WebGL 规范

最终稿（1.0）2011年二月18日

翻译: 樊虹剑 (c) 2011

原文: http://www.khronos.org/registry/webgl/specs/latest/

译者前言 5
1 介绍 6
2 环境创建和绘图缓存表示 8
2.1 canvas 元素 8
2.2 绘图缓存 9
2.3 WebGL视口 11
示例1: 11
2.4 预乘Alpha, Canvas APIs和texImage2D 12
3 WebGL资源.....14
4 安全.....15
4.1 资源局限 15
4.2 起源局限 15
4.3 支持的GLSL Constructs 16
4.4 防卫 Denial of Service 17
5 DOM界面 19
5.1 类型 19
5.2 WebGLContextAttributes 20
5.2.1 环境生成参数 20
示例2 21
5.3 WebGLObject 22
5.4 WebGLBuffer 22
5.5 WebGLFramebuffer 22
5.6 WebGLProgram 23
5.7 WebGLRenderbuffer 23

5.8 WebGLShader 24
5.9 WebGLTexture 24
5.10 WebGLUniformLocation 24
5.11 WebGLActiveInfo 25
5.12 ArrayBuffer 和 Typed Arrays 25
示例3 26
5.13 WebGL 环境 27
5.14 WebGLContextEvent 59
示例4 60
6 WebGL和OpenGL ES 2.0的区别 64
6.1 缓存物件绑定 64
6.2 没有客户端数组 64
6.3 缓存偏移和跨度需求 65
6.4 启用顶点属性和范围检查 65
6.5 帧存物件挂接 66
6.6 像素存储参数 67
6.7 从帧存读出像素 68
6.8 模板分离掩码和参考值 68
6.9 顶点属性数据跨度 69
6.10 视口深度范围 69
6.11 常量颜色融合 69
6.12 定点数支持 70
6.13 GLSL Constructs 70
6.14 扩展查询 70
6.15 实现的颜色读取格式与类型 71
6.16 压缩纹理支持 71
6.17 GLSL 标识名最长限制 72

6.18 GLSL源字符集外的字符 72
6.19 字串长度查询 73
7 引用.....74
7.1 Normative references 74
7.2 Other references 75
8 Acknowledgments.....76

译者前言

WebGL是目前最令人着迷的电脑绘图技术之一。因为，尽管GPU已经在电脑和高端手机中无所不在，其计算能力也已超越CPU几倍有余，但游戏之外，GPU基本处于闲置状态。固然是三维绘图的难度所致，但其难度相对于程序员所要掌握的语言和电脑知识，并非高不可攀。恐怕更重要的是观念，认为三维绘图只属游戏或制图等有限领域，用户接受度不高，硬件兼容不佳，且数据量庞大。WebGL可以打破这一认知误区。就像网络视频一下子被用户接受，由WebGL支持的各种应用，也会自然而然的籍由浏览器，进入千家万户。

由于WebGL刚刚推出，支持度不高，中文资料更少。本人在自学只余，自觉自发的翻译一些关键文档，供国人参考。希望有志者能带领中国突破低端产业链，融入科技发展的最前沿。附带一句，冲破藩篱靠自己，社会没主义更要坚持正义。国人加油！

另，错误在所难免，一切以原文为准。

反犬的虫

2011年二月

1 介绍

WebGL是针对万维网的即时三维绘图API。它来自 OpenGL ES 2.0，提供类似的绘图功能，但应用在HTML的环境。WebGL设计为HTML画板Canvas元素的绘图环境。HTML画板提供了在网页中编程绘画的目标，并允许使用不同的绘画API执行。Canvas 规范只定义了二维的绘图环境界面CanvasRenderingContext2D。此文件讲述另一界面WebGLRenderingContext，提供 WebGL的API。

此API的即时性背离了大多数的web API。基于三维图形的多数使用案例，WebGL选用此方案，来提供可应用于所有使用案例的灵活原语。库函数可在WebGL之上提供为专门领域定制的API，来给WebGL添加便利层，加速和简化开发。但是，因为它的OpenGL ES 2.0遗传，对于熟悉现代桌面 OpenGL 或 OpenGL ES 2.0的开发员，转移到 WebGL 的开发，应该是直截了当的。

此文件的许多函数带有 OpenGL ES 手册页的链接（译注：函数的链接和说明没有列出，详见原文）。尽管尽了最大努力使这些页匹配OpenGL ES 2.0的规范，它们还可能有错误。在矛盾时，以 OpenGL ES 2.0规范为准。

此文件的后面章节需要和OpenGL ES 2.0规范一起阅读。除特别指出，每个方法的行为由OpenGL ES 2.0规范定义。本规范可以背离OpenGL ES 2.0规范，以用来确保互操作或安全性，通常是OpenGL ES2.0规范留给实现自定义的领域。这些不同总结在“WebGL和OpenGL ES 2.0的区别”一节。

2 环境创建和绘图缓存表示

使用WebGL API之前，作者必须从如下指定的 HTMLCanvasElement 取得一个WebGLRenderingContext物件。这个物件用来管理OpenGL状态，并在绘图缓存绘画，此缓存也必须在创建环境时生成。作者可以提供此绘图缓存的配置选项，否则此文件中给出的默认值会被使用。此绘图缓存在HTML页复合(composite)操作之前提供给HTML复合器，但仅当绘图缓存在上次复合操作后被更动。

2.1 canvas 元素

WebGLRenderingContext 应在给定的 HTMLCanvasElement 上使用字串 ‘webgl’ 调用 getContext() 方法得到。此字串大小写敏感。第一次调用时，一个WebGLRenderingContext物件被创建返回。在此时一个绘图缓存也应该被创建。后续使用同样字串调用 getContext() 应返回同样物件。

HTML Canvas 规范定义了试图对同一 canvas 元素提取两个或更多不兼容的环境的行为。

第二个参数可以传递给 getContext() 方法。如传递，此参数应是一个 WebGLContextAttributes 物件，带有用于创建绘图缓存的配置参数。详见 WebGLContextAttributes 一节。

alpha, depth, stencil 和 antialias属性是请求，不是需求。WebGL实现不保证它们被遵守，但应尽量满足。WebGL实现或图形硬件不支持的属性组合不应导致 WebGLRenderingContext创建的失败。实际用来创建环境的属性，可用WebGLRenderingContext上的getContextAttributes()方法取得。WebGL实现必须遵循 premultipliedAlpha 和 preserveDrawingBuffer 属性。

后续以 ‘webgl’ 调用getContext()时，传入的 WebGLContextAttributes 物件应被忽略。

2.2 绘图缓存

API绘画用的绘图缓存应在创建 WebGLRenderingContext 物件时定义。下表给出组成绘图缓存所有缓存，以及其最小尺寸和是否默认定义。绘图缓存的最小尺寸应由 HTMLCanvasElement 的 width 和 height 属性决定。下表同时给出创建或尺寸改变时的清零值。

如果请求的宽或高不能满足，不管是绘图缓存或 HTMLCanvasElement的width 和 height属性改变时，一个较小的绘图

缓存应被创建。实际使用的尺寸有实现决定，并不保证同样长宽比的缓存会被创建。实际绘图缓存的尺寸可由 drawingBufferWidth 和 drawingBufferHeight 得到。

默认的缓存尺寸如下所示。可选的 WebGLContextAttributes 物件可用于改变缓存是否定义。还可用于定义颜色缓存是否包含alpha信道。如有，alpha信道用于HTML复合器用来和网页的其它颜色缓存组合。WebGLContextAttributes 物件只在 getContex t第一次调用时使用。在绘图缓存创建后，没有工具可以改变其属性。

WebGL绘图缓存在 HTML 页复合操作之前提供给 HTML 复合器，但仅当绘图缓存在上次复合操作后被更动。在提供绘图缓存给复合操作之前，实现应确保所有的绘画操作已经冲洗（flush）到绘图缓存。默认的，复合后绘图缓存的内容应清为如上表所示默认值。

此默认行为可由设置 WebGLContextAttributes 物件的 preserveDrawingBuffer 属性改变。如果此设置为真，绘图缓存的内容应保留，直到作者清除或覆盖它们。如果此设置为假，在绘画函数返回后，试图用此环境作为源图像的操作，都会导致无定义的行为。这包括 readPixels 和 toDataURL 调用，及使用此环境作为另一环境的 texImage2D 或 drawImage的源图像。

非正式:

尽管有时希望能保留绘图缓存，但这将在某些平台导致明显的性能下降。　尽量将此设置为假，并使用其它的技巧。例如同步绘图缓存存取可用来取得其内容（例如在对缓存绘画的同一函数中调用readPixels 或 toDataURL）。如果作者希望在不同的调用中对同一缓存绘图，可以使用 Framebuffer 物件。

实现可以把隐含的绘图缓存的清空操作优化掉，只要能确保作者不能从另一进程取得缓存内容。例如，如果作者明确清空，则不再需要隐含清空。

2.3 WebGL视口

OpenGL　管理一个方形的视口（viewport）作为状态的一部分，定义在绘图缓存绘画结果的方位。在 WebGL 环境创建后，视口初始为原点（0，0），长宽等同于 (canvas.width, canvas.height) 的方形。

WebGL的实现"不应该"针对 canvas 元素的尺寸改变影响OpenGL视口的状态。

示例1:

注意如果 WebGL 程序不包含设置视口的逻辑，它将不能正确处理canvas尺寸的改变。下例的 ECMAScript 示范 WebGL 程序如何编程的改变 canvas 尺寸。

```
var canvas = document.getElementById('canvas1');
var gl = canvas.getContext('webgl');
canvas.width = newWidth;
canvas.height = newHeight;
gl.viewport(0, 0, canvas.width, canvas.height);
```

理论依据：自动设置视口会干扰手动设置它们的程序。程序期待使用onresize经手者（handler）回应canvas尺寸的改变，并相应设置OpenGL视口。

2.4 预乘Alpha, Canvas APIs和texImage2D

OpenGL API 允许程序在绘画时改变融合（blend）模式，因此允许控制绘图缓存的aplha值如何解释；参照 WebGLContextAttributes 一节的 premultipliedAlpha 参数。

HTML canvas API 的 toDataURL 和 drawImage 必须遵循 premultipliedAlpha 环境创建参数。当 toDataURL 调用在正进行绘画的 WebGL 内容的 canvas 时，如果要求的图像格式未指定预乘（premultiplied）的alpha，并且WebGL环境的 premultipliedAlpha 设

为真，则像素值必须反乘 （demultiplied）；即，颜色信道被alpha信道除。注意，此操作有损失。

传递WebGL绘画的 canvas给CanvasRenderingContext2D的 drawImage方法，可以或不需在绘图操作时修改已绘的WebGL内容，这取决于CanvasRenderingContext2D 实现的预乘需要。

当把WebGL绘画的 canvas 传递给 texImage2D API 时，取决于被代入的 canvas的 premultipliedAlpha 环境创建参数设置，和目标WebGL 环境的 UNPACK_PREMULTIPLY_ALPHA_WEBGL 像素存储参数，像素值可能需要改变预乘格式。

