PowerVR Supported Extensions

OpenGL ES and EGL

Filename	:	PowerVR.Supported Extensions.OpenGL ES.EGL
Version	:	PowerVR SDK REL_3.2@2607465a External Issue
Issue Date	:	10 Sep 2013
Author	:	Imagination Technologies Limited

Contents

1. Introduction 4

2. EGL Extensions 5

2.1. EGL_IMG_context_priority 5
2.2. EGL_IMG_image_plane_attribs 6
2.3. EGL_KHR_create_context 7
2.4. EGL_KHR_fence_sync 9
2.5. EGL_KHR_gl_renderbuffer_image 10
2.6. EGL_KHR_gl_texture_2D_image 11
2.7. EGL_KHR_gl_texture_cubemap_image 12
2.8. EGL_KHR_image 13
2.9. EGL_KHR_image_base 14
2.10. EGL_KHR_image_pixmap 15

3. OpenGL ES Extensions 16

3.1. GL_APPLE_texture_2D_limited_npot 16
3.2. GL_EXT_blend_minmax 17
3.3. GL_EXT_color_buffer_float 18
3.4. GL_EXT_debug_marker 19
3.5. GL_EXT_discard_framebuffer 20
3.6. GL_EXT_draw_buffers 21
3.7. GL_EXT_multi_draw_arrays 22
3.8. GL_EXT_multisampled_render_to_texture 23
3.9. GL_EXT_occlusion_query_boolean 24
3.10. GL_EXT_shader_texture_lod 26
3.11. GL_EXT_texture_filter_anisotropic 27
3.12. GL_EXT_texture_format_BGRA8888 28
3.13. GL_EXT_texture_rg 29
3.14. GL_IMG_multisampled_render_to_texture 30
3.15. GL_IMG_program_binary 31
3.16. GL_IMG_read_format 32
3.17. GL_IMG_shader_binary 33
3.18. GL_IMG_texture_compression_pvrtc 34
3.19. GL_IMG_texture_compression_pvrtc2 35
3.20. GL_IMG_texture_format_BGRA8888 36
3.21. GL_IMG_texture_npot 37
3.22. GL_OES_blend_equation_separate 38
3.23. GL_OES_blend_func_separate 39
3.24. GL_OES_blend_subtract 40
3.25. GL_OES_byte_coordinates 41
3.26. GL_OES_compressed_ETC1_RGB8_texture 42
3.27. GL_OES_compressed_paletted_texture 43
3.28. GL_OES_depth_texture 44
3.29. GL_OES_depth24 45
3.30. GL_OES_draw_texture 46
3.31. GL_OES_egl_image 47
3.32. GL_OES_egl_image_external 48
3.33. GL_OES_egl_sync 49
3.34. GL_OES_element_index_uint 51
3.35. GL_OES_extended_matrix_palette 52
3.36. GL_OES_fixed_point 53
3.37. GL_OES_fragment_precision_high 54
3.38. GL_OES_framebuffer_object 55
3.39. GL_OES_get_program_binary 57
3.40. GL_OES_mapbuffer 58
3.41. GL_OES_matrix_get 59
3.42. GL_OES_matrix_palette 60

3.43. GL_OES_packed_depth_stencil 61
3.44. GL_OES_point_size_array 62
3.45. GL_OES_point_sprite 63
3.46. GL_OES_query_matrix 64
3.47. GL_OES_read_format 65
3.48. GL_OES_required_internalformat 67
3.49. GL_OES_rgb8_rgba8 68
3.50. GL_OES_single_precision 69
3.51. GL_OES_standard_derivatives 70
3.52. GL_OES_stencil_wrap 71
3.53. GL_OES_stencil8 72
3.54. GL_OES_texture_cube_map 73
3.55. GL_OES_texture_env_crossbar 74
3.56. GL_OES_texture_float 75
3.57. GL_OES_texture_half_float 76
3.58. GL_OES_texture_mirrored_repeat 77
3.59. GL_OES_texture_npot 78
3.60. GL_OES_vertex_array_object 79
3.61. GL_OES_vertex_half_float 80

4. Contact Details 81

List of Figures

No table of figures entries found.

1. Introduction

Document Overview

OpenGL extensions are a means for OpenGL implementations to provide new or expanded functionality that the core of OpenGL does not provide. Using extensions should not be looked on as something to be avoided; it should be accepted as standard practice for the OpenGL user. The purpose of this document is to serve as a reference for all of the extensions supported by Imagination's reference driver implementation.

Note

While Imagination's reference driver implementation supports all of the extensions in this document it is the decision of our licensees as to which are exposed in their own implementations.

2. EGL Extensions

2.1. EGL_IMG_context_priority

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension allows a developer to set the execution priority of an EGL context, by setting a new attribute when creating the context: EGL_CONTEXT_PRIORITY_LEVEL_IMG. This can be useful in a multi-process environment, where an API-utilising task can be considered lower or higher priority.

Three priority levels are described in this extension, EGL_CONTEXT_PRIORITY_HIGH_IMG, EGL_CONTEXT_PRIORITY_MEDIUM_IMG and EGL_CONTEXT_PRIORITY_LOW_IMG, meaning high, medium and low priority, respectively. By default, all contexts are created with medium priority, and so are on equal execution footing. The specification of the priority is only a hint though, and so can be ignored by the implementation - this is typical when, for example, high priority contexts are reserved for system processes.

A query is provided to obtain the real priority of the client after it has been created. Note that currently, all implementations of this extension set a priority per process, no matter how many contexts are in that process. To set multiple priorities, multiple processes must be used, and EGLImages typically used for sharing.

Registry Link

http://www.khronos.org/registry/egl/extensions/IMG/EGL_IMG_context_priority.txt

Example Code

```
// Set up the context attributes, specifying OpenGL ES 3.0 support and a low priority context.
EGLint contextAttribs[] =
{
	EGL_CONTEXT_CLIENT_VERSION, 3,
	EGL_CONTEXT_PRIORITY_LEVEL_IMG, EGL_CONTEXT_PRIORITY_LOW_IMG,
	EGL_NONE
};

// Create the context with the context attributes supplied
eglContext = eglCreateContext(eglDisplay, eglConfig, NULL, contextAttribs);
```

2.2. EGL_IMG_image_plane_attribs

Supported Hardware

Series5XT, Series6

Extension Description

This extension is specific to Android, and allows the creation of EGLImages from individual planes of a multi-planar Android native image buffer (i.e. "ANativeWindowBuffer"). As such, EGL_KHR_image_base and EGL_ANDROID_image_native_buffer are required. Multi-planar surfaces typically include YUV images or video data. The idea of splitting the planes of an image like this allow a user to process multi-planar data in an intuitive way, without worrying about how data is interleaved or laid out in memory too much after the initial definition. Once the EGLImage is created, the user can access the data from a client API (e.g. using GL_OES_egl_image) in the same way they would any other EGLImage, and treat the channels as simple values. It is then the application developer's responsibility to handle any colour space conversions in the client APIs.

Example Code

```
// Setup plane attributes that get the first plane from a native Android object as an
// EGLImageKHR. For YUV Images this is the "Y" plane.
EGLint imageAttribsPlanes[] =
{
        EGL_NATIVE_BUFFER_MULTIPLANE_SEPARATE_IMG, EGL_TRUE,
        EGL_NATIVE_BUFFER_PLANE_OFFSET_IMG, 0,
        EGL_NONE
};

// Create an EGLImageKHR from the first plane of a native Android Image object. EGL_NO_CONTEXT
// must always be specified, as Android native images are not supported by an EGLContext.
EGLImageKHR eglImageFirstPlane = eglCreateImageKHR(eglDisplay, EGL_NO_CONTEXT,
EGL_NATIVE_BUFFER_ANDROID, (EGLClientBuffer)aNativeWindowBuffer, imageAttribsPlanes);

// Change the attributes to get an EGLImageKHR from the second plane of the image. For YUV
// Images this is the "UV" plane.
imageAttribsPlanes[3] = 1;

// Create an EGLImageKHR from the second plane of a native Android Image object.
// EGL_NO_CONTEXT must always be specified, as Android native images are not supported by an
// EGLContext.
EGLImageKHR eglImageSecondPlane = eglCreateImageKHR(eglDisplay, EGL_NO_CONTEXT,
EGL_NATIVE_BUFFER_ANDROID, (EGLClientBuffer)aNativeWindowBuffer, imageAttribsPlanes);
```

2.3. EGL_KHR_create_context

Supported Hardware

Series6

Extension Description

This extension enables a number of new context attributes to be specified in EGL to allow greater control of the returned context. The new functionality consists of four aspects; version control, core vs compatibility layer control for OpenGL, debugging (see extension GL_KHR_debug), and robustness in OpenGL (see desktop extension "GL_ARB_robustness"). This extension mirrors the desktop WGL_ARB_create_context and GLX_ARB_create_context extensions, and provides similar functionality through EGL.

Version Control

This part of the extension enables developers to explicitly create a context with a given level of API support. An explicitly defined EGL_OPENGL_ES3_BIT is provided to query the new API via the original mechanism to make sure a compatible context is available, and then this extension allows developers to explicitly state which version of OpenGL ES they are actually going to be using. The two additional attributes to specify this are EGL_CONTEXT_MAJOR_VERSION_KHR (which is an alias for EGL_CONTEXT_CLIENT_VERSION) and EGL_CONTEXT_MINOR_VERSION_KHR. Each then accepts an integer value specifying what version you require.

OpenGL Core vs. Compatibility

EGL_OPENGL_PROFILE_MASK_KHR allows a developer to specify EGL_CONTEXT_OPENGL_CORE_PROFILE_BIT_KHR or EGL_CONTEXT_OPENGL_COMPATIBILITY_PROFILE_BIT_KHR to explicitly choose a core or compatibility context when using the OpenGL client API.

