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] refer to sections and tables in the OpenGL 4.2 core specification.
• [n.n.n] refers to sections in the OpenGL Shading Language 4.20 specification.

OpenGL Errors [2.5] enum GetError(void); Returns the numeric error code.

OpenGL Operation
Floating-Point Numbers [2.1.1 - 2.1.2]
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 [Table 2.1]

<table>
<thead>
<tr>
<th>Command Letters</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>INT</td>
</tr>
<tr>
<td>b</td>
<td>BYTE</td>
</tr>
<tr>
<td>c</td>
<td>SHORT</td>
</tr>
<tr>
<td>d</td>
<td>USHORT</td>
</tr>
<tr>
<td>e</td>
<td>CHAR</td>
</tr>
<tr>
<td>f</td>
<td>USHORT</td>
</tr>
<tr>
<td>g</td>
<td>INT64</td>
</tr>
<tr>
<td>h</td>
<td>UINT64</td>
</tr>
<tr>
<td>i</td>
<td>INT8</td>
</tr>
<tr>
<td>j</td>
<td>UINT8</td>
</tr>
<tr>
<td>k</td>
<td>INT16</td>
</tr>
<tr>
<td>l</td>
<td>UINT16</td>
</tr>
<tr>
<td>m</td>
<td>INT32</td>
</tr>
<tr>
<td>n</td>
<td>UINT32</td>
</tr>
<tr>
<td>o</td>
<td>INT64</td>
</tr>
<tr>
<td>p</td>
<td>UINT64</td>
</tr>
</tbody>
</table>

Vertex Arrays [2.8]
void VertexAttribPointer(uint index, int size, enum type, boolean normalized, sizei stride, const void *pointer);
type: SHORT, INT, FLOAT, HALF, FLOAT, DOUBLE, UNSIGNED INT_2_10_10_10_REV, FIXED, BYTE, UINT, UNSIGNED BYTE, SHORT.
void VertexAttribPointerPointere(int index, int size, enum type, boolean normalized, sizei stride, const void *pointer);
type: BYTE, SHORT, UNSIGNED BYTE, SHORT, INT, UNSIGNED INT_10_10_10_2, FLOAT, DOUBLE, UNSIGNED INT_2_10_10_10_REV, FIXED, UNSIGNED BYTE, SHORT.
void VertexAttribPointerPointere(int index, int size, enum type, sizei stride, const void *pointer);
type: DOUBLE
index: [0, MAX_VERTEX_ATTRIBS - 1]
void Enable/DisableVertexArray(uint index);
void DisableVertexArray(uint index);
index: [0, MAX_VERTEX_ATTRIBS - 1]
void Enable/DisablePRIMITIVE_RESTART;
void PrimitiveRestartIndex(uint index);

Drawing Commands [2.8.3]
For all the functions in this section:
mode: POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES, LINES_ADJACENCY, LINE_ADJACENCY, TRIANGLE_ADJACENCY, PATCHES, TRIANGLES_ADJACENCY, TESSellation_ADJACENCY, PATCHES, TESSellation_ADJACENCY, Puncture, TESSellation_ADJACENCY, Puncture

<table>
<thead>
<tr>
<th>Command Letters</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>INT64</td>
</tr>
<tr>
<td>G</td>
<td>UINT64</td>
</tr>
<tr>
<td>J</td>
<td>INT32</td>
</tr>
<tr>
<td>K</td>
<td>UINT32</td>
</tr>
<tr>
<td>L</td>
<td>INT32</td>
</tr>
<tr>
<td>N</td>
<td>UINT32</td>
</tr>
<tr>
<td>P</td>
<td>INT32</td>
</tr>
<tr>
<td>R</td>
<td>UINT32</td>
</tr>
<tr>
<td>S</td>
<td>INT32</td>
</tr>
<tr>
<td>T</td>
<td>UINT32</td>
</tr>
<tr>
<td>U</td>
<td>INT32</td>
</tr>
<tr>
<td>V</td>
<td>UINT32</td>
</tr>
<tr>
<td>W</td>
<td>INT32</td>
</tr>
<tr>
<td>X</td>
<td>UINT32</td>
</tr>
<tr>
<td>Y</td>
<td>INT32</td>
</tr>
<tr>
<td>Z</td>
<td>UINT32</td>
</tr>
</tbody>
</table>