3 WebGL资源

OpenGL管理一些类型的资源作为它状态的一部分。它们由整型物件名指明，并从不同的创建调用取得。与之不同，WebGL 把这些资源作为DOM 物件。每个物件都从 WebGLObject 界面导出。目前支持的资源有：纹理 texture，缓存 buffer（即VBO），帧存 framebuffer，绘存renderbuffer，渲染shader和程序program。WebGLRenderingContext界面有个方法可以创建每类的 WebGLObject子类。底层图像库的数据存入这些物件并被之全权管理。只要物件存在，这些资源一定存在。并且，只要作者明确的引用，或被底层图像库使用，DOM 物件一定存在。当这些条件不具备时，用户代理可以随时使用对应的delete调用(例如 deleteTexture） 删除物件。如果作者希望掌控何时释放底层资源，可以直接这些使用 delete 调用。

4 安全

4.1 资源局限

WebGL 的资源，例如纹理和顶点缓冲物件 （VBO）必须带有已经初始化的数据，即时它们创建时没有初始值。无初始值的创建资源通常用来给纹理或 VBO 留位，然后被 texSubImage 或 bufferSubData 调用修改。如果没有初始值提供给这些调用，WebGL 实现必须初始其内容为0，深度绘存 （depth renderbuffers） 必须清为默认的1.0。例如，可能需要创建一个0值的暂存，尺寸等同于要求的VBO，以便正确初始化。所有其他形式的加载数据于纹理或 VBO，涉及 ArrayBuffers 或例如图像的 DOM 物件，都需要已经初始化。

当 WebGL 资源被渲染通过 drawElements 或 drawArrays 这样的调用存取时，WebGL 实现必须确保渲染不能存取越界或存取未初始化的数据。参照 “Enabled Vertex Attributes and Range Checking" WebGL 实现所必须的限定。

4.2 起源局限

为避免资讯泄漏，HTML5 画板元素有个 origin-clean 设置（见HTML5, 4.8.11.3 节 境，如果需要以下行为，此 origin-clean 必须为假:

- 使用起源不同于画板元素的 Document 物件的 HTMLImageElement 或 HTMLVideoElement 调用 texImage2D。
- 使用origin-clean为假的 HTMLCanvasElement 调用texImage2D。

只要origin-clean为假，即便其它参量正确，其canvas元素的2D环境的readPixels方法都会引发SECURITY_ERR异常。

4.3 支持的GLSL Constructs

WebGL实现必须仅仅接受符合 OpenGL ES Shading Language 版本1.0的渲染，并且不超越其附录A强制的最小功能。特别是，可用于其它版本GLSL(例如桌面OpenGL版本）的渲染引用状态变量或函数，必须禁止加载。

除了上面规范中的保留标识之外，以“webgl_” 和 “_webgl_”开始的标识被WebGL保留。渲染中由此前缀开始的函数，变量，结构名或结构域名，必须禁止加载。

4.4 防卫 Denial of Service

非正式:

有可能故意或无意间组合渲染和几何geometry使得绘画时间过长。此问题类似于长时间运行的脚本，用户代理已经可以自卫。但是，长时间运行的绘图调用可导致整个窗口系统，而不仅仅是用户代理，失去交互性。

基本上不可能对入侵的渲染结构加以限制来对抗此问题。实验已经证明，即使最严格的限制也不能防止长时间绘图，且此类限制阻碍了渲染的作者实现常用的算法。

用户代理应采取自卫防止过长的绘图和对应的交互损失。建议的自卫包括:

- 把带有大量元素的绘图调用分为较小的绘图调用。
- 对绘图调用计时，如超时，则禁止后续的此页绘图。

- 使用用户层，图像API层或操作系统层提供的看门狗工具限制绘图调用的时长。
- 隔离用户代理的图像绘制为单独的操作系统进程，以便对其无损程序状态的结束和重启。

在OS和图像API层的基础支持，会日后改进，因此，防卫的精确性质此处并未指明。

5 DOM界面

此节描述加入DOM的界面和函数，用于运行态得到上面的功能。

5.1 类型

如下类型在后续章节的界面中使用。

```
typedef events::Event Event;
typedef html::HTMLCanvasElement HTMLCanvasElement;
typedef html::HTMLImageElement HTMLImageElement;
typedef html::HTMLVideoElement HTMLVideoElement;
typedef html::ImageData ImageData;
typedef unsigned long GLenum;
typedef boolean GLboolean;
typedef unsigned long GLbitfield;
typedef byte GLbyte; /* 'byte' 应是8位有符号类型 */
typedef short GLshort;
typedef long GLint;
typedef long GLsizei;
typedef long long GLintptr;
typedef long long GLsizeiptr;
typedef unsigned byte GLubyte; /*unsigned byte是8位无符号型*/
typedef unsigned short GLushort;
typedef unsigned long GLuint;
typedef float GLfloat;
typedef float GLclampf;
```

5.2 WebGLContextAttributes

WebGLContextAttributes 接口带有绘图表面（drawing surface）属性，并作为第二个参数传给 getContext。本机物件（native object）也可作为此参数；指定的属性能从此物件查询。

```
[Callback] interface WebGLContextAttributes {
     attribute boolean alpha;
     attribute boolean depth;
     attribute boolean stencil;
     attribute boolean antialias;
     attribute boolean premultipliedAlpha;
     attribute boolean preserveDrawingBuffer;
};
```

5.2.1 环境生成参数

下表列出 WebGLContextAttributes 物件的每个属性及用途，并给出默认值。此默认值用于传给 getContext 的第二个参数不存在，或传入的本机物件不存在给出的名字时。

- alpha 默认：真。如真，绘图缓存带有alpha信道，用于OpenGL目标alpha操作和页面的复合。如假，aplha信道不存在。
- depth 默认：真。如真，绘图缓存带有最少16位的深度缓存。如假，深度缓存不存在。

- stencil 默认：假。如真，绘图缓存带有最少8位的模板缓存。如假，模板缓存不存在。
- antialias 默认：真。如真，支持抗锯齿绘图缓存的实现使用其选择的技术（多采样/超采样）和质量执行抗锯齿操作。如假或实现不支持抗锯齿操作，不做抗锯齿操作。
- premultipliedAlpha 默认：真。如真，页面复合器假定绘图缓存的颜色带有预乘的alpha值。如假，页面复合器假定绘图缓存的颜色没有预乘。如果alpha设置为假，此设置忽略。见 Premultiplied Alpha 关于 premultipliedAlpha 效果的更多资料。
- preserveDrawingBuffer 默认：假。如假，当绘图缓存如 Drawing Buffer 一节所示时，绘图缓存的内容清为默认值。所有绘图缓存的元素（颜色，深度和模板）被清空。如真，缓存不清空，保持其值直到被作者清除或覆盖。 某些硬件设置 preserveDrawingBuffer 为真会有严重的性能影响。

示例2

此例的 ECMAScript 会把一个 WebGLContextAttributes 参量传给 getContext。 假设有个名为 canvas1 的画板元素存在页面上。

```
var canvas = document.getElementById('canvas1');
var context = canvas.getContext('webgl', { antialias: false,
stencil: true });
```

5.3 WebGLObject

WebGLObject 界面是所有 GL 物件的父界面。

```
interface WebGLObject {
};
```

5.4 WebGLBuffer

WebGLBuffer 界面代表一个 OpenGL 的缓存物件。底层物件使用类似 glGenBuffers 的调用创建，和类似 glBindBuffer 的调用绑定， 及类似 glDeleteBuffers 的调用摧毁。

```
interface WebGLBuffer : WebGLObject {
}
```

5.5 WebGLFramebuffer

WebGLFramebuffer 界面代表一个 OpenGL Framebuffer 物件。底层物件使用类似 glGenFramebuffers 的调用创建，用类似 glBindFramebuffer 的调用绑定，及类似 glDeleteFramebuffers 的调用摧毁。

```
interface WebGLFramebuffer : WebGLObject {
}
```

5.6 WebGLProgram

WebGLProgram 界面代表一个 OpenGL Program 物件。底层物件使用类似 glCreateProgram 的调用创建，用类似 glUseProgram 的调用绑定，及类似 glDeleteProgram 的调用摧毁。

```
interface WebGLProgram : WebGLObject {
}
```

5.7 WebGLRenderbuffer

WebGLRenderbuffer 界面代表一个 OpenGL Renderbuffer 物件。底层物件使用类似 glGenRenderbuffers 的调用创建，用类似 glBindRenderbuffer 的调用绑定，及类似 glDeleteRenderbuffers 的调用摧毁。

```
interface WebGLRenderbuffer : WebGLObject {
}
```

5.8 WebGLShader

WebGLShader 界面代表一个 OpenGL Shader 物件。底层物件使用类似 glCreateShader 的调用创建，用类似 glAttachShader 的调用挂接，及类似 glDeleteShader 的调用摧毁。

```
interface WebGLShader : WebGLObject {
}
```

5.9 WebGLTexture

WebGLTexture 界面代表一个 OpenGL Texture 物件。底层物件使用类似 glGenTextures 的调用创建，用类似 glBindTexture 的调用绑定，及类似 glDeleteTextures 的调用摧毁。

```
interface WebGLTexture : WebGLObject {
}
```

5.10 WebGLUniformLocation

WebGLUniformLocation 界面代表一个渲染程序用的 uniform 变量。

```
interface WebGLUniformLocation {
}
```

5.11 WebGLActiveInfo

WebGLActiveInfo 界面代表 getActiveAttrib 和 getActiveUniform 返回的信息。

```
interface WebGLActiveInfo {
      readonly attribute GLint size;
      readonly attribute GLenum type;
      readonly attribute DOMString name;
}
```

5.11.1 属性

存在下列属性:

- size of type GLint 请求变量的尺寸
- type of type GLenum请求变量的数据类型
- name of type DOMString 请求变量名

5.12 ArrayBuffer 和 Typed Arrays

顶点，下标，纹理，和其它数据使用有型数组 (Typed Arrays) 规范里定义的 ArrayBuffer 和 views 传递给 WebGL 实现。除 Float64Array 之外的所有 typed array views 都可和WebGL共用。OpenGL ES 2.0, 及其 WebGL，不支持双精度浮点类型数据。

有型数组支持创建交叉 （interleaved） 的，异质 （heterogeneous） 的顶点数据; 以及上载不同的数据块给大型的顶点缓冲物件，及 OpenGL 程序中绝大部分的使用案例。

示例3

此ECMAScript示例展示用不同类型的有型数组存取同一 ArrayBuffer 。例中的缓存包含一个浮点的顶点位置 (x, y, z), 后跟一个颜色为 4 个无符号字节 (r, g, b, a) 。

```
var numVertices = 100; // for example

// Compute the size needed for the buffer, in bytes and
floats
var vertexSize = 3 * Float32Array.BYTES_PER_ELEMENT +
     4 * Uint8Array.BYTES_PER_ELEMENT;
var vertexSizeInFloats = vertexSize /
Float32Array.BYTES_PER_ELEMENT;

// Allocate the buffer
var buf = new ArrayBuffer(numVertices * vertexSize);

// Map this buffer to a Float32Array to access the positions
```

```
var positionArray = new Float32Array(buf);

// Map the same buffer to a Uint8Array to access the color
var colorArray = new Uint8Array(buf);

// Set up the initial offset of the vertices and colors
within the buffer
var positionIdx = 0;
var colorIdx = 3 * Float32Array.BYTES_PER_ELEMENT;

// Initialize the buffer
for (var i = 0; i < numVertices; i++) {
    positionArray[positionIdx] = ...;
    positionArray[positionIdx + 1] = ...;
    positionArray[positionIdx + 2] = ...;
    colorArray[colorIdx] = ...;
    colorArray[colorIdx + 1] = ...;
    colorArray[colorIdx + 2] = ...;
    colorArray[colorIdx + 3] = ...;
    positionIdx += vertexSizeInFloats;
    colorIdx += vertexSize;
}
```