There is also a bit that can be set for EGL_CONTEXT_FLAGS_KHR which tells the implementation to return only OpenGL contexts which do not support functionality marked as deprecated by any version of OpenGL after OpenGL 3.0. Further information on this is available in the OpenGL specification.

Debug Contexts

When the GL_KHR_debug extension exists in a client OpenGL/ES API, this bit flag tells the implementation to enable debug functionality for that API. This is necessary to inform the underlying implementation that debug information should be tracked. This bit should only be set for purposes of debugging, and not in shipping code.

OpenGL Robustness

When setting EGL_CONTEXT_FLAGS_KHR, the EGL_CONTEXT_OPENGL_ROBUST_ACCESS_BIT_KHR bit is provided to allow a user to require buffer access to remain robust as defined in the GL_ARB_robustness extension.

Various problems in either hardware or software can occasionally cause a hardware reset in a GPU. Unlike a CPU, these can typically be recovered from without having to restart the device - though setting up the underlying context again is usually a necessity. Setting EGL_CONTEXT_OPENGL_RESET_NOTIFICATION_STRATEGY_KHR to either EGL_NO_RESET_NOTIFICATION_KHR or EGL_LOSE_CONTEXT_ON_RESET_KHR will set the behaviour to either notify the user or ignore it, as defined by GL_ARB_robustness using GL_NO_RESET_NOTIFICATION_ARB and GL_LOSE_CONTEXT_ON_RESET_ARB respectively.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_create_context.txt

Example Code

```
// Set up the config attributes, specifying that we want a config with Window support and ES3
// compatibility.
EGLint configAttribs[] =
{
        EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
        EGL_RENDERABLE_TYPE , EGL_OPENGL_ES3_BIT_KHR,
        EGL_NONE
};

// Choose an appropriate configuration - just get the first available one that matches here
EGLint iConfigs;
EGLConfig eglConfig;
eglChooseConfig(eglDisplay, configAttribs, &eglConfig, 1, &iConfigs);

// Set up the context attributes, specifying OpenGL ES 3.0 support and a debug context.
EGLint contextAttribs[] =
{
        EGL_CONTEXT_MAJOR_CLIENT_VERSION, 3,
        EGL CONTEXT MINOR CLIENT VERSION, 0,
        EGL CONTEXT FLAGS KHR, EGL CONTEXT OPENGL DEBUG BIT KHR,
        EGL_NONE
};

// Create the context with the context attributes supplied
eglContext = eglCreateContext(eglDisplay, eglConfig, NULL, contextAttribs);
```

2.4. EGL_KHR_fence_sync

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension introduces sync objects to EGL, and provides application developers with a mechanism of notifying the CPU when a GPU operation has completed. Fence sync objects are inserted into the GL command stream immediately after the call which the user wants to be notified about or wait on. Any thread can then make a call to eglClientWaitSync() to wait for the GPU to finish what it's doing, with a user specified timeout. A timeout of 0 equates to simply querying whether it's complete yet, similar to how queries work in later versions of OpenGL ES. A common use case for this is to be loading resources on one thread, and informing the CPU when it can start submitting calls on another thread that can actually draw with them. The benefits of such multi-threaded rendering can allow developers to create games with seamless loading. To actually interact with client APIs such as OpenGL ES, the client API needs to have an extension that specifies it will be inserted into the same command stream. For OpenGL ES this is GL_OES_EGL_sync.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/EGL_KHR_fence_sync.txt

Example Code

```
// Create a fence sync object - there are no valid attributes that can be passed in so pass
// NULL or an attribute list containing only EGL NONE.
EGLSyncKHR eglFenceSync = eglCreateSyncKHR(eglDisplay, EGL SYNC FENCE KHR, NULL);

/*  Wait for the sync object, flushing the context so it definitely completes in finite time.
    This call will wait forever until the context finishes what it's doing.
    This also works as a more flexible and guaranteed version of glFlush/glFinish, as you can
    specify a timeout.
*/
EGLint waitResult = eglClientWaitSyncKHR(eglDisplay, eglFenceSync,
EGL_SYNCH_FLUSH_COMMANDS_BIT_KHR, EGL_FOREVER_KHR);

// Destroy the fence sync object once we're done with it.
EGLBoolean success = eglDestroySyncKHR(eglDisplay, eglFenceSync);
```

2.5. EGL_KHR_gl_renderbuffer_image

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension extends EGL_KHR_image_base, and enables EGLImages to be created from a source renderbuffer from OpenGL. This particular extension string denotes that an EGLImage can be created from a renderbuffer. This extension is also required to create an OpenGL renderbuffer from an EGLImage when GL_KHR_image is supported in the implementation.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_gl_image.txt

Example Code

```
// Create an EGLImageKHR from an OpenGL ES Renderbuffer
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglOpenGLESContext,
EGL_GL_RENDERBUFFER_KHR, (EGLClientBuffer)anOpenGLRenderbuffer, NULL);
```

2.6. EGL_KHR_gl_texture_2D_image

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension extends EGL_KHR_image_base, and enables EGLImages to be created from a source texture from OpenGL. This particular extension string denotes that an EGLImage can be created from a basic 2D texture. This extension is also required to create an OpenGL texture from an EGLImage when GL_KHR_image is supported in the implementation.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_gl_image.txt

Example Code

```
// Attributes that tell the CreateImage command to use the first level of the texture
// (NB: This is the default and doesn't actually need to be specified, it's just for
// illustration.)
EGLint imageAttributes[] =
{
        EGL GL TEXTURE LEVEL,  0,
        EGL NONE
};

// Create an EGLImageKHR from an OpenGL ES Texture. The context which contains the OpenGL ES
// texture must be specified, as OpenGL ES is supported by an EGLContext.
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglOpenGLESContext,
EGL_GL_TEXTURE_2D_KHR, (EGLClientBuffer)anOpenGLTexture, imageAttributes);
```

2.7. EGL_KHR_gl_texture_cubemap_image

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension extends EGL_KHR_image_base, and enables EGLImages to be created from a source texture from OpenGL. This particular extension string denotes that an EGLImage can be created from a cubemap texture. This extension is also required to create an OpenGL texture from an EGLImage when GL_KHR_image is supported in the implementation.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_gl_image.txt

Example Code

```
// Attributes that tell the CreateImage command to use the first level of the texture
// (NB: This is the default and doesn't actually need to be specified, it's just for
// illustration.)
EGLint imageAttributes[] =
{
        EGL GL TEXTURE LEVEL,  0,
        EGL NONE
};

// Create an EGLImageKHR from an OpenGL ES Cubemap Texture. The target in this case specifies
// that the image is created from the positive X face of the cubemap, as EGLImages are 2D
only.
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglOpenGLESContext,
EGL_GL_TEXTURE_CUBE_MAP_POSITIVE_X_KHR , (EGLClientBuffer)anOpenGLCubeMap, imageAttributes);
```

2.8. EGL_KHR_image

Supported Hardware

Series5/5XT, Series6

Extension Description

This was originally the extension which defined EGLImages, but pixmaps were core to this extension's operation. Subsequent implementations have wanted to support EGLImages without having pixmap support, and so this has been split into two extensions. EGL_KHR_image_base and EGL_KHR_image_pixmap are the two child extensions which when supported together are the equivalent of EGL_KHR_image.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_image.txt

Example Code

```
// Create an EGLImageKHR from an EGL Pixmap
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglOpenGLESContext,
EGL_NATIVE_PIXMAP_KHR, (EGLClientBuffer)anEGLPixmap, NULL);
```

2.9. EGL_KHR_image_base

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension introduces image objects to EGL, which are a way of defining a somewhat generic object which stores something that may be considered an array of image data; textures, renderbuffers, pixmaps and the like. This extension defines what an EGLImage is to a developer, and makes no assumptions about the underlying data - internally data is stored in a format that is friendly to whatever client APIs are supported by a given implementation. This extension also does not define any source targets for creating an EGLImage, this extension on its own is of little use beyond defining the object itself and the interface. Extensions which add functionality to create EGLImages from various sources are listed below:

- EGL
  - EGL_KHR_image_pixmap
- OpenGL/ES
  - EGL_KHR_gl_texture_2D_image
  - EGL_KHR_gl_texture_cubemap_image
  - EGL_KHR_gl_texture_3D_image
  - EGL_KHR_gl_renderbuffer_image
- OpenCL
  - EGL_IMG_cl_image
- Android
  - EGL_ANDROID_image_native_buffer
- Linux
  - EGL_EXT_image_dma_buf_import
- OpenVG
  - EGL_KHR_vg_parent_image

Other various client extensions enable the creation of a client object from an EGLImage, such as GL_KHR_EGL_image.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_image_base.txt

Example Code

```
// Create an EGLImageKHR from an EGL Pixmap
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglOpenGLESContext,
EGL_NATIVE_PIXMAP_KHR, (EGLClientBuffer)anEGLPixmap, NULL);
```

2.10. EGL_KHR_image_pixmap

Supported Hardware

Series5/5XT, Series6

Extension Description

This extension extends EGL_KHR_image_base, and enables the creation of an EGLImage an EGLNativePixmapType.

Note: This extension is not explicitly exposed on PowerVR hardware, but is implied as supported whenever EGL_KHR_image is present, as it is a compound extension which includes both this and EGL_KHR_image_base.

