x4
mat4x2	->	dmat4x2
mat4x3	->	dmat4x4

Aggregation of Basic Types

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, vec*n*.

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

Exponential Functions [8.2]

Component-wise operation. Tf=float, vec*n*. Td= double, dvec*n*. Tfd= Tf, Td

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

Common Functions [8.3]

Component-wise operation. Tf=float, vec*n*. Tb=bool, bvec*n*. Ti=int, ivec*n*. Tu=uint, uvec*n*.
Td= double, dvec*n*. Tfd= Tf, Td. Tiu= Ti, Tu.

Returns absolute value:
Tfd **abs**(Tfd *x*)
Ti **abs**(Ti *x*)

Returns -1.0, 0.0, or 1.0:
Tfd **sign**(Tfd *x*)
Ti **sign**(Ti *x*)

Returns nearest integer <= *x*:
Tfd **floor**(Tfd *x*)

Returns nearest integer with absolute value <= absolute value of *x*:
Tfd **trunc**(Tfd *x*)

Returns nearest integer, implementation-dependent rounding mode:
Tfd **round**(Tfd *x*)

Returns nearest integer, 0.5 rounds to nearest even integer:
Tfd **roundEven**(Tfd *x*)

Returns nearest integer >= *x*:
Tfd **ceil**(Tfd *x*)

Returns *x* - floor(*x*):
Tfd **fract**(Tfd *x*)

Returns modulus:
Tfd **mod**(Tfd *x*, Tfd *y*)
Tf **mod**(Tf *x*, float *y*)
Td **mod**(Td *x*, double *y*)

Returns separate integer and fractional parts:
Tfd **modf**(Tfd *x*, out Tfd *i*)

Returns minimum value:
Tfd **min**(Tfd *x*, Tfd *y*)
Tf **min**(Tf *x*, float *y*)
Td **min**(Td *x*, double *y*)
Tiu **min**(Tiu *x*, Tiu *y*)
Ti **min**(Ti *x*, int *y*)
Tu **min**(Tu *x*, uint *y*)

(Continue ↑)

Common Functions (cont.)

Returns maximum value:
Tfd **max**(Tfd *x*, Tfd *y*)
Tf **max**(Tf *x*, float *y*)
Td **max**(Td *x*, double *y*)
Tiu **max**(Tiu *x*, Tiu *y*)
Ti **max**(Ti *x*, int *y*)
Tu **max**(Tu *x*, uint *y*)

Returns min(max(*x*, *minVal*), *maxVal*):
Tfd **clamp**(Tfd *x*, Tfd *minVal*, Tfd *maxVal*)
Tf **clamp**(Tf *x*, float *minVal*, float *maxVal*)
Td **clamp**(Td *x*, double *minVal*, double *maxVal*)
Tiu **clamp**(Tiu *x*, Tiu *minVal*, Tiu *maxVal*)
Ti **clamp**(Ti *x*, int *minVal*, int *maxVal*)
Tu **clamp**(Tu *x*, uint *minVal*, uint *maxVal*)

Returns linear blend of *x* and *y*:
Tfd **mix**(Tfd *x*, Tfd *y*, Tfd *a*)
Tf **mix**(Tf *x*, Tf *y*, float *a*)
Td **mix**(Td *x*, Td *y*, double *a*)

Returns true if components in *a* select components from *y*, else from *x*:
Tfd **mix**(Tfd *x*, Tfd *y*, Tb *a*)

Returns 0.0 if *x* < *edge*, else 1.0:
Tfd **step**(Tfd *edge*, Tfd *x*)
Tf **step**(float *edge*, Tf *x*)
Td **step**(double *edge*, Td *x*)

Clamps and smoothes:
Tfd **smoothstep**(Tfd *edge0*, Tfd *edge1*, Tfd *x*)
Tf **smoothstep**(float *edge0*, float *edge1*, Tf *x*)
Td **smoothstep**(double *edge0*, double *edge1*, Td *x*)