5.13 WebGL 环境

WebGLRenderingContext 一个 API代表允许以 OpenGL ES 2.0 的风格在画板元素上绘图。

```
interface WebGLRenderingContext {
/* ClearBufferMask */
const GLenum DEPTH_BUFFER_BIT = 0x00000100;
const GLenum STENCIL_BUFFER_BIT = 0x00000400;
const GLenum COLOR_BUFFER_BIT = 0x00004000;
/* BeginMode */
```

```
const GLenum POINTS = 0x0000;
const GLenum LINES = 0x0001;
const GLenum LINE_LOOP = 0x0002;
const GLenum LINE_STRIP = 0x0003;
const GLenum TRIANGLES = 0x0004;
const GLenum TRIANGLE_STRIP = 0x0005;
const GLenum TRIANGLE_FAN = 0x0006;
/* AlphaFunction (not supported in ES20) */
/* NEVER */
/* LESS */
/* EQUAL */
/* LEQUAL */
/* GREATER */
/* NOTEQUAL */
/* GEQUAL */
/* ALWAYS */
/* BlendingFactorDest */
const GLenum ZERO = 0;
const GLenum ONE = 1;
const GLenum SRC_COLOR = 0x0300;
const GLenum ONE_MINUS_SRC_COLOR = 0x0301;
const GLenum SRC_ALPHA = 0x0302;
const GLenum ONE_MINUS_SRC_ALPHA = 0x0303;
const GLenum DST_ALPHA = 0x0304;
const GLenum ONE_MINUS_DST_ALPHA = 0x0305;
/* BlendingFactorSrc */
/* ZERO */
/* ONE */
const GLenum DST_COLOR = 0x0306;
const GLenum ONE_MINUS_DST_COLOR = 0x0307;
const GLenum SRC_ALPHA_SATURATE = 0x0308;
/* SRC_ALPHA */
/* ONE_MINUS_SRC_ALPHA */
/* DST_ALPHA */
/* ONE_MINUS_DST_ALPHA */
/* BlendEquationSeparate */
const GLenum FUNC_ADD = 0x8006;
const GLenum BLEND_EQUATION = 0x8009;
const GLenum BLEND_EQUATION_RGB = 0x8009; /* same as
BLEND_EQUATION */
```

```
const GLenum BLEND_EQUATION_ALPHA = 0x883D;
/* BlendSubtract */
const GLenum FUNC_SUBTRACT = 0x800A;
const GLenum FUNC_REVERSE_SUBTRACT = 0x800B;
/* Separate Blend Functions */
const GLenum BLEND_DST_RGB = 0x80C8;
const GLenum BLEND_SRC_RGB = 0x80C9;
const GLenum BLEND_DST_ALPHA = 0x80CA;
const GLenum BLEND_SRC_ALPHA = 0x80CB;
const GLenum CONSTANT_COLOR = 0x8001;
const GLenum ONE_MINUS_CONSTANT_COLOR = 0x8002;
const GLenum CONSTANT_ALPHA = 0x8003;
const GLenum ONE_MINUS_CONSTANT_ALPHA = 0x8004;
const GLenum BLEND_COLOR = 0x8005;
/* Buffer Objects */
const GLenum ARRAY_BUFFER = 0x8892;
const GLenum ELEMENT_ARRAY_BUFFER = 0x8893;
const GLenum ARRAY_BUFFER_BINDING = 0x8894;
const GLenum ELEMENT_ARRAY_BUFFER_BINDING = 0x8895;
const GLenum STREAM_DRAW = 0x88E0;
const GLenum STATIC_DRAW = 0x88E4;
const GLenum DYNAMIC_DRAW = 0x88E8;
const GLenum BUFFER_SIZE = 0x8764;
const GLenum BUFFER_USAGE = 0x8765;
const GLenum CURRENT_VERTEX_ATTRIB = 0x8626;
/* CullFaceMode */
const GLenum FRONT = 0x0404;
const GLenum BACK = 0x0405;
const GLenum FRONT_AND_BACK = 0x0408;
/* DepthFunction */
/* NEVER */
/* LESS */
/* EQUAL */
/* LEQUAL */
/* GREATER */
/* NOTEQUAL */
/* GEQUAL */
/* ALWAYS */
/* EnableCap */
/* TEXTURE_2D */
```

```
const GLenum CULL_FACE = 0x0B44;
const GLenum BLEND = 0x0BE2;
const GLenum DITHER = 0x0BD0;
const GLenum STENCIL_TEST = 0x0B90;
const GLenum DEPTH_TEST = 0x0B71;
const GLenum SCISSOR_TEST = 0x0C11;
const GLenum POLYGON_OFFSET_FILL = 0x8037;
const GLenum SAMPLE_ALPHA_TO_COVERAGE = 0x809E;
const GLenum SAMPLE_COVERAGE = 0x80A0;
/* ErrorCode */
const GLenum NO_ERROR = 0;
const GLenum INVALID_ENUM = 0x0500;
const GLenum INVALID_VALUE = 0x0501;
const GLenum INVALID_OPERATION = 0x0502;
const GLenum OUT_OF_MEMORY = 0x0505;
/* FrontFaceDirection */
const GLenum CW = 0x0900;
const GLenum CCW = 0x0901;
/* GetPName */
const GLenum LINE_WIDTH = 0x0B21;
const GLenum ALIASED_POINT_SIZE_RANGE = 0x846D;
const GLenum ALIASED_LINE_WIDTH_RANGE = 0x846E;
const GLenum CULL_FACE_MODE = 0x0B45;
const GLenum FRONT_FACE = 0x0B46;
const GLenum DEPTH_RANGE = 0x0B70;
const GLenum DEPTH_WRITEMASK = 0x0B72;
const GLenum DEPTH_CLEAR_VALUE = 0x0B73;
const GLenum DEPTH_FUNC = 0x0B74;
const GLenum STENCIL_CLEAR_VALUE = 0x0B91;
const GLenum STENCIL_FUNC = 0x0B92;
const GLenum STENCIL_FAIL = 0x0B94;
const GLenum STENCIL_PASS_DEPTH_FAIL = 0x0B95;
const GLenum STENCIL_PASS_DEPTH_PASS = 0x0B96;
const GLenum STENCIL_REF = 0x0B97;
const GLenum STENCIL_VALUE_MASK = 0x0B93;
const GLenum STENCIL_WRITEMASK = 0x0B98;
const GLenum STENCIL_BACK_FUNC = 0x8800;
const GLenum STENCIL_BACK_FAIL = 0x8801;
const GLenum STENCIL_BACK_PASS_DEPTH_FAIL = 0x8802;
const GLenum STENCIL_BACK_PASS_DEPTH_PASS = 0x8803;
```

```
const GLenum STENCIL_BACK_REF = 0x8CA3;
const GLenum STENCIL_BACK_VALUE_MASK = 0x8CA4;
const GLenum STENCIL_BACK_WRITEMASK = 0x8CA5;
const GLenum VIEWPORT = 0x0BA2;
const GLenum SCISSOR_BOX = 0x0C10;
/* SCISSOR_TEST */
const GLenum COLOR_CLEAR_VALUE = 0x0C22;
const GLenum COLOR_WRITEMASK = 0x0C23;
const GLenum UNPACK_ALIGNMENT = 0x0CF5;
const GLenum PACK_ALIGNMENT = 0x0D05;
const GLenum MAX_TEXTURE_SIZE = 0x0D33;
const GLenum MAX_VIEWPORT_DIMS = 0x0D3A;
const GLenum SUBPIXEL_BITS = 0x0D50;
const GLenum RED_BITS = 0x0D52;
const GLenum GREEN_BITS = 0x0D53;
const GLenum BLUE_BITS = 0x0D54;
const GLenum ALPHA_BITS = 0x0D55;
const GLenum DEPTH_BITS = 0x0D56;
const GLenum STENCIL_BITS = 0x0D57;
const GLenum POLYGON_OFFSET_UNITS = 0x2A00;
/* POLYGON_OFFSET_FILL */
const GLenum POLYGON_OFFSET_FACTOR = 0x8038;
const GLenum TEXTURE_BINDING_2D = 0x8069;
const GLenum SAMPLE_BUFFERS = 0x80A8;
const GLenum SAMPLES = 0x80A9;
const GLenum SAMPLE_COVERAGE_VALUE = 0x80AA;
const GLenum SAMPLE_COVERAGE_INVERT = 0x80AB;
/* GetTextureParameter */
/* TEXTURE_MAG_FILTER */
/* TEXTURE_MIN_FILTER */
/* TEXTURE_WRAP_S */
/* TEXTURE_WRAP_T */
const GLenum NUM_COMPRESSED_TEXTURE_FORMATS = 0x86A2;
const GLenum COMPRESSED_TEXTURE_FORMATS = 0x86A3;
/* HintMode */
const GLenum DONT_CARE = 0x1100;
const GLenum FASTEST = 0x1101;
const GLenum NICEST = 0x1102;
/* HintTarget */
const GLenum GENERATE_MIPMAP_HINT = 0x8192;
```

```
/* DataType */
const GLenum BYTE = 0x1400;
const GLenum UNSIGNED_BYTE = 0x1401;
const GLenum SHORT = 0x1402;
const GLenum UNSIGNED_SHORT = 0x1403;
const GLenum INT = 0x1404;
const GLenum UNSIGNED_INT = 0x1405;
const GLenum FLOAT = 0x1406;
/* PixelFormat */
const GLenum DEPTH_COMPONENT = 0x1902;
const GLenum ALPHA = 0x1906;
const GLenum RGB = 0x1907;
const GLenum RGBA = 0x1908;
const GLenum LUMINANCE = 0x1909;
const GLenum LUMINANCE_ALPHA = 0x190A;
/* PixelType */
/* UNSIGNED_BYTE */
const GLenum UNSIGNED_SHORT_4_4_4_4 = 0x8033;
const GLenum UNSIGNED_SHORT_5_5_5_1 = 0x8034;
const GLenum UNSIGNED_SHORT_5_6_5 = 0x8363;
/* Shaders */
const GLenum FRAGMENT_SHADER = 0x8B30;
const GLenum VERTEX_SHADER = 0x8B31;
const GLenum MAX_VERTEX_ATTRIBS = 0x8869;
const GLenum MAX_VERTEX_UNIFORM_VECTORS = 0x8DFB;
const GLenum MAX_VARYING_VECTORS = 0x8DFC;
const GLenum MAX_COMBINED_TEXTURE_IMAGE_UNITS = 0x8B4D;
const GLenum MAX_VERTEX_TEXTURE_IMAGE_UNITS = 0x8B4C;
const GLenum MAX_TEXTURE_IMAGE_UNITS = 0x8872;
const GLenum MAX_FRAGMENT_UNIFORM_VECTORS = 0x8DFD;
const GLenum SHADER_TYPE = 0x8B4F;
const GLenum DELETE_STATUS = 0x8B80;
const GLenum LINK_STATUS = 0x8B82;
const GLenum VALIDATE_STATUS = 0x8B83;
const GLenum ATTACHED_SHADERS = 0x8B85;
const GLenum ACTIVE_UNIFORMS = 0x8B86;
const GLenum ACTIVE_ATTRIBUTES = 0x8B89;
const GLenum SHADING_LANGUAGE_VERSION = 0x8B8C;
const GLenum CURRENT_PROGRAM = 0x8B8D;
/* StencilFunction */
```

```
const GLenum NEVER = 0x0200;
const GLenum LESS = 0x0201;
const GLenum EQUAL = 0x0202;
const GLenum LEQUAL = 0x0203;
const GLenum GREATER = 0x0204;
const GLenum NOTEQUAL = 0x0205;
const GLenum GEQUAL = 0x0206;
const GLenum ALWAYS = 0x0207;
/* StencilOp */
/* ZERO */
const GLenum KEEP = 0x1E00;
const GLenum REPLACE = 0x1E01;
const GLenum INCR = 0x1E02;
const GLenum DECR = 0x1E03;
const GLenum INVERT = 0x150A;
const GLenum INCR_WRAP = 0x8507;
const GLenum DECR_WRAP = 0x8508;
/* StringName */
const GLenum VENDOR = 0x1F00;
const GLenum RENDERER = 0x1F01;
const GLenum VERSION = 0x1F02;
/* TextureMagFilter */
const GLenum NEAREST = 0x2600;
const GLenum LINEAR = 0x2601;
/* TextureMinFilter */
/* NEAREST */
/* LINEAR */