Registry Link

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_image_pixmap.txt

Example Code

```
// Create an EGLImageKHR from an EGL Pixmap
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglOpenGLESContext,
EGL_NATIVE_PIXMAP_KHR, (EGLClientBuffer)anEGLPixmap, NULL);
```

3. OpenGL ES Extensions

3.1. GL_APPLE_texture_2D_limited_npot

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension adds limited support for non-power-of-two (POT) textures in OpenGL ES 1.x. specifically; users are now able to create 2D textures with NPOT dimensions. This relaxation does not apply to cubemaps or 3D textures. There are also several limitations:

- Texture Wrap Modes
  - Only GL_CLAMP_TO_EDGE may be used as a wrap mode when sampling from an NPOT texture.
- MIPMapping
  - MIP Mapping is not supported.
  - Only minification filters GL_NEAREST or GL_LINEAR are allowed.
  - glGenerateMIPMap does not work.

Most of these restrictions are lifted if GL_OES_texture_npot is present.

Note

This functionality is core to OpenGL ES 2.0 and 3.0. All limitations on non-power of two textures are lifted for OpenGL ES 3.0

Example Code

```
// Upload a texture with dimensions of 15 by 47. Typically this isn't supported without
// extension support.
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 15, 47, 0, GL_RGBA, GL_UNSIGNED_BYTE, pixelData);

// Only the following parameters are specifiable, anything else is invalid (including the
// default texture parameters!).
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

3.2. GL_EXT_blend_minmax

Supported Hardware

Series 5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

This extension adds two additional blending modes to OpenGL ES; GL_MIN_EXT and GL_MAX_EXT. These two blend modes compare the source and destination colours, and the result of the comparison will be the minimum/maximum colour that was written. Each component is operated on separately, so that it is entirely possible to have an output colour that has channel values from multiple different polygons.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/blend_minmax.txt

Example Code

```
// Set the blend equation to use additive blending
glBlendEquation(GL_FUNC_ADD_EXT);

// Set the blend equation to use minimum value blending
glBlendEquation(GL_MIN_EXT);

// Set the blend equation to use maxmum value blending
glBlendEquation(GL_MAX_EXT);
```

3.3. GL_EXT_color_buffer_float

Supported Hardware

Series6

Valid APIs

OpenGL ES 3.0

Extension Description

Rendering to a floating point buffer has been a core part of the desktop GL specification for a long time now, but was not included in the OpenGL ES 3.0 specification. This extension adds that functionality to ES3.0 class hardware, allowing users to output floating point values from fragments, enabling things like HDR rendering and advanced post-processing techniques that need a higher range or precision than fixed point values provide.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_color_buffer_float.txt

Example Code

```
// Create a floating point colour buffer (texture)
GLint floatingPointColourBuffer = glTexStorage(GL TEXTURE 2D, 1, GL RGBA32F, 1024, 1024);

// Attach it to a framebuffer. This would usually result in an incomplete framebuffer without
// this extension.
glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
floatingPointColourBuffer, 0);
```

3.4. GL_EXT_debug_marker

Supported Hardware

Series5XT

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension enables developers to insert debug markers into the command stream, which are essentially a way of annotating code in GL tracing tools such as PVRTrace. There are no other benefits to this extension beyond this, as runtime markers could be introduced by an application developer themselves if they needed a way to label parts of their code.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_debug_marker.txt

Example Code

```
// Insert a regular event marker, used effectively as a simple string annotation to your code.
glInsertEventMarkerEXT(0, "This is a debug marker annotation");

// Signal that any following functions belong to a nested functionality group. For instance,
this might denote the start of a function which renders a specific object.
glPushGroupMarkerEXT(0, "This is a group of functionality.");

// End the previously defined functionality group
glPopGroupMarkerEXT();
```

3.5. GL_EXT_discard_framebuffer

Supported Hardware

Series 5/5XT and Series6 (ES2/3 only)

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

Generally, OpenGL ES is expected to have the result of a render stored into memory so that it can be used for later computation. If the storage memory is also the memory that is being rendered to then this isn't a huge problem. However, in cases where the result of a render is stored in a temporary buffer to be written outer later, bandwidth is wasted in copying this data out if it isn't going to be used. This extension provides a mechanism for users to explicitly state that they don't need to store part of the render output, allowing the underlying implementation to avoid the costly data copy, and potentially improving performance.

Note

This functionality is core to OpenGL ES 3.0 however a different function,"glInvalidateFramebuffer", is used instead. This function works identically to glDiscardFramebufferEXT, but was renamed and re-specified to be brought in line with other OpenGL functionality.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_discard_framebuffer.txt

Example Code

```
// Finish rendering to a particular framebuffer
glDraw(...);

// Specify attachments to discard, typically depth and stencil are discarded in this way.
GLenum discardAttachments[] =
{
        GL_DEPTH_ATTACHMENT,
        GL_STENCIL_ATTACHMENT
};

// Discard the framebuffer's contents which we aren't interested in
glDiscardFramebufferEXT(GL_FRAMEBUFFER, sizeof(discardAttachments)/sizeof(GLenum),
discardAttachments);

// The next call should always be a framebuffer change, nothing should occur between this and
// the discard.
glBindFramebuffer(0);

// Clearing will prevent any data being written back to on-chip memory as well.
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
```

3.6. GL_EXT_draw_buffers

Supported Hardware

Series5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

Typically only one colour output is enabled by default in OpenGL ES 2.0, and whilst querying for additional colour buffers has exists, the base OpenGL ES 2.0 specification does not allow for additional colour buffers. This extension adds language and functionality that enable up to a maximum of 16 total colour buffers per framebuffer object, though the actual available number varies by platform. The main benefit of this extension is to reduce the number of draw passes in a scene, as a number of techniques need more than 4 channels of data to output for a given scene, usually for further calculation. Without additional colour buffers, developers are forced to render the same data multiple times, or sacrifice precision by packing data.

This functionality is also more commonly known as "Multiple Render Targets" or MRTs.

Note

This functionality is core to OpenGL ES 3.0.

Example Code

```
// Attach a texture to a framebuffer, using a colour attachment beyond 0.
glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_COLOR_ATTACHMENT1, GL_TEXTURE_2D,
aTextureColourBuffer, 0);

// The writable draw buffers also need to be set separately. In this instance there are two
// colour buffers. Create a list and allow them to be rendered to.
GLenum buffers[2] = {GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1};
glDrawBuffers(2, buffers);
```

3.7. GL_EXT_multi_draw_arrays

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension adds two new functions to OpenGL ES; glMultiDrawArrays and glMultiDrawElements. These functions perform the same tasks as glDrawArrays/Elements, but also allow the users to render multiple primitive groups in one function call. In other words, one "glMultiDrawArrays" is equivalent to multiple "glDrawArrays" with the same rendering state.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/multi_draw_arrays.txt

Example Code

```
// Setup lists of buffer offsets and triangle counts
GLint firsts[] = {0, 32};
GLsizei counts[] = {32, 32};

// If multiple vertex objects share the same draw state (blending, depth testing, shaders,
// textures, etc.) then multi-draw can be called on them all at once, rather than calling
// glDraw multiple times.
glMultiDrawArrays(GL_TRIANGLES, firsts, counts, 2);
```

3.8. GL_EXT_multisampled_render_to_texture

Supported Hardware

Series 5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension adds multisampled rendering capabilities to OpenGLES, allowing a user to create multisampled render buffers, and attach textures to a framebuffer in a way that they can be used for multisampled renders. By enabling textures to be attached as multisampled but without using actual multisample textures, a significant amount of bandwidth can be saved. This mode of operation enables multisampled anti-aliasing to smooth lines at the edges of polygons and reduce the appearance of jagged lines. This is functionally equivalent to GL_IMG_multisampled_render_to_texture on PowerVR hardware.

Note

Part of this functionality is core to OpenGL ES 3.0. The renderbuffer functionality is present in core; however the multisampled texture attachment is not.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_multisampled_render_to_texture.txt

Example Code

```
// Create a multisampled depth renderbuffer
GLint depthBuffer = glRenderbufferStorageMultisampleEXT(GL RENDERBUFFER, 4,
GL DEPTH COMPONENT16, 1024, 1024);

// Attach the renderbuffer to the framebuffer as per usual. Note: If one attachment is
// multi-sampled, all attachments need to be. If there is a mix, the framebuffer will be
// incomplete.
glFramebufferRenderbuffer(GL FRAMEBUFFER, GL DEPTH ATTACHMENT, GL RENDERBUFFER, depthBuffer);

// Create a regular colour buffer (texture). This extension doesn't introduce multi-sample
// textures, instead attaching regular textures that multi-sampling will resolve to before
// writing out.
GLint colourBuffer = glTexStorage(GL TEXTURE 2D, 1, GL RGBA8, 1024, 1024);

// Attach it to a framebuffer as multisampled, so that OpenGL ES knows that it should resolve
// a multisampled render to this texture.
glFramebufferTexture2DMultisampleEXT(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
colourBuffer, 0);
```

3.9. GL_EXT_occlusion_query_boolean

Supported Hardware

Series 5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

This extension adds two new mechanisms to OpenGL ES 2.0 - Querying and Boolean Occlusion Queries.

Querying is a mechanism which allows the user to request information from the hardware directly, often allowing access to information that the hardware can gather with easily that would otherwise require expensive software checks. As modern GPUs work asynchronously to the CPU, querying does not instantly return a value; instead, a result is requested, and then at some point in the future the GL implementation will have a response ready. App developers can check when that query is ready, and then access that value.

Boolean occlusion queries use the querying mechanism to allow app developers to find out if an object they've been sending down the pipeline is actually visible or not. The result of the query can then be used in subsequent frames to determine whether or not to draw the object again. This allows a developer to better balance the load between software culling and hardware visibility tests.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_occlusion_query_boolean.txt

Example Code

```
// An occlusion query object to use.
GLuint occlusionQueryObject = 0;

void Render()
{
        if (occlusionQueryObject == 0)
        {
                // Generate a new occlusion query object
                glGenQueriesEXT(1, &occlusionQueryObject);

                // Begin the query
                glBeginQueryEXT(ANY SAMPLES PASSED EXT, occlusionQueryObject);

                // Submit the geometry you'd like to submit for occlusion testing - typically#
                // a bounding box of the actual geometry
                glDraw(...)