Shaders and Programs
Shader Objects [2.11.1-2]
uint CreateShader(enum type);
type: VERTEX, FRAGMENT, GEOMETRY, SHADER, TESSellation, EVALUATION, CONTROL_SHADER
void ShaderSource(uint shader, sizei count, const char **string, const int *length);
void CompileShader(uint shader);
void ReleaseShaderCompiler(void);
void DeleteShader(uint shader);
void ShaderBinary(sizei count, const char *shaders, enum binformat, const void *binary, sizei length);

Program Objects [2.11.3]
uint CreateProgram(void);
void AttachShader(uint program, uint shader);
void DetachShader(uint program, uint shader);
void LinkProgram(uint program);
void UseProgram(uint program);
uint CreateShaderProgram(uint program, sizei count, const char **strings);
void ProgramParameter(uint program, uint name, int value);

OpenGL Command Syntax [2.3]
GL commands are formed from a return type, a name, and optionally up to 4 characters (or character pairs) from the Command Letters table (above), as shown by the prototype:

return-type Name[1234]|b i i64 f d ub us ui u64|v | [args ,] | T arg 1 , . . . , T arg n ; | args |

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

Vertex Specification [2.7]
Vertices have 2, 3, or 4 coordinates. The VertexAttrib* commands specify generic attributes with components of type float (VertexAttrib*, int or uint (VertexAttrib*)), or double (VertexAttrib*).
void VertexAttrib3f(3d, f, uv, i, t);
void VertexAttrib3d(3d, f, uv, i, t);
void VertexAttrib4f(4d, f, uv, i, t);
void VertexAttrib4d(4d, f, uv, i, t);
void VertexAttribNbi(4d, f, uv, i, t);
void VertexAttribNbu(4d, f, uv, i, t);
void VertexAttribNbi(4d, f, uv, i, t);

Buffer Objects [2.9-10]
void GenBuffers(sizei n, uint *buffers);
void DeleteBuffers(sizei n, uint *buffers);
Creating and Binding Buffer Objects [2.9.1]
void BindBuffer(enum target, uint buffer);
target: PIXEL, FRAMEBUFFER, UNIFORM, TEXTURE, BUFFER, COPY, READ, WRITE, ELEMENT ARRAY, TRANSFORM, FLOOBACK, ATOMIC_COUNTER

Copying Between Buffers [2.9.5]
void CopyBufferSubData(enum readtarget, enum writetarget, intptr offset, intptr size, uint writebuffer, uint readbuffer); see BindBuffer

Vertex Array Objects [2.10]
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, uint *arrays);
void BindVertexArray(uint array);

Vertex Array Object Queries [6.1.10]
boolean IsVertexArray(uint array);

Buffer Object Queries [6.1.9]
boolean IsBuffer(uint buffer);

UseProgramStages(3uint program, bitfield stages, uint program);
stages: ALL_SHADER, BITS or the bitwise OR of TESSellation, EVALUATION, CONTROL_SHADER BIT, VERTEX, GEOMETRY, FRAGMENT, SHADER_BIT

Active ShaderProgram(uint program, uint program);

Program Binaries [2.11.5]
void GetProgramBinary(uint program, sizei size, const void *, uint *length);
void GetProgramBinaryData(uint program, uint program, const char *, void *data); (Shaders and Programs Continue)
Asynchronous Queries [6.1.7]
void GetQueryiv(enum target, enum pname, int *params);
    target: see BeginQuery, plus TIMESTAMP
    pname: CURRENT_QUERY, QUERY_COUNTER_BITS
boolean IsQuery(uint id);

Texturing [3.9]
void ActiveTexture(uint texture);
    texture: TEXTUREi where i is
        {0, MAX_TEXTURE_COORDS, MAX_COMBINED_TEXTURE_IMAGE_UNITS-1}

Texture Objects [3.9.1]
void BindTexture(enum target, uint texture);
    target: {TEXTURE, 1D, 2D, 3D, 1D_ARRAY, 2D_ARRAY, TEXTURE_CUBE_MAP_ARRAY, 2D_MULTISAMPLE_ARRAY}
void DeleteTextures(size_t n, const uint *textures);
void GenTextures(size_t n, uint *textures);

Sampler Objects [3.9.2]
void GenSamplers(size_t n, uint *samplers);
void BindSampler(uint unit, uint sampler);
void SamplerParameteriv(uint unit, enum pname, const T *params);
    pname: TEXTURE_WRAP_S, TEXTURE_WRAP_T, TEXTURE_BORDER_COLOR, TEXTURE_COMPARE_MODE, TEXTURE_COMPARE_FUNC
void DeleteSamplers(size_t n, const uint *samplers);