Returns true if *x* is NaN:
Tb **isnan**(Tfd *x*)

Returns true if *x* is positive or negative infinity:
Tb **isinf**(Tfd *x*)

Returns signed int or uint value of the encoding of a float:
Ti **floatBitsToInt**(Tf *value*)
Tu **floatBitsToUint**(Tf *value*)

Returns float value of a signed int or uint encoding of a float:
Tf **intBitsToFloat**(Ti *value*)
Tf **uintBitsToFloat**(Tu *value*)

Computes and returns a*b + c. Treated as a single operation when using **precise**:
Tfd **fma**(Tfd *a*, Tfd *b*, Tfd *c*)

Splits *x* into a floating-point significand in the range [0.5, 1.0) and an integer exponent1 ò

Built-In Functions (cont.)

Image Functions (cont.)

Adds the value of *data* to the contents of the selected texel:

uint **imageAtomicAdd**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicAdd**(*IMAGE_PARAMS*, int *data*)

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

uint **imageAtomicMin**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicMin**(*IMAGE_PARAMS*, int *data*)

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

uint **imageAtomicMax**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicMax**(*IMAGE_PARAMS*, int *data*)

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

uint **imageAtomicAnd**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicAnd**(*IMAGE_PARAMS*, int *data*)

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

uint **imageAtomicOr**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicOr**(*IMAGE_PARAMS*, int *data*)

(Continue ↑)

Image Functions (cont.)

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

uint **imageAtomicXor**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicXor**(*IMAGE_PARAMS*, int *data*)

Copies the value of *data*:

uint **imageAtomicExchange**(*IMAGE_PARAMS*, uint *data*)

int **imageAtomicExchange**(*IMAGE_PARAMS*, int *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**(*IMAGE_PARAMS*, uint *compare*, uint *data*)

int **imageAtomicCompSwap**(*IMAGE_PARAMS*, int *compare*, int *data*)

Fragment Processing Functions [8.13]

Available only in fragment shaders.
Tf=float, vec*n*.

Derivative fragment-processing functions

Tf **dFdx**(Tf *p*)	derivative in *x*
Tf **dFdy**(Tf *p*)	derivative in *y*
Tf **fwidth**(Tf *p*)	sum of absolute derivative in *x* and *y*; **abs**(**dFdx**(*p*)) + **abs**(**dFdy**(*p*));

Interpolation fragment-processing functions

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

Tf **interpolateAtCentroid**(Tf *interpolant*)

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

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

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

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

Noise Functions [8.14]

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

float **noise1**(Tf *x*)   vec*n* **noise*n***(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. gvec4=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*)

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*.

gvec4 **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**(gsamplerCubeArrayShadow *sampler*, vec4 *P*, float *compare*)

Texture lookup with projection.

gvec4 **textureProj**(gsampler{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.

gvec4 **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.

gvec4 **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*.

gvec4 **texelFetch**(gsampler{1D[Array],2D[Array,Rect],3D} *sampler*, {int,ivec2,ivec3} *P*[, {int,ivec2} *lod*])

gvec4 **texelFetch**(gsampler{Buffer, 2DMS[Array]} *sampler*, {int,ivec2,ivec3} *P*[, int *sample*])

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

gvec4 **texelFetchOffset**(gsampler{1D[Array],2D[Array],3D} *sampler*, {int,ivec2,ivec3} *P*, int *lod*, {int,ivec2,ivec3} *offset*)

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

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

gvec4 **textureProjOffset**(gsampler{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.

gvec4 **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.

gvec4 **textureProjLod**(gsampler{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.

gvec4 **textureProjLodOffset**(gsampler{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.

gvec4 **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.

gvec4 **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**.

gvec4 **textureProjGrad**(gsampler{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.**

gvec4 **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.

gvec4 **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.

gvec4 **textureGatherOffset**(gsampler2D[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.

gvec4 **textureGatherOffsets**(gsampler2D[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.

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

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

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