const GLenum NEAREST_MIPMAP_NEAREST = 0x2700;
const GLenum LINEAR_MIPMAP_NEAREST = 0x2701;
const GLenum NEAREST_MIPMAP_LINEAR = 0x2702;
const GLenum LINEAR_MIPMAP_LINEAR = 0x2703;
/* TextureParameterName */
const GLenum TEXTURE_MAG_FILTER = 0x2800;
const GLenum TEXTURE_MIN_FILTER = 0x2801;
const GLenum TEXTURE_WRAP_S = 0x2802;
const GLenum TEXTURE_WRAP_T = 0x2803;
/* TextureTarget */
const GLenum TEXTURE_2D = 0x0DE1;
const GLenum TEXTURE = 0x1702;
const GLenum TEXTURE_CUBE_MAP = 0x8513;
```

```
const GLenum TEXTURE_BINDING_CUBE_MAP = 0x8514;
const GLenum TEXTURE_CUBE_MAP_POSITIVE_X = 0x8515;
const GLenum TEXTURE_CUBE_MAP_NEGATIVE_X = 0x8516;
const GLenum TEXTURE_CUBE_MAP_POSITIVE_Y = 0x8517;
const GLenum TEXTURE_CUBE_MAP_NEGATIVE_Y = 0x8518;
const GLenum TEXTURE_CUBE_MAP_POSITIVE_Z = 0x8519;
const GLenum TEXTURE_CUBE_MAP_NEGATIVE_Z = 0x851A;
const GLenum MAX_CUBE_MAP_TEXTURE_SIZE = 0x851C;
/* TextureUnit */
const GLenum TEXTURE0 = 0x84C0;
const GLenum TEXTURE1 = 0x84C1;
const GLenum TEXTURE2 = 0x84C2;
const GLenum TEXTURE3 = 0x84C3;
const GLenum TEXTURE4 = 0x84C4;
const GLenum TEXTURE5 = 0x84C5;
const GLenum TEXTURE6 = 0x84C6;
const GLenum TEXTURE7 = 0x84C7;
const GLenum TEXTURE8 = 0x84C8;
const GLenum TEXTURE9 = 0x84C9;
const GLenum TEXTURE10 = 0x84CA;
const GLenum TEXTURE11 = 0x84CB;
const GLenum TEXTURE12 = 0x84CC;
const GLenum TEXTURE13 = 0x84CD;
const GLenum TEXTURE14 = 0x84CE;
const GLenum TEXTURE15 = 0x84CF;
const GLenum TEXTURE16 = 0x84D0;
const GLenum TEXTURE17 = 0x84D1;
const GLenum TEXTURE18 = 0x84D2;
const GLenum TEXTURE19 = 0x84D3;
const GLenum TEXTURE20 = 0x84D4;
const GLenum TEXTURE21 = 0x84D5;
const GLenum TEXTURE22 = 0x84D6;
const GLenum TEXTURE23 = 0x84D7;
const GLenum TEXTURE24 = 0x84D8;
const GLenum TEXTURE25 = 0x84D9;
const GLenum TEXTURE26 = 0x84DA;
const GLenum TEXTURE27 = 0x84DB;
const GLenum TEXTURE28 = 0x84DC;
const GLenum TEXTURE29 = 0x84DD;
const GLenum TEXTURE30 = 0x84DE;
```

```
const GLenum TEXTURE31 = 0x84DF;
const GLenum ACTIVE_TEXTURE = 0x84E0;
/* TextureWrapMode */
const GLenum REPEAT = 0x2901;
const GLenum CLAMP_TO_EDGE = 0x812F;
const GLenum MIRRORED_REPEAT = 0x8370;
/* Uniform Types */
const GLenum FLOAT_VEC2 = 0x8B50;
const GLenum FLOAT_VEC3 = 0x8B51;
const GLenum FLOAT_VEC4 = 0x8B52;
const GLenum INT_VEC2 = 0x8B53;
const GLenum INT_VEC3 = 0x8B54;
const GLenum INT_VEC4 = 0x8B55;
const GLenum BOOL = 0x8B56;
const GLenum BOOL_VEC2 = 0x8B57;
const GLenum BOOL_VEC3 = 0x8B58;
const GLenum BOOL_VEC4 = 0x8B59;
const GLenum FLOAT_MAT2 = 0x8B5A;
const GLenum FLOAT_MAT3 = 0x8B5B;
const GLenum FLOAT_MAT4 = 0x8B5C;
const GLenum SAMPLER_2D = 0x8B5E;
const GLenum SAMPLER_CUBE = 0x8B60;
/* Vertex Arrays */
const GLenum VERTEX_ATTRIB_ARRAY_ENABLED = 0x8622;
const GLenum VERTEX_ATTRIB_ARRAY_SIZE = 0x8623;
const GLenum VERTEX_ATTRIB_ARRAY_STRIDE = 0x8624;
const GLenum VERTEX_ATTRIB_ARRAY_TYPE = 0x8625;
const GLenum VERTEX_ATTRIB_ARRAY_NORMALIZED = 0x886A;
const GLenum VERTEX_ATTRIB_ARRAY_POINTER = 0x8645;
const GLenum VERTEX_ATTRIB_ARRAY_BUFFER_BINDING = 0x889F;
/* Shader Source */
const GLenum COMPILE_STATUS = 0x8B81;
/* Shader Precision-Specified Types */
const GLenum LOW_FLOAT = 0x8DF0;
const GLenum MEDIUM_FLOAT = 0x8DF1;
const GLenum HIGH_FLOAT = 0x8DF2;
const GLenum LOW_INT = 0x8DF3;
const GLenum MEDIUM_INT = 0x8DF4;
const GLenum HIGH_INT = 0x8DF5;
/* Framebuffer Object. */
```

```
const GLenum FRAMEBUFFER = 0x8D40;
const GLenum RENDERBUFFER = 0x8D41;
const GLenum RGBA4 = 0x8056;
const GLenum RGB5_A1 = 0x8057;
const GLenum RGB565 = 0x8D62;
const GLenum DEPTH_COMPONENT16 = 0x81A5;
const GLenum STENCIL_INDEX = 0x1901;
const GLenum STENCIL_INDEX8 = 0x8D48;
const GLenum DEPTH_STENCIL = 0x84F9;
const GLenum RENDERBUFFER_WIDTH = 0x8D42;
const GLenum RENDERBUFFER_HEIGHT = 0x8D43;
const GLenum RENDERBUFFER_INTERNAL_FORMAT = 0x8D44;
const GLenum RENDERBUFFER_RED_SIZE = 0x8D50;
const GLenum RENDERBUFFER_GREEN_SIZE = 0x8D51;
const GLenum RENDERBUFFER_BLUE_SIZE = 0x8D52;
const GLenum RENDERBUFFER_ALPHA_SIZE = 0x8D53;
const GLenum RENDERBUFFER_DEPTH_SIZE = 0x8D54;
const GLenum RENDERBUFFER_STENCIL_SIZE = 0x8D55;
const GLenum FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE = 0x8CD0;
const GLenum FRAMEBUFFER_ATTACHMENT_OBJECT_NAME = 0x8CD1;
const GLenum FRAMEBUFFER_ATTACHMENT_TEXTURE_LEVEL = 0x8CD2;
const GLenum FRAMEBUFFER_ATTACHMENT_TEXTURE_CUBE_MAP_FACE =
0x8CD3;
const GLenum COLOR_ATTACHMENT0 = 0x8CE0;
const GLenum DEPTH_ATTACHMENT = 0x8D00;
const GLenum STENCIL_ATTACHMENT = 0x8D20;
const GLenum DEPTH_STENCIL_ATTACHMENT = 0x821A;
const GLenum NONE = 0;
const GLenum FRAMEBUFFER_COMPLETE = 0x8CD5;
const GLenum FRAMEBUFFER_INCOMPLETE_ATTACHMENT = 0x8CD6;
const GLenum FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT =
0x8CD7;
const GLenum FRAMEBUFFER_INCOMPLETE_DIMENSIONS = 0x8CD9;
const GLenum FRAMEBUFFER_UNSUPPORTED = 0x8CDD;
const GLenum FRAMEBUFFER_BINDING = 0x8CA6;
const GLenum RENDERBUFFER_BINDING = 0x8CA7;
const GLenum MAX_RENDERBUFFER_SIZE = 0x84E8;
const GLenum INVALID_FRAMEBUFFER_OPERATION = 0x0506;
/* WebGL-specific enums */
const GLenum UNPACK_FLIP_Y_WEBGL = 0x9240;
```

```
const GLenum UNPACK_PREMULTIPLY_ALPHA_WEBGL = 0x9241;
const GLenum CONTEXT_LOST_WEBGL = 0x9242;
const GLenum UNPACK_COLORSPACE_CONVERSION_WEBGL = 0x9243;
const GLenum BROWSER_DEFAULT_WEBGL = 0x9244;
readonly attribute HTMLCanvasElement canvas;
readonly attribute GLsizei drawingBufferWidth;
readonly attribute GLsizei drawingBufferHeight;
WebGLContextAttributes getContextAttributes();
boolean isContextLost();
DOMString[ ] getSupportedExtensions();
object getExtension(DOMString name);
void activeTexture(GLenum texture);
void attachShader(WebGLProgram program, WebGLShader shader);
void bindAttribLocation(WebGLProgram program, GLuint index,
DOMString name);
void bindBuffer(GLenum target, WebGLBuffer buffer);
void bindFramebuffer(GLenum target, WebGLFramebuffer frame-
buffer);
void bindRenderbuffer(GLenum target, WebGLRenderbuffer ren-
derbuffer);
void bindTexture(GLenum target, WebGLTexture texture);
void blendColor(GLclampf red, GLclampf green, GLclampf blue,
GLclampf alpha);
void blendEquation(GLenum mode);
void blendEquationSeparate(GLenum modeRGB, GLenum modeAl-
pha);
void blendFunc(GLenum sfactor, GLenum dfactor);
void blendFuncSeparate(GLenum srcRGB, GLenum dstRGB,
GLenum srcAlpha, GLenum dstAlpha);
void bufferData(GLenum target, GLsizeiptr size, GLenum us-
age);
void bufferData(GLenum target, ArrayBufferView data, GLenum
usage);
void bufferData(GLenum target, ArrayBuffer data, GLenum us-
age);
void bufferSubData(GLenum target, GLintptr offset, Array-
BufferView data);
void bufferSubData(GLenum target, GLintptr offset, Array-
Buffer data);
GLenum checkFramebufferStatus(GLenum target);
```

```
void clear(GLbitfield mask);
void clearColor(GLclampf red, GLclampf green, GLclampf blue,
GLclampf alpha);
void clearDepth(GLclampf depth);
void clearStencil(GLint s);
void colorMask(GLboolean red, GLboolean green, GLboolean
blue, GLboolean alpha);
void compileShader(WebGLShader shader);
void copyTexImage2D(GLenum target, GLint level, GLenum in-
ternalformat,
GLint x, GLint y, GLsizei width, GLsizei height,
GLint border);
void copyTexSubImage2D(GLenum target, GLint level, GLint
xoffset, GLint yoffset,
GLint x, GLint y, GLsizei width, GLsizei height);
WebGLBuffer createBuffer();
WebGLFramebuffer createFramebuffer();
WebGLProgram createProgram();
WebGLRenderbuffer createRenderbuffer();
WebGLShader createShader(GLenum type);
WebGLTexture createTexture();
void cullFace(GLenum mode);
void deleteBuffer(WebGLBuffer buffer);
void deleteFramebuffer(WebGLFramebuffer framebuffer);
void deleteProgram(WebGLProgram program);
void deleteRenderbuffer(WebGLRenderbuffer renderbuffer);