Texture Image Spec. [3.9.3]
void TexImage3D(enum target, int level, internalformat, size_t width, size_t height, size_t depth, enum format, enum type, const void *data);
    target: {TEXTURE_1D, TEXTURE_2D, TEXTURE_3D, TEXTURE_1D_ARRAY, TEXTURE_2D_ARRAY, TEXTURE_3D_ARRAY, TEXTURE_CUBE_MAP, TEXTURE_CUBE_MAP_ARRAY, TEXTURE_RECTANGLE, TEXTURE_RECTANGLE_ARRAY}
    internalformat: {UN}PACK_ALIGNMENT, COMPRESSED_TEXTURE_FORMATS
    format: {UN}PACK_RED_GREEN_BGRA, {UN}PACK_RED_GREEN_565, {UN}PACK_RGB, {UN}PACK_RGBA, {UN}PACK_LUMINANCE_5_5_5_1, {UN}PACK_INTENSITY_5_5_5_1, {UN}PACK_RED_GREEN_8_8, {UN}PACK_RED_GREEN_5_9_1_1, {UN}PACK_RGB8, {UN}PACK_RGB5_6_5, {UN}PACK_RGBA8, {UN}PACK_RGBA5_5_5_1, {UN}PACK_RGB10_10_10_2
    type: {UN}PACK_RGB, {UN}PACK_RGBA, COMPRESSED_RGB, COMPRESSED_RGBA, COMPRESSED_RGB8_PAT, COMPRESSED_RGBA8_PAT, COMPRESSED_RGB8_PAT, COMPRESSED_RGBA8_PAT

Multisample [3.3.1]
Use to antialias points, and lines.

void GetMultisamplefv( enum pname, uint n, float *params);
    pname: SAMPLE_COUNT, SAMPLE_FALSE, SAMPLE_TRUE, SAMPLE_REJECT, SPRITE_SAMPLE_COUNT, SPRITE_SAMPLE_MASK, SPRITE_SAMPLE_REJECT, SPRITE_SAMPLE_TRUE

Points [3.4]
void PointSize(float size);

Polygon Rast. & Depth Offset [3.6.3-4]
void EnableVertexAttrib(index, uint index);
void VertexAttribI{I_2I}_I{I_2I}_v(index, int value);
void VertexAttribI{I_2I}_I{I_2I}_sv(index, size_t size, const T *data);

Lighting and Color
Flatshading [2.19]
void ProvokingVertex( enum provokeMode );

Polylines [3.6]
void Enable( enum target );
void POLYGON_SMOOTH, CULL_FACE

Polygon Stencil [3.7.1]
void PixelStoref( enum pname, float param );
void PixelStorei( enum pname, int param );

Pixel Storage Modes [3.7.1]
void PixelStoref( enum pname, float param );

Texture Compare [3.9.5]
void TexImage1D( enum target, int level, internalformat, size_t width, size_t height, size_t border, enum format, enum type, const void *data );

Compressed Texture Images [3.9.5]
void CompressedTexImage3D( enum target, int level, internalformat, size_t width, size_t height, size_t depth, int border, enum format, enum type, const void *data );
void CompressedTexImage2D( enum target, int level, internalformat, size_t width, size_t height, int border, enum format, enum type, const void *data );

Lighting and Color
Flatshading [2.19]
void ProvokingVertex( enum provokeMode );

Polylines [3.6]
void Enable( enum target );
void POLYGON_SMOOTH, CULL_FACE

Polygon Stencil [3.7.1]
void PixelStoref( enum pname, float param );
void PixelStorei( enum pname, int param );

Pixel Storage Modes [3.7.1]
void PixelStoref( enum pname, float param );
void PixelStorei( enum pname, int param );

Texture Compare [3.9.5]
void TexImage1D( enum target, int level, internalformat, size_t width, size_t height, size_t border, enum format, enum type, const void *data );
void CompressedTexImage3D( enum target, int level, internalformat, size_t width, size_t height, size_t depth, int border, enum format, enum type, const void *data );
void CompressedTexImage2D( enum target, int level, internalformat, size_t width, size_t height, int border, enum format, enum type, const void *data );

Lighting and Color
Flatshading [2.19]
void ProvokingVertex( enum provokeMode );

Polylines [3.6]
void Enable( enum target );
void POLYGON_SMOOTH, CULL_FACE

Polygon Stencil [3.7.1]
void PixelStoref( enum pname, float param );
void PixelStorei( enum pname, int param );