                // Signal that the query has ended
                glEndQueryEXT(occlusionQueryObject);
        }
        // Check if the query has finished or not
        GLuint resultAvailable = 0, result = 0;
        glGetQueryObjectuivEXT(occlusionQueryObject, GL QUERY RESULT AVAILABLE,
       &resultAvailable);

        if (resultAvailable == GL_TRUE)
        {
                glGetQueryObjectuivEXT(occlusionQueryObject, GL_QUERY_RESULT, &result);

                // Delete the occlusion query object and reset it to zero to signal that a new
                // query can be started.
                glDeleteQueries(1, &occlusionQueryObject);
                occlusionQueryObject = 0;
        }

        // If the result isn't yet available, or if the query indicates that the sample
       // geometry IS visible, render the object.
        if (resultAvailable == GL_FALSE || result >= 1)
        {
                glDraw(...);
        }
}
```

3.10. GL_EXT_shader_texture_lod

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

This extension allows the user to manually select a level of detail from the fragment shader, rather than leaving it up to the implementation. Vanilla OpenGL ES 2.0 allows the user to add a level of detail bias to the MIP Map selection, but only allows explicit selection in the vertex shader. By exposing this in the fragment shader too, it means that custom level of details can be achieved - allowing for example a simple blurring effect that can be used to simulate focussing on different parts of the scene.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_shader_texture_lod.txt

Example Code (GLSL)

```
// Samples a texture explicitly from the 5th MIP Map level
lowp vec4 colour = texture2DLodEXT(aSampler, texCoords.xy, 4.0);
```

3.11. GL_EXT_texture_filter_anisotropic

Supported Hardware

Series6

Valid APIs

OpenGL ES 2.0 3

Extension Description

This extension enables anisotropic texture filtering, which is a method that can improve the quality of textures rendered at an oblique angle to the camera. Typically, texture filtering is performed isotropically, which means that the filtering is performed the same way across each axis (e.g. x and y). By effectively allowing the filter to vary with each axis independently, better results can be obtained when the scaling on each axis varies from its counterpart. Anisotropic filtering is a more computationally intensive method of filtering, but can come with significant quality improvements in some cases and should be used accordingly.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/texture_filter_anisotropic.txt

Example Code

```
// Enable anisotropic filtering with a maximum anisotropy of 4
glTexParameterf(GL_TEXTURE_2D, GL_MAX_ANISOTROPY, 4.0);
```

3.12. GL_EXT_texture_format_BGRA8888

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

Typically, textures in OpenGL ES run in RGBA order; this extension adds BGRA8888 as a new format so that textures in this byte order can be used instead. This can be effectively simulated in OpenGL ES 2.0 and 3.0 by using texture swizzling when it is not supported. Whilst this extension should be preferentially used, GL_IMG_texture_format_BGRA8888 mirrors this extension and can be used when this is not available. Apple hardware exposes a variation on this extension: GL_APPLE_texture_format_BGRA888, which is identical except that when loading the texture, the internal format should be RGBA instead of being equivalent to the format.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_texture_format_BGRA8888.txt

Example Code

```
// Create a BGRA8888 texture
glTexImage2D(GL TEXTURE 2D, 0, GL BGRA EXT, 1024, 1024, 0, GL BGRA EXT, GL UNSIGNED BYTE,
pixelData);
```

3.13. GL_EXT_texture_rg

Supported Hardware

Series 5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

OpenGL ES exposed luminance/alpha texture formats as the only single and double channel texture formats available, and these were not available as colour buffers when rendering. This extension adds the RG88 and R8 texture formats which can additionally be used as colour buffers. These also have the benefit of dropping the baggage previously associated with the LA formats, such as luminance being replicated across all channels - something that is no longer necessary with programmable shaders.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/EXT/EXT_texture_rg.txt

Example Code

```
// Create a two-channel byte texture
glTexImage2D(GL_TEXTURE_2D, 0, GL_RG_EXT, 1024, 1024, 0, GL_RG_EXT, GL_UNSIGNED_BYTE,
pixelData);
```

3.14. GL_IMG_multisampled_render_to_texture

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension adds multisampled rendering capabilities to OpenGLES, allowing a user to create multisampled render buffers, and attach textures to a framebuffer in a way that they can be used for multisampled renders. By enabling textures to be attached as multisampled but without using actual multisample textures, a significant amount of bandwidth can be saved. This mode of operation enables multisampled anti-aliasing to smooth lines at the edges of polygons and reduce the appearance of jagged lines. This is functionally equivalent to GL_EXT_multisampled_render_to_texture on PowerVR hardware.

Note

Part of this functionality is core to OpenGL ES 3.0. The renderbuffer functionality is present in core; however the multisampled texture attachment is not.

Registry Link

http://www.khronos.org/registry/gles/extensions/IMG/IMG_multisampled_render_to_texture.txt

Example Code

```
// Create a multisampled depth renderbuffer
GLint depthBuffer = glRenderbufferStorageMultisampleIMG(GL RENDERBUFFER, 4,
GL DEPTH COMPONENT16, 1024, 1024);

// Attach the renderbuffer to the framebuffer as per usual. Note: If one attachment is
// multi-sampled, all attachments need to be. If there is a mix, the framebuffer will be
// incomplete.
glFramebufferRenderbuffer(GL FRAMEBUFFER, GL DEPTH ATTACHMENT, GL RENDERBUFFER, depthBuffer);

// Create a regular colour buffer (texture). This extension doesn't introduce multi-sample
// textures, instead attaching regular textures that multi-sampling will resolve to before
// writing out.
GLint colourBuffer = glTexStorage(GL TEXTURE 2D, 1, GL RGBA8, 1024, 1024);

// Attach it to a framebuffer as multisampled, so that OpenGL ES knows that it should resolve
// a multisampled render to this texture.
glFramebufferTexture2DMultisampleIMG(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
colourBuffer, 0);
```

3.15. GL_IMG_program_binary

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 2.0 3

Extension Description

This extension enables PowerVR program binary formats to be queried and handled by the GL_OES_get_program_binary extension, which allows binary shader programs to be retrieved, stored and reloaded, avoiding compilation steps on subsequent runs of an application.

Registry Link

http://www.khronos.org/registry/gles/extensions/IMG/IMG_program_binary.txt

Example Code

```
// Create a shader program object
GLuint shaderProgram = glCreateProgram();

// If there isn't already a binary, create a program
if (!binaryExists)
{
	// Create shaders, compile them, attach them and link them as per normal, then...

	// Retrieve the program binary size
	GLint length=0;
	glGetProgramiv(shaderProgram, GL PROGRAM BINARY LENGTH, &length);

	// Pointer to the binary shader program in memory, needs to be allocated with the
	// right size.
	GLvoid* programBinaryData = (GLvoid*)malloc(length);

	// The format that the binary is retrieved in.
	GLenum binaryFormat=0;

	// Error checking variable - this should be greater than 0 after
	// glGetProgramBinaryOES, otherwise there was an error.
	GLsizei lengthWritten=0;

	glGetProgramBinaryOES(shaderProgram, length, &lengthWritten, &binaryFormat,
	programBinaryData);
}
else
{
	// Get the binary data, how much data there is, and what format it's in from whatever
	// cache it's stored in (e.g. a file)
	// ...

	// Instead of creating, compiling, attaching and linking shaders, upload a binary
	// program data from a program that previously underwent all of these stages.
	glProgramBinaryOES(shaderProgram, binaryFormat, programBinaryData,
	programBinaryDataLength);
}
```

3.16. GL_IMG_read_format

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

Reading pixels from a framebuffer in OpenGL ES is performed via the function glReadPixels(). This function is able to read data out in only two formats; RGBA8888, and another, implementation defined format, which can be queried with glGetIntegerv(). In OpenGL ES 1.x, only one read format is available in core, so GL_OES_read_format is required for this extension. Normally these additional formats are limited to being RGBA, RGB or Alpha-Only formats. This extension adds a new readable format (GL_BGRA_IMG) and a new readable type (UNSIGNED_SHORT_4_4_4_4_REV_IMG), allowing the function to be extended to use a different native format than typically allowed.

Registry Link

http://www.khronos.org/registry/gles/extensions/IMG/IMG_read_format.txt

Example Code

```
// Read pixels can now be performed with GL BGRA data as well as any other default formats.
glReadPixels(0, 0, 1024, 1024, GL BGRA IMG, GL UNSIGNED SHORT 4 4 4 4 REV IMG,
pixelDataOutput);
```

3.17. GL_IMG_shader_binary

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 2.0 3

Extension Description

The OpenGL ES 2.0 and 3.0 specifications include functions "glGetShaderBinary/glShaderBinary" - allowing an application developer to retrieve, store and load binary shaders after compilation. By using this extension to cache compiled shaders, it is possible to avoid the run time cost of compiling shaders on subsequent runs. The specifications do not, however, define any binary formats for shaders, which is instead left up to each implementer. This extension enables Imagination's shader binary format to be exposed through these mechanisms.

Typically it is more efficient to use GL_OES_get_program_binary/GL_IMG_program_binary where available, as these also avoid the cost of linking and binding attributes - saving additional time on start up.

Registry Link

http://www.khronos.org/registry/gles/extensions/IMG/IMG_shader_binary.txt

Example Code