void deleteShader(WebGLShader shader);
void deleteTexture(WebGLTexture texture);
void depthFunc(GLenum func);
void depthMask(GLboolean flag);
void depthRange(GLclampf zNear, GLclampf zFar);
void detachShader(WebGLProgram program, WebGLShader shader);
void disable(GLenum cap);
void disableVertexAttribArray(GLuint index);
void drawArrays(GLenum mode, GLint first, GLsizei count);
void drawElements(GLenum mode, GLsizei count, GLenum type,
GLintptr offset);
void enable(GLenum cap);
void enableVertexAttribArray(GLuint index);
void finish();
```

```
void flush();
void framebufferRenderbuffer(GLenum target, GLenum attach-
ment,
GLenum renderbuffertarget,
WebGLRenderbuffer renderbuffer);
void framebufferTexture2D(GLenum target, GLenum attachment,
GLenum textarget,
WebGLTexture texture, GLint level);
void frontFace(GLenum mode);
void generateMipmap(GLenum target);
WebGLActiveInfo getActiveAttrib(WebGLProgram program, GLuint
index);
WebGLActiveInfo getActiveUniform(WebGLProgram program, GLu-
int index);
WebGLShader[ ] getAttachedShaders(WebGLProgram program);
GLint getAttribLocation(WebGLProgram program, DOMString
name);
any getParameter(GLenum pname);
any getBufferParameter(GLenum target, GLenum pname);
GLenum getError();
any getFramebufferAttachmentParameter(GLenum target, GLenum
attachment,
GLenum pname);
any getProgramParameter(WebGLProgram program, GLenum pname);
DOMString getProgramInfoLog(WebGLProgram program);
any getRenderbufferParameter(GLenum target, GLenum pname);
any getShaderParameter(WebGLShader shader, GLenum pname);
DOMString getShaderInfoLog(WebGLShader shader);
DOMString getShaderSource(WebGLShader shader);
any getTexParameter(GLenum target, GLenum pname);
any getUniform(WebGLProgram program, WebGLUniformLocation
location);
WebGLUniformLocation getUniformLocation(WebGLProgram pro-
gram, DOMString name);
any getVertexAttrib(GLuint index, GLenum pname);
GLsizeiptr getVertexAttribOffset(GLuint index, GLenum
pname);
void hint(GLenum target, GLenum mode);
GLboolean isBuffer(WebGLBuffer buffer);
GLboolean isEnabled(GLenum cap);
```

```
GLboolean isFramebuffer(WebGLFramebuffer framebuffer);
GLboolean isProgram(WebGLProgram program);
GLboolean isRenderbuffer(WebGLRenderbuffer renderbuffer);
GLboolean isShader(WebGLShader shader);
GLboolean isTexture(WebGLTexture texture);
void lineWidth(GLfloat width);
void linkProgram(WebGLProgram program);
void pixelStorei(GLenum pname, GLint param);
void polygonOffset(GLfloat factor, GLfloat units);
void readPixels(GLint x, GLint y, GLsizei width, GLsizei
height,
GLenum format, GLenum type, ArrayBufferView pixels);
void renderbufferStorage(GLenum target, GLenum internalfor-
mat,
GLsizei width, GLsizei height);
void sampleCoverage(GLclampf value, GLboolean invert);
void scissor(GLint x, GLint y, GLsizei width, GLsizei
height);
void shaderSource(WebGLShader shader, DOMString source);
void stencilFunc(GLenum func, GLint ref, GLuint mask);
void stencilFuncSeparate(GLenum face, GLenum func, GLint
ref, GLuint mask);
void stencilMask(GLuint mask);
void stencilMaskSeparate(GLenum face, GLuint mask);
void stencilOp(GLenum fail, GLenum zfail, GLenum zpass);
void stencilOpSeparate(GLenum face, GLenum fail, GLenum
zfail, GLenum zpass);
void texImage2D(GLenum target, GLint level, GLenum internal-
format,
GLsizei width, GLsizei height, GLint border, GLenum format,
GLenum type, ArrayBufferView pixels);
void texImage2D(GLenum target, GLint level, GLenum internal-
format,
GLenum format, GLenum type, ImageData pixels);
void texImage2D(GLenum target, GLint level, GLenum internal-
format,
GLenum format, GLenum type, HTMLImageElement image);
void texImage2D(GLenum target, GLint level, GLenum internal-
format,
GLenum format, GLenum type, HTMLCanvasElement canvas);
```

```
void texImage2D(GLenum target, GLint level, GLenum internal-
format,
GLenum format, GLenum type, HTMLVideoElement video);
void texParameterf(GLenum target, GLenum pname, GLfloat
param);
void texParameteri(GLenum target, GLenum pname, GLint
param);
void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset,
GLsizei width, GLsizei height,
GLenum format, GLenum type, ArrayBufferView pixels);
void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset,
GLenum format, GLenum type, ImageData pixels);
void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset,
GLenum format, GLenum type, HTMLImageElement image);
void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset,
GLenum format, GLenum type, HTMLCanvasElement canvas);
void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset,
GLenum format, GLenum type, HTMLVideoElement video);
void uniform1f(WebGLUniformLocation location, GLfloat x);
void uniform1fv(WebGLUniformLocation location, Float32Array
v);
void uniform1fv(WebGLUniformLocation location, float[] v);
void uniform1i(WebGLUniformLocation location, GLint x);
void uniform1iv(WebGLUniformLocation location, Int32Array
v);
void uniform1iv(WebGLUniformLocation location, long[] v);
void uniform2f(WebGLUniformLocation location, GLfloat x,
GLfloat y);
void uniform2fv(WebGLUniformLocation location, Float32Array
v);
void uniform2fv(WebGLUniformLocation location, float[] v);
void uniform2i(WebGLUniformLocation location, GLint x, GLint
y);
void uniform2iv(WebGLUniformLocation location, Int32Array
v);
```

```
void uniform2iv(WebGLUniformLocation location, long[] v);
void uniform3f(WebGLUniformLocation location, GLfloat x,
GLfloat y, GLfloat z);
void uniform3fv(WebGLUniformLocation location, Float32Array
v);
void uniform3fv(WebGLUniformLocation location, float[] v);
void uniform3i(WebGLUniformLocation location, GLint x, GLint
y, GLint z);
void uniform3iv(WebGLUniformLocation location, Int32Array
v);
void uniform3iv(WebGLUniformLocation location, long[] v);
void uniform4f(WebGLUniformLocation location, GLfloat x,
GLfloat y, GLfloat z, GLfloat w);
void uniform4fv(WebGLUniformLocation location, Float32Array
v);
void uniform4fv(WebGLUniformLocation location, float[] v);
void uniform4i(WebGLUniformLocation location, GLint x, GLint
y, GLint z, GLint w);
void uniform4iv(WebGLUniformLocation location, Int32Array
v);
void uniform4iv(WebGLUniformLocation location, long[] v);
void uniformMatrix2fv(WebGLUniformLocation location, GLboo-
lean transpose,
Float32Array value);
void uniformMatrix2fv(WebGLUniformLocation location, GLboo-
lean transpose,
float[] value);
void uniformMatrix3fv(WebGLUniformLocation location, GLboo-
lean transpose,
Float32Array value);
void uniformMatrix3fv(WebGLUniformLocation location, GLboo-
lean transpose,
float[] value);
void uniformMatrix4fv(WebGLUniformLocation location, GLboo-
lean transpose,
Float32Array value);
void uniformMatrix4fv(WebGLUniformLocation location, GLboo-
lean transpose,
float[] value);
void useProgram(WebGLProgram program);
```

```
void validateProgram(WebGLProgram program);
void vertexAttrib1f(GLuint indx, GLfloat x);
void vertexAttrib1fv(GLuint indx, Float32Array values);
void vertexAttrib1fv(GLuint indx, float[] values);
void vertexAttrib2f(GLuint indx, GLfloat x, GLfloat y);
void vertexAttrib2fv(GLuint indx, Float32Array values);
void vertexAttrib2fv(GLuint indx, float[] values);
void vertexAttrib3f(GLuint indx, GLfloat x, GLfloat y,
GLfloat z);
void vertexAttrib3fv(GLuint indx, Float32Array values);
void vertexAttrib3fv(GLuint indx, float[] values);
void vertexAttrib4f(GLuint indx, GLfloat x, GLfloat y,
GLfloat z, GLfloat w);
void vertexAttrib4fv(GLuint indx, Float32Array values);
void vertexAttrib4fv(GLuint indx, float[] values);
void vertexAttribPointer(GLuint indx, GLint size, GLenum
type,
GLboolean normalized, GLsizei stride, GLintptr offset);
void viewport(GLint x, GLint y, GLsizei width, GLsizei
height);
}
```