Pixel Storage Modes [3.7.1]
void PixelStoref( enum pname, float param );
void PixelStorei( enum pname, int param );

Texture Compare [3.9.5]
void TexImage1D( enum target, int level, internalformat, size_t width, size_t height, size_t border, enum format, enum type, const void *data );
void CompressedTexImage3D( enum target, int level, internalformat, size_t width, size_t height, size_t depth, int border, enum format, enum type, const void *data );
void CompressedTexImage2D( enum target, int level, internalformat, size_t width, size_t height, int border, enum format, enum type, const void *data );

Copying Image [3.9.1]
void CopyTexImage1D( enum target, int level, internalformat, size_t width, size_t height, int border, const void *data );

Copy Image [3.9.1]
void CopyTexImage2D( enum target, int level, internalformat, size_t width, size_t height, int border, const void *data );

Copy Image [3.9.1]
void CopyTexImage3D( enum target, int level, internalformat, size_t width, size_t height, int border, const void *data );

Copy Image [3.9.1]
void CopyTexImage1D( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );
void CopyTexImage2D( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );
void CopyTexImage3D( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );

Sampling [3.3.1]
void GetTexImage( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );
void GetTexImage2D( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );
void GetTexImage3D( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );
void GetTexImage1D( enum target, int level, internalformat, size_t width, size_t height, enum format, enum type, const void *data );

Copyright © 2012 Khronos Group - Rev. 0712
www.opengl.org/registry

OpenGL 4.2 API Reference Card
Page 3
Texturing (cont.)
Texture Image Loads/Stores [3.9.20]
void BindTexImage(texture unit index, uint texture, int level, boolean layered, int layer, enum access, enum format);
access: READ, READ_WRITE, ONLY_WRITE_READ, READ_ONLY, READ_WRITE, ONLY_WRITE_READ
Buffering Multisample (void nsizei
DeleteRenderbuffers
(void, enum renderbuffers)
void GenFramebuffers (void)
void Disable (enum target, int id, void *msg);
target: see "tex" for GetInteger
boolean IsTexture (uint texture);
Sampler Queries [6.1.5]
boolean IsSampler (uint sampler);
void GetSamplerParameterfv (void nsizei
GetSamplerParameteriv (void nsizei
Texture Queries [6.1.4]
void GetTexImage (texture unit, int tex, enum format, enum type, void *img);
texture: (1, 2D, 3D, 2D_ARRAY TEXTURE_CUBE_MAP_ARRAY)
texture: TEXTURE_3D, TEXTURE_2D_MULTISAMPLE
value: TEXTURE_WIDTH, HEIGHT, DEPTH, TEXTURE_SAMPLES, TEXTURE_FIXED_SAMPLE_LOCATIONS, TEXTURE_INTERNAL_FORMAT, SHARED_SIZE, TEXTURE_COMPRESSED_IMAGE_SIZE, TEXTURE_BUFFER_DATA_STORE_BINDING, texture_e_x_y_z, TYPE, (where x can be RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL)

Whole Framebuffer
Selecting Buffers for Writing [4.2.1]
void Drawbuffer (enum buf);
buf: NONE, FRONT_LEFT, RIGHT, LEFT, FRONT_BACK, BACK_LEFT, BACK_RIGHT, COLOR_ATTACHMENT i = 0, MAX_COLOR_ATTACHMENTS - 1,
AUX i = 0, AUX_BUFFERS - 1
void Drawbuffers (sizei
Drawbuffers (sizei
value: texture_e_x_y_z, TYPE, (where x can be RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL)

Fine Control of Buffer Updates [4.2.2]
void ColorMask (boolean r, boolean g, boolean b, boolean a);
void ClearDepth (float d);
void ClearDepthf (float d);
void ClearStencil (int s);
void ClearBufferf (sizei
value: texture_e_x_y_z, TYPE, (where x can be RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL)

Clearing the Buffers [4.2.3]
void ClearColor (clampf r, clampf g, clampf b, clampf a);
void ClearDepth (clampf d);
void ClearDepthf (clampf d);
void ClearStencil (int s);
void ClearBufferf (sizei
value: texture_e_x_y_z, TYPE, (where x can be RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL)

Reading, and Copying Pixels
Reading Pixels [4.3.1]
void ReadPixels (sizei
Source i = 0, MAX_COLOR_ATTACHMENTS - 1
value: texture_e_x_y_z, TYPE, (where x can be RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL)

Copy Pixels [4.3.2]
void BIfframebuffer (sizei
value: texture_e_x_y_z, TYPE, (where x can be RED, GREEN, BLUE, ALPHA, DEPTH, STENCIL)

Also see DrawPixels, ClampColor, PixelZoom in the Rasterization section of this card.
OpenGL 4.2 API Reference Card

Timer Queries [5.1]
Timer queries use query objects to track the amount of time needed to fully complete a set of GL commands.