```
// Create a fragment shader
GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);

// Instead of uploading shader source and compiling, upload pre-compiled binary data
glShaderBinary(1, &fragmentShader, GL_SGX_BINARY_IMG, binaryShaderData,
lengthOfBinaryShaderData);
```

3.18. GL_IMG_texture_compression_pvrtc

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

Compressed textures are a core part of modern rendering solutions, enabling much more texture content to be pushed and with greater performance due to the lower bandwidth and memory requirements. PVRTC is PowerVR's proprietary texture compression format which achieves high quality, full RGBA compression at data rates of 2 or 4 bits per pixel. Textures are restricted to being square, with power of two dimensions - though typically rectangular textures are often supported; there is no way to query for this.

This extension specifically enables supported devices to use these textures directly, often dramatically improving performance over uncompressed textures. This extension does not enable compression of these textures as it is too computationally expensive; this can be done on Desktop platforms with PVRTexTool (http://www.imgtec.com/powervr/insider/powervr-pvrtextool.asp)

Registry Link

http://www.khronos.org/registry/gles/extensions/IMG/IMG_texture_compression_pvrtc.txt

Example Code

```
// Upload a 4bpp PVRTC1 texture, RGBA and RGB are uploaded in the same way.
GLuint bitsPerPixel = 4;
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGBA_PVRTC_4BPPV1_IMG, 1024, 1024, 0,
(width*height*bitsPerPixel)/8, pixelData);

// Upload a 2bpp PVRTC1 texture, RGBA and RGB are uploaded in the same way.
GLuint bitsPerPixel = 2;
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGB_PVRTC_2BPPV1_IMG, 1024, 1024, 0,
(width*height*bitsPerPixel)/8, pixelData);
```

3.19. GL_IMG_texture_compression_pvrtc2

Supported Hardware

Series 5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

Compressed textures are a core part of modern rendering solutions, enabling much more texture content to be pushed and with greater performance due to the lower bandwidth and memory requirements. PVRTC2 is PowerVR's proprietary texture compression format which achieves high quality, full RGBA compression at data rates of 2 or 4 bits per pixel. PVRTC2 is related to the original PVRTC format, but adds a number of additional features which allow it to improve further on the original's quality levels. In particular, textures with arbitrary dimensions are now supported, and there is a hard edge mode which can improve areas of high discontinuity and non-tiled textures.

This extension specifically enables supported devices to use these textures directly, often dramatically improving performance over uncompressed textures. This extension does not enable compression of these textures as it is too computationally expensive; this can be done on Desktop platforms with PVRTexTool (http://www.imgtec.com/powervr/insider/powervr-pvrtextool.asp)

Registry Link

http://www.khronos.org/registry/gles/extensions/IMG/IMG_texture_compression_pvrtc2.txt

Example Code

```
// Upload a 4bpp PVRTC2 texture
GLuint bitsPerPixel = 4;
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGBA_PVRTC_4BPPV2_IMG, 1024, 1024, 0,
(width*height*bitsPerPixel)/8, pixelData);

// Upload a 2bpp PVRTC2 texture
GLuint bitsPerPixel = 2;
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGBA_PVRTC_2BPPV2_IMG, 1024, 1024, 0,
(width*height*bitsPerPixel)/8, pixelData);
```

3.20. GL_IMG_texture_format_BGRA8888

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

Typically, textures in OpenGL ES run in RGBA order; this extension adds BGRA8888 as a new format so that textures in this byte order can be used instead. This can be effectively simulated in OpenGL ES 2.0 and 3.0 by using texture swizzling when it is not supported.
GL_EXT_texture_format_BGRA8888 mirrors this extension and should be preferentially used. Apple hardware exposes a variation on this extension: GL_APPLE_texture_format_BGRA888, which is identical except that when loading the texture, the internal format should be RGBA instead of being equivalent to the format.

Example Code

```
// Create a BGRA8888 texture
glTexImage2D(GL_TEXTURE_2D, 0, GL_BGRA_IMG, 1024, 1024, 0, GL_BGRA_IMG, GL_UNSIGNED_BYTE,
pixelData);
```

3.21. GL_IMG_texture_npot

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 2.0

Extension Description

This extension adds MIPMap capable minification filters to OpenGL ES 2 for non-power of two (NPOT) textures. OpenGL ES 2 limits NPOT textures as it does not support MIPMapping for them. This extension allows users to call 'glGenerateMipMap' to create a full MIP chain, and then adds minification filters that allow them to be accessed. This extension does not add functionality allowing applications to supply a full NPOT MIP Map chain prior to runtime, however.

Note

This functionality is core to OpenGL ES 3.0.

Example Code

```
// Upload a texture with dimensions of 15 by 47. Typically this isn't supported without
// extension support.
glTexImage2D(GL TEXTURE 2D, 0, GL RGBA, 15, 47, 0, GL RGBA, GL UNSIGNED BYTE, pixelData);

// Mip-Maps can't be specified by this extension, but can be auto-generated for non-compressed
// textures
glGenerateMipmap(GL TEXTURE 2D);

// Any filter mode specified by OpenGL is now valid with this extension, whereas it would not
// have been previously.
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameter(GL TEXTURE 2D, GL TEXTURE MAG FILTER, GL LINEAR);

// Only the following parameters are specifiable, anything else is invalid (including the
// default texture parameters!).
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

3.22. GL_OES_blend_equation_separate

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This function extends GL_OES_blend_subtract, which adds functionality allowing a blend equation to operate on all the channels in an output colour at once. However, often a different operation is wanted for the alpha and colour channels. This extension adds a new function, glBlendEquationSeparateOES, which operates in the same way as glBlendEquationOES, except that different equations can now be specified for RGB and Alpha.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_blend_equation_separate.txt

Example Code

```
// Set the blend equation to use additive blending for the RGB channels, but subtractive for
// the Alpha channel.
glBlendEquationSeparate(GL_FUNC_ADD_OES, GL_FUNC_SUBTRACT_OES);
```

3.23. GL_OES_blend_func_separate

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0 3*

Extension Description

OpenGL ES 1.x has functionality allowing a blend function to operate on all the channels in an output colour at once. However, often a different operation is wanted for the alpha and colour channels. This extension adds a new function, glBlendFuncSeparateOES, which operates in the same way as glBlendFunc, except that different equations can now be specified for RGB and Alpha.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_blend_func_separate.txt

Example Code

```
// Set the blend equation to use a value of transparent white for the source colour, but the
// destination colour remains as it otherwise would.
glBlendFuncSeparate(GL_ONE, GL_DST_COLOR, GL_ZERO, GL_DST_ALPHA);
```

3.24. GL_OES_blend_subtract

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

OpenGL ES 1.x has functionality allowing various basic blending functions to occur via glBlendFunc, by allowing the user to pick the source and destination values used in blend calculations. This feature enables further specification of how blending works by adding a new function - glBlendEquationOES, which allows a developer to specify, further what actually happens to those values once they have been selected. Usual GL operation adds the values together, using "GL_FUNC_ADD_OES" as a default. Two new modes are included by this function; "subtraction" and "reverse subtraction". So the equations used look as follows:

- GL_FUNC_ADD_OES:
  - Result = Source + Destination
- GL_FUNC_SUBTRACT_OES:
  - Result = Source - Destination
- GL_FUNC_REVERSE_SUBTRACT_OES:
  - Result = Destination - Source

This extension is further extended by GL_OES_blend_equation_separate.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_blend_subtract.txt

Example Code

```
// Set the blend equation to use additive blending
glBlendEquation(GL_FUNC_ADD_OES);

// Set the blend equation to use subtractive blending: source - destination
glBlendEquation(GL FUNC SUBTRACT OES);

// Set the blend equation to use reverse subtractive blending: destination - source
glBlendEquation(GL_FUNC_REVERSE_SUBTRACT_OES);
```

3.25. GL_OES_byte_coordinates

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension allows users to pass byte values as texture coordinates and vertex positions in OpenGL ES 1.x, which are otherwise not allowed by the API. Byte sized data has the advantage of consuming less bandwidth than larger types and can be useful for certain scenes.

Note

This functionality is superseded by programmable shader support in subsequent versions of OpenGL ES.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_byte_coordinates.txt

Example Code

```
// Input byte coordinates
GLbyte coordinates[] =
{
        0, 255, 0,
        255, 255, 0,
        255, 0, 0,
};

// Give the coordinates to OpenGL ES
glVertexPointer(3, GL_BYTE, 0, coordinates);
```

3.26. GL_OES_compressed_ETC1_RGB8_texture

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension adds support for loading textures that have been compressed with the ETC1 texture compression scheme, directly into the API via glCompressedTexImage2D. This format provides RGB texture encoding at a lower bitrate than would otherwise be available, whilst still providing reasonable quality.

As this specification was written against OpenGL ES1, when used in ES2, some features applicable to ES2 are not available due to the non-trivial nature of colour read back from an ETC texture. In particular; Non-power of two ETC1 is not directly supported in some first generation ES2 hardware, so it's generally best to ensure that all ETC1 textures are encoded with square, power-of-two dimensions.

Note

Whilst this functionality can be exposed in OpenGL ES 3.0, it is effectively enabled anyway by the inclusion in core 3.0 of ETC2, which is backwards compatible. By passing the same data with the new enum "GL_COMPRESSED_RGB8_ETC2" the behaviour can be replicated.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_compressed_ETC1_RGB8_texture.txt

Example Code

```
// Upload an ETC1 texture.
GLuint bitsPerPixel = 4;
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGB8_ETC2, 1024, 1024, 0,
(width*height*bitsPerPixel)/8, pixelData);
```

3.27. GL_OES_compressed_paletted_texture

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension allows the use of paletted textures in OpenGL ES as a compressed texture. Paletted formats are textures which store a small number of colours as a palette, and then the texel array is a simple array of indices into this palette. The benefit of this is that the palette can be cached and the indices have a lower bandwidth cost than a full colour. This extension allows the colour formats of RGB565, RGBA4444, RGBA551, RGB888 or RGBA8888. Each texel is then allowed to either have a 4 or 8 bit index value, and an appropriate number of palette colours to match.

