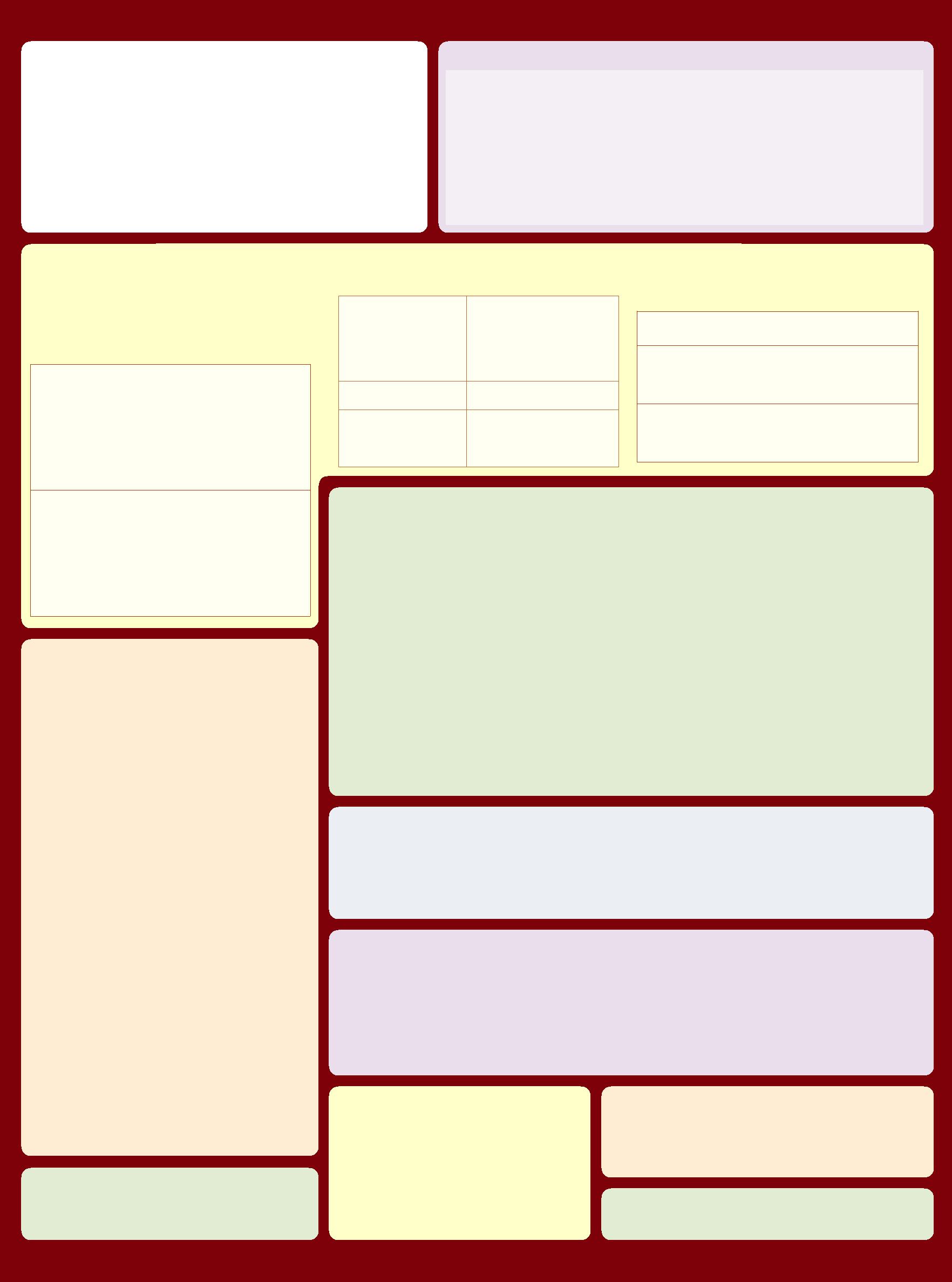
**WebGL 1.0 API Quick Reference Card**

**WebGL®** is a software interface for accessing graphics hardwarefrom within a web browser. Based on OpenGL ES 2.0, WebGL allows a

**Page 1**



**The WebGL Context and getContext() [2.5]**

programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of 3D objects.

* **[n.n.n]** refers to sections in the WebGL 1.0 specification, available atwww.khronos.org/webgl
* **Content marked in purple** does **not** have a corresponding function inOpenGL ES. The OpenGL ES 2.0 specification is available at www.khronos.org/registry/gles

**除非另有说明，否则WebGL函数调用的行为与OpenGL ES对应的函数相同。.**

This object manages OpenGL state and renders to a drawing buffer, which must is also be created at the same time of as the context creation.

Create the WebGLRenderingContext object and drawing buffer by calling the **getContext** method of a given HTMLCanvasElement object with the exact string ‘webgl’. The drawing buffer is also created by **getContext**.

For example:

<!DOCTYPE html>

<html><body>

<canvas id=”c”></canvas> <script type=”text/javascript”>

var canvas = document.getElementById(“c”); var gl = canvas.getContext(“webgl”); gl.clearColor(1.0, 0.0, 0.0, 1.0); gl.clear(gl.COLOR\_BUFFER\_BIT);

</script>

</body></html>

**Interfaces**

接口是可选的请求，可被实现忽略. 实际值可以通过**getContextAttributes** 查看.

**WebGLContextAttributes [5.2]**

接口包含请求的绘图表面属性，并作为第二个参数传递给gl.**getContext**.

**Attributes:**

|  |  |
| --- | --- |
| **alpha** | 默认值: true |
| 如果为true,绘制的buffer中的alpha通道执行OpenGL目标alpha和网页合成。 |  |
|  |  |
| **depth** | 默认值: true |
| 如果为true绘制的buffer的深度至少为16bits |  |
|  |  |
| **stencil** | 默认值: false |
| If true, requests a stencil buffer of at least 8 bits. |  |

**WebGLObject [5.3]**

This is the parent interface for all WebGL resource objects.

**Resource interface objects:**

|  |  |
| --- | --- |
| **WebGLBuffer [5.4]** | OpenGL Buffer Object. |
| **WebGLProgram [5.6]** | OpenGL Program Object. |
| **WebGLRenderbuffer [5.7]** | OpenGL Renderbuffer Object. |
| **WebGLShader [5.8]** | OpenGL Shader Object. |
| **WebGLTexture [5.9]** | OpenGL Texture Object. |

**WebGLUniformLocation [5.10]** Location of a uniform variable in ashader program.

|  |  |
| --- | --- |
| **WebGLActiveInfo [5.11]** | Information returned from calls to |
|  | **getActiveAttrib** and **getActiveUniform**. |
|  | Has the following read-only |
|  | properties: name, size, type. |

**WebGLRenderingContext [5.13]**

This is the prinicpal interface in WebGL. The functions listed on this reference card are available within this interface.

**Attributes:**

**canvas** Type: HTMLCanvasElement A reference to the canvas element which created this context.

**drawingBufferWidth** Type: GLsizei

The actual width of the drawing buffer, which may differ from the width attribute of the HTMLCanvasElement if the implementation is unable to satisfy the requested width or height.

**drawingBufferHeight** Type: GLsizei

The actual height of the drawing buffer, which may differ from the height attribute of the HTMLCanvasElement if the implementation is unable to satisfy the requested width or height

|  |  |
| --- | --- |
| **antialias** | 默认值: true |
| If true, requests drawing buffer with antialiasing using its choice | |
| of technique (multisample/supersample) and quality. | |
|  |  |
| **premultipliedAlpha** | 默认值: true |
| If true, requests drawing buffer which contains colors with | |
| premultiplied alpha. (Ignored if Alpha is false.) |  |
|  |  |
| **preserveDrawingBuffer** | 默认值: false |

If true, requests that contents of the drawing buffer remain in between frames, at potential performance cost.