Synchronization
Flush and Finish [5.2]
void Flush(void);
void Finish(void);

Sync Objects and Fences [5.3]
deleteSync(synch sync);
synch FenceSync(enum condition, bitfield flags);
condition: SYNC_GPU_COMMANDS_COMPLETE, COMPLETE
flags: must be 0

Waiting for Sync Objects [5.3.1]
enum ClientWaitSync(synch sync, bitfield flags, uint64 timeout ns);
flags: SYNCH_FLUSH_COMMANDS, BIT or zero

GetWaitSync(synch sync, bitfield flags, timeout ns);
timeout: TIMEOUT_IGNORED

Sync Object Queries [5.6.1]
void GetSyncinfo(synch sync, enum pname, size_t bufsiz, size_t length, int *values);
pname: OBJECT_TYPE, SYM_STATUS, CONDITION, FLAGS

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

Simple Queries [6.1.1]
void GetBoolean(enum pname, boolean *data);
void GetBooleanv(enum pname, int *data);
void GetIntegerv(enum pname, int64 *data);
void GetFloatv(enum pname, float *data);
void GetDoublev(enum pname, double *data);
void GetBooleani_v(uint target, uint index, boolean *data);
void GetIntegeri_v(uint target, uint index, int64 *data);
void GetInteger64i_v(uint target, uint index, int64 *data);

String Queries [6.1.6]
ubyte *GetString(name enum);
names: RENDERER, VENDOR, VERSION, SHADING_LANGUAGE_VERSION
ubyte *GetString(name enum, uint index);
name: EXTENSIONS
index: range is [0, NUM_EXTENSIONS - 1]

Hints [5.4]
void Hint(uint target, enum hint);
target: RENDERER_OFFSET, RENDERER_DRIVER_NAME
hint: STANDARD_RENDERING_HINT, POSTPROCESSING_HINT, COMPRESSION_HINT, LINE_PATTERN_HINT, FILL_MODE_HINT

OpenGL Shading Language 4.20 Reference Card

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, and fragment processors.

[n.n.n] and [Table n.n] refer to sections and tables in the OpenGL Shading Language 4.20 specification at www.opengl.org/registry

Operators & Expressions [5.1]
The following operators are numbered in order of precedence. Relational and equality operators evaluate to Boolean. Also see lessThan(), equal(), etc.

Types [4.1]

Floating-Point Opaque Types

Signed Integer Opaque Types (cont’d)

Unsigned Integer Opaque Types (cont’d)

Implicit Conversions

Aggregation of Basic Types

Arrays

Structs

Blocks

Preprocessor [3.3]
Preprocessor Directives

Preprocessor Operators

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\}] Points or normals
[\{f, g, b, a\}] Colors
[\{s, t, p, a\}] Texture coordinates

Continued

Continued
Subroutine type variables are assigned to functions through the subroutine type in a subroutine uniform variable declaration:

```
subroutine uniform subroutinename = subroutineVariableName;
```

For tessellation evaluation shaders:

- `triangles, quads, equal_spacing, isolines, fractional_even_odd_spacing, cw, ccw, point_mode`
- `division, integer_inverse_spacing`
- `fractional_even_spacing, fractional_odd_spacing`
- `triangular`, `linear` (for `triangles`)
- `equal_spacing` (for `quad`)
- `linear_spacing` (for `quad`)
- `invariant` (for `quad`)

For geometry shader inputs:

- `points, lines, lines_triangles_adjacency`, `triangles, invocations = integer-constant`

For fragment shaders only for redrawing built-in variable `gl_FragCoord`:

- `integer, origin_upper_left, pixel_center_integer`

For "in" only (not with variable declarations):

```
early_fragment_tests
```

Opaque Uniform Layout Qualifiers

```
opaque uniform layout(qualifiers) variableName;
```

Atomic Counter Layout Qualifiers

```
atomic_counter uniform layout(qualifiers) variableName;
```