5.13.1 属性

存在如下属性:

- canvas 类型为 HTMLCanvasElement，此环境创建的画板元素的引用
- drawingBufferWidth 类型为 GLsizei，绘图缓存的真实宽度。如果实现不能满足请求的宽度，则与 HTMLCanvasElement 的 width 属性不同。

- drawingBufferHeight 类型为 GLsizei，绘图缓存的真实高度。如果实现不能满足请求的高度，则与 HTMLCanvasElement 的 height 属性不同。

5.13.2 取得环境信息

- WebGLContextAttributes getContextAttributes() 返回现绘图缓存的 WebGLContextAttributes。

5.13.3 设置和取得状态

OpenGL ES 2.0 维护用于绘图的状态值。此组中的所有调用，如无声明，和 OpenGL 对应的调用行为相同。

```
void activeTexture(GLenum texture)
void blendEquation(GLenum mode)
void blendEquationSeparate(GLenum modeRGB, GLenum modeAlpha)
void blendFunc(GLenum sfactor, GLenum dfactor) 局限参见“常量颜色融合”
void blendFuncSeparate(GLenum srcRGB, GLenum dstRGB, GLenum srcAlpha, GLenum dstAlpha) 局限参见“常量颜色融合”
void clearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
void clearDepth(GLclampf depth) 深度值强制在0到1。
void clearStencil(GLint s)
```

```
void colorMask(GLboolean red, GLboolean green, GLboolean
blue, GLboolean alpha)
void cullFace(GLenum mode)
void depthFunc(GLenum func)
void depthMask(GLboolean flag)
void depthRange(GLclampf zNear, GLclampf zFar) zNear和zFar值
强制在0到1。zNear必须小于等于zFar。参见“视口深度范围”
void disable(GLenum cap)
void enable(GLenum cap)
void frontFace(GLenum mode)
any getParameter(GLenum pname) 返回传入pname的值。返回类型是请求
pname的自然类型，如下给出：

    pname               返回类型
    ACTIVE_TEXTURE      unsigned long
    ALIASED_LINE_WIDTH_RANGE  Float32Array (with 2 elements)
    ALIASED_POINT_SIZE_RANGE  Float32Array (with 2 elements)
    ALPHA_BITS          long
    ARRAY_BUFFER_BINDING      WebGLBuffer
    BLEND               boolean
    BLEND_COLOR   Float32Array (with 4 values)
    BLEND_DST_ALPHA     unsigned long
    BLEND_DST_RGB unsigned long
    BLEND_EQUATION_ALPHA      unsigned long
    BLEND_EQUATION_RGB  unsigned long
    BLEND_SRC_ALPHA     unsigned long
    BLEND_SRC_RGB       unsigned long
    BLUE_BITS           long
    COLOR_CLEAR_VALUE   Float32Array (with 4 values)
    COLOR_WRITEMASK     boolean[] (with 4 values)
    COMPRESSED_TEXTURE_FORMATS      null
    CULL_FACE           boolean
    CULL_FACE_MODE      unsigned long
    CURRENT_PROGRAM     WebGLProgram
    DEPTH_BITS          long
    DEPTH_CLEAR_VALUE   float
    DEPTH_FUNC          unsigned long
    DEPTH_RANGE         Float32Array (with 2 elements)
    DEPTH_TEST          boolean
    DEPTH_WRITEMASK     boolean
```

```
DITHER                          boolean
ELEMENT_ARRAY_BUFFER_BINDING    WebGLBuffer
FRAMEBUFFER_BINDING             WebGLFramebuffer
FRONT_FACE                      unsigned long
GENERATE_MIPMAP_HINT            unsigned long
GREEN_BITS                      long
LINE_WIDTH                      float
MAX_COMBINED_TEXTURE_IMAGE_UNITS      long
MAX_CUBE_MAP_TEXTURE_SIZE             long
MAX_FRAGMENT_UNIFORM_VECTORS          long
MAX_RENDERBUFFER_SIZE     long
MAX_TEXTURE_IMAGE_UNITS   long
MAX_TEXTURE_SIZE          long
MAX_VARYING_VECTORS       long
MAX_VERTEX_ATTRIBS        long
MAX_VERTEX_TEXTURE_IMAGE_UNITS        long
MAX_VERTEX_UNIFORM_VECTORS            long
MAX_VIEWPORT_DIMS         Int32Array (with 2 elements)
NUM_COMPRESSED_TEXTURE_FORMATS        long
PACK_ALIGNMENT            long
POLYGON_OFFSET_FACTOR     float
POLYGON_OFFSET_FILL       boolean
POLYGON_OFFSET_UNITS      float
RED_BITS                  long
RENDERBUFFER_BINDING      WebGLRenderbuffer
RENDERER                  DOMString
SAMPLE_BUFFERS            long
SAMPLE_COVERAGE_INVERT    boolean
SAMPLE_COVERAGE_VALUE     float
SAMPLES                         long
SCISSOR_BOX               Int32Array (with 4 elements)
SCISSOR_TEST                    boolean
SHADING_LANGUAGE_VERSION        DOMString
STENCIL_BACK_FAIL               unsigned long
STENCIL_BACK_FUNC               unsigned long
STENCIL_BACK_PASS_DEPTH_FAIL    unsigned long
STENCIL_BACK_PASS_DEPTH_PASS    unsigned long
STENCIL_BACK_REF                long
STENCIL_BACK_VALUE_MASK         unsigned long
STENCIL_BACK_WRITEMASK          unsigned long
```

```
    STENCIL_BITS                    long
    STENCIL_CLEAR_VALUE             long
    STENCIL_FAIL unsigned           long
    STENCIL_FUNC unsigned           long
    STENCIL_PASS_DEPTH_FAIL         unsigned long
    STENCIL_PASS_DEPTH_PASS         unsigned long
    STENCIL_REF                     long
    STENCIL_TEST                    boolean
    STENCIL_VALUE_MASK              unsigned long
    STENCIL_WRITEMASK               unsigned long
    SUBPIXEL_BITS                   long
    TEXTURE_BINDING_2D              WebGLTexture
    TEXTURE_BINDING_CUBE_MAP        WebGLTexture
    UNPACK_ALIGNMENT                int
    UNPACK_COLORSPACE_CONVERSION_WEBGL   unsigned long
    UNPACK_FLIP_Y_WEBGL             boolean
    UNPACK_PREMULTIPLY_ALPHA_WEBGL  boolean
    VENDOR                          DOMString
    VERSION                         DOMString
    VIEWPORT                 Int32Array (with 4 elements)

GLenum getError() 参见“WebGLContextEvent”
void hint(GLenum target, GLenum mode)
GLboolean isEnabled(GLenum cap)
void lineWidth(GLfloat width)
void pixelStorei(GLenum pname, GLint param) 参见“像素存储参数”
void polygonOffset(GLfloat factor, GLfloat units)
void sampleCoverage(GLclampf value, GLboolean invert)
void stencilFunc(GLenum func, GLint ref, GLuint mask)
void stencilFuncSeparate(GLenum face, GLenum func, GLint
ref, GLuint mask) 参见“模板分离掩码和参考值”
void stencilMask(GLuint mask) 参见“模板分离掩码和参考值”
void stencilMaskSeparate(GLenum face, GLuint mask)
void stencilOp(GLenum fail, GLenum zfail, GLenum zpass)
void stencilOpSeparate(GLenum face, GLenum fail, GLenum
zfail, GLenum zpass)
```

5.13.4 视图和裁剪

视口指定从归一的设备坐标系到窗口坐标系的 x 和 y 的仿射变换。绘图缓存的尺寸由 HTMLCanvasElement 决定。剪刀框定义一个包含绘图的方形。当剪刀测试启用时，只有剪刀框内的像素可以被绘图命令改变。启用后绘图只发生在视口，画板和剪刀框的交接处。如未启用剪刀测试， 绘图只发生在视口和画板的交接处。

```
void scissor(GLint x, GLint y, GLsizei width, GLsizei
height)
void viewport(GLint x, GLint y, GLsizei width, GLsizei
height)
```

5.13.5 缓存物件

缓存物件（有时称为 VBO）持有GLSL渲染用的顶点属性数据。

```
void bindBuffer(GLenum target, WebGLBuffer buffer)
void bufferData(GLenum target, GLsizeiptr size, GLenum us-
age)
void bufferData(GLenum target, ArrayBufferView data, GLenum
usage)
void bufferData(GLenum target, ArrayBuffer data, GLenum us-
age)
void bufferSubData(GLenum target, GLintptr offset, Array-
BufferView data)
void bufferSubData(GLenum target, GLintptr offset, Array-
Buffer data)
```

```
WebGLBuffer createBuffer() 类似OpenGL ES 2.0 glGenBuffers
void deleteBuffer(WebGLBuffer buffer) 类似OpenGL ES glDelete-
Buffers
any getBufferParameter(GLenum target, GLenum pname) 类似
OpenGL ES glGetBufferParameteriv。 返回传入pname的值。返回类型是
请求pname的自然类型，如下给出：

    pname          返回类型
    BUFFER_SIZE    long
    BUFFER_USAGE   unsigned long

GLboolean isBuffer(WebGLBuffer buffer)
```

5.13.6 帧存物件

帧存物件 （Framebuffer object） 提供除绘图缓存之外的另一绘画目标。它们是颜色，alpha，深度和模板缓存的集合，常用于绘制图像，作为后续的纹理。

```
void bindFramebuffer(GLenum target, WebGLFramebuffer frame-
buffer)

GLenum checkFramebufferStatus(GLenum target)

WebGLFramebuffer createFramebuffer() 类似OpenGL ES 2.0 glGen-
Framebuffers

void deleteFramebuffer(WebGLFramebuffer buffer) 类似OpenGL ES
2.0 glDeleteFramebuffers

void framebufferRenderbuffer(GLenum target, GLenum attach-
ment, GLenum renderbuffertarget, WebGLRenderbuffer render-
buffer)
```

```
void framebufferTexture2D(GLenum target, GLenum attachment,
GLenum textarget, WebGLTexture texture, GLint level)

any getFramebufferAttachmentParameter(GLenum target, GLenum
attachment, GLenum pname) 类似OpenGL ES 2.0 glGetFramebuffer-
AttachmentParameteriv
```

pname	返回类型
FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE	unsigned long
FRAMEBUFFER_ATTACHMENT_OBJECT_NAME	WebGLRenderbuffer 或 WebGLTexture
FRAMEBUFFER_ATTACHMENT_TEXTURE_LEVEL	long
FRAMEBUFFER_ATTACHMENT_TEXTURE_CUBE_MAP_FACE	long

```
GLboolean isFramebuffer(WebGLFramebuffer framebuffer)
```

5.13.7 绘存物件

绘存物件（Renderbuffer object）用于给帧存物件提供独立的存放空间。

```
void bindRenderbuffer(GLenum target, WebGLRenderbuffer ren-
derbuffer)

WebGLRenderbuffer createRenderbuffer()

void deleteRenderbuffer(WebGLRenderbuffer renderbuffer)

any getRenderbufferParameter(GLenum target, GLenum pname)
```

pname	返回类型
RENDERBUFFER_WIDTH	long

```
  RENDERBUFFER_HEIGHT               long
  RENDERBUFFER_INTERNAL_FORMAT      unsigned long
  RENDERBUFFER_RED_SIZE             long
  RENDERBUFFER_GREEN_SIZE           long
  RENDERBUFFER_BLUE_SIZE            long
  RENDERBUFFER_ALPHA_SIZE           long
  RENDERBUFFER_DEPTH_SIZE           long
  RENDERBUFFER_STENCIL_SIZE         long
```

```
GLboolean isRenderbuffer(WebGLRenderbuffer renderbuffer)
```

```
void renderbufferStorage(GLenum target, GLenum internalfor-
mat, GLsizei width, GLsizei height)
```

5.13.8 纹理物件

纹理物件 （Texture object）提供纹理操作的存储和状态。如果没有绑定 WebGLTexture （例如传递 null 或 0 给 bindTexture），则试图更改或查询纹理物件应产生 INVALID_OPERATION 错误。如下面函数指出的:

```
void bindTexture(GLenum target, WebGLTexture texture)
```

```
void copyTexImage2D(GLenum target, GLint level, GLenum in-
ternalformat, GLint x, GLint y, GLsizei width, GLsizei
height, GLint border)
```

```
void copyTexSubImage2D(GLenum target, GLint level, GLint
xoffset, GLint yoffset, GLint x, GLint y, GLsizei width,
GLsizei height)
```

```
WebGLTexture createTexture()

void deleteTexture(WebGLTexture texture)

void generateMipmap(GLenum target)

any getTexParameter(GLenum target, GLenum pname)

  pname                 返回类型
  TEXTURE_MAG_FILTER    unsigned long
  TEXTURE_MIN_FILTER    unsigned long
  TEXTURE_WRAP_S        unsigned long
  TEXTURE_WRAP_T        unsigned long

GLboolean isTexture(WebGLTexture texture)

void texImage2D(GLenum target, GLint level, GLenum internal-
format, GLsizei width, GLsizei height, GLint border, GLenum
format, GLenum type, ArrayBufferView pixels)

void texImage2D(GLenum target, GLint level, GLenum internal-
format, GLenum format, GLenum type, ImageData pixels)

void texImage2D(GLenum target, GLint level, GLenum internal-
format, GLenum format, GLenum type, HTMLImageElement image)

void texImage2D(GLenum target, GLint level, GLenum internal-
format, GLenum format, GLenum type, HTMLCanvasElement can-
vas)

void texImage2D(GLenum target, GLint level, GLenum internal-
format, GLenum format, GLenum type, HTMLVideoElement video)

void texParameterf(GLenum target, GLenum pname, GLfloat
param)

void texParameteri(GLenum target, GLenum pname, GLint param)
```

```
void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset, GLsizei width, GLsizei height, GLenum
format, GLenum type, ArrayBufferView pixels)

void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset, GLenum format, GLenum type, ImageData
pixels)

void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset, GLenum format, GLenum type, HTMLImageE-
lement image)

void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset, GLenum format, GLenum type, HTMLCanvasE-
lement canvas)

void texSubImage2D(GLenum target, GLint level, GLint xoff-
set, GLint yoffset, GLenum format, GLenum type, HTMLVi-
deoElement video)
```

5.13.9 程序和渲染

OpenGL ES 2.0 绘图需要使用 OpenGL ES 的渲染语言 GLSL ES 。渲染必须用一个源字串 (shaderSource) 加载，用 (compileShader) 编译，用 (attachShader) 挂接在一个程序上，必须链接 (linkProgram) 然后使用 (useProgram)。

```
void attachShader(WebGLProgram program, WebGLShader shader)

void bindAttribLocation(WebGLProgram program, GLuint index,
DOMString name)
```

```
void compileShader(WebGLShader shader)

WebGLProgram createProgram()

WebGLShader createShader(type)

void deleteProgram(WebGLProgram program)

void deleteShader(WebGLShader shader)

void detachShader(WebGLProgram program, WebGLShader shader)

WebGLShader[ ] getAttachedShaders(WebGLProgram program) 、

any getProgramParameter(WebGLProgram program, GLenum pname)
(OpenGL ES 2.0 §6.1.8, similar to man page)

     pname               返回类型
     DELETE_STATUS       boolean
     LINK_STATUS         boolean
     VALIDATE_STATUS     boolean
     ATTACHED_SHADERS    long
     ACTIVE_ATTRIBUTES   long
     ACTIVE_UNIFORMS     long

DOMString getProgramInfoLog(WebGLProgram program)

any getShaderParameter(WebGLShader shader, GLenum pname)



     pname               返回类型
     SHADER_TYPE         unsigned long
     DELETE_STATUS       boolean
     COMPILE_STATUS      boolean

DOMString getShaderInfoLog(WebGLShader shader)

DOMString getShaderSource(WebGLShader shader)
```

```
GLboolean isProgram(WebGLProgram program)

GLboolean isShader(WebGLShader shader)

void linkProgram(WebGLProgram program)

void shaderSource(WebGLShader shader, DOMString source)

void useProgram(WebGLProgram program)

void validateProgram(WebGLProgram program)
```

5.13.10 统一和属性

渲染使用的值作为统一（uniform） 或顶点属性 （attribute）。

```
void disableVertexAttribArray(GLuint index)

void enableVertexAttribArray(GLuint index)

WebGLActiveInfo getActiveUniform(WebGLProgram program, GLu-
int index)

GLint getAttribLocation(WebGLProgram program, DOMString
name)

any getUniform(WebGLProgram program, WebGLUniformLocation
location)
```

uniform类型	返回类型
boolean	boolean
int	long
float	float
vec2	Float32Array (with 2 elements)
ivec2	Int32Array (with 2 elements)

```
    bvec2         boolean[] (with 2 elements)
    vec3          Float32Array (with 3 elements)
    ivec3         Int32Array (with 3 elements)
    bvec3         boolean[] (with 3 elements)
    vec4          Float32Array (with 4 elements)
    ivec4         Int32Array (with 4 elements)
    bvec4         boolean[] (with 4 elements)
    mat2          Float32Array (with 4 elements)
    mat3          Float32Array (with 9 elements)
    mat4          Float32Array (with 16 elements)

WebGLUniformLocation getUniformLocation(WebGLProgram pro-
gram, DOMString name)

any getVertexAttrib(GLuint index, GLenum pname)

  pname                                  返回类型
  VERTEX_ATTRIB_ARRAY_BUFFER_BINDING     WebGLBuffer
  VERTEX_ATTRIB_ARRAY_ENABLED            boolean
  VERTEX_ATTRIB_ARRAY_SIZE               long
  VERTEX_ATTRIB_ARRAY_STRIDE             long
  VERTEX_ATTRIB_ARRAY_TYPE               unsigned long
  VERTEX_ATTRIB_ARRAY_NORMALIZED         boolean
  CURRENT_VERTEX_ATTRIB                  Float32Array (with
  4 elements)

GLsizeiptr getVertexAttribOffset(GLuint index, GLenum pname)

void uniform[1234][fi](WebGLUniformLocation location, ...)
void uniform[1234][fi]v(WebGLUniformLocation location, ...)
void uniformMatrix[234]fv(WebGLUniformLocation location,
GLboolean transpose, ...)

void vertexAttrib[1234]f(GLuint indx, ...)
void vertexAttrib[1234]fv(GLuint indx, ...)

void vertexAttribPointer(GLuint indx, GLint size, GLenum
type, GLboolean normalized, GLsizei stride, GLintptr offset)
```

5.13.11 写入绘图缓存

OpenGL ES 2.0 有3个调用可以在绘图缓存绘画: clear, drawArrays 和 drawElements。 后续的绘画可以作用于绘图缓存或一个帧存物件。当在绘图缓存绘画时，调用三者之一都应导致下一次复合操作时提交此绘图缓存给 HTML 页面复合器。

```
void clear(GLbitfield mask)

void drawArrays(GLenum mode, GLint first, GLsizei count)

void drawElements(GLenum mode, GLsizei count, GLenum type,
GLintptr offset)

void finish() (OpenGL ES 2.0 §5.1, man page)

void flush() (OpenGL ES 2.0 §5.1, man page)
```

5.13.12 读回像素

当前帧存的像素可以读回一个 ArrayBufferView 物件。

```
void readPixels(GLint x, GLint y, GLsizei width, GLsizei
height, GLenum format, GLenum type, ArrayBufferView pixels)
```

5.13.13 检查环境丢失事件

移动装置的电源事件等情况可能导致 WebGL 绘图环境随时丢失，并需要程序重建。细节参考 WebGLContextEvent。下面的方法可帮助发现丢失事件。

```
boolean isContextLost()
```

5.13.14 检查和启动扩展

在未使用扩展机制启用对应功能之前，WebGL实现不准支持多余的参数，常量和函数。getSupportedExtensions 函数返回一个此实现所支持的扩展的字串数组。扩展字串大小写无关。对此字串使用 getExtension 可启动对应扩展。此调用返回一个物件，包含所有此扩展定义的常量和函数。 此物件的定义由 扩展指定，并必须在扩展规范中定义。

一旦扩展启动，没有机制可以对其禁用。对 getExtension 使用相同的扩展字串的多次调用应返回同一物件。对未曾调用 getExtension 启动就使用扩展的任何企图，必须产生适当的 GL错误，并一定不能使用其功能。

本规范不定义任何扩展。单独的 WebGL extension registry 定义某特定的 webGL 实现所支持的扩展。

```
DOMString[ ] getSupportedExtensions()

object getExtension(DOMString name)
```

5.14 WebGLContextEvent

WebGL 产生一个 WebGLContextEvent 回应绘画环境的状态改变。对应的 HTMLCanvasElement 可有一个 Listener 响应此事件。事件由 DOM Event System 发送。 事件类型可包括状态的丢失和恢复，或不能创建环境等。

```
interface WebGLContextEvent : Event {
      readonly attribute DOMString statusMessage;
      void initWebGLContextEvent(DOMString typeArg,
      boolean canBubbleArg,
      boolean cancelableArg,
      DOMString statusMessageArg);
};
```

5.14.1 属性

现有属性为:

```
statusMessage of type DOMString
```

5.14.2 方法

现有方法为:

```
void initWebGLContextEvent(DOMString typeArg, boolean can-
BubbleArg, boolean cancelableArg, DOMString statusMes-
sageArg)
```

5.14.3 事件类型

```
webglcontextlost

webglcontextrestored

webglcontextcreationerror
```

示例4

下面的 ECMAScript 展示如何在 “canvas1”的画板上登记事件接收者，接收环境丢失和恢复事件，并重启程序。完整起见，它还展示如何让一个执行异步图像加载的程序，在任意时间处理环境丢失事件。

```
window.onload = init;
```

```
var g_gl;
var g_canvas;
var g_intervalId;
var g_images = [];
var g_imgURLs = [
 "someimage.jpg",
 "someotherimage.jpg",
 "yetanotherimage.png"
];

function init() {
  g_canvas = document.getElementById("canvas1");
  g_canvas.addEventListener("webglcontextlost", contextLost-
Handler, false);
  g_canvas.addEventListener("webglcontextrestored", contex-
tRestoredHandler, false);
  g_gl = canvas.getContext("webgl");

  for (var ii = 0; ii < g_imgURLs.length; ++ii) {
    // Create an image tag.
    var image = document.createElement('img');

    // Create a texture for this image.
    image.texture = g_gl.createTexture();

    // Mark the image as not loaded.
    image.loaded = false;

    // Setup a load callback.
    image.onload = (function(image) {
        return function() {
          imageLoadedHandler(image);
        };
      }(image));

    // Start the image loading.
    image.src = g_imgURLs[ii];

    // Remember the image.
```

```
    g_images.push(image);
  }

  g_intervalId = window.setInterval(renderHandler, 1000/60);
}

function renderHandler() {
  // draw with textures.
  // ...
}

function imageLoadedHandler(image) {
  // Mark the image as loaded.
  image.loaded = true;

  // Copy the image to the texture.
  updateTexture(image);
}

function updateTexture(image) {
  if (!g_gl.isContextLost() && image.loaded) {
    g_gl.bindTexture(g_gl.TEXTURE_2D, image.texture);
    g_gl.texImage2D(g_gl.TEXTURE_2D, 0, image);
  }
}

function contextLostHandler() {
  // stop rendering.
  window.clearInterval(g_internvalId);
}

function contextRestoredHandler() {
  // create new textures for all images and restore their
contents.
  for (var ii = 0; ii < g_images.length; ++ii) {
    g_images[ii].texture = g_gl.createTexture();
    updateTexture(g_images[ii]);
  }

  // Start rendering again.
```

```
  g_intervalId = window.setInterval(renderHandler, 1000/60);
}
```