Note

In subsequent versions of OpenGL ES, this functionality can be easily replicated using programmable shaders by having a small palette texture, and a vertex attribute or another texture specifying the index to use. This allows the functionality to be much more flexible than provided by this extension, so it was dropped.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_compressed_paletted_texture.txt

Example Code

```
// Upload a palletted texture. The size of the data will be dependent on how it was
// compressed, and is thus dependent on whatever tool was used.
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_PALETTE4_RGB8_OES, 1024, 1024, 0, pixelDataSize,
pixelData);
```

3.28. GL_OES_depth_texture

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

When using a framebuffer, z-buffers need to be specifically supported by attaching a depth render target. Originally this was done with a depth renderbuffer; but these are limited as they cannot be read back later. This extension adds depth textures, which allow the user to read back depth output from a render, enabling techniques such as shadow mapping to occur without resorting to storing the depth in a colour texture.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_depth_texture.txt

Example Code

```
// Create a depth texture - generally texture data isn't uploaded for this - but it can be.
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 1024, 1024, 0, GL_DEPTH_COMPONENT,
GL_UNSIGNED_INT, NULL);
```

3.29. GL_OES_depth24

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

This extension adds the sized internal format GL_DEPTH_COMPONENT24 to the list of internal formats accepted by the function "glRenderBufferStorage". This format is a single channel data format with 24-bits per channel, used to represent the depth values stored in a z-buffer.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_depth24.txt

Example Code

```
// Create a 24-bit depth renderbuffer
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24_OES, 1024, 1024);
```

3.30. GL_OES_draw_texture

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension adds a simple method of blitting a texture to the screen, for things such as backgrounds and 2D GUI elements. The user can provide a crop rectangle to specify the part of the texture that they wish to draw, and then call DrawTex**OES() with coordinates for the part of the screen they wish to draw to.

Note

This functionality as specified cannot be made to work with the programmable pipeline of OpenGL ES 2.0. In OpenGL ES 3.0, this functionality is effectively replaced by the function "glBlitFramebuffer".

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_draw_texture.txt

Example Code

```
// Bind a texture
glBindTexture(GL_TEXTURE_2D, texture);

// Set the texture crop rectangle to set the texture coordinates to draw from
GLint textureCropRectangle[4] = {0, 0, 256, 256};
glTexParameteriv(GL_TEXTURE_2D, GL_TEXTURE_CROP_RECT_OES, textureCropRectangle);

// Draw the texture to the whole screen, which will be resized appropriately from the texture
// rectangle
glDrawTexfOES(-1.0, -1.0, 2.0, 2.0);
```

3.31. GL_OES_egl_image

Naming

This extension is named GL_OES_EGL_image on Series5/5XT and GL_OES_egl_image on Series6.

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension enables the use of EGLSync objects in an OpenGL ES command stream. EGLSync objects are defined in separate extensions, such as EGL_KHR_fence_sync.

This extension enables the an OpenGL ES implementation use EGLImages as textures, using the functions glEGLImageTargetTexture2DOES and glEGLImageTargetRenderbufferStorageOES. The concept of an EGLImage is introduced by EGL_KHR_image_base or EGL_KHR_image.

Despite allowing the creation of renderbuffers, whose sole purpose is to be used as a backing buffer for a framebuffer, this extension does not enable EGLImages to be used as a framebuffer attachments. This functionality is instead added by GL_OES_EGL_image_external. Typically you will not see an implementation supporting renderbuffer creation from an EGLImage without also supporting this additional extension.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_EGL_image.txt

Example Code

```
// Create an EGL image via EGL KHR image base. In this case from an EGL pixmap
// (EGL KHR image pixmap), but the source is irrelevant to this example
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglContext, EGL_NATIVE_PIXMAP_KHR,
(EGLNativeBuffer)eglPixmap, NULL);

// Generate a texture object to map the EGLImage to, and bind it.
GLint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

// Map the EGLImage into an OpenGL ES texture using this extension.
glEGLImageTargetTexture2DOES(GL_TEXTURE_2D, (GLeglImageOES)eglImage);
```

3.32. GL_OES_egl_image_external

Naming

This extension is named GL_OES_EGL_image_external on Series5/5XT and GL_OES_egl_image_external on Series6.

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0

Extension Description

This extension layers on top of GL_OES_EGL_image_external, and allows textures that have been created by an EGLImage to be used as a framebuffer attachment.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_EGL_image_external.txt

Example Code

```
// Create an EGL image via EGL_KHR_image_base. In this case from an EGL pixmap
(EGL_KHR_image_pixmap), but the source is irrelevant to this example
EGLImageKHR eglImage = eglCreateImageKHR(eglDisplay, eglContext, EGL_NATIVE_PIXMAP_KHR,
(EGLNativeBuffer)eglPixmap, NULL);

// Generate a texture object to map the EGLImage to, and bind it.
GLint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

// Map the EGLImage into an OpenGL ES texture using this extension.
glEGLImageTargetTexture2DOES(GL_TEXTURE_2D, (GLeglImageOES)eglImage);

// This extension now allows an application to attach this to a framebuffer, using the
// standard OpenGL ES framebuffer commands
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
```

3.33. GL_OES_egl_sync

Naming

This extension is named GL_OES_EGL_sync on Series5/5XT and GL_OES_egl_sync on Series6.

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

This extension enables the use of EGLSync objects in an OpenGL ES command stream. EGLSync objects are defined in separate extensions, such as EGL_KHR_fence_sync.

Note

This functionality is superseded by GLSync objects in OpenGL ES 3.0

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/EGL_KHR_fence_sync.txt

Example Code

```
// Disclaimer: None of this is good thread-safe code, it's pseudo-code for illustration. The
exact CPU-side synchronisation mechanism will depend on your code.

// A fence sync handle accessible from both threads.
EGLSyncKHR g_eglFenceSync = 0;

// Pseudo-Function representing operations on a thread, in particular this thread is acting as
// a data producer.
void Thread1()
{
        // Render something
        glDraw(...);

        // Create a fence sync object in the command stream after the draw call, so a second
        // thread knows when the render operation is completed.
        g_eglFenceSync = eglCreateSyncKHR(eglDisplay, EGL_SYNC_FENCE_KHR, NULL);

        /*  Call eglClientWaitSync to flush the context, so that it's guaranteed to finish at
            some point. It only flushes the current context for the thread, hence calling it
            here - Flushing from the thread which actually needs to wait on this sync object
            would not flush the previous draw operations. By setting the timeout to 0 we've
            effectively called "glFlush" but a little more precisely, and it must work on all
            drivers.
        */
        EGLint waitResult = eglClientWaitSyncKHR(eglDisplay, eglFenceSync,
        EGL_SYNCH_FLUSH_COMMANDS_BIT_KHR, 0);
}

// Pseudo-Function representing operations on a second thread. This thread acts as a data
consumer.
void Thread2()
{
        while(g_eglFenceSync == 0)
        {
                // Do some other work, the first thread isn't ready.
        }

        // Wait for the synchronisation object
        EGLint waitResult = eglClientWaitSyncKHR(eglDisplay, eglFenceSync,
        EGL SYNCH FLUSH COMMANDS BIT KHR, EGL FOREVER KHR);

        // Destroy the fence sync object once we're done with it.
        EGLBoolean success = eglDestroySyncKHR(eglDisplay, eglFenceSync);

        // Now that the first thread's rendering has completed, do something with the result.
        // E.g. composite the result of the draw into a windowing system.
        glDraw(...);
}
```

3.34. GL_OES_element_index_uint

Supported Hardware

Series 5/5XT and Series6 (ES2/3 Only)

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

This extension adds the ability to use an array of unsigned integers (GL_UNSIGNED_INT) as indices in glDrawElements - typically only unsigned short or unsigned byte types are supported. This is more costly than using byte or short in terms of bandwidth, but does allow a much greater range of data to be specified - allowing larger models to be sent through one draw call where they would otherwise have to be batched.