Qualifiers

Storage Qualifiers

```
uniform const sampler[] = defaultValue;
```

Auxiliary Storage Qualifiers

```
centroid float centroid[dimension];
```

Uniform Qualifiers

```
declares global variables with same values across entire primitive processed. Examples:

- vec4 lightPosition;
- uniform vec3 color = vec3(0.7, 0.7, 0.2);
```

Uniform-Block Layout Qualifiers

```
layout(qualifiers) blockName {

type1 arg1, ..., typeN argN;
}
```

Operations and Constructors

Vector & Matrix

```
dvec2(dvec3, dvec2); 

dvec2(dvec2, dvec2); 

Vec3 & Mat 2

- mat2x3(vec2, float, vec2, float);
- mat2(vec2, vec2);
```

Uniform Language

```
// sets 4th element, 3rd col. to 2.0
// sets 2nd col. to all 2.0
// m is a matrix
```

Structure Fields or Operations on Matrices and Vectors

```
m = f * m; 

v = f * v;

f = dot(v, v);
```

Parameter Qualifiers

```
for function parameters passed into function
const for function parameters that cannot be written to
out for function parameters passed back out of function, but not initialized when passed in
in for function parameters passed both into and out of a function
```

Built-In Variables

```
float gl_TessLevelInner[2];
float gl_TessLevelOuter[4];
```

Tessellation Control Language

```
uint gl_tessCoord;
```

Tessellation Evaluation Language

```
uint gl_tessCoord;
```

Statements and Structure

Function call by value-return

```
for (;;) { break, continue }
```

Selection

```
if ( ) { } else { }
```

Switch

```
case integer: break; 
```

Try/Except

```
try { ... } catch ( ) { ... } 
```

Return

```
return in main();
```

Subroutines

```
subroutine subroutinename = subroutineVariableName;
```

If pragma STDLGL

```
Invertinval gl_Position;
```

Building Shaders

```
void glUseProgram(GLuint program);
```

Invertinval in gl_Position;

```
if (!ccw) return;
```

Version Numbers

```
at version 4.0, the following new keywords apply:
```

Inventor

```
set to current shader interpretation
```

Memory Qualifiers

```
coherent reads and writes are coherent with other shader invocations
```

Order of Qualification

```
when multiple qualifiers are present in a declaration they may appear in any order, but must all appear before the type. the layout qualifier is the only qualifier that can appear more than once.
```

Precise Qualifier

```
ensures that operations are executed in stated order with operator consistency. requires two identical multiplications, followed by an add.
```

Special Qualifiers

```
none (default) local read/write memory or input parameter
```

Statements and Structure Iteration and Jumps

```
call by value-return
```

Iteration

```
for (;;) { break, continue }
```

Selection

```
if ( ) { } else { }
```

Switch

```
case integer: break; 
```

Try/Except

```
try { ... } catch ( ) { ... } 
```

Return

```
return in main();
```

Subroutines

```
subroutine subroutinename = subroutineVariableName;
```

If pragma STDLGL

```
Invertinval gl_Position;
```

Building Shaders

```
void glUseProgram(GLuint program);
```

Invertinval in gl_Position;

```
if (!ccw) return;
```

Version Numbers

```
at version 4.0, the following new keywords apply:
```

Inventor

```
set to current shader interpretation
```

Memory Qualifiers

```
coherent reads and writes are coherent with other shader invocations
```

Order of Qualification

```
when multiple qualifiers are present in a declaration they may appear in any order, but must all appear before the type. the layout qualifier is the only qualifier that can appear more than once.
```

Precise Qualifier

```
ensures that operations are executed in stated order with operator consistency. requires two identical multiplications, followed by an add.
```

Special Qualifiers

```
none (default) local read/write memory or input parameter
```

Statements and Structure Iteration and Jumps

```
call by value-return
```

Iteration

```
for (;;) { break, continue }
```

Selection

```
if ( ) { } else { }
```

Switch

```
case integer: break; 
```

Try/Except

```
try { ... } catch ( ) { ... } 
```

Return

```
return in main();
```

Subroutines

```
subroutine subroutinename = subroutineVariableName;
```

If pragma STDLGL

```
Invertinval gl_Position;
```

Building Shaders

```
void glUseProgram(GLuint program);
```

Invertinval in gl_Position;

```
if (!ccw) return;
```

Version Numbers

```
at version 4.0, the following new keywords apply:
```

Inventor

```
set to current shader interpretation
```

Memory Qualifiers

```
coherent reads and writes are coherent with other shader invocations
```