6 WebGL和OpenGL ES 2.0的区别

此节列出 WebGL 对 OpenGL ES 2.0 API的改动，以用于改善不同操作系统和设备间的可移植性。

6.1 缓存物件绑定

WebGLAPI中，某个缓存物件在其生存期内只能绑定 ARRAY_BUFFER 或 ELEMENT_ARRAY_BUFFER 的绑定点之一。此限制指出某缓存物件只能包含顶点或下标，不能兼备。

在第一次作为 bindBuffer 的参量时一个 WebGLBuffer 被初始化。后续的 bindBuffer 如果试图绑定同一个 WebGLBuffer 到不同的绑定点，会产生 INVALID_OPERATION 错误，且绑定点的状态不变。

6.2 没有客户端数组

WebGLAPI不支持客户端数组。如果调用vertexAttribPointer时WebGLBuffer未绑定在ARRAY_BUFFER的绑定点，产生 INVALID_OPERATION 错误。如果调用 drawElements 时 count 大

于0，且没有 WebGLBuffer 绑定在 ELEMENT_ARRAY_BUFFER 的绑定点， 产生 INVALID_OPERATION 错误。

6.3 缓存偏移和跨度需求

drawElements和vertexAttribPointer的offset参量，以及vertexAttribPointer的stride参量，必须是传入调用的数据类型尺寸的整倍数，否则，产生 INVALID_OPERATION 错误。

6.4 启用顶点属性和范围检查

如果顶点属性作为enableVertexAttribArray的数组启用，但bindBuffer和 vertexAttribPointer没有属性绑定，则drawArrays 或 drawElements的调用产生 INVALID_OPERATION 错误。

如果顶点属性作为数组启用，又有缓存绑定在属性上，且此属性被当前程序使用，则drawArrays和drawElements的调用会确认每个引用的顶点在绑定缓存的存储内。如果drawArrays指定的范围或drawElements引用的下标不在绑定缓存的存储内，产生 INVALID_OPERATION 错误，且没有几何绘图。

如果顶点属性作为数组启用，又有缓存绑定在属性上，但此属性没有被当前程序使用，则无论绑定缓存的大小，皆不会在drawArrays或drawElements的调用时产生任何错误。

6.5 帧存物件挂接

WebGL把DEPTH_STENCIL_ATTACHMENT帧存物件挂接点和DEPTH_STENCIL绘存内部格式相加。要挂接深度和模板缓存两者到一个帧存物件，使用DEPTH_STENCIL绘存内部格式调用renderbufferStorage，再使用DEPTH_STENCIL_ATTACHMENT帧存物件挂接点调用framebufferRenderbuffer。

挂接在DEPTH_ATTACHMENT挂接点的绘存必须使用DEPTH_COMPONENT16内部格式分配。挂接在STENCIL_ATTACHMENT挂接点的绘存必须使用STENCIL_INDEX8内部格式分配。挂接在DEPTH_STENCIL_ATTACHMENT挂接点的绘存必须使用DEPTH_STENCIL内部格式分配。

WebGL API中，同时将挂接绘存到如下组合的挂接点会出错:

- DEPTH_ATTACHMENT + DEPTH_STENCIL_ATTACHMENT
- STENCIL_ATTACHMENT+DEPTH_STENCIL_ATTACHMENT

- DEPTH_ATTACHMENT + STENCIL_ATTACHMENT

如果违反了上述的限定，则:

- checkFramebufferStatus 必须返回FRAMEBUFFER_UNSUPPORTED
- 下述调用，无论是否更改或读取帧存，必须产生INVALID_FRAMEBUFFER_OPERATION 错误，且帧存，目标纹理和目标内存不得更改
  - clear
  - copyTexImage2D
  - copyTexSubImage2D
  - drawArrays
  - drawElements
  - readPixels

6.6 像素存储参数

WebGL API支持如下附加的pixelStorei 参数。

- UNPACK_FLIP_Y_WEBGL of type boolean，如真，则后续texImage2D和texSubImage2D的调用中，源数据随竖轴反转，因此，概念上最后一行首先传入。默认假。非零值表示真。
- UNPACK_PREMULTIPLY_ALPHA_WEBGL of type boolean 如真，后续texImage2D和texSubImage2D的调用中，源数据的alpha

信道如果存在，在数据传输时和颜色信道相乘。 默认假。非零值表示真。

- UNPACK_COLORSPACE_CONVERSION_WEBGL of type unsigned long 如设为BROWSER_DEFAULT_WEBGL，浏览器的默认色空间转换，在后续texImage2D和texSubImage2D的调用中应用HTMLImageElement。之前的转换可以特指为浏览器和文件类型两者。如为NONE，不应用色空间转换。默认为BROWSER_DEFAULT_WEBGL。

6.7 从帧存读出像素

WebGL API的读帧存函数（copyTexImage2D，copyTexSubImage2D和readPixels）定义为对帧存边界外的所有像素产生RGBA值(0,0,0,0)。

6.8 模板分离掩码和参考值

WebGL API中在模板操作时，对前后面的三角指定不同的掩码和参考值是非法的。如下的drawArrays 或 drawElements的调用会产生INVALID_OPERATION错误:

- STENCIL_WRITEMASK != STENCIL_BACK_WRITEMASK（分别由 stencilMaskSeperate 指定mask 参数的FRONT和BACK的face值）
- STENCIL_VALUE_MASK != STENCIL_BACK_VALUE_MASK （分别由 stencilFuncSeperate 指定mask 参数的FRONT和BACK的face值）
- STENCIL_REF != STENCIL_BACK_REF（分别由 stencilFuncSeperate 指定ref参数的FRONT和BACK的face值）

6.9 顶点属性数据跨度

WebGL API 支持顶点属性数据跨度到255字节。如果stride参数超过255，则vertexAttribPointer调用产生INVALID_VALUE错误。

6.10 视口深度范围

WebGL API 不支持近距平面的映射值大于远距平面的深度范围。如果zNear大于zFar，则depthRange调用产生INVALID_OPERATION错误。

6.11 常量颜色融合

WebGL API中，常量颜色和常量aplha不可在blend函数中作为源和目标因子一起使用。blendFunc调用会产生INVALID_OPERATION错误，如果两个因子之一设为CONSTANT_COLOR ONE_MINUS_CONSTANT_COLOR，而另一个为CONSTANT_ALPHA或ONE_MINUS_CONSTANT_ALPHA。blendFuncSeperate调用会产生INVALID_OPERATION错误，如果srcRGB设为CONSTANT_COLOR或ONE_MINUS_CONSTANT_COLOR，而dstRGB为CONSTANT_ALPHA 或 ONE_MINUS_CONSTANT_ALPHA，或反之。

6.12 定点数支持

WebGL不支持 GL_FIXED数据类型。

6.13 GLSL Constructs

根据 “支持的 GLSL Constructs”，以“webgl_” 和 “_webgl_”开始的标识被WebGL保留。

6.14 扩展查询

在OpenGL ES 2.0 API中，glGetString(GL_EXTENSIONS) 调用可以确定可用的扩展，它返回一列空格分隔的扩展字串。WebGL API的EXTENSIONS枚举被取消，代之以 getSupportedExtensions 调用确定可用的扩展。

6.15 实现的颜色读取格式与类型

在OpenGL ES 2.0 的API中，

IMPLEMENTATION_COLOR_READ_FORMAT和IMPLEMENTATION_COLOR_READ_TYPE参数用来告知应用程序，除所需的RGBA/UNSIGNED_BYTE对儿之外，还可传递给ReadPixels的格式与类型组合。WebGL 1.0中，ReadPixels支持的格式与类型组合在“读回像素”一节记录。

IMPLEMENTATION_COLOR_READ_FORMAT和IMPLEMENTATION_COLOR_READ_TYPE枚举被取消。

6.16 压缩纹理支持

WebGL API 不支持任何类型的纹理压缩。

因此，CompressedTexImage2D和CompressedTexSubImage2D不在此规范。并且，getParameter的COMPRESSED_TEXTURE_FORMATS和NUM_COMPRESSED_TEXTURE_FORMATS参数返回null或零值。

6.17 GLSL 标识名最长限制

GLSL ES 规范未定义最长标识名的限制。WebGL要求的支持为最少256个字符。

6.18 GLSL源字符集外的字符

GLSL ES 规范定义的OpenGL ES 渲染程序用的源字符集是ISO/IEC 646:1991，即ASCII。如果字串带有非此字符集外的字符，传递给渲染相关的入口bindAttribLocation,getAttribLocation, getUniformLocation, 或shaderSource时，产生INVALID_VALUE错误。例外情况是，所有允许在HTML DOMString的字符，可用作GLSL的注释。此用法必须不能导致错误。

非正式:

某些GLSL实现不允许ASCII之外的字符出现在注释中。WebGL实现需要防止此用法出错。推荐的技术是预处理GLSL字串，去掉所有注释，但通过添加新行字符，为除错保留行号。

6.19 字串长度查询

枚举INFO_LOG_LENGTH,SHADER_SOURCE_LENGTH,ACTIVE_UNIFORM_MAX_LENGTH和ACTIVE_ATTRIB_MAX_LENGTH已从WebGL API中删除。在OpenGL ES 2.0 API里，这些枚举用了决定传给glGetActiveAttrib这些调用的缓存大小。

WebGL API，类似的调用（getActiveAttrib，getActiveUniform，getProgramInfoLog，getShaderInfoLog，和getShaderSource）都返回DOMString。

7 引用

7.1 Normative references

[CANVAS]

HTML5: The Canvas Element, World Wide Web Consortium (W3C).

[TYPEDARRAYS]

Typed Array Specification: Editor's Draft, V. Vukicevic, K. Russell, May 2010.

[GLES20]

OpenGL® ES Common Profile Specification Version 2.0.25, A. Munshi, J. Leech, November 2010.

[GLES20GLSL]

The OpenGL® ES Shading Language Version 1.00, R. Simpson, May 2009.

[REGISTRY]

WebGL Extension Registry

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner. IETF, March 1997.

[WEBIDL]

Web IDL: W3C Editor’ s Draft, C. McCormack, September 2009.

[ASCII]

International Standard ISO/IEC 646:1991. Information technology - ISO 7-bit coded character set for information interchange

[DOMSTRING]

Document Object Model Core: The DOMString type, World Wide Web Consortium (W3C).

7.2 Other references

8 Acknowledgments

This specification is produced by the Khronos WebGL Working Group.

Special thanks to: Arun Ranganathan (Mozilla), Jon Leech, Kenneth Russell (Google), Kenneth Waters (Google), Mark Callow (HI), Mark Steele (Mozilla), Oliver Hunt (Apple), Tim Johansson (Opera), Vangelis Kokkevis (Google), Vladimir Vukicevic (Mozilla), Gregg Tavares (Google)

Additional thanks to: Alan Hudson (Yumetech), Bill Licea Kane (AMD), Cedric Vivier (Zegami), Dan Gessel (Apple), David Ligon (Qualcomm), Greg Ross (Nvidia), Jacob Strom (Ericsson), Kari Pulli (Nokia), Leddie Stenvie (ST-Ericsson), Neil Trevett (Nvidia), Per Wennersten (Ericsson), Per-Erik Brodin (Ericsson), Shiki Okasaka (Google), Tom Olson (ARM), Zhengrong Yao (Ericsson), and the members of the Khronos WebGL Working Group.