Note

This functionality is core to OpenGL ES 3.0

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_element_index_uint.txt

Example Code

```
// Draw with 32-bit, unsigned integer indices
glDrawElements(GL_TRIANGLES, 32, GL_UNSIGNED_INT, indices);
```

3.35. GL_OES_extended_matrix_palette

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension extends GL_OES_matrix_palette, which specifies that a minimum of only 9 matrices are paletted, and that only 3 matrices can be blended per vertex. Based on developer feedback, this extension has been created to up the minimum palette size to 32, and now 4 matrices can be blended per vertex.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_extended_matrix_palette.txt

Example Code

```
// Bind the 32nd matrix as the current matrix, not normally allowed but defined by this
// extension.
glCurrentPaletteMatrixOES(31);
```

3.36. GL_OES_fixed_point

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension provides fixed point functions for hardware that supports the OpenGL ES 1.x common profile, but has inefficient floating point support. The functions provided are equivalent to the Common-Lite profile, and are denoted by ending in an "x" instead of an "f" for floating point functions. Fixed point values are unsigned integers that are scaled to a S15.16 representation. The S15.16 representation means that there is one sign bit, the top 15 bits represent integer values, and the bottom 16 bits represent values below zero. In essence, it represents a signed short with 16 bits of sub-zero values.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_fixed_point.txt

Example Code

```
// Use the fixed-point version of a function to scale by 1.0, 0.5 and 2.0 in the x, y and z
// axes respectively. The value "65536" is effectively treated as 1.0.
GLfixed x = 65536;
GLfixed y = 65536 / 2;
GLfixed z = 65536 * 2;
glScalex(x, y, z);
```

3.37. GL_OES_fragment_precision_high

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

This extension allows the use of high precision floating point values in fragment shaders. This affects GLSL ES 1.00 only, as it is a core feature in GLSL ES 3.00. High precision floating point allows a higher range of values to be used than would otherwise be provided by medium precision floating point, allowing more complex and accurate work to be done in the shader.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_fragment_precision_high.txt

Example Code (GLSL)

```
// Declare a highp variable in a fragment shader, not allowed in Core OpenGL ES
varying highp vec4 someVariable;
```

3.38. GL_OES_framebuffer_object

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension defines a simple interface for drawing to rendering destinations other than the buffers provided to the GL by the windowing-system. This allows things to be drawn independently of VSync and enables multi-pass algorithms that previously would have been limited to using CopyTexImage2D on the main framebuffer. Framebuffers allow you to attach colour, depth and stencil attachments to be used as drawing buffers, and then bind the framebuffer to be drawn into. Subsequent draw operations will then take place in this framebuffer, until the main framebuffer is rebound. Textures attached in this way can then be read back in subsequent drawing operations. This extension also introduces "Renderbuffers", which work similarly to a texture but are managed by the GPU entirely, and cannot be read back later. They are typically used for things like a depth buffer where the end result does not matter and will never need flushing.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_framebuffer_object.txt

Example Code

```
void Initialise()
{
	// Create a framebuffer object
	GLint framebufferObject;
	glGenFramebuffersOES(1, &framebufferObject);

	// Bind the framebuffer object to the current state
	glBindFramebufferOES(GL_FRAMEBUFFER_OES, framebufferObject)

	// Create a depth renderbuffer
	GLint renderBuffer;
	glGenRenderbuffersOES(1, &renderBuffer);

	// Bind and initialise it.
	glBindRenderbufferOES(GL_RENDERBUFFER_OES, renderBuffer);
	glRenderbufferStorageOES(GL RENDERBUFFER OES, GL DEPTH COMPONENT16 OES, 1024, 1024);

	// Attach the renderbuffer to the framebuffer
	glFramebufferRenderbufferOES(GL FRAMEBUFFER OES, GL DEPTH ATTACHMENT,
	GL_RENDERBUFFER_OES,  renderBuffer);

	// Create a colour texture
	GLint texture;
	glGenTextures(1, &texture);

	// Bind and initialise the texture
	glBindTexture(GL_TEXTURE_2D, texture);
	glTexImage2D(GL TEXTURE 2D, 0, GL RGBA, 1024, 1024, 0, GL RGBA, GL UNSIGNED BYTE,
	NULL);

	// Attach the texture to the framebuffer
	glFramebufferTexture2DOES(GL_FRAMEBUFFER_OES, GL_COLOUR_ATTACHMENT0_OES,
	GL TEXTURE 2D, texture, 0);

	// Check framebuffer is complete
	if (glCheckFramebufferStatusOES(GL_FRAMEBUFFER_OES) != GL_FRAMEBUFFER_COMPLETE_OES)
	{
		// An error has occurred
	}
}

void Render()
{
	// Bind the framebuffer for rendering
	glBindFramebufferOES(GL FRAMEBUFFER OES, framebufferObject)

	// Draw something
	glDraw(...);

	// Bind the default (EGL provided) framebuffer to actually draw to the screen
	glBindFramebufferOES(GL FRAMEBUFFER OES, 0)

	// Draw something else, generally using the result of the framebuffer object's render.
	glDraw(...);

	// SwapBuffers
	eglSwapBuffers();
}
```

3.39. GL_OES_get_program_binary

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 2.0

Extension Description

One of the larger costs of using shaders is that when initialising your application, they need to be compiled by an implementation's default compiler. Mechanisms exist to get the compiled shader binary back from the API after compilation, but there is still a linking stage after this which needs to be performed to create a program. This extension allows users to retrieve the fully compiled and linked program binary from the API, and then load it in on subsequent runs. Using this extension can save additional time on application start up compared to shader binaries as it also includes the attribute location binding and linking stages. It's worth noting that this extension does not itself define any binary formats, this is up to each vendor. For PowerVR hardware, these formats are defined by GL_IMG_program_binary.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_get_program_binary.txt

Example Code

```
// Create a shader program object
GLuint shaderProgram = glCreateProgram();

// If there isn't already a binary, create a program
if (!binaryExists)
{
	// Create shaders, compile them, attach them and link them as per normal, then...

	// Retrieve the program binary size
	GLint length=0;
	glGetProgramiv(shaderProgram, GL PROGRAM BINARY LENGTH, &length);

	// Pointer to the binary shader program in memory, needs to be allocated with the
	// right size.
	GLvoid* programBinaryData = (GLvoid*)malloc(length);

	// The format that the binary is retrieved in.
	GLenum binaryFormat=0;

	// Error checking variable - this should be greater than 0 after
	// glGetProgramBinaryOES, otherwise there was an error.
	GLsizei lengthWritten=0;

	glGetProgramBinaryOES(shaderProgram, length, &lengthWritten, &binaryFormat,
	programBinaryData);
}
else
{
	// Get the binary data, how much data there is, and what format it's in from whatever
	// cache it's stored in (e.g. a file)
	// ...

	// Instead of creating, compiling, attaching and linking shaders, upload a binary
	// program data from a program that previously underwent all of these stages.
	glProgramBinaryOES(shaderProgram, binaryFormat, programBinaryData,
	programBinaryDataLength);
}
```

3.40. GL_OES_mapbuffer

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

This extension enables the user to upload data directly into memory that the driver provides for them, by mapping the storage of a buffer object into client address space. In many cases this means that the user can avoid an additional memory allocation on top of one performed by the driver.

Note

This functionality is core to OpenGL ES 3.0 in the form of `glMapBufferRange`

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_mapbuffer.txt

Example Code

```
// Bind a buffer
glBindBuffer(GL ARRAY BUFFER, aVertexBuffer);

// Map the buffer's data to a CPU addressable pointer, in a way that allows us to write to it,
// but not read the data back.
void* bufferData = glMapBufferOES(GL_ARRAY_BUFFER, GL_WRITE_ONLY_OES);

// Get the size of the buffer
GLint bufferSize = 0;
glGetBufferParameteriv(GL_ARRAY_BUFFER, GL_BUFFER_SIZE, &bufferSize);

// Write some data into the pointer
memcpy(bufferData, newDataForTheBuffer, bufferSize);

// Unmap the pointer
glUnmapBufferOES(GL_ARRAY_BUFFER);

// The pointer is now invalid, so NULL it
bufferData = NULL;
```

3.41. GL_OES_matrix_get

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension is specifically targeted at platforms where only fixed point functionality is supported on the CPU, but data may be represented internally to a GPU as floating point. In these cases, an application developer is only able to query floating point matrices via glGetFixedv, which will result in a loss of information. To work around this, this extension allows floating point matrices to be queried back from the hardware by packing each floating point value into an integer instead, using the IEEE 754 floating point representation. This value can then be queried using glGetIntegerv, using a number of new "float as int" tokens for affected matrix types.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_matrix_get.txt

Example Code

```
// Get the values of the projection matrix that was previously loaded into OpenGL ES, but with
// the floating point bits stored as integers (for systems that don't support floats)
GLint projectionMatrix[16];
glGetIntegerv(GL_MODELVIEW_MATRIX_FLOAT_AS_INT_BITS_OES, projectionMatrix);
```

3.42. GL_OES_matrix_palette

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

Rather than using a fixed matrix for an entire model, allowing a mixture of different matrices with appropriate weightings to be used in a single piece of geometry can allow greater flexibility in a renderer. In particular this enables better control of skeletal animation. This extension adds a palette of matrices to be specified, with indices passed in an array to specify which matrix to use for each vertex, and an array of weights to control how much it affects the vertex.

Note

This extension is effectively part of the OpenGL ES 1.x Extension Pack Specification (required for GL_OES_extended_matrix_palette) and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_matrix_palette.txt

Example Code

```
// Set the matrix pallette stack as the current matrix mode
glMatrixMode(GL_MATRIX_PALETTE_OES);

// Bind the 9th matrix as the current matrix - the maximum guaranteed by this extension
glCurrentPaletteMatrixOES(8);

// Load a matrix into the palette
glLoadMatrix(someTranslationMatrix);

// Use translation functions as with any other matrix
glTranslatef(0.0f, 10.0f, 0.0f);
```

3.43. GL_OES_packed_depth_stencil

Supported Hardware

Series 5/5XT and Series6 (ES2/3 Only)

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

Typical use of a depth buffer can generally be satisfied with a 24-bit buffer, but this is generally unaligned with respect to power-of-two boundaries. Having 8-bits of stencil data is typically more than enough for most applications, so it is a natural fit to interleave the two buffer types. By doing so, the overall buffer can be aligned to power-of-two boundaries for speed, without wasting any memory.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_packed_depth_stencil.txt

Example Code

```
// Create a packed depth and stencil texture format, normally you would not upload data here
// for this
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_STENCIL_OES, 1024, 1024, 0, GL_DEPTH_STENCIL_OES,
GL_UNSIGNED_INT_24_8_OES, NULL);
```

3.44. GL_OES_point_size_array

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

Points are usually rendered at a fixed size by OpenGL ES, according to the value set by glPointSize. This size is applied uniformly to all rendered points, which limits their capabilities. This extension adds the ability to use an array of values instead, so that each point can have different sizes.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_point_size_array.txt