**ArrayBuffer and Typed Arrays [5.12]**

Data is transferred to WebGL using ArrayBuffer and views. Buffers represent unstructured binary data, which can be modified using one or more typed array views.

**Buffers**

**ArrayBuffer**(ulong*byteLength*)

ulong byteLength: read-only, length of view in bytes.

Creates a new buffer. To modify the data, create one or more views referencing it.

*ViewType*(ArrayBuffer *buffer*, [optional] ulong *byteOffset*,[optional] ulong *length*)

Create a new view of given buffer, starting at optional byte offset, extending for optional length elements.

ArrayBuffer *buffer*: Read-only, buffer backing this view

ulong *byteOffset*: Read-only, byte offset of view start in buffer

ulong *length*: Read-only, number of elements in this view

Other Properties

ulong *byteLength*: Read-only, length of view in bytes.

const ulong *BYTES\_PER\_ELEMENT*: element size in bytes.

**Per-Fragment Operations [5.13.3]**

void **blendColor**(float *red*, float *green*, float *blue*, float *alpha*)

void **blendEquation**(enum *mode*)

*mode:* See *modeRGB* for**blendEquationSeparate**

void **blendEquationSeparate**(enum *modeRGB*, enum *modeAlpha*)

*modeRGB,* and *modeAlpha*: FUNC\_ADD, FUNC\_SUBTRACT,FUNC\_REVERSE\_SUBTRACT

void **blendFunc**(enum *sfactor*, enum *dfactor*)

*sfactor:* Same as for *dfactor*, plus SRC\_ALPHA\_SATURATE

*dfactor:* ZERO, ONE, [ONE\_MINUS\_]SRC\_COLOR,

[ONE\_MINUS\_]DST\_COLOR, [ONE\_MINUS\_]SRC\_ALPHA,

[ONE\_MINUS\_]DST\_ALPHA, [ONE\_MINUS\_]CONSTANT\_COLOR,

[ONE\_MINUS\_]CONSTANT\_ALPHA

**Note:** Src and dst factors may not both reference constant color

void **blendFuncSeparate**(enum *srcRGB*, enum *dstRGB*, enum *srcAlpha*, enum *dstAlpha*)

*srcRGB, srcAlpha*: See *sfactor* for**blendFunc**

*dstRGB, dstAlpha*: See *dfactor* for**blendFunc**

**Note:** Src and dst factors may not both reference constant color

void **depthFunc**(enum *func*)

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

void **sampleCoverage**(float *value*, bool *invert*)

void **stencilFunc**(enum *func*, int *ref*, uint *mask*)

*func:* NEVER, ALWAYS, LESS, LEQUAL, [NOT]EQUAL, GREATER,GEQUAL

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

*face:* FRONT, BACK, FRONT\_AND\_BACK

*func:* NEVER, ALWAYS, LESS, LEQUAL, [NOT]EQUAL, GREATER,GEQUAL

void **stencilOp**(enum *fail*, enum *zfail*, enum *zpass*)

**Views**

In the following, *ViewType* may be Int8Array, Int16Array, Int32Array, Uint8Array, Uint16Array, Uint32Array, Float32Array.

*ViewType*(ulong *length*)

Creates a view and a new underlying buffer.

ulong *length*: Read-only, number of elements in this view.

*ViewType*(*ViewType other*)

Creates new underlying buffer and copies ‘other’ array.

*ViewType*(type[] *other*)

Creates new underlying buffer and copies ‘other’ array.

**Whole Framebuffer Operations [5.13.3]**

void **clear**(ulong *mask*) **[5.13.11]**

*mask:* Bitwise OR of {COLOR, DEPTH, STENCIL}\_BUFFER\_BIT

void **clearColor**(float *red*, float *green*, float *blue*, float *alpha*)

void **clearDepth**(float *depth*)

*depth:* Clamped to the range 0 to 1.

**Buffer Objects [5.13.5]**

Once bound, buffers may not be rebound with a different Target.

void **bindBuffer**(enum *target*, Object *buffer*)

*target:* ARRAY\_BUFFER, ELEMENT\_ARRAY\_BUFFER

void **bufferData**(enum *target*, long *size*, enum *usage*)

*target:* ARRAY\_BUFFER, ELEMENT\_ARRAY\_BUFFER

*usage*:STATIC\_DRAW, STREAM\_DRAW, DYNAMIC\_DRAW

void **bufferData**(enum *target*, Object *data*, enum *usage*)

*target* and *usage:* Same as for**bufferData**above

Methods

*view*[i] = get/set element i

**set**(*ViewType*other,[optional] ulongoffset)

**set**(type[]*other*,[optional] ulong*offset*)

Replace elements in this view with those from other, starting at optional offset.

*ViewType* **subset**(long *begin*,[optional] long *end*)

Return a subset of this view, referencing the same underlying buffer.

void **clearStencil**(int *s*)

void **colorMask**(bool *red*, bool *green*, bool *blue*, bool *alpha*)

void **depthMask**(bool *flag*)

void **stencilMask**(uint *mask*)

void **stencilMaskSeparate**(enum *face*, uint *mask*)

*face:* FRONT, BACK, FRONT\_AND\_BACK

void **bufferSubData**(enum *target*, long *offset*, Object *data*)

*target:* ARRAY\_BUFFER, ELEMENT\_ARRAY\_BUFFER

Object **createBuffer**()

**Note:** Corresponding OpenGL ES function is **GenBuffers**

void **deleteBuffer**(Object *buffer*)

any **getBufferParameter**(enum *target*, enum *pname*)

*target:* ARRAY\_BUFFER, ELEMENT\_ ARRAY\_BUFFER

*pname:* BUFFER\_SIZE, BUFFER\_USAGE

bool **isBuffer**(Object *buffer*)

*fail, zfail*, and *zpass*: KEEP, ZERO, REPLACE, INCR, DECR, INVERT,INCR\_WRAP, DECR\_WRAP

void **stencilOpSeparate**(enum *face*, enum *fail*, enum *zfail*, enum *zpass*)

*face:* FRONT, BACK, FRONT\_AND\_BACK

*fail, zfail*, and *zpass*: See fail, *zfail*, and *zpass* for**stencilOp**

**Detect and Enable Extensions [5.13.14]**

string[ ] **getSupportedExtensions**()

object **getExtension**(string *name*)

**View and Clip [5.13.3 - 5.13.4]**

The viewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Drawing buffer size is determined by the HTMLCanvasElement.

void **depthRange**(float *zNear*, float *zFar*)

*zNear:* Clamped to the range 0 to 1 Must be <= *zFar*

*zFar:* Clamped to the range 0 to 1.

void **scissor**(int *x*, int *y*, long *width*, long *height*)

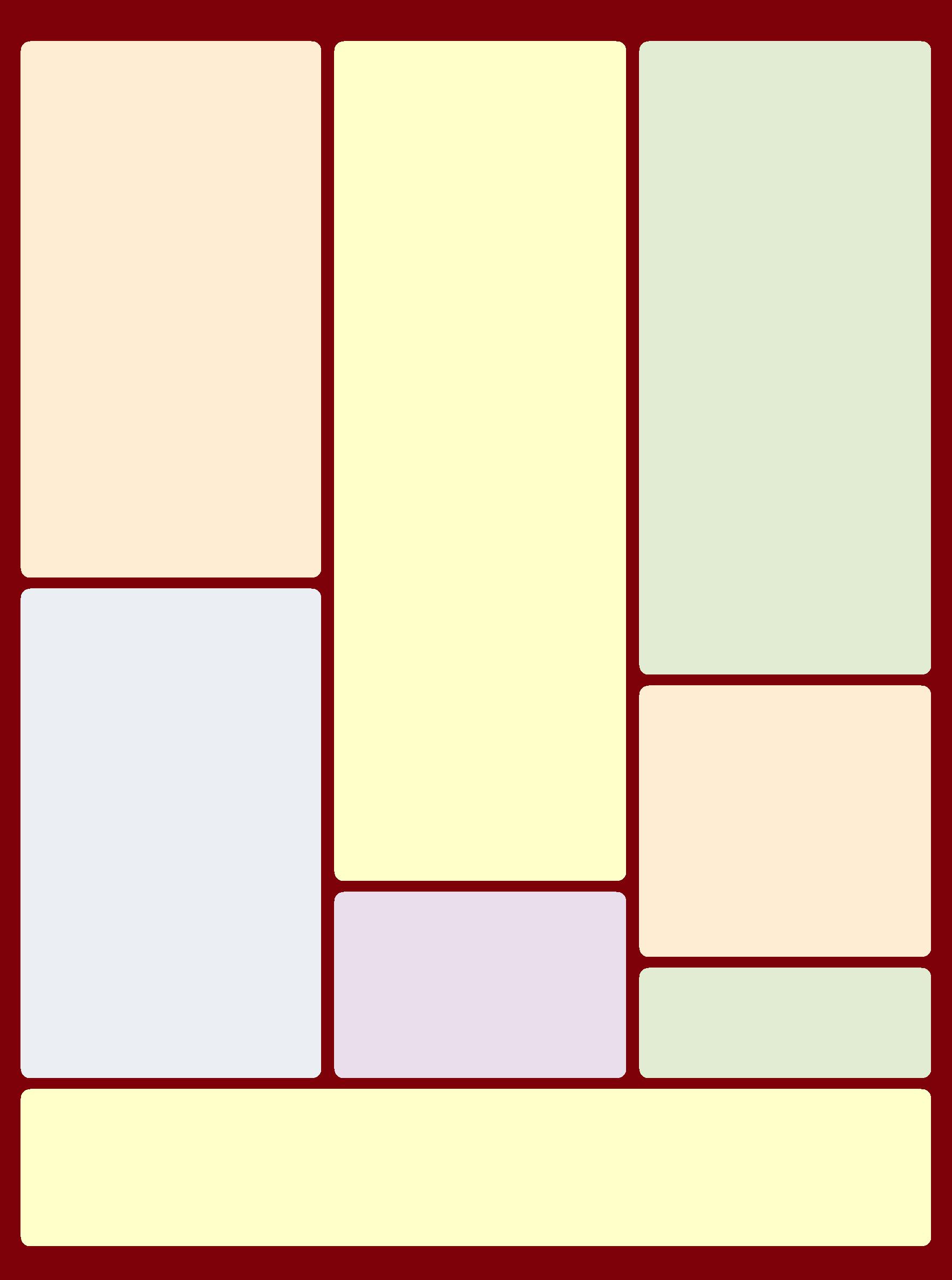
void **viewport**(int *x*, int *y*, long *width*, long *height*)

|  |  |  |
| --- | --- | --- |
| **Rasterization [5.13.3]** | void **frontFace**(enum *mode*) |  |
| void **cullFace**(enum *mode*) | *mode:* CCW, CW |  |
| void **lineWidth**(float *width*) |  |
| *mode:* BACK, FRONT\_AND\_BACK, |  |
| void **polygonOffset**(float *factor*, |  |
| FRONT |  |
|  | float *units*) |  |

**Detect context lost events [5.13.13]**

bool **isContextLost**()

**©2011 Khronos Group - Rev. 0618** **www.khronos.org/webgl**



**WebGL 1.0 API Quick Reference Card - Page 2**

**Page 2**

**Programs and Shaders [5.13.9]**

Rendering with OpenGL ES 2.0 requires the use of shaders.

Shaders must be loaded with a source string (**shaderSource**),

compiled (**compileShader**), and attached to a program

(**attachShader**) which must be linked (**linkProgram**) and then

used (**useProgram**).

void **attachShader**(Object *program*, Object *shader*)

void **bindAttribLocation**(Object *program*, uint *index*, string *name*)

void **compileShader**(Object *shader*)

Object **createProgram**()

Object **createShader**(enum *type*)

*type:* VERTEX\_SHADER, FRAGMENT\_SHADER

void **deleteProgram**(Object *program*)

void **deleteShader**(Object *shader*)

void **detachShader**(Object *program*, Object *shader*)

Object[ ] **getAttachedShaders**(Object *program*)

any **getProgramParameter**(Object *program*, enum *pname*)

**Note:** Corresponding OpenGL ES function is **GetProgramiv**

*pname:* DELETE\_STATUS, LINK\_STATUS, VALIDATE\_STATUS,ATTACHED\_SHADERS, ACTIVE\_{ATTRIBUTES, UNIFORMS}

string **getProgramInfoLog**(Object *program*)

any **getShaderParameter**(Object *shader*, enum *pname*)

**Note:** Corresponding OpenGL ES function is **GetShaderiv**

*pname:* SHADER\_TYPE, DELETE\_STATUS, COMPILE\_STATUS

string **getShaderInfoLog**(Object *shader*)

string **getShaderSource**(Object *shader*)

bool **isProgram**(Object *program*)

bool **isShader**(Object *shader*)

void **linkProgram**(Object *program*)

void **shaderSource**(Object *shader*, string *source*)

void **useProgram**(Object *program*)

void **validateProgram**(Object *program*)

**Uniforms and Attributes [5.13.10]**

Values used by the shaders are passed in as uniform of vertex attributes.

void **disableVertexAttribArray**(uint *index*)

*index:* [0, MAX\_VERTEX\_ATTRIBS - 1]

void **enableVertexAttribArray**(uint *index*)

*index:* [0, MAX\_VERTEX\_ATTRIBS - 1]

Object **getActiveAttrib**(Object *program*, uint *index*)

Object **getActiveUniform**(Object *program*, uint *index*)

ulong **getAttribLocation**(Object *program*, string *name*)

any **getUniform**(Object *program*, uint *location*)

uint **getUniformLocation**(Object *program*, string *name*)

any **getVertexAttrib**(uint *index*, enum *pname*)

*pname:* CURRENT\_VERTEX\_ATTRIB , VERTEX\_ATTRIB\_ARRAY\_

{BUFFER\_BINDING, ENABLED, SIZE, STRIDE, TYPE, NORMALIZED}

long **getVertexAttribOffset**(uint *index*, enum *pname*)

**Note:** Corres. OpenGL ES function is **GetVertexAttribPointerv**

*pname:* VERTEX\_ATTRIB\_ARRAY\_POINTER

void **uniform[1234][fi]**(uint *location*, ...)

void **uniform[1234][fi]v**(uint *location*, Array *value*)

void **uniformMatrix[234]fv**(uint *location*, bool *transpose*, *Array*)

*transpose:* FALSE

void **vertexAttrib[1234]f**(uint *index*, ...)

void **vertexAttrib[1234]fv**(uint *index*, Array *value*)

void **vertexAttribPointer**(uint *index*, int *size*, enum *type*, bool *normalized*, long *stride*, long *offset*)

*type:* BYTE, SHORT, UNSIGNED\_{BYTE, SHORT}, FIXED, FLOAT

*index:* [0, MAX\_VERTEX\_ATTRIBS - 1]

*stride:* [0, 255]

*offset, stride*: must be a multiple of the type size in WebGL

**Framebuffer Objects [5.13.6]**

Framebuffer objects provide an alternative rendering target to the drawing buffer.

void **bindFramebuffer**(enum *target*, Object *framebuffer*)

*target:* FRAMEBUFFER

enum **checkFramebufferStatus**(enum *target*)

*target:* FRAMEBUFFER

*Returns*:FRAMEBUFFER\_{COMPLETE, UNSUPPORTED},FRAMEBUFFER\_INCOMPLETE\_{ATTACHMENT, DIMENSIONS, MISSING\_ATTACHMENT}

**Texture Objects [5.13.8]**

Texture objects provide storage and state for texturing operations. WebGL adds an error for operations relating to the currently bound texture if no texture is bound.

void **activeTexture**(enum *texture*) **[5.13.3]**

*texture:* [TEXTURE0..TEXTURE*i*] where *i* =

MAX\_COMBINED\_TEXTURE\_IMAGE\_UNITS - 1

void **bindTexture**(enum *target*, Object *texture*)

*target:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP

void **copyTexImage2D**(enum *target*, int *level*, enum *internalformat*, int *x*, int *y*, long *width*, long *height*, int *border*)

*target:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP\_POSITIVE\_{X,Y,Z}TEXTURE\_CUBE\_MAP\_NEGATIVE\_{X,Y,Z}

*internalformat:* ALPHA, LUMINANCE, LUMINANCE\_ALPHA, RGB[A]

void **copyTexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *x*, int *y*, long *width*, long *height*)

*target:* See *target* for**copyTexImage2D**

Object **createTexture**()

**Note:** Corresponding OpenGL ES function is **GenTextures**

void **deleteTexture**(Object *texture*)

void **generateMipmap**(enum *target*)

*target:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP

any **getTexParameter**(enum *target*, enum *pname*)

*target:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP

*pname:* TEXTURE\_WRAP\_{S, T}, TEXTURE\_{MIN, MAG}\_FILTER

bool **isTexture**(Object *texture*)

void **texImage2D**(enum *target*, int *level*,

enum *internalformat*, long *width*, long *height*,

int *border*, enum *format*, enum *type*, Object *pixels*)

void **texImage2D**(enum *target*, int *level*,

enum *internalformat*, enum *format*, enum *type*, Object *object*)

**Note:** The following values apply to all variations of **texImage2D**.

*target:* See *target* for**copyTexImage2D**

*internalformat:* See *internalformat* for**copyTexImage2D**

*format:* ALPHA, RGB, RGBA, LUMINANCE, LUMINANCE\_ALPHA

*type*:UNSIGNED\_BYTE, UNSIGNED\_SHORT\_5\_6\_5,UNSIGNED\_SHORT\_4\_4\_4\_4, UNSIGNED\_SHORT\_5\_5\_5\_1

*object:* pixels of type ImageData, image of type HTMLImageElement,canvas of type HTMLCanvasElement,

video of type HTMLVideoElement

void **texParameterf**(enum *target*, enum *pname*, float *param*)

*target:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP

*pname:* TEXTURE\_WRAP\_{S, T}, TEXTURE\_{MIN, MAG}\_FILTER

void **texParameteri**(enum *target*, enum *pname*, int *param*)

*target:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP

*pname:* TEXTURE\_WRAP\_{S, T}, TEXTURE\_{MIN, MAG}\_FILTER

void **texSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, long *width*, long *height*, enum *format*, enum *type*, Object *pixels*)

void **texSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, enum *format*, enum *type*, Object *object*)

**Note:** Following values apply to all variations of **texSubImage2D**.

*target:* TEXTURE\_CUBE\_MAP\_POSITIVE\_{X, Y, Z},TEXTURE\_CUBE\_MAP\_NEGATIVE\_{X, Y, Z}

*format* and *type:* See *format* and *type* for**texImage2D**

*object:* Same as for**texImage2D**

**Writing to the Draw Buffer [5.13.11]**

When rendering is directed to drawing buffer, OpenGL ES 2.0 rendering calls cause the drawing buffer to be presented to the HTML page compositor at start of next compositing operation.

void **drawArrays**(enum *mode*, int *first*, long *count*)

*mode:* POINTS, LINE\_STRIP, LINE\_LOOP, LINES, TRIANGLE\_STRIP,TRIANGLE\_FAN, TRIANGLES

*first:* May not be a negative value.

void **drawElements**(enum *mode*, long *count*, enum *type*, long *offset*)

*mode:* POINTS, LINE\_STRIP, LINE\_LOOP, LINES, TRIANGLE\_STRIP,TRIANGLE\_FAN, TRIANGLES

*type:* UNSIGNED\_BYTE, UNSIGNED\_SHORT

Object **createFramebuffer**()

**Note:** Corresponding OpenGL ES function is **GenFramebuffers**

void **deleteFramebuffer**(Object *buffer*)

void **framebufferRenderbuffer**(enum *target*, enum *attachment*, enum *renderbuffertarget*, Object *renderbuffer*)

*target:* FRAMEBUFFER

*attachment:* COLOR\_ATTACHMENT0, {DEPTH, STENCIL}\_ATTACHMENT

*renderbuffertarget:* RENDERBUFFER

bool **isFramebuffer**(Object *framebuffer*)

**Special Functions [5.13.3]**

contextStruct **getContextAttributes**() **[5.13.2]** void **disable**(enum *cap*)

*cap:* BLEND, CULL\_FACE, DEPTH\_TEST, DITHER,

POLYGON\_OFFSET\_FILL, SAMPLE\_ALPHA\_TO\_COVERAGE,

SAMPLE\_COVERAGE, SCISSOR\_TEST, STENCIL\_TEST

void **enable**(enum *cap*)

*cap:* See *cap* for**disable**

void **finish**() **[5.13.11]**

void **flush**() **[5.13.11]**

enum **getError**()

*Returns:* OUT\_OF\_MEMORY, INVALID\_{ENUM, OPERATION,FRAMEBUFFER\_OPERATION, VALUE}, NO\_ERROR, CONTEXT\_LOST\_WEBGL

any **getParameter**(enum *pname*)

*pname:* {ALPHA,RED, GREEN, BLUE, SUBPIXEL}\_BITS,

ACTIVE\_TEXTURE, ALIASED\_{LINE\_WIDTH, POINT\_SIZE}\_RANGE,

ARRAY\_BUFFER\_BINDING, BLEND\_DST\_{ALPHA, RGB},

BLEND\_EQUATION\_{ALPHA, RGB}, BLEND\_SRC\_{ALPHA, RGB},

BLEND[\_COLOR], COLOR\_{CLEAR\_VALUE, WRITEMASK},

[NUM\_]COMPRESSED\_TEXTURE\_FORMATS, CULL\_FACE[\_MODE],

CURRENT\_PROGRAM, DEPTH\_{BITS, CLEAR\_VALUE, FUNC,

RANGE, TEST, WRITEMASK}, ELEMENT\_ARRAY\_BUFFER\_BINDING,

DITHER, FRAMEBUFFER\_BINDING, FRONT\_FACE,

GENERATE\_MIPMAP\_HINT, LINE\_WIDTH,

MAX\_[COMBINED\_]TEXTURE\_IMAGE\_UNITS,

MAX\_{CUBE\_MAP\_TEXTURE, RENDERBUFFER, TEXTURE}\_SIZE,

MAX\_VARYING\_VECTORS, MAX\_VERTEX\_{ATTRIBS,

TEXTURE\_IMAGE\_UNITS, UNIFORM\_VECTORS},

MAX\_VIEWPORT\_DIMS, PACK\_ALIGNMENT,

POLYGON\_OFFSET\_{FACTOR, FILL, UNITS},

RENDERBUFFER\_BINDING, RENDERER, SAMPLE\_BUFFERS,

SAMPLE\_COVERAGE\_{INVERT, VALUE}, SAMPLES,

SCISSOR\_{BOX, TEST}, SHADING\_LANGUAGE\_VERSION,

STENCIL\_{BITS, CLEAR\_VALUE, TEST}, STENCIL\_[BACK\_]{FAIL,

FUNC, REF,VALUE\_MASK, WRITEMASK},

STENCIL\_[BACK\_]PASS\_DEPTH\_{FAIL, PASS},

TEXTURE\_BINDING\_{2D, CUBE\_MAP}, UNPACK\_ALIGNMENT,

UNPACK\_{COLORSPACE\_CONVERSION\_WEBGL, FLIP\_Y\_WEBGL,

PREMULTIPLY\_ALPHA\_WEBGL}, VENDOR, VERSION, VIEWPORT

void **hint**(enum *target*, enum *mode*)

*target:* GENERATE\_MIPMAP\_HINT

*hint:* FASTEST, NICEST, DONT\_CARE

bool **isEnabled**(enum *cap*)

*cap:* cap: See *cap* for**disable**

void **pixelStorei**(enum *pname*, int *param*) *pname:* UNPACK\_ALIGNMENT, PACK\_ALIGNMENT,

UNPACK\_{FLIP\_Y\_WEBGL, PREMULTIPLY\_ALPHA\_WEBGL}, UNPACK\_COLORSPACE\_CONVERSION\_WEBGL

**Renderbuffer Objects [5.13.7]**

Renderbuffer objects are used to provide storage for the individual buffers used in a framebuffer object.

void **bindRenderbuffer**(enum *target*, Object *renderbuffer*)

*target:* RENDERBUFFER

Object **createRenderbuffer**()

**Note:** Corresponding OpenGL ES function is **GenRenderbuffers**

void **deleteRenderbuffer**(Object *renderbuffer*)

any **getRenderbufferParameter**(enum *target*, enum *pname*)

*target:* RENDERBUFFER

*pname:* RENDERBUFFER\_{WIDTH,HEIGHT,INTERNAL\_FORMAT},

RENDEDRBUFFER\_{RED,GREEN,BLUE,ALPHA,DEPTH,STENCIL}\_SIZE

bool **isRenderbuffer**(Object *renderbuffer*)

void **renderbufferStorage**(enum *target*,

enum *internalformat*, long *width*, long *height*)

*target:* RENDERBUFFER

*internalformat:* DEPTH\_COMPONENT16, RGBA4, RGB5\_A1,RGB565, STENCIL\_INDEX8

**Read Back Pixels [5.13.12]**

Pixels in the current framebuffer can be read back into an ArrayBufferView object.

void **readPixels**(int *x*, int *y*, long *width*, long *height*, enum *format*, enum *type*, Object *pixels*)

*format:* RGBA

*type*:UNSIGNED\_BYTE

void **framebufferTexture2D**(enum *target*, enum *attachment*, enum *textarget*, Object *texture*, int *level*)

*target* and *attachment*: Same as for**framebufferRenderbuffer**

*textarget:* TEXTURE\_2D, TEXTURE\_CUBE\_MAP\_POSITIVE{X, Y, Z},TEXTURE\_CUBE\_MAP\_NEGATIVE{X, Y, Z},

any **getFramebufferAttachmentParameter**(enum *target*, enum *attachment*, enum *pname*)

*target* and *attachment*: Same as for**framebufferRenderbuffer**

*pname:* FRAMEBUFFER\_ATTACHMENT\_OBJECT\_{TYPE, NAME},

FRAMEBUFFER\_ATTACHMENT\_TEXTURE\_LEVEL, FRAMEBUFFER\_ATTACHMENT\_TEXTURE\_CUBE\_MAP\_FACE

**©2011 Khronos Group - Rev. 021168** **www.khronos.org/webgl**

**WebGL 1.0 API Quick Reference Card**

**The OpenGL® ES Shading Language** is two closely-related languages which are used to create shaders for the vertex and fragment processors contained in the OpenGL ES processing pipeline.

**[n.n.n]** refers to sections in the OpenGL ESShading Language 1.0 specification at [www.khronos.org/registry/gles](http://www.opengl.org/registry)

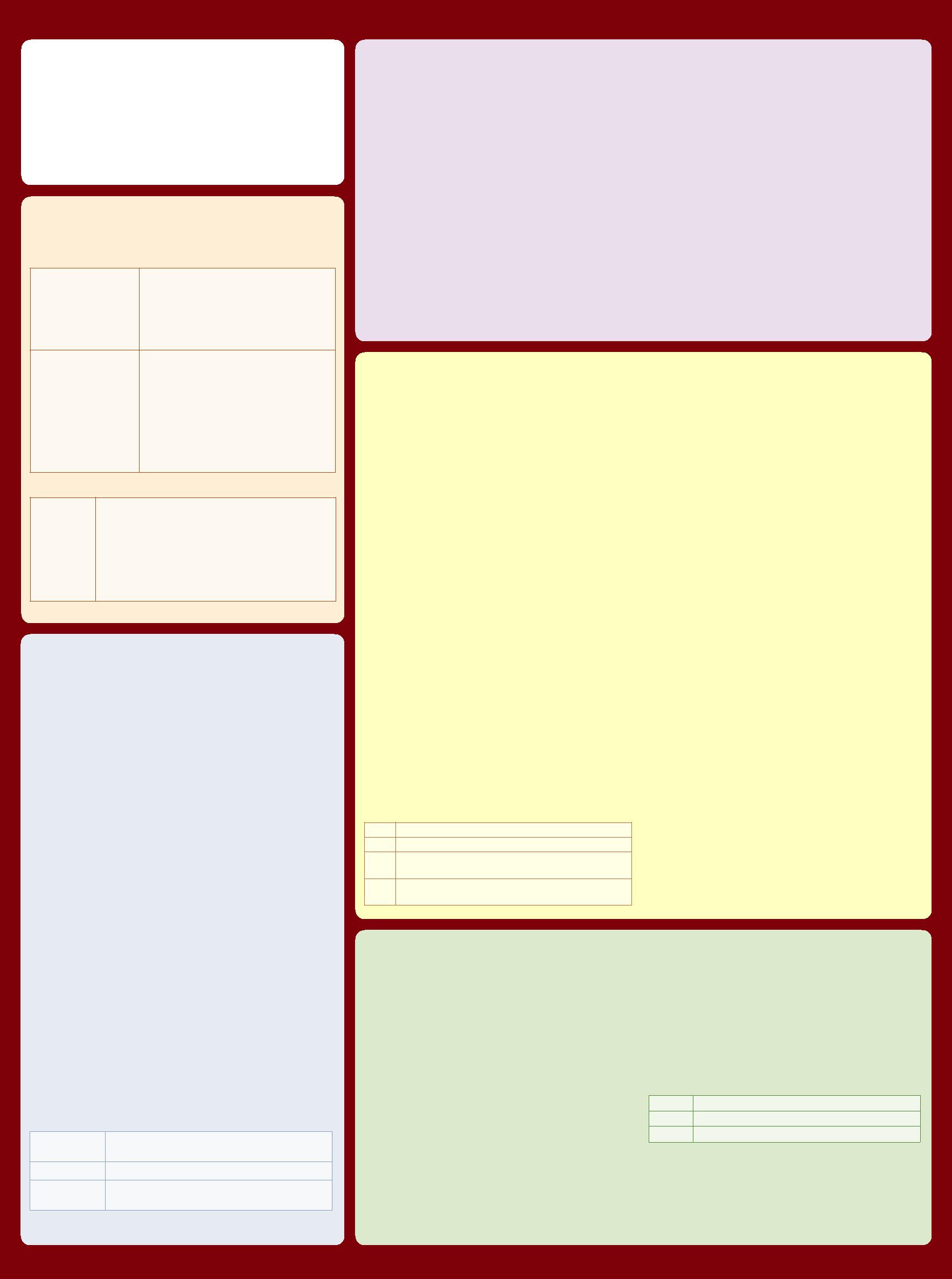
**Types [4.1]**

A shader can aggregate these using arrays and structures to build more complex types. There are no pointer types.

**Basic Types**

|  |  |
| --- | --- |
| **void** | no function return value or empty parameter list |
|  |  |
| **bool** | Boolean |
|  |  |
| **int** | signed integer |
|  |  |
| **float** | floating scalar |

**Page 3**



**Preprocessor [3.4]**

**Preprocessor Directives**

The number sign (#) can be immediately preceded or followed in its line by spaces or horizontal tabs.

*#* *#define* *#undef* *#if* *#ifdef* *#ifndef* *#else*

*#elif* *#endif* *#error* *#pragma* *#extension* *#version* *#line*

**Examples of Preprocessor Directives**

* “#version 100” in a shader program specifies that the program is written in GLSL ES version 1.00. It is optional. If used, it must occur before anything else in the program other than whitespace or comments.
* #extension *extension\_name* : *behavior*, where *behavior* can be require, enable, warn, or disable; and where *extension\_name is* the extension supported by the compiler

**Predefined Macros**

|  |  |  |
| --- | --- | --- |
| \_\_LINE\_\_ | Decimal integer constant that is one more than the number of preceding new-lines in the current |  |
| source string |  |
|  |  |
| \_\_VERSION\_\_ | Decimal integer, e.g.: 100 |  |
| GL\_ES | Defined and set to integer 1 if running on an OpenGL-ES Shading Language. |  |
| GL\_FRAGMENT\_PRECISION\_HIGH | 1 if highp is supported in the fragment language, else undefined [4.5.4] |  |
|  |  |  |

|  |  |
| --- | --- |
| **vec2, vec3, vec4** | n-component floating point vector |
|  |  |
| **bvec2, bvec3, bvec4** | Boolean vector |
|  |  |
| **ivec2, ivec3, ivec4** | signed integer vector |
|  |  |
| **mat2, mat3, mat4** | 2x2, 3x3, 4x4 float matrix |
|  |  |
| **sampler2D** | access a 2D texture |
|  |  |
| **samplerCube** | access cube mapped texture |

**Structures and Arrays [4.1.8, 4.1.9]**

|  |  |  |
| --- | --- | --- |
| **Structures** | **struct** *type-name*{ |  |
|  | *members* |  |
|  | } *struct-name*[]; | // optional variable declaration, |
|  |  | // optionally an array |
| **Arrays** | **float** foo[3]; |  |

* structures and blocks can be arrays
* only 1-dimensional arrays supported
* structure members can be arrays

**Operators and Expressions**

**Operators [5.1]** Numbered in order of precedence. Therelational and equality operators > < <= >= == != evaluate to a Boolean. To compare vectors component-wise, use functions such as lessThan(), equal(), etc.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Operator** | | **Description** | **Associativity** |  |
| 1. | **( )** |  | parenthetical grouping | N/A |  |
|  | **[ ]** |  | array subscript |  |  |
| 2. | **( )** |  | function call & constructor structure | L - R |  |
| **.** |  | field or method selector, swizzler |  |
|  |  |  |  |
|  | **++** | **--** | postfix increment and decrement |  |  |
| 3. | **++** | **--** | prefix increment and decrement | R - L |  |
| **+ -** | **!** | unary |  |
|  |  |  |
| 4. | **\*** | **/** | multiplicative | L - R |  |
| 5. | **+ -** | | additive | L - R |  |
| 7. | **< > <= >=** | | relational | L - R |  |
| 8. | **==** | **!=** | equality | L - R |  |
| 12. | **&&** | | logical and | L - R |  |
| 13. | **^^** | | logical exclusive or | L - R |  |
| 14. | **| |** | | logical inclusive or | L - R |  |
|  |  |  | selection (Selects one entire operand. |  |  |
| 15. | **? :** | | Use mix() to select individual components | L - R |  |
|  |  |  | of vectors.) |  |  |
| 16. | **=** |  | assignment | L - R |  |
| **+=** | **-=** |  |
| arithmetic assignments |  |
|  | **\*=** | **/=** |  |  |
|  |  |  |  |
| 17. | **,** |  | sequence | L - R |  |
|  |  |  |  |  |  |

**Vector Components [5.5]**

In addition to array numeric subscript syntax, names of vector components are denoted by a single letter. Components can be swizzled and replicated, e.g.: pos.xx, pos.zy

***{x, y, z, w}*** Use when accessing vectors that represent points or normals

***{r, g, b, a}*** Use when accessing vectors that represent colors

***{s, t, p, q}*** Use when accessing vectors that represent texture coordinates

**Qualifiers**

**Storage Qualifiers [4.3]**

Variable declarations may be preceded by one storage qualifier.

|  |  |
| --- | --- |
| *none* | (Default) local read/write memory, or input parameter |
| **const** | Compile-time constant, or read-only function |
|  | parameter |
| **attribute** | Linkage between a vertex shader and OpenGL ES for |
|  | per-vertex data |
| **uniform** | Value does not change across the primitive being |
|  | processed, uniforms form the linkage between a |
|  | shader, OpenGL ES, and the application |
| **varying** | Linkage between a vertex shader and fragment shader |
|  | for interpolated data |
|  |  |

**Uniform [4.3.4]**

Use to declare global variables whose values are the same across the entire primitive being processed. All uniform variables are read-only. Use uniform qualifiers with any basic data types, to declare a variable whose type is a structure, or an array of any of these. For example:

uniform **vec4** lightPosition;

**Varying [4.3.5]**

The varying qualifier can be used only with the data types float, vec2, vec3, vec4, mat2, mat3, mat4, or arrays of these. Structures cannot be varying. Varying variables are required to have global scope. Declaration is as follows:

varying **vec3** normal;

**Parameter Qualifiers [4.4]**

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

*none* (Default) same as **in**

**in** For function parameters passed into a function

**out** For function parameters passed back out of a function, but not initialized for use when passed in

**inout** For function parameters passed both into and out of a function

**Aggregate Operations and Constructors**

**Matrix Constructor Examples [5.4]**

mat2(float) // init diagonal

mat2(vec2, vec2); // column-major order

mat2(float, float, float, float); // column-major order

**Structure Constructor Example [5.4.3]**

struct light {float *intensity;* vec3 *pos;* };

light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));

**Matrix Components [5.6]**

Access components of a matrix with array subscripting syntax.

For example:

mat4 m; // m represents a matrix

m[1] = vec4(2.0); // sets second column to all 2.0

m[0][0] = 1.0; // sets upper left element to 1.0

m[2][3] = 2.0; // sets 4th element of 3rd column to 2.0

Examples of operations on matrices and vectors:

m = f \* m; // scalar \* matrix component-wise

v = f \* v; // scalar \* vector component-wise

v = v \* v; // vector \* vector component-wise

**Precision and Precision Qualifiers [4.5]**

Any floating point, integer, or sampler declaration can have the type preceded by one of these precision qualifiers:

|  |  |
| --- | --- |
| **highp** | Satisfies minimum requirements for the vertex language. |
|  | Optional in the fragment language. |
| **mediump** | Satisfies minimum requirements for the fragment |
|  | language. Its range and precision is between that |
|  | provided by **lowp** and **highp**. |
| **lowp** | Range and precision can be less than **mediump**, but still |
|  | represents all color values for any color channel. |

For example:

lowp float color;

varying mediump vec2 Coord;

lowp ivec2 foo(lowp mat3);

highp mat4 m;

Ranges & precisions for precision qualifiers (FP=floating point):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **FP Magnitude** |  | **Integer** | |
|  | **FP Range** | **Range** | **FP Precision** | **Range** | |
| **highp** | (−262 , 262) | (2–62 , 262) | Relative 2–16 | (−216 | , 216) |
| **mediump** | (−214 , 214) | (2–14 , 214) | Relative 2–10 | (−210 | , 210) |
| **lowp** | (−2, 2) | (2–8 , 2) | Absolute 2–8 | (−28 | , 28) |

A precision statement establishes a default precision qualifier for subsequent int, float, and sampler declarations, e.g.:

precision **highp** int;

**Invariant Qualifiers Examples [4.6]**

|  |  |
| --- | --- |
| **#pragma STDGL invariant(all)** | Force all output variables to be |
|  | invariant |
| **invariant gl\_Position;** | Qualify a previously declared |
|  | variable |
| **invariant varying mediump** | Qualify as part of a variable |
| **vec3 Color;** | declaration |

**Order of Qualification [4.7]**

When multiple qualifications are present, they must follow a strict order. This order is as follows.

*invariant, storage, precision*

*storage, parameter, precision*

m = m +/- m; // matrix component-wise addition/subtraction m = m \* m; // linear algebraic multiply

m = v \* m; // row vector \* matrix linear algebraic multiply

m = m \* v; // matrix \* column vector linear algebraic multiply f = dot(v, v); // vector dot product v = cross(v, v); // vector cross product

m = matrixCompMult(m, m); // component-wise multiply

**Structure Operations [5.7]**

Select structure fields using the period (.) operator. Other operators include:

* field selector

|  |  |
| --- | --- |
| **== !=** | equality |

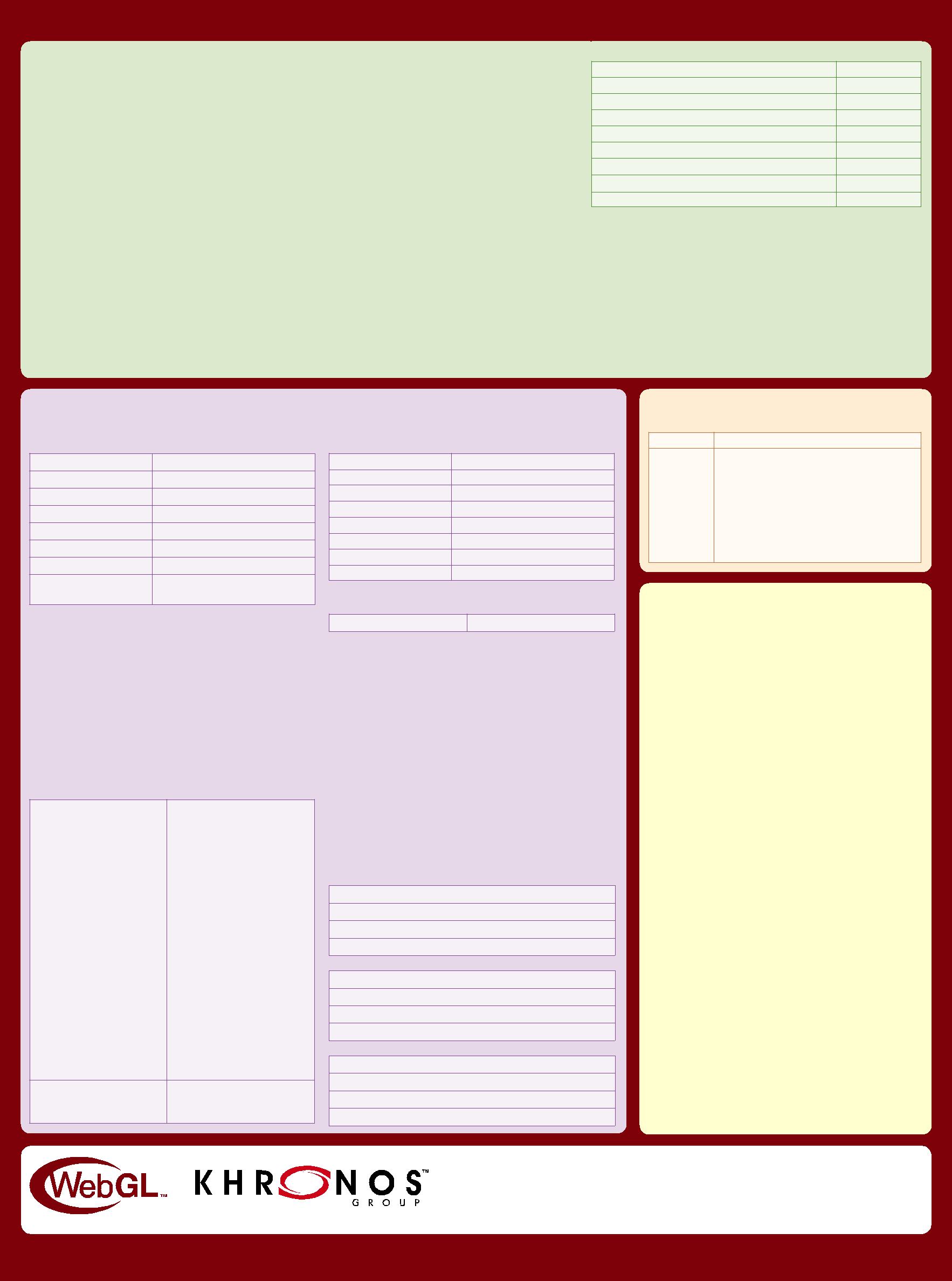
* assignment

**Array Operations [4.1.9]**

Array elements are accessed using the array subscript operator “[ ]”. For example:

diffuseColor += lightIntensity[3] \* NdotL;

**©2011 Khronos Group - Rev. 0618** **www.khronos.org/webgl**



**WebGL 1.0 API Quick Reference Card**

**Built-In Inputs, Outputs, and Constants [7]**

Shader programs use Special Variables to communicate with fixed-function parts of the pipeline. Output Special Variables may be read back after writing. Input Special Variables are read-only. All Special Variables have global scope.

**Vertex Shader Special Variables [7.1]**

**Outputs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** |  | **Description** | **Units or coordinate system** |
| highp vec4 | gl\_Position; | transformed vertex position | clip coordinates |
| mediump float | gl\_PointSize; | transformed point size (point rasterization only) | pixels |

**Fragment Shader Special Variables [7.2]**

Fragment shaders may write to **gl\_FragColor** or to one or more elements of **gl\_FragData[]**, but not both.

The size of the **gl\_FragData** array is given by the built-in constant **gl\_MaxDrawBuffers**.

**Inputs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** |  | **Description** | **Units or coordinate system** |
| mediump vec4 | gl\_FragCoord; | fragment position within frame buffer | window coordinates |
| bool | gl\_FrontFacing; | fragment belongs to a front-facing primitive | Boolean |
| mediump vec2 | gl\_PointCoord; | fragment position within a point (point rasterization only) | 0.0 to 1.0 for each component |
| **Outputs:** |  |  |  |
| **Variable** |  | **Description** | **Units or coordinate system** |
| mediump vec4 | gl\_FragColor; | fragment color | RGBA color |
| mediump vec4 | gl\_FragData[*n*] | fragment color for color attachment *n* | RGBA color |

**Page 4**

**Built-In Constants With Minimum Values [7.4]**

|  |  |
| --- | --- |
| **Built-in Constant** | **Minimum value** |
| const mediump int gl\_MaxVertexAttribs | 8 |
| const mediump int gl\_MaxVertexUniformVectors | 128 |
| const mediump int gl\_MaxVaryingVectors | 8 |
| const mediump int gl\_MaxVertexTextureImageUnits | 0 |
| const mediump int gl\_MaxCombinedTextureImageUnits | 8 |
| const mediump int gl\_MaxTextureImageUnits | 8 |
| const mediump int gl\_MaxFragmentUniformVectors | 16 |
| const mediump int gl\_MaxDrawBuffers | 1 |

**Built-In Uniform State [7.5]**

Specifies depth range in window coordinates. If an implementation does not support highp precision in the fragment language, and state is listed as highp, then that state will only be available as mediump in the fragment language.

struct **gl\_DepthRangeParameters** {

highp float near; // n

highp float far; // f

}; highp float diff; // f - n

uniform **gl\_DepthRangeParameters gl\_DepthRange**;

**Built-In Functions**

**Angle & Trigonometry Functions [8.1]**

Component-wise operation. Parameters specified as *angle* are assumed to be in units of radians. T is float, vec2, vec3, vec4.

**Geometric Functions [8.4]**

These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

**Statements and Structure**

**Iteration and Jumps [6]**

**Function Call** call by value-return

T **radians**(T *degrees*)

T **degrees**(T *radians*)

T **sin**(T *angle*)

T **cos**(T *angle*)

T **tan**(T *angle*)

T **asin**(T *x*)

T **acos**(T *x*)

T **atan**(T *y*, T *x*)

degrees to radians

radians to degrees

sine

cosine

tangent

arc sine

arc cosine

arc tangent

|  |  |
| --- | --- |
| float **length**(T *x*) | length of vector |
| float **distance**(T *p0*, T *p1*) | distance between points |
| float **dot**(T *x*, T *y*) | dot product |
| vec3 **cross**(vec3 *x*, vec3 *y*) | cross product |
| T **normalize**(T *x*) | normalize vector to length 1 |
| T **faceforward**(T *N*, T *I*, T *Nref*) returns *N* if **dot**(*Nref*, *I*) < 0, else -*N* | |
| T **reflect**(T *I*, T *N*) | reflection direction *I* - 2 \* **dot**(*N*,*I*) \* *N* |
| T **refract**(T *I*, T *N*, float *eta*) | refraction vector |

|  |  |  |
| --- | --- | --- |
| **Iteration** | for (;;) { break, continue } | |
|  | while ( ) { break, continue } | |
|  | do { break, continue } while ( ); | |
| **Selection** | if ( ) { } |  |
|  | if ( ) { } else { } |  |
|  |  | |
| **Jump** | break, continue, return | |
|  | discard | // Fragment shader only |
| **Entry** | void main() |  |

T **atan**(T *y\_over\_x*)

**Matrix Functions [8.5]**

Type mat is any matrix type.

**Sample Program**

A shader pair that applies diffuse and ambient lighting to a

**Exponential Functions [8.2]**

Component-wise operation. T is float, vec2, vec3, vec4.

|  |  |
| --- | --- |
| T **pow**(T *x*, T *y*) | *xy* |
| T **exp**(T *x*) | e*x* |
| T **log**(T *x*) | ln |
| T **exp2**(T *x*) | 2*x* |
| T **log2**(T *x*) | log*2* |
| T **sqrt**(T *x*) | square root |
| T **inversesqrt**(T *x*) | inverse square root |

**Common Functions [8.3]**

Component-wise operation. T is float, vec2, vec3, vec4.

|  |  |  |
| --- | --- | --- |
| T **abs**(T *x*) | absolute value |  |
| T **sign**(T *x*) | returns -1.0, 0.0, or 1.0 |  |
| T **floor**(T *x*) | nearest integer <= *x* |  |
| T **ceil**(T *x*) | nearest integer >= *x* |  |
| T **fract**(T *x*) | *x* -**floor**(*x*) |  |
| T **mod**(T *x*, T *y*) | modulus |  |
| T **mod**(T *x*, float *y*) |  |
|  |  |
| T **min**(T *x*, T *y*) | minimum value |  |
| T **min**(T *x*, float *y*) |  |
|  |  |
| T **max**(T *x*, T *y*) | maximum value |  |
| T **max**(T *x*, float *y*) |  |
|  |  |
| T **clamp**(T *x*, T *minVal*, T *maxVal*) |  |  |
| T **clamp**(T *x*, float *minVal*, | **min**(**max**(*x*, **minVal**), **maxVal**) |  |
| float *maxVal*) |  |  |
| T **mix**(T *x*, T *y*, T *a*) | linear blend of *x* and *y* |  |
| T **mix**(T *x*, T *y*, float *a*) |  |
|  |  |
| T **step**(T *edge*, T *x*) | 0.0 if *x* < *edge*, else 1.0 |  |
| T **step**(float *edge*, T *x*) |  |
|  |  |

T **smoothstep**(T *edge0*, T *edge1*, T *x*)

T **smoothstep**(float *edge0*, clip and smooth float *edge1*, T *x*)

mat **matrixCompMult**(mat *x*, mat *y*) multiply *x* by *y* component-wise

**Vector Relational Functions [8.6]**

Compare *x* and *y* component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is bvec*n*; vec is vec*n*; ivec is ivec*n* (where *n* is 2, 3, or 4). T is the union of vec and ivec.

|  |  |
| --- | --- |
| bvec **lessThan**(T *x*, T *y*) | x < y |
| bvec **lessThanEqual**(T *x*, T *y*) | x <= y |
| bvec **greaterThan**(T *x*, T *y*) | x > y |
| bvec **greaterThanEqual**(T *x*, T *y*) | x >= y |
| bvec **equal**(T *x*, T *y*) | x == y |
| bvec **equal**(bvec *x*, bvec *y*) |  |
| bvec **notEqual**(T *x*, T *y*) | x!= y |
| bvec **notEqual**(bvec *x*, bvec *y*) |  |
| bool **any**(bvec *x*) | true if any component of *x* is true |
| bool **all**(bvec *x*) | true if all components of *x* are true |
| bvec **not**(bvec *x*) | logical complement of *x* |

**Texture Lookup Functions [8.7]**

Available only in vertex shaders.

vec4 **texture2DLod**(sampler2D *sampler*, vec2 *coord,* float *lod*)

vec4 **texture2DProjLod**(sampler2D *sampler*, vec3 *coord,* float *lod*)

vec4 **texture2DProjLod**(sampler2D *sampler*, vec4 *coord,* float *lod*)

vec4 **textureCubeLod**(samplerCube *sampler*, vec3 *coord*, float *lod*)

Available only in fragment shaders.

vec4 **texture2D**(sampler2D *sampler*, vec2 *coord*, float *bias*)

vec4 **texture2DProj**(sampler2D *sampler*, vec3 *coord*, float *bias*)

vec4 **texture2DProj**(sampler2D *sampler*, vec4 *coord*, float *bias*)

vec4 **textureCube**(samplerCube *sampler*, vec3 *coord*, float *bias*)

Available in vertex and fragment shaders. vec4 **texture2D**(sampler2D *sampler*, vec2 *coord*)

vec4 **texture2DProj**(sampler2D *sampler*, vec3 *coord*)

vec4 **texture2DProj**(sampler2D *sampler*, vec4 *coord*)

vec4 **textureCube**(samplerCube *sampler*, vec3 *coord*)

textured object.

**Vertex Shader**

|  |  |  |  |
| --- | --- | --- | --- |
| uniform | mat4 | mvp\_matrix; | // model-view-projection matrix |
| uniform | mat3 | normal\_matrix; | // normal matrix |
| uniform | vec3 | ec\_light\_dir; | // light direction in eye coords |
| attribute | vec4 | a\_vertex; | // vertex position |
| attribute | vec3 | a\_normal; | // vertex normal |
| attribute | vec2 | a\_texcoord; | // texture coordinates |
| varying | float | v\_diffuse; |  |
| varying | vec2 | v\_texcoord; |  |

void main(void)

{

// put vertex normal into eye coords

vec3 ec\_normal = normalize(normal\_matrix \* a\_normal); // emit diffuse scale factor, texcoord, and position

v\_diffuse = max(dot(ec\_light\_dir, ec\_normal), 0.0);

v\_texcoord = a\_texcoord;

gl\_Position = mvp\_matrix \* a\_vertex;

}

**Fragment Shader**

|  |  |  |  |
| --- | --- | --- | --- |
| precision | mediump | | float; |
| uniform | sampler2D | | t\_reflectance; |
| uniform | vec4 |  | i\_ambient; |
| varying | float |  | v\_diffuse; |
| varying | vec2 |  | v\_texcoord; |
| void main (void) | | |  |
| { |  |  |  |
| vec4 | color | = texture2D(t\_reflectance, v\_texcoord); | |
| gl\_FragColor | | = color \* (vec4(v\_diffuse) + i\_ambient); | |
| } |  |  |  |

WebGL and OpenGL ES are registered trademarks of Khronos Group.

The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of

parallel computing, graphics and dynamic media on a wide variety of platforms and devices.

See www.khronos.org to learn about the Khronos Group. See www.khronos.org/webgl to learn about WebGL.

See www.khronos.org/opengles to learn about OpenGL ES.

**©2011 Khronos Group - Rev. 0618** **Reference card production by Miller *&* Mattson www.millermattson.com** **www.khronos.org/webgl**