Order of Qualification

```
when multiple qualifiers are present in a declaration they may appear in any order, but must all appear before the type. the layout qualifier is the only qualifier that can appear more than once.
```

Precise Qualifier

```
ensures that operations are executed in stated order with operator consistency. requires two identical multiplications, followed by an add.
```

Special Qualifiers

```
none (default) local read/write memory or input parameter
```
Built-In Functions

Angle & Trig. Functions

- `sin(T)` computes the sine of the angle.
- `cos(T)` computes the cosine of the angle.
- `tan(T)` computes the tangent of the angle.
- `asin(T)` computes the inverse sine (in radians).
- `acos(T)` computes the inverse cosine (in radians).
- `atan(T)` computes the inverse tangent (in radians).

Exponential Functions

- `exp(T)` computes the exponential function of `T`.
- `log(T)` computes the natural logarithm of `T`.
- `pow(T, T)` computes `T` raised to the power of `T`.

Common Functions

- `sqrt(T)` computes the square root of `T`.
- `abs(T)` returns the absolute value of `T`.
- `sign(T)` returns 1 if `T` is positive, -1 if `T` is negative, and 0 if `T` is zero.

Vector Functions

- `cross(T)` returns the cross product of two vectors.
- `refract(T, N, N)` returns the refraction vector.
- `reflect(T, N, N)` returns the reflection vector.
- `distance(T)` returns the distance between two vectors.
- `normalize(T)` normalizes a vector.
- `dot(T, T)` returns the dot product of two vectors.
- `cross(T)` returns the cross product of two vectors.
- `magnitude(T)` returns the magnitude of a vector.

Matrix Functions

- `transpose(T)` returns the transpose of a matrix.
- `determinant(T)` returns the determinant of a matrix.
- `inverse(T)` returns the inverse of a matrix.

Geometric Functions

- `frustum(T)` returns the perspective matrix.
- `ortho(T)` returns the orthographic projection matrix.
- `lookAt(T)` returns the view matrix.
- `projection(T)` returns the projection matrix.
- `modelView(T)` returns the model-view matrix.
- `modelViewProjection(T)` returns the model-view-projection matrix.
- `viewport(T)` returns the viewport matrix.

Floating-Point Pack/Unpack

- `packUnorm4x8(T)` packs 4 float values into a 32-bit integer.
- `unpackUnorm4x8(T)` unpacks a 32-bit integer into 4 float values.

Use of `T` or `N` within each function call must be the same.

Type Abbreviations for Built-In Functions:

- Tf: float, vecn, dvecn, bool
- Td: float, vecd, dvecd, bvecd
- Ti: int, uint, ivec, uvec
- Tiu: int, uint
- Ts: svec, uvec
- Tfd: float, vec, dvec
- Tiu: int, uint
- Td: float, vecd, dvecd
- Tiu: int, uint
- Ts: svec, uvec
- Tfd: float, vec, dvec
- Ti: int, uint
- Ts: svec, uvec

Matrix Functions (continued)

- `transpose(T)` returns the transpose of a matrix.
- `determinant(T)` returns the determinant of a matrix.
- `inverse(T)` returns the inverse of a matrix.

Vector Relational Functions

- `greaterThanEqual(T, T)` returns true if `T` is greater than or equal to `T`.
- `lessThanEqual(T, T)` returns true if `T` is less than or equal to `T`.
- `equal(T, T)` returns true if `T` is equal to `T`.
- `notEqual(T, T)` returns true if `T` is not equal to `T`.

Integer Functions

- `add(T)` returns the sum of `T`.
- `sub(T)` returns the difference of `T`.
- `mul(T)` returns the product of `T`.
- `div(T)` returns the quotient of `T`.
- `mod(T)` returns the remainder of `T`.

Built-In Variables (cont.)

- `MaxTextureImageUnits` is the maximum number of texture units.
- `MaxVertexUniformComponents` is the maximum number of vertex uniform components.
- `MaxFragmentUniformComponents` is the maximum number of fragment uniform components.
- `MaxGeometryOutputVertices` is the maximum number of geometry output vertices.
- `MaxGeometryOutputComponents` is the maximum number of geometry output components.
- `MaxCombinerInputPorts` is the maximum number of combiner input ports.
- `MaxCombinerOutputPorts` is the maximum number of combiner output ports.
- `MaxGeometryTextureImageUnits` is the maximum number of geometry texture image units.
- `MaxSampleBuffers` is the maximum number of sample buffers.
- `MaxClipDistances` is the maximum number of clip distances.
- `MaxViewports` is the maximum number of viewports.
- `MaxVertexTextureImageUnits` is the maximum number of vertex texture image units.
- `MaxProgramsInEffect` is the maximum number of programs in effect.
- `MaxProgramsInUse` is the maximum number of programs in use.
- `MaxVertexAtomicCounters` is the maximum number of vertex atomic counters.
- `MaxFragmentAtomicCounters` is the maximum number of fragment atomic counters.
- `MaxGeometryAtomicCounters` is the maximum number of geometry atomic counters.
- `MaxCombinerInputPorts` is the maximum number of combiner input ports.
- `MaxCombinerOutputPorts` is the maximum number of combiner output ports.
- `MaxProgramsInEffect` is the maximum number of programs in effect.
- `MaxProgramsInUse` is the maximum number of programs in use.
- `MaxGeometryTextureImageUnits` is the maximum number of geometry texture image units.
- `MaxProgramsInEffect` is the maximum number of programs in effect.
- `MaxProgramsInUse` is the maximum number of programs in use.
- `MaxGeometryTextureImageUnits` is the maximum number of geometry texture image units.
- `MaxProgramsInEffect` is the maximum number of programs in effect.
- `MaxProgramsInUse` is the maximum number of programs in use.
- `MaxGeometryTextureImageUnits` is the maximum number of geometry texture image units.
Texture Functions [8.9]  
Available to vertex, geometry, and fragment shaders. 

Texture atomicCounter  
uint atomicCounterIncrement()  
Atomically returns the value of an atomic counter.  

Texture atomicCounterDecrement()  
Atomically decrements counter for c, then returns value of counter for c.  

Texture atomicCounter()  
Atomically returns the value of an atomic counter.  

Image Functions [8.8]  
In these image functions, /IMAGE_PARAMS may be one of the following:  
gimage[1D] Buffer image, int P  
gimage[2D] 2D image, ivec2 P  
gimage[3D] 3D image, ivec3 P  
gimage[2D] 2DMS image, ivec2 P, P sample  
gimage[2D] 2DMS image, ivec3 P, P sample  

Texture Functions [8.9]  
Available to vertex, geometry, and fragment shaders. 

TexCoord Lookup Functions [8.9]  
Texture lookup as in gimage.  

TexCoord Lookup with Explicit LOD  
Texture lookup both projectively as in gimage, and with explicit gradient as in gimageProjGrad.  

TexCoord Lookup with Explicit LOD  
Texture lookup both projectively as in gimage, and with explicit gradient as in gimageProjGrad.  

TexCoord Gather Offset  
Texture gather as in gimage, except offset added before texture lookup.  

TexCoord Gather Offset  
Texture gather as in gimage, except offset added before texture lookup.  

ViewController  
Shader Invocation: Synchronizes across shader invocations.  

Memory Barrier  
Shader Invocation: Controls the ordering of memory transactions issued by a single shader invocation.
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. In order to fully take advantage of modern OpenGL, pay close attention to how to most efficiently use the new buffer types.

- Blue blocks indicate various buffers that feed or get fed by the OpenGL pipeline.
- Green blocks indicate features new or significantly changed with OpenGL 4.x.

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.
### OpenGL 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.

<table>
<thead>
<tr>
<th>Column</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>ActiveShaderProgram</td>
<td>1</td>
</tr>
<tr>
<td>ActiveTexture</td>
<td>3</td>
</tr>
<tr>
<td>Angle Functions</td>
<td>7</td>
</tr>
<tr>
<td>Arrays</td>
<td>6</td>
</tr>
<tr>
<td>Asynchronous Queries</td>
<td>3</td>
</tr>
<tr>
<td>Atomic-Counter Functions</td>
<td>8</td>
</tr>
<tr>
<td>AttachShader</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Begin/EndConditionalRender</td>
<td>2</td>
</tr>
<tr>
<td>Begin/EndQuery(Indexed)</td>
<td>2</td>
</tr>
<tr>
<td>Begin/EndTransformFeedback</td>
<td>2</td>
</tr>
<tr>
<td>BindAttribLocation</td>
<td>2</td>
</tr>
<tr>
<td>BindBuffer</td>
<td>1</td>
</tr>
<tr>
<td>BindFramebuffer</td>
<td>4</td>
</tr>
<tr>
<td>BindFramebufferRange</td>
<td></td>
</tr>
</tbody>
</table>