Example Code

```
// Setup an array of points to be rendered - three in this case
GLfloat pointPositions[] =
{
         0.4f, 0.7f, 0.2f,
         0.1f,-0.5f, 0.2f,
        -0.0f, 0.7f, 0.2f
};

// Set the vertex pointer as normal
glVertexPointer(3, GL FLOAT, 0, pointPositions);

// Enable the vertex array
glEnableClientState(GL_VERTEX_ARRAY);

// Create an array of sizes for these points
GLfloat pointSizes[] =
{
        1.0f,
        2.0f,
        0.7f
};

// Set the point sizes via this extension
glPointSizePointerOES(3, GL_FLOAT, 0, pointPositions);

// Enable the point size array
glEnableClientState(GL POINT SIZE ARRAY OES);

// Draw the points
glDraw(GL_POINTS, ...);
```

3.45. GL_OES_point_sprite

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension extends the standard functionality of OpenGL points to allow more flexible rendering than otherwise provided. For a core point object, OpenGL ES specifies that texture coordinates are identical across the entire body of the point, and it is then antialiased to fade out. To work around this, developers often use a quad with an alpha blended/tested texture to allow the full range of a texture to be expressed. When in use, this extension disables the antialiasing on the point, and instead interpolates the texture coordinates provided over the entire body of the point, allowing a point to be effectively textured to represent a sprite.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_point_sprite.txt

Example Code

```
// Enable point sprite drawing
glEnable(GL_POINT_SPRITE_OES);

// Set the texture environment to use point sprite coordinate replace to true, so that texture
// coordinates are interpolated automatically across the rendered point, rather than using a
// single coordinate.
glTexEnvi(GL_POINT_SPRITE_OES, GL_COORD_REPLACE_OES, GL_TRUE);

// Draw some point sprites at the locations specified in
glDraw(GL_POINTS, ...);
```

3.46. GL_OES_query_matrix

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension allows a user to query a matrix in a type independent of the underlying implemented value, by returning the mantissa and exponent of each matrix value separately. This solves a similar issue to OES_matrix_get, but in a slightly more flexible way. The mantissa value is returned as an S15.16 fixed point value, and the exponents are returned as integers. It works similarly to the C function "modf". This extension relies on either the Common-Lite profile or the GL_OES_fixed_point extension to provide the definition of a fixed point value.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_query_matrix.txt

Example Code

```
// Get the project matrix back as separate exponents and mantissas

// Set the projection matrix stack as the current matrix mode
glMatrixMode(GL PROJECTION);

// Get the exponents and mantissa values of the projection matrix that was previously loaded
// into OpenGL ES
GLfixed projectionMatrixMantissas[16];
GLint projectionMatrixExponents[16];
glQueryMatrixxOES(projectionMatrixMantissas, projectionMatrixExponents);
```

3.47. GL_OES_read_format

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x

Extension Description

Reading pixels from a framebuffer in OpenGL ES is performed via the function glReadPixels(). This function is able to read data out in only one format for OpenGL ES 1.x; RGBA8888. This extension adds another, implementation defined format, which can be queried with glGetIntegerv(). These additional formats are limited to being RGBA, RGB or Alpha-Only formats.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_read_format.txt

Example Code

```
// Find out the preferred format and type to read back pixels on a platform
GLint preferredFormat;
GLint preferredType;
glGetIntegerv(GL_IMPLEMENTATION_COLOR_READ_FORMAT_OES, &preferredFormat);
glGetIntegerv(GL_IMPLEMENTATION_COLOR_READ_FORMAT_OES, &preferredType);

// Figure out how big the texture is going to be.
GLuint pixelReadDataSize = 0;
switch(preferredFormat)
{
case GL RGBA:
	{
		switch(preferredType)
		{
		case GL_UNSIGNED_BYTE:
			pixelReadDataSize = readWidth * readHeight * 4;
			break;
		case GL UNSIGNED SHORT 4 4 4 4:
		case GL UNSIGNED SHORT 5 5 5 1:
			pixelReadDataSize = readWidth * readHeight * 2;
			break;
		}
		break;
	}
case GL RGB:
	{
		switch(preferredType)
		{
		case GL UNSIGNED BYTE:
			pixelReadDataSize = readWidth * readHeight * 3;
			break;
		case GL_UNSIGNED_SHORT_5_6_5:
			pixelReadDataSize = readWidth * readHeight * 2;
			break;
		}
		break;
	}
case GL_LUMINANCE_ALPHA:
	{
		pixelReadDataSize = readWidth * readHeight * 2;
		break;
	}
case GL_LUMINANCE:
	{
		pixelReadDataSize = readWidth * readHeight;
		break;
	}
case GL ALPHA:
	{
		pixelReadDataSize = readWidth * readHeight;
		break;
	}
}

// Allocate enough space to read the data
GLvoid* pixelReadData = malloc(pixelReadDataSize);

// Specify the pixel unpack operation to be tightly packed (no padding) when reading back
// data, otherwise this needs to be handled when working out the size
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

// Read pixels using these formats
if (pixelReadData)
{
	glReadPixels(readOriginX, readOriginY, readWidth, readHeight, preferredFormat,
preferredType, pixelReadData);
}
```

3.48. GL_OES_required_internalformat

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0, 3.0*

Extension Description

By default, OpenGL ES implementations are free to store texture data as they see fit when using unsized types such as "GL_RGBA". This extension adds a number of "sized" internal formats to the specification which mandate a minimum storage precision for these data types. More precision may be used internally, but it can never store data at a lower precision than requested which would otherwise cause a loss of information.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_required_internalformat.txt

Example Code

```
// Create a texture with RGBA8 so that it uses no less than 8 bits per pixel
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8_OES, 1024, 1024, 0, GL_RGBA, GL_UNSIGNED_BYTE,
pixelData);
```

3.49. GL_OES_rgb8_rgba8

Supported Hardware

Series 5/5XT and Series6

Valid APIs

OpenGL ES 1.x, 2.0

Extension Description

This extension adds the sized internal formats RGB8 and RGBA8 to the list of internal formats accepted by the function "glRenderBufferStorage". These formats represent 3 and 4 channel data formats with 8-bits per channel, used to represent colour data.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_rgb8_rgba8.txt

Example Code

```
// Create a renderbuffer with RGBA8 so that it uses no less than 8 bits per pixel
glRenderbufferStorage(GL_RENDERBUFFER, GL_RGBA8_OES, 1024, 1024);
```

3.50. GL_OES_single_precision

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This function is basically the equivalent of the GL_OES_fixed_point, but for floating point functions. This extension doesn't actually express anything, as these functions exist in the common profile anyway. Theoretically it could have made some sort of sense in a Common-Lite Profile (fixed point only), but this never happened in practice.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_single_precision.txt

Example Code

```
// Use a floating point function
GLfloat rgbaBlendValues[4] = {0.0f, 0.0f, 0.0f, 0.0f};
glGetFloatv(GL_BLEND_COLOR, rgbaBlendValues);
```

3.51. GL_OES_standard_derivatives

Supported Hardware

Series/5XT, Series6

Valid APIs

OpenGL ES 2.0, OpenGL ES 2.0

Extension Description

Standard derivative functions are optionally available in the GL shading language to give an approximate delta to the values in neighbouring fragments in the x or y directions. The function will evaluate the local difference for any value passed to it. It uses dFdx() to return the difference in the x direction, and dFdy() to return the difference in the y direction.

For example, if you were to call 'dFdx(gl_FragCoord.x)', you'd get a result of 1.0, as the neighbouring fragment will have an x coordinate exactly one position over. 'dFdx(gl_FragCoord.y)' on the other hand would return 0.0, as the y coordinate is static as you move left or right.

Typical use of this functionality is to estimate the filter width used for anti-aliasing procedural textures. To facilitate this use case, a third function is provided: fwidth(), which returns the sum of the absolute difference in x and y (e.g. abs(dFdx(val)) + abs(dFdy(val))).

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_standard_derivatives.txt

Example Code (GLSL)

```
// Get the texture colour for a given fragment
lowp vec4 colourHere = texture2D(sTexture, TexCoord.xy);

// Get the value that was used in a neighbouring fragment in the x direction
lowp vec4 colourNextDoor = dFdx(colourHere) + colourHere;

// Get the value that was used in a neighbouring fragment in the y direction
lowp vec4 colourNextDoorVertically = dFdy(colourHere) + colourHere;
```

3.52. GL_OES_stencil_wrap

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x

Extension Description

Core OpenGL ES 1.0 stencil buffers require that when a value is incremented to the maximum value or decreased to the minimum, further operations will clamp that value. However, a number of algorithms use this buffer as a counter, calculating the difference between the total number of increments and the total number of decrements. This extension adds two new stencil operations: INCR_WRAP and DECR_WRAP. These allow the value to wrap round when the value changes, and gives more flexibility to the stencil buffer.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_stencil_wrap.txt

Example Code

```
// Change the stencil operations to decrease and wrap when failing the depth or stencil test,
// and increase and wrap when succeeding
glStencilOp(GL_DECR_WRAP, GL_DECR_WRAP, GL_INCR_WRAP);
```

3.53. GL_OES_stencil8

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x

Extension Description

Despite OpenGL ES 1.x supporting Depth and Stencil testing in core, the specification doesn't define any stencil buffer representations, and does not mandate any stencil buffer support. This extension exposes an 8 bit stencil buffer for stencil operations.

Note

This functionality is core to OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_stencil8.txt

Example Code

```
// Create a 1024x1024, 8-bit stencil renderbuffer
glRenