# Appendix B
# WebGL Reference

## WebGL API Reference

This appendix provides an overview of the WebGL JavaScript API with brief descriptions of all functions, their parameters, and return values. For further details on the WebGL and the OpenGL ES 2 API, refer directly to the WebGL specification, available at `www.khronos.org/registry/webgl/specs/1.0/`, and the OpenGL ES 2 specification, available at `www.khronos.org/registry/gles/`. Listing B.1 demonstrates how to create a `canvas` element and get a WebGL context.

**Listing B.1 Creating a WebGL Context**

```
// create a canvas element
var canvas = document.createElement("canvas");

// add the canvas to the document
document.body.appendChild(canvas);

// set a few parameters
var parms = {
    alpha : true,
    stencil : false,
    antialias : true,
};

// create webgl context
// note: use "experimental-webgl" until "webgl" is supported
var gl = canvas.getContext("webgl", parms);
```

Table B-1 lists the available parameters. Note that browsers are required to support only the `alpha`, `premultipliedAlpha`, and `preserveDrawingBuffer` parameters.

**Table B-1 WebGL parameters**

| Parameter | Default | Description |
|---|---|---|
| `alpha` | `true` | If `true`, enables the alpha channel of the drawing buffer. |
| `depth` | `true` | If supported and `true`, enables the depth buffer. |
| `stencil` | `false` | If supported and `true`, enables the stencil buffer. |
| `antialias` | `false` | If supported and `true`, enables antialiasing using an implementation-specific technique. |
| `premultipliedAlpha` | `true` | If `true`, enables premultiplied alpha in the drawing buffer. Ignored if alpha is `false`. |
| `preserveDrawingBuffer` | `false` | If supported and true, the drawing buffer is preserved until explicitly cleared. |

## Data types

WebGL inherits the data types used in OpenGL ES, including several numeric types that are all represented in JavaScript by the basic `number` type. Other data types are used in this appendix only to illustrate the types of values expected by the functions. Table B-2 lists the data types used in this appendix.

In addition to strings, objects, and numbers, WebGL also uses symbolic constants for many values. They are identified by uppercase properties on the `webgl` context object. This appendix uses the enum data type to refer to such constants.

**Table B-2 Data types**

| Type name | Description |
|---|---|
| `enum` | WebGL enum value, for example, `gl.BUFFER_SIZE` |
| `int` | Signed integer value |
| `uint` | Unsigned integer value |
| `float` | Floating-point value |

**NOTE**

The data types in Table B-2 are a simplification of the ones described in the WebGL specification. You generally don't have to worry about these types, however, as they all translate to the `number` type in JavaScript.

## Typed arrays

WebGL uses the new typed array objects to handle various types of data. See `www.khronos.org/registry/typedarray/specs/latest/` for further details on the Typed Array specification. Table B-3 lists the array types used here.

**Table B-3 Array buffer views**

| Type | Description |
|---|---|
| `Uint8Array` | Unsigned 8-bit integer values |
| `Uint16Array` | Unsigned 16-bit integer values |
| `Uint32Array` | Unsigned 32-bit integer values |
| `Int32Array` | Signed 32-bit integer values |
| `Float32Array` | 32-bit floating-point values |

## Buffers

Table B-4 lists the functions related to creating, deleting, and setting data for buffer objects.

**Table B-4 Buffer functions**

| Function | Description |
|---|---|
| `gl.createBuffer()` | Creates and returns a new `WebGLBuffer` object. |
| `gl.deleteBuffer(`<br>`WebGLBuffer buffer`<br>`)` | Deletes a `WebGLBuffer` object. |
| `gl.bindBuffer(`<br>`enum target,`<br>`WebGLBuffer buffer`<br>`)` | Binds a `WebGLBuffer` object to the specified target.<br>`Valid values for target are gl.ARRAY_BUFFER` and `gl.ELEMENT_ARRAY_BUFFER`. |
| `gl.bufferData(`<br>`enum target,`<br>`Object data,`<br>`enum usage`<br>`)`<br>`gl.bufferData(`<br>`enum target,`<br>`uint size,`<br>`enum usage`<br>`)` | Stores data for the `WebGLBuffer` object bound to the specified target. The second argument can be an `ArrayBuffer` object, an `ArrayBufferView` object, or a `ulong` value indicating the initial size of the data.<br>Valid values for `usage` are `gl.STREAM_DRAW`, `gl.STATIC_DRAW`, and `gl.DYNAMIC_DRAW`. |
| `gl.bufferSubData(`<br>`enum target,`<br>`Object data,`<br>`enum usage`<br>`)` | Updates a subsection of the data store of the buffer object bound to the target. The data argument can be an `ArrayBuffer` object or an `ArrayBufferView` object.<br>Valid values for `usage` are `gl.STREAM_DRAW`, `gl.STATIC_DRAW`, and `gl.DYNAMIC_DRAW`. |
| `gl.getBufferParameter(`<br>`enum target,`<br>`enum pname`<br>`)` | Returns the value of a parameter for the buffer object currently bound to the specified `target`.<br>Valid values for `name` are `gl.BUFFER_SIZE` and `gl.BUFFER_USAGE`. |
| `gl.isBuffer(`<br>`WebGLBuffer buffer`<br>`)` | Returns `true` if `buffer` is a `WebGLObject` and has been bound to a target with `gl.bindBuffer()`. |

## Shaders

Table B-5 lists the functions related to creating, deleting, and compiling shader objects.

**Table B-5 Shader functions**

| Function | Description |
|---|---|
| `gl.createShader(`<br>`  enum type`<br>`)` | Returns a new `WebGLShader` object.<br>Valid values for `type` are `gl.FRAGMENT_SHADER` and `gl.VERTEX_SHADER`. |
| `gl.deleteShader(`<br>`  WebGLShader shader`<br>`)` | Deletes the specified `WebGLShader` object. The delete status is stored in the `gl.DELETE_STATUS` parameter. |
| `gl.shaderSource(`<br>`  WebGLShader shader,`<br>`  string source`<br>`)` | Sets the GLSL source code of the specified `WebGLShader` object. |
| `gl.getShaderSource(`<br>`  WebGLShader shader`<br>`)` | Returns a `string` containing the GLSL source code of the specified `WebGLShader` object. |
| `gl.compileShader(`<br>`  WebGLShader shader`<br>`)` | Compiles a `WebGLShader` object. |
| `gl.getShaderInfoLog(`<br>`  WebGLShader shader`<br>`)` | Returns a `string` containing the information log for the specified `WebGLShader` object, including any compilation errors. |
| `gl.isShader(`<br>`  WebGLShader shader`<br>`)` | Returns `true` if `shader` is a `WebGLShader` object that has not yet been deleted; otherwise returns `false`. |
| `gl.getShaderParameter(`<br>`  WebGLShader shader,`<br>`  enum pname`<br>`)` | Returns the value of a parameter for the specified `WebGLShader` object.<br>Valid values for `pname` are `gl.SHADER_TYPE`, `gl.DELETE_STATUS`, and `gl.COMPILE_STATUS`. |

## Program objects

Table B-6 lists the functions related to creating, deleting, and linking program objects.

**Table B-6 Program object functions**

| Function | Description |
|---|---|
| `gl.createProgram()` | Creates and returns a new `WebGLProgram` program object. |
| `gl.deleteProgram( WebGLProgram program )` | Deletes the specified `WebGLProgram` object. The delete status is stored in the `gl.DELETE_STATUS` parameter. |
| `gl.linkProgram( WebGLProgram program )` | Links the specified `WebGLProgram` object and creates executables for the programmable vertex and fragment processors. The status is stored in the `gl.LINK_STATUS` parameter. |
| `gl.getProgramInfoLog( WebGLProgram program )` | Returns a `string` containing the information log for the specified `WebGLProgram` object, including any linker errors. |
| `gl.validateProgram( WebGLProgram program )` | Validates the specified `WebGLProgram` object and stores the status in the `gl.VALIDATE_STATUS` parameter. |
| `gl.attachShader( WebGLProgram program, WebGLShader shader )` | Attaches a `WebGLShader` object to the specified `WebGLProgram` object. The number of attached shaders is stored in the `gl.ATTACHED_SHADERS` parameter. |
| `gl.detachShader( WebGLProgram program, WebGLShader shader )` | Detaches a `WebGLShader` object from the specified `WebGLProgram` object. |
| `gl.getAttachedShaders( WebGLProgram program )` | Returns an array containing the `WebGLShader` objects currently attached to the specified `WebGLProgram` object. |
| `gl.getActiveAttrib( WebGLProgram program, uint index )` | Returns information about the vertex attribute at the specified `index` on the specified `WebGLProgram` object.<br>The return value is a `WebGLActiveInfo` object with the properties `size`, `type`, and `name`. |

| Function | Description |
|---|---|
| `gl.getActiveUniform(`<br>`  WebGLProgram program,`<br>`  uint index`<br>`)` | Returns information about the uniform variable at the specified `index` on the specified `WebGLProgram` object.<br>The return value is a `WebGLActiveInfo` object with the properties `size`, `type`, and `name`. |
| `gl.useProgram(`<br>`  WebGLProgram program`<br>`)` | Activates the specified `WebGLProgram` object for the current rendering. |
| `gl.isProgram(`<br>`  WebGLProgram program`<br>`)` | Returns `true` if `program` is a `WebGLProgram` object that has not yet been deleted; otherwise returns `false`. |
| `gl.getProgramParameter(`<br>`  WebGLProgram program,`<br>`  enum pname`<br>`)` | Returns the value of a parameter for the specified `WebGLProgram` object.<br>Valid values for `name` are `gl.DELETE_STATUS`, `gl.LINK_STATUS`, `gl.VALIDATE_STATUS`, `gl.ATTACHED_SHADERS`, `gl.ACTIVE_UNIFORMS`, and `gl.ACTIVE_ATTRIBUTES`. |

## Uniform variables

Table B-7 lists the functions related to accessing and setting values for uniform variables.

**Table B-7 Uniform variable functions**

| Function | Description |
|---|---|
| `gl.getUniformLocation(`<br>`  WebGLProgram program,`<br>`  string name`<br>`)` | Returns a `WebGLUniformLocation` object pointing to the location of the uniform with the specified `name` for the specified `WebGLProgram` object. |
| `gl.getUniform(`<br>`  WebGLProgram program,`<br>`  WebGLUniformLocation location`<br>`)` | Returns the value of the uniform variable at the specified `location` for the specified `WebGLProgram` object. The type of the return value depends on the type of the uniform variable. |
| `gl.uniform[1234][fi](`<br>`  WebGLUniformLocation location,`<br>`  ...`<br>`)` | Sets the value of the uniform variable at the specified `location` for the active `WebGLProgram` object.<br>Examples:<br>`gl.uniform1i(location, 17);`<br>`gl.uniform3f(location, 1.5, 2.3, 3.7);` |

*continued*

**Table B-7 continued**

| Function | Description |
|---|---|
| `gl.uniform[1234][fi]v(`<br>`  WebGLUniformLocation location,`<br>`  Array value`<br>`)` | Sets the values of the uniform at the specified `location` for the current `WebGLProgram` object.<br>Example:<br>`gl.uniform3fv(location, new Float32Array([`<br>`  0.5, -2.0, 5.5,`<br>`  6.2, 1.0, -2.5`<br>`]);` |
| `gl.uniformMatrix[234]fv(`<br>`  WebGLUniformLocation location,`<br>`  boolean transpose,`<br>`  Float32Array value`<br>`)` | Sets the value of the matrix uniform at the specified `location` for the current `WebGLProgram` object. The `transpose` parameter must be set to `false`. If necessary, you must transpose the matrix manually before loading.<br>Example:<br>`gl.uniformMatrix3fv(location, new Float32Array([`<br>`  1.0, 0.0, 0.0,`<br>`  0.0, 1.0, 0.0,`<br>`  0.0, 0.0, 1.0`<br>`]);` |

## Vertex attributes

Table B-8 lists the functions related to enabling and setting values for vertex attributes.

**Table B-8 Vertex attribute functions**

| Function | Description |
|---|---|
| `gl.enableVertexAttribArray(`<br>`  uint index`<br>`)` | Enables the vertex attribute at the specified `index`. |
| `gl.disableVertexAttribArray(`<br>`  uint index`<br>`)` | Disables the vertex attribute at the specified `index`. |

| Function | Description |
|---|---|
| `gl.getAttribLocation(`<br>`  WebGLProgram program,`<br>`  string name`<br>`)` | Returns the location of the vertex attribute with the specified `name` for the specified program. |
| `gl.bindAttribLocation(`<br>`  WebGLProgram program,`<br>`  uint index,`<br>`  string name`<br>`)` | Binds the vertex attribute with the specified `name` to the specified `index` on the specified program. |
| `gl.getVertexAttrib(`<br>`  uint index,`<br>`  enum pname`<br>`)` | Returns information about the vertex attribute at the specified `index`. The parameter, specified by `pname`, dictates the type of the return value.<br><br>Valid values for `pname` are `gl.VERTEX_ATTRIB_ARRAY_ENABLED`, `gl.VERTEX_ATTRIB_ARRAY_SIZE`, `gl.VERTEX_ATTRIB_ARRAY_STRIDE`, `gl.VERTEX_ATTRIB_ARRAY_TYPE`, `gl.VERTEX_ATTRIB_ARRAY_NORMALIZED`, `gl.CURRENT_VERTEX_ATTRIB`, and `gl.VERTEX_ATTRIB_ARRAY_BUFFER_BINDING`. |
| `gl.getVertexAttribOffset(`<br>`  uint index,`<br>`  enum pname`<br>`)` | Returns the address of the pointer to the vertex attribute at the specified index. The value of `pname` must be `gl.VERTEX_ATTRIB_ARRAY_POINTER`. |
| `gl.vertexAttrib[1234]f(`<br>`  uint index,`<br>`  ...`<br>`)` | Sets a constant value for the vertex attribute at the specified `index`.<br><br>Example:<br><br>`gl.vertexAttrib3f(index, 2.3, 5.4, 1.5);` |
| `gl.vertexAttrib[1234]fv(`<br>`  uint index,`<br>`  Float32Array values`<br>`)` | Sets a constant value for the vertex attribute at the specified `index`.<br><br>Example:<br><br>`gl.vertexAttrib3fv(index, [2.3, 5.4, 1.5]);` |

*continued*

**Table B-8 continued**

| Function | Description |
|---|---|
| `gl.vertexAttribPointer(`<br>`  uint index,`<br>`  int size,`<br>`  enum type,`<br>`  boolean normalized,`<br>`  int stride,`<br>`  int offset`<br>`)` | Assigns the currently bound `WebGLBuffer` object to the vertex attribute at the specified `index`.<br>The `size` parameter specifies the dimension of the elements in the data, for example, 3 for `vec3` values. Must be 1, 2, 3, or 4.<br>The `type` parameter specifies the data type of the data and must be `BYTE`, `UNSIGNED_BYTE`, `SHORT`, `UNSIGNED_SHORT`, `INT`, `UNSIGNED_INT`, or `FLOAT`.<br>The `normalized` parameter specifies whether the values should be normalized to [-1, 1].<br>The `stride` parameter specifies the number of bytes from the start of one vertex to the start of the next. If the data is tightly packed, a value of 0 can be used to autocalculate the stride.<br>The `offset` parameter specifies the first element. |

## Drawing

Table B-9 lists the functions related to drawing geometry on the `canvas` element.

**Table B-9 Drawing functions**

| Function | Description |
|---|---|
| `gl.viewport(`<br>`  int x,`<br>`  int y,`<br>`  int width,`<br>`  int height`<br>`)` | Sets the viewport to a rectangle with its upper-left corner at (`x`, `y`) and dimensions `width` x `height`.<br>The viewport specifies the area where content is rendered. |
| `gl.drawArrays(`<br>`  enum mode,`<br>`  int first,`<br>`  int count`<br>`)` | Renders primitives from array data in the currently bound buffers.<br>The value of `count` specifies the number of elements to render. The value of `first` specifies the first element to render.<br>Valid values for `mode` are `gl.POINTS`, `gl.LINES`, `gl.LINE_LOOP`, `gl.LINE_STRIP`, `gl.TRIANGLES`, `gl.TRIANGLE_STRIP`, and `gl.TRIANGLE_FAN`. |

| Function | Description |
|---|---|
| `gl.drawElements(`<br>`  enum mode,`<br>`  int count,`<br>`  enum type,`<br>`  int offset`<br>`)` | Draws indexed primitives from the currently bound buffers.<br>The value of `count` specifies the number of elements to render. The value of `offset` specifies the first element to render.<br>See the `gl.drawArrays()` description for valid values for `mode`. |
| `gl.flush()` | Causes any buffered WebGL commands to execute immediately. |
| `gl.finish()` | Does not return until all WebGL commands have executed and finished. |

## Textures

Table B-10 lists the functions related to creating, deleting, and loading textures.

**Table B-10 Texture functions**

| Function | Description |
|---|---|
| `gl.createTexture()` | Returns a new `WebGLTexture` object. |
| `gl.deleteTexture(`<br>`  WebGLTexture texture`<br>`)` | Deletes the specified `WebGLTexture` object. |
| `gl.bindTexture(`<br>`  enum target,`<br>`  WebGLTexture texture`<br>`)` | Binds the specified `WebGLTexture` object to the specified `target`. Valid values for `target` are `gl.TEXTURE_2D` and `gl.TEXTURE_CUBE_MAP`. |
| `gl.activeTexture(`<br>`  enum texture`<br>`)` | Activates the specified texture unit. Valid values for `texture` are `gl.TEXTURE0` to `gl.TEXTUREn` where `n` = `gl.MAX_COMBINED_TEXTURE_IMAGE_UNITS` |
| `gl.generateMipmap(`<br>`  enum target`<br>`)` | Generates mipmaps for the texture currently bound to the specified `target`. Valid values for `target` are `gl.TEXTURE_2D` and `gl.TEXTURE_CUBE_MAP`. |

*continued*

**Table B-10 continued**

| Function | Description |
|---|---|
| `gl.texImage2D(`<br>`  enum target,`<br>`  int level,`<br>`  enum internalformat,`<br>`  int width,`<br>`  int height,`<br>`  int border,`<br>`  enum format,`<br>`  enum type,`<br>`  ArrayBufferView pixels`<br>`)` | Loads pixel data from the `pixels` array into the texture bound to the specified target. Valid values for `target` are `gl.TEXTURE_2D`, `gl.TEXTURE_CUBE_MAP_POSITIVE_X`, `gl.TEXTURE_CUBE_MAP_NEGATIVE_X`, `gl.TEXTURE_CUBE_MAP_POSITIVE_Y`, `gl.TEXTURE_CUBE_MAP_NEGATIVE_Y`, `gl.TEXTURE_CUBE_MAP_POSITIVE_Z`, and `gl.TEXTURE_CUBE_MAP_NEGATIVE_Z`.<br><br>Valid values for `format` and `internalformat` are `gl.ALPHA`, `gl.RGB`, `gl.RGBA`, `gl.LUMINANCE`, and `gl.LUMINANCE_ALPHA`.<br><br>The `type` parameter indicates the type of data in `pixels`. Valid values for `type` and their corresponding `ArrayBufferView` types are `gl.UNSIGNED_BYTE (UInt8Array)`, `gl.UNSIGNED_SHORT_4_4_4_4 (UInt16Array)`, `gl.UNSIGNED_SHORT_5_5_5_1 (UInt16Array)`, and `gl.UNSIGNED_SHORT_5_6_5 (UInt16Array)`.<br><br>The value of `border` must be 0. The value of `level` indicates the mipmap level where 0 is the base image. |
| `gl.texImage2D(`<br>`  enum target,`<br>`  int level,`<br>`  enum internalformat,`<br>`  enum format,`<br>`  enum type,`<br>`  Object pixels`<br>`)` | Loads pixel data from the `pixels` object into the texture bound to the specified target.<br><br>The `pixels` object can be an `img` element, a `canvas` element, a `video` element, or an `ImageData` object created, for example, by the `ctx.getImageData()` method of a 2d canvas context.<br><br>See the previous `gl.texImage2D()` description for information on the other parameters. |
| `gl.texSubImage2D(`<br>`  enum target,`<br>`  int level,`<br>`  int xoffset,`<br>`  int yoffset,`<br>`  int width,`<br>`  int height,`<br>`  enum format,`<br>`  enum type,`<br>`  ArrayBufferView pixels`<br>`)` | Loads pixel data into a subregion of the texture bound to the specified `target`.<br><br>The subregion has the dimensions `width x height`. The position (`xoffset`, `yoffset`) specifies the upper-left corner of the subregion.<br><br>See the `gl.texImage2D()` description for information on the other parameters. |

| Function | Description |
|---|---|
| `gl.texSubImage2D(`<br>`  enum target,`<br>`  int level,`<br>`  int xoffset,`<br>`  int yoffset,`<br>`  enum format,`<br>`  enum type,`<br>`  Object pixels`<br>`)` | Loads pixel data into a subregion of the texture bound to the specified `target`.<br>The position (`xoffset`, `yoffset`) specifies the upper-left corner of the subregion.<br>The `pixels` object can be an `img` element, a `canvas` element, a `video` element, or an `ImageData` object created, for example, by the `ctx.getImageData()` method of a 2d canvas context.<br>See `gl.texImage2D()` description for information on the other parameters. |
| `gl.copyTexImage2D(`<br>`  enum target,`<br>`  int level,`<br>`  enum internalformat,`<br>`  int x,`<br>`  int y,`<br>`  int width,`<br>`  int height,`<br>`  int border`<br>`)` | Copies image data from the frame buffer into the texture bound to the specified `target`.<br>See `gl.texImage2D()` description for information on the other parameters. |
| `gl.copyTexSubImage2D(`<br>`  enum target,`<br>`  int level,`<br>`  int xoffset,`<br>`  int yoffset,`<br>`  int x,`<br>`  int y,`<br>`  int width,`<br>`  int height`<br>`)` | Copies image data from the frame buffer into a subregion of the texture bound to the specified `target`.<br>See the `gl.texSubImage2D()` description for information on the other parameters. |
| `gl.isTexture(`<br>`  WebGLTexture texture`<br>`)` | Returns `true` if texture is a `WebGLTexture` that has been bound to a target with `gl.bindTexture()`; otherwise returns `false`. |

*continued*

**Table B-10 continued**

| Function | Description |
|---|---|
| `gl.texParameterf(`<br>`  enum target,`<br>`  enum pname,`<br>`  float param`<br>`)`<br>`gl.texParameteri(`<br>`  enum target,`<br>`  enum pname,`<br>`  int param`<br>`)` | Sets a texture parameter for the texture currently bound to the specified `target`.<br>Valid values for `target` are `gl.TEXTURE_2D` and `gl.TEXTURE_CUBE_MAP`.<br>If `pname` is `gl.TEXTURE_MIN_FILTER`, `param` must be `gl.NEAREST`, `gl.LINEAR`, `gl.NEAREST_MIPMAP_NEAREST`, `gl.LINEAR_MIPMAP_NEAREST`, `gl.NEAREST_MIPMAP_LINEAR`, or `gl.LINEAR_MIPMAP_LINEAR`.<br>If `pname` is `gl.TEXTURE_MIN_FILTER`, `param` must be `gl.NEAREST` or `gl.LINEAR`.<br>If `pname` is `gl.TEXTURE_WRAP_S` or `gl.TEXTURE_WRAP_T`, `param` must be `gl.REPEAT`, `gl.CLAMP_TO_EDGE`, or `gl.MIRRORED_REPEAT`. |
| `gl.getTexParameter(`<br>`  enum target,`<br>`  enum pname`<br>`)` | Returns the value of a texture parameter for the texture currently bound to `target`.<br>See `gl.texParameter[fi]()` for valid `pname` values. |

## Blending

Table B-11 lists the functions related to blending equations and functions.

**Table B-11 Blending functions**

| Function | Description |
|---|---|
| `gl.blendEquation(`<br>`  enum mode`<br>`)` | Sets the blending equation.<br>Valid values for `mode` are `gl.FUNC_ADD`, `gl.FUNC_SUBTRACT`, and `gl.FUNC_REVERSE_SUBTRACT`. |
| `gl.blendEquationSeparate(`<br>`  enum modeRGB,`<br>`  enum modeAlpha`<br>`)` | Sets the blending equation separately for RGB and alpha.<br>See description for `gl.blendEquation()` for valid values for `mode`. |

| Function | Description |
|---|---|
| `gl.blendFunc(`<br>`  enum sfactor,`<br>`  enum dfactor`<br>`)` | Sets the source and destination blending factors.<br>Valid values for `sfactor` and `dfactor` are `gl.ZERO`, `gl.ONE`, `gl.SRC_COLOR`, `gl.ONE_MINUS_SRC_COLOR`, `gl.DST_COLOR`, `gl.ONE_MINUS_DST_COLOR`, `gl.SRC_ALPHA`, `gl.ONE_MINUS_SRC_ALPHA`, `gl.DST_ALPHA`, `gl.ONE_MINUS_DST_ALPHA`, `gl.CONSTANT_COLOR`, `gl.ONE_MINUS_CONSTANT_COLOR`, `gl.CONSTANT_ALPHA`, and `gl.ONE_MINUS_CONSTANT_ALPHA`.<br>Additionally, the value of `sfactor` can be `gl.SRC_ALPHA_SATURATE`.<br>Constant alpha and constant color cannot be used at the same time. |
| `gl.blendFuncSeparate(`<br>`  enum srcRGB,`<br>`  enum dstRGB,`<br>`  enum srcAlpha,`<br>`  enum dstAlpha`<br>`)` | Sets the blending factors separately for RGB and alpha.<br>See `gl.blendFunc()` for valid values for source and destination factors. |
| `gl.blendColor(`<br>`  float red,`<br>`  float green,`<br>`  float blue,`<br>`  float alpha`<br>`)` | Sets the constant blending color. |

## Stencil buffer

Table B-12 lists the functions related to setting functions and operations for the stencil buffer.

**Table B-12 Stencil buffer functions**

| Function | Description |
|---|---|
| `gl.clearStencil(`<br>`  int s`<br>`)` | Sets the stencil index used when the stencil buffer is cleared. |
| `gl.stencilFunc(`<br>`  enum func,`<br>`  int ref,`<br>`  int mask`<br>`)` | Sets the functions and reference value used for stencil testing.<br>Valid values for `func` are `gl.NEVER`, `gl.LESS`, `gl.EQUAL`, `gl.LEQUAL`, `gl.GREATER`, `gl.NOTEQUAL`, `gl.GEQUAL`, and `gl.ALWAYS`. |
| `gl.stencilFuncSeparate(`<br>`  enum face,`<br>`  enum func,`<br>`  int ref,`<br>`  int mask`<br>`)` | Sets the stencil functions and reference value separately for front- and back-facing polygons.<br>Valid values for `face` are `gl.FRONT`, `gl.BACK`, and `gl.FRONT_AND_BACK`. |
| `gl.stencilMask(`<br>`  uint mask`<br>`)` | Sets the mask that controls writing of individual bits to the stencil buffer. |
| `gl.stencilMaskSeparate(`<br>`  enum face,`<br>`  uint mask`<br>`)` | Sets the stencil mask separately for front- and back-facing polygons.<br>See `gl.stencilFuncSeparate()` for valid `face` values. |
| `gl.stencilOp(`<br>`  enum fail,`<br>`  enum zfail,`<br>`  enum zpass`<br>`)` | Sets the operations used in the stencil test.<br>The `fail` value is the operation used when the stencil test fails. The `zfail` value is the operation used when the stencil test passes but the depth test fails. The `zpass` value is the operation used when both tests pass.<br>Valid values for all parameters are `gl.ZERO`, `gl.KEEP`, `gl.REPLACE`, `gl.INCR`, `gl.DECR`, `gl.INVERT`, `gl.INCR_WRAP`, and `gl.DECR_WRAP`. |
| `gl.stencilOpSeparate(`<br>`  enum face,`<br>`  enum fail,`<br>`  enum zfail,`<br>`  enum zpass`<br>`)` | Sets the stencil test operations separately for front- and back-facing polygons.<br>See `gl.stencilOp()` for valid operation values. |

## Depth buffer

Table B-13 lists the functions related to setting depth buffer values.

**Table B-13 Depth buffer functions**

| Function | Description |
|---|---|
| `gl.depthFunc(`<br>`  enum func`<br>`)` | Sets the depth buffer function.<br>Valid values for `func` are `gl.NEVER`, `gl.LESS`, `gl.EQUAL`, `gl.LEQUAL`, `gl.GREATER`, `gl.NOTEQUAL`, `gl.GEQUAL`, and `gl.ALWAYS`. |
| `gl.depthMask(`<br>`  boolean flag`<br>`)` | Enables or disables writing to the depth buffer. |
| `gl.depthRange(`<br>`  float zNear,`<br>`  float zFar`<br>`)` | Sets the range of the depth buffer.<br>The value of `zNear` must be less than the value of `zFar`. |
| `gl.clearDepth(`<br>`  float depth`<br>`)` | Sets the depth value used to clear the depth buffer. |
| `gl.polygonOffset(`<br>`  float factor,`<br>`  float units`<br>`)` | Sets the scale factor and the offset units used to calculate depth values. |

## Render buffers

Table B-14 lists the functions related to creating, deleting, and using render buffers.

**Table B-14 Render buffer functions**

| Function | Description |
|---|---|
| `gl.createRenderbuffer()` | Returns a new `WebGLRenderBuffer` object. |
| `gl.deleteRenderbuffer(`<br>`  WebGLRenderbuffer renderbuffer`<br>`)` | Deletes the specified `WebGLRenderBuffer` object. |

*continued*

**Table B-14 continued**

| Function | Description |
|---|---|
| `gl.bindRenderbuffer(`<br>`  enum target,`<br>`  WebGLRenderbuffer renderbuffer`<br>`)` | Binds a `WebGLRenderBuffer` object to the specified `target`.<br>The value of `target` must be `gl.RENDERBUFFER`. |
| `gl.renderbufferStorage(`<br>`  enum target,`<br>`  enum internalformat,`<br>`  int width,`<br>`  int height`<br>`)` | Initializes the data store for the currently bound render buffer.<br>The `width` and `height` parameters specify the dimensions of the render buffer.<br>Valid values for `internalformat` are `gl.RGBA4`, `gl.RGB565`, `gl.RGB5_A1`, `gl.DEPTH_COMPONENT16`, and `gl.STENCIL_INDEX8`. |
| `gl.framebufferRenderbuffer(`<br>`  enum target,`<br>`  enum attachment,`<br>`  enum renderbuffertarget,`<br>`  WebGLRenderbuffer renderbuffer`<br>`)` | Attaches the specified `WebGLRenderbuffer` object to the frame buffer currently bound to `target`.<br>The value of `target` must be `gl.FRAMEBUFFER`.<br>The value of `renderbuffertarget` must be `gl.RENDERBUFFER`.<br>Valid values for `attachment` are `gl.COLOR_ATTACHMENT0`, `gl.DEPTH_ATTACHMENT`, `gl.STENCIL_ATTACHMENT`, and `gl.DEPTH_STENCIL_ATTACHMENT`. |
| `gl.isRenderbuffer(`<br>`  WebGLRenderbuffer renderbuffer`<br>`)` | Returns `true` if `renderbuffer` is a `WebGLRenderBuffer` object that has been bound with `gl.bindRenderBuffer()`; otherwise returns `false`. |
| `gl.getRenderbufferParameter(`<br>`  enum target,`<br>`  enum pname`<br>`)` | Returns the value of a parameter for the currently bound render buffer.<br>Valid values for `pname` are `gl.RENDERBUFFER_WIDTH`, `gl.RENDERBUFFER_HEIGHT`, `gl.RENDERBUFFER_INTERNAL_FORMAT`, `gl.RENDERBUFFER_RED_SIZE`, `gl.RENDERBUFFER_GREEN_SIZE`, `gl.RENDERBUFFER_BLUE_SIZE`, `gl.RENDERBUFFER_ALPHA_SIZE`, `gl.RENDERBUFFER_DEPTH_SIZE`, and `gl.RENDERBUFFER_STENCIL_SIZE`. |

## Frame buffers

Table B-15 lists the functions related to creating, deleting, and using frame buffers.

**Table B-15 Frame buffer functions**

| Function | Description |
|---|---|
| `gl.createFramebuffer()` | Returns a new `WebGLFramebuffer` object. |
| `gl.deleteFramebuffer(`<br>`WebGLFramebuffer framebuffer`<br>`)` | Deletes the specified `WebGLFramebuffer` object. |
| `gl.bindFramebuffer(`<br>`enum target,`<br>`WebGLFramebuffer framebuffer`<br>`)` | Binds the specified `WebGLFramebuffer` object to the specified `target`. The value of `target` must be `gl.FRAMEBUFFER`. |
| `gl.checkFramebufferStatus(`<br>`enum target`<br>`)` | Returns the status of the currently bound frame buffer.<br><br>Return value is one of `gl.FRAMEBUFFER_COMPLETE`, `gl.FRAMEBUFFER_INCOMPLETE_ATTACHMENT`, `gl.FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT`, `gl.FRAMEBUFFER_INCOMPLETE_DIMENSIONS`, or `gl.FRAMEBUFFER_UNSUPPORTED`. |
| `gl.isFramebuffer(`<br>`WebGLFramebuffer framebuffer`<br>`)` | Returns `true` if `framebuffer` is a `WebGLFramebuffer` object that has been bound with `gl.bindFrameBuffer()`. |
| `gl.framebufferTexture2D(`<br>`enum target,`<br>`enum attachment,`<br>`enum textarget,`<br>`WebGLTexture texture,`<br>`int level`<br>`)` | Attaches the specified `WebGLTexture` object to the currently bound frame buffer.<br><br>See `gl.framebufferRenderbuffer()` for valid attachment values.<br><br>Valid values for `textarget` are `gl.TEXTURE_2D`, `gl.TEXTURE_CUBE_MAP_POSITIVE_X`, `gl.TEXTURE_CUBE_MAP_NEGATIVE_X`, `gl.TEXTURE_CUBE_MAP_POSITIVE_Y`, `gl.TEXTURE_CUBE_MAP_NEGATIVE_Y`, `gl.TEXTURE_CUBE_MAP_POSITIVE_Z`, and `gl.TEXTURE_CUBE_MAP_NEGATIVE_Z`.<br><br>The value of `level` must be 0. |

*continued*

**Table B-15 continued**

| Function | Description |
|---|---|
| `gl.getFramebufferAttachmentParameter(`<br>`  enum target,`<br>`  enum attachment,`<br>`  enum pname`<br>`)` | Returns the value of an attachment parameter for the currently bound frame buffer.<br>Valid values for `attachment` are `gl.COLOR_ATTACHMENT0`, `gl.DEPTH_ATTACHMENT`, and `gl.STENCIL_ATTACHMENT`.<br>Valid values for `pname` are `gl.FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE`, `gl.FRAMEBUFFER_ATTACHMENT_OBJECT_NAME`, `gl.FRAMEBUFFER_ATTACHMENT_TEXTURE_LEVEL`, and `gl.FRAMEBUFFER_ATTACHMENT_TEXTURE_CUBE_MAP_FACE`. |
| `gl.colorMask(`<br>`  boolean red,`<br>`  boolean green,`<br>`  boolean blue,`<br>`  boolean alpha`<br>`)` | Enables or disables writing the red, green, blue, and alpha components of the frame buffer. |
| `gl.readPixels(`<br>`  int x,`<br>`  int y,`<br>`  int width,`<br>`  int height,`<br>`  enum format,`<br>`  enum type,`<br>`  ArrayBufferView pixels`<br>`)` | Reads pixel data from the frame buffer.<br>The `x`, `y`, `width`, and `height` values specify the rectangular region that is read.<br>The value of `format` must be `gl.RGBA`. The value of `type` must be `gl.UNSIGNED_BYTE`.<br>The pixel data is loaded into the `pixels` array, which must be an `UInt8Array` to match the `gl.UNSIGNED_BYTE` type. |
| `gl.pixelStorei(`<br>`  enum pname,`<br>`  any param`<br>`)` | Sets pixel storage modes. The data type of `param` depends on the parameter.<br>If pname is `gl.PACK_ALIGNMENT` or `gl.UNPACK_ALIGNMENT`, `param` must be an `int` value.<br>If pname is `gl.UNPACK_FLIP_Y_WEBGL` or `gl.UNPACK_PREMULTIPLY_ALPHA_WEBGL`, `param` must be a `boolean` value.<br>If `pname` is `gl.UNPACK_COLORSPACE_CONVERSION_WEBGL`, `param` must be `gl.BROWSER_DEFAULT_WEBGL` or `gl.NONE`. |

## Other functions

Table B-16 lists functions that do not fit into any other sections.

**Table B-16 Other functions**

| Function | Description |
|---|---|
| `gl.enable(` `enum cap` `)` | Enables a capability. Valid values for `cap` are `gl.CULL_FACE`, `gl.BLEND`, `gl.DITHER`, `gl.STENCIL_TEST`, `gl.DEPTH_TEST`, `gl.SCISSOR_TEST`, `gl.POLYGON_OFFSET_FILL`, `gl.SAMPLE_ALPHA_TO_COVERAGE`, and `gl.SAMPLE_COVERAGE`. |
| `gl.disable(` `enum cap` `)` | Disables a capability. See `gl.enable()` description for valid values for `cap`. |
| `gl.isEnabled(` `enum cap` `)` | Returns `true` if the specified capability is enabled; otherwise returns `false`. See `gl.enable()` description for valid values for `cap`. |
| `gl.cullFace(` `enum mode` `)` | Sets the face culling mode. Valid values for `mode` are `gl.FRONT`, `gl.BACK`, and `gl.FRONT_AND_BACK`. |
| `gl.frontFace(` `enum mode` `)` | Sets the winding order used for face culling. Valid values for `mode` are `gl.CW` and `gl.CCW`. |
| `gl.clear(` `uint mask` `)` | Clears the color, depth, and stencil buffers. The value of `mask` is a bitmask specifying which buffers to clear. Example: `gl.clear(` `gl.DEPTH_BUFFER_BIT \|\|` `gl.STENCIL_BUFFER_BIT \|\|` `gl.COLOR_BUFFER_BIT` `);` |

continued

**Table B-16 continued**

| Function | Description |
|---|---|
| `gl.clearColor(`<br>`  float red,`<br>`  float green,`<br>`  float blue,`<br>`  float alpha`<br>`)` | Sets the color used to clear the color buffer. |
| `gl.lineWidth(`<br>`  float width`<br>`)` | Sets the width of rendered lines. |
| `gl.scissor(`<br>`  int x,`<br>`  int y,`<br>`  int width,`<br>`  int height`<br>`)` | Sets the scissor box.<br>The `x` and `y` values specify the upper-left corner of the scissor box. The `width` and `height` values specify the dimensions. |
| `gl.sampleCoverage(`<br>`  float value,`<br>`  boolean invert`<br>`)` | Sets multisample coverage parameters.<br>The value of `invert` specifies whether the coverage masks are inverted. |
| `gl.getError()` | Returns an `enum` value indicating the error status of the last executed WebGL command.<br>Possible return values are `gl.INVALID_ENUM`, `gl.INVALID_VALUE`, `gl.INVALID_OPERATION`, `gl.OUT_OF_MEMORY`, `gl.CONTEXT_LOST_WEBGL`, and `gl.INVALID_FRAMEBUFFER_OPERATION`. |
| `gl.hint(`<br>`  enum target,`<br>`  enum mode`<br>`)` | Sets hints for the implementation.<br>The value of target must be `gl.GENERATE_MIPMAP_HINT`.<br>Valid values for `mode` are `gl.DONT_CARE`, `gl.FASTEST`, and `gl.NICEST`. |
| `gl.getSupportedExtensions()` | Returns an array of strings, listing the supported extensions. |

| Function | Description |
|---|---|
| `gl.getExtension(`<br>`  string name`<br>`)` | Returns an object if the extension with the specified `name` is supported; otherwise returns `null`. |
| `gl.getContextAttributes()` | Returns the `WebGLContextAttributes` object specified when creating the `webgl` context. |
| `gl.isContextLost()` | Returns `true` if the `webgl` context is lost and must be re-created, for example, due to power events on mobile devices. |
| `gl.getParameter(`<br>`  enum pname`<br>`)` | Returns the value of the WebGL parameter with the name `pname`. See Table B-17 for valid `pname` values. |

## Parameters

Table B-17 lists the parameters that can be accessed with the `gl.getParameter()` method.

**Table B-17 Parameters**

| Parameter name | Description |
|---|---|
| `gl.ACTIVE_TEXTURE` | An `int` value indicating the active texture unit. See `gl.activeTexture()` in Table B-10. |
| `gl.ALIASED_LINE_WIDTH_RANGE` | A `Float32Array` with two elements: the smallest and largest supported line widths for aliased lines. |
| `gl.ALIASED_POINT_SIZE_RANGE` | A `Float32Array` with two elements: the smallest and largest supported point sizes for aliased points. |
| `gl.ALPHA_BITS` | The number of alpha bitplanes in the current color buffer. |
| `gl.ARRAY_BUFFER_BINDING` | The `WebGLBuffer` object currently bound to the `gl.ARRAY_BUFFER` target. See `gl.bindBuffer()` in Table B-4. |
| `gl.BLEND` | A `boolean` value indicating whether blending is enabled. See `gl.enable()` in Table B-16. |
| `gl.BLEND_COLOR` | A `Float32Array` with four elements: the red, green, blue, and alpha components of the blend color. See `gl.blendColor()` in Table B-11. |
| `gl.BLEND_DST_ALPHA` | The `enum` value of the destination alpha blend function. See `gl.blendFuncSeparate()` in Table B-11. |
| `gl.BLEND_DST_RGB` | The `enum` value of the destination RGB blend function. See `gl.blendFuncSeparate()` in Table B-11. |

*continued*

**Table B-17 continued**

| Function | Description |
|---|---|
| `gl.BLEND_EQUATION_ALPHA` | The `enum` value of the alpha blend equation. See `gl.blendEquationSeparate()` in Table B-11. |
| `gl.BLEND_EQUATION_RGB` | The `enum` value of the RGB blend equation. See `gl.blendEquationSeparate()` in Table B-11. |
| `gl.BLEND_SRC_ALPHA` | The `enum` value of the source alpha blend function. See `gl.blendFuncSeparate()` in Table B-11. |
| `gl.BLEND_SRC_RGB` | The `enum` value of the source RGB blend function. See `gl.blendFuncSeparate()` in Table B-11. |
| `gl.BLUE_BITS` | The number of blue bitplanes in the current color buffer. |
| `gl.COLOR_CLEAR_VALUE` | A `Float32Array` with four elements: the red, green, blue, and alpha components of the color used to clear the color buffer. See `gl.clearColor()` in Table B-16. |
| `gl.COLOR_WRITEMASK` | An array with four `boolean` values indicating whether the writing to the red, green, blue, and alpha components of the color buffer is enabled. See `gl.colorMask()` in Table B-15. |
| `gl.COMPRESSED_TEXTURE_FORMATS` | Always `null` because WebGL does not support any compressed texture formats. |
| `gl.CULL_FACE` | A `boolean` value indicating whether face culling is enabled. See `gl.enable()` in Table B-16. |
| `gl.CULL_FACE_MODE` | The `enum` value of the current face culling mode. See `gl.cullFace()` in Table B-16. |
| `gl.CURRENT_PROGRAM` | The active `WebGLProgram` object. See `gl.useProgram()` in Table B-6. |
| `gl.DEPTH_BITS` | The number of bitplanes in the current depth buffer. |
| `gl.DEPTH_CLEAR_VALUE` | A `float` value indicating the depth value used to clear the depth buffer. See `gl.clearDepth()` in Table B-13. |
| `gl.DEPTH_FUNC` | The `enum` value of the depth comparison function. See `gl.depthFunc()` in Table B-13. |
| `gl.DEPTH_RANGE` | A `Float32Array` with two elements indicating the depth range in the depth buffer. See `gl.depthRange()` in Table B-13. |
| `gl.DEPTH_TEST` | A `boolean` value indicating whether depth testing is enabled. See `gl.enable()` in Table B-16. |
| `gl.DEPTH_WRITEMASK` | A `boolean` value indicating whether writing to the depth buffer is enabled. See `gl.depthMask()` in Table B-13. |
| `gl.DITHER` | A `boolean` value indicating whether fragment dithering is enabled. See `gl.enable()` in Table B-16. |

| Function | Description |
|---|---|
| `gl.ELEMENT_ARRAY_BUFFER_BINDING` | The `WebGLBuffer` object currently bound to the `gl.ELEMENT_ARRAY_BUFFER` target. See `gl.bindBuffer()` in Table B-4. |
| `gl.FRAMEBUFFER_BINDING` | The currently bound `WebGLFramebuffer` object. See `gl.bindFramebuffer()` in Table B-15. |
| `gl.FRONT_FACE` | An `enum` value indicating the triangle winding direction. See `gl.frontFace()` in Table B-16. |
| `gl.GENERATE_MIPMAP_HINT` | The `enum` value of the mipmap generation hint mode. See `gl.hint()` in Table B-16. |
| `gl.GREEN_BITS` | The number of green bitplanes in the current color buffer. |
| `gl.LINE_WIDTH` | A `float` value indicating the current line width. See `gl.lineWidth()` in Table B-16. |
| `gl.MAX_COMBINED_TEXTURE_IMAGE_UNITS` | The maximum supported texture units in the vertex shader and fragment shader combined. The value is at least 8. |
| `gl.MAX_CUBE_MAP_TEXTURE_SIZE` | An estimate of the largest cube map texture size. The value is at least 16. |
| `gl.MAX_FRAGMENT_UNIFORM_VECTORS` | The maximum number of four-element uniform variables in the fragment shader. The value is at least 16. |
| `gl.MAX_RENDERBUFFER_SIZE` | The largest supported width and height for the render buffer. The value is at least 1. |
| `gl.MAX_TEXTURE_IMAGE_UNITS` | The maximum supported texture units in the fragment shader. The value is at least 8. |
| `gl.MAX_TEXTURE_SIZE` | An estimate of the largest texture size. The value is at least 64. |
| `gl.MAX_VARYING_VECTORS` | The maximum number of four-element varying variables in the vertex and fragment shader. The value is at least 8. |
| `gl.MAX_VERTEX_ATTRIBS` | The maximum number of four-element vertex attributes available in the vertex shader. The value is at least 8. |
| `gl.MAX_VERTEX_TEXTURE_IMAGE_UNITS` | The maximum supported texture units in the vertex shader. Can be 0. |
| `gl.MAX_VERTEX_UNIFORM_VECTORS` | The maximum number of four-element uniform variables in the vertex shader. The value is at least 128. |
| `gl.MAX_VIEWPORT_DIMS` | An `Int32Array` with two elements: the maximum width and height of the viewport. |
| `gl.NUM_COMPRESSED_TEXTURE_FORMATS` | Always 0 because WebGL does not support any compressed texture formats. |
| `gl.PACK_ALIGNMENT` | An `int` value indicating the byte alignment used when writing pixel data to memory. See `gl.pixelStorei()` in Table B-15. |

*continued*

**Table B-17 continued**

| Function | Description |
|---|---|
| `gl.POLYGON_OFFSET_FACTOR` | The `float` scaling factor used for polygon offset. See `gl.polygonOffset()` in Table B-13. |
| `gl.POLYGON_OFFSET_FILL` | A `boolean` value indicating whether polygon offset mode is enabled for fill mode. See `gl.enable()` in Table B-16. |
| `gl.POLYGON_OFFSET_UNITS` | The `float` value that is used to create a constant depth offset. See `gl.polygonOffset()` in Table B-13. |
| `gl.RED_BITS` | The number of red bitplanes in the current color buffer. |
| `gl.RENDERBUFFER_BINDING` | The currently bound `WebGLRenderbuffer` object. See `gl.bindRenderbuffer()` in Table B-14. |
| `gl.RENDERER` | A string containing the name of the renderer. |
| `gl.SAMPLE_BUFFERS` | An `int` value indicating the number of sample buffers associated with the current frame buffer. |
| `gl.SAMPLE_COVERAGE_INVERT` | A `boolean` value indicating whether the coverage value should be inverted. See `gl.sampleCoverage()` in Table B-16. |
| `gl.SAMPLE_COVERAGE_VALUE` | A `float` value indicating the current coverage value. See `gl.sampleCoverage()` in Table B-16. |
| `gl.SAMPLES` | An `int` value indicating the coverage mask size of the current frame buffer. |
| `gl.SCISSOR_BOX` | An `Int32Array` with four elements: the `x`, `y`, `width`, and `height` values of the current scissor box. See `gl.scissor()` in Table B-16. |
| `gl.SCISSOR_TEST` | A `boolean` value indicating whether scissor testing is enabled. See `gl.enable()` in Table B-16. |
| `gl.SHADING_LANGUAGE_VERSION` | A `string` containing the version of the shader language used in the implementation. Example: `WebGL GLSL ES 1.0` |
| `gl.STENCIL_BACK_FAIL` | The `enum` value of the operation used for back-facing polygons when the stencil test fails. See `gl.stencilOpSeparate()` in Table B-12. |
| `gl.STENCIL_BACK_FUNC` | The `enum` value of the comparison function used for back-facing polygons. See `gl.stencilFuncSeparate()` in Table B-12. |
| `gl.STENCIL_BACK_PASS_DEPTH_FAIL` | The `enum` value of the operation used for back-facing polygons when the stencil test passes but the depth test fails. See `gl.stencilOpSeparate()` in Table B-12. |

| Function | Description |
|---|---|
| `gl.STENCIL_BACK_PASS_DEPTH_PASS` | The `enum` value of the operation used for back-facing polygons when both the stencil test and depth test pass. See `gl.stencilOpSeparate()`in Table B-12. |
| `gl.STENCIL_BACK_REF` | The reference value used for back-facing polygons. See `gl.stencilFuncSeparate()` in Table B-12. |
| `gl.STENCIL_BACK_VALUE_MASK` | An `int` mask that is used for back-facing polygons to mask the reference value and stencil buffer before comparison. See `gl.stencilFuncSeparate()` in Table B-12. |
| `gl.STENCIL_BACK_WRITEMASK` | An `int` mask that controls writing for back-facing polygons. See `gl.stencilMaskSeparate()` in Table B-12. |
| `gl.STENCIL_BITS` | The number of bitplanes in the stencil buffer. |
| `gl.STENCIL_CLEAR_VALUE` | The `int` index value used to clear the stencil buffer. See `gl.clearStencil()` in Table B-12. |
| `gl.STENCIL_FAIL` | The `enum` value of the operation used for front-facing polygons when the stencil test fails. See `gl.stencilOpSeparate()`in Table B-12. |
| `gl.STENCIL_FUNC` | The enum value of the comparison function used for front-facing polygons. See `gl.stencilFuncSeparate()` in Table B-12. |
| `gl.STENCIL_PASS_DEPTH_FAIL` | The `enum` value of the operation used for front-facing polygons when the stencil test passes but the depth test fails. See `gl.stencilOpSeparate()` in Table B-12. |
| `gl.STENCIL_PASS_DEPTH_PASS` | The `enum` value of the operation used for front-facing polygons when both the stencil test and depth test pass. See `gl.stencilOpSeparate()` in Table B-12. |
| `gl.STENCIL_REF` | The reference value used for front-facing polygons. See `gl.stencilFuncSeparate()` in Table B-12. |
| `gl.STENCIL_TEST` | A `boolean` value indicating whether stencil testing is enabled. See `gl.enable()` in Table B-16. |
| `gl.STENCIL_VALUE_MASK` | An `int` mask used for front-facing polygons to mask the reference value and stencil buffer before comparison. See `gl.stencilFuncSeparate()` in Table B-12. |
| `gl.STENCIL_WRITEMASK` | An `int` mask that controls writing for front-facing polygons. See `gl.stencilMaskSeparate()` in Table B-12. |
| `gl.SUBPIXEL_BITS` | An estimate of the number of subpixel bits. The value is at least 4. |
| `gl.TEXTURE_BINDING_2D` | The `WebGLTexture` object currently bound to the `gl.TEXTURE_2D` target. See `gl.bindTexture()` in Table B-10. |

*continued*

**Table B-17 continued**

| Function | Description |
|---|---|
| `gl.TEXTURE_BINDING_CUBE_MAP` | The `WebGLTexture` object currently bound to the `gl.TEXTURE_CUBE_MAP` target. See `gl.bindTexture()` in Table B-10. |
| `gl.UNPACK_ALIGNMENT` | An `int` value indicating the byte alignment used when reading pixel data from memory. See `gl.pixelStorei()` in Table B-15. |
| `gl.UNPACK_COLORSPACE_CONVERSION_WEBGL` | The `enum` value of the colorspace conversion used when loading image data. Initially set to `gl.BROWSER_DEFAULT_WEBGL`. See `gl.pixelStorei()` in Table B-15. |
| `gl.UNPACK_FLIP_Y_WEBGL` | A `boolean` value indicating whether texture image data is flipped along the vertical axis. See `gl.pixelStorei()` in Table B-15. |
| `gl.UNPACK_PREMULTIPLY_ALPHA_WEBGL` | A `boolean` value indicating whether the alpha channel is multiplied into the RGB channels when loading image data. See `gl.pixelStorei()` in Table B-15. |
| `gl.VENDOR` | A `string` containing the name of the company responsible for the implementation. |
| `gl.VERSION` | A string containing the WebGL version used in the implementation. Example: `WebGL 1.0` |
| `gl.VIEWPORT` | An `Int32Array` with four elements: the `x`, `y`, `width`, and `height` values of the current viewport. See `gl.viewport()` in Table B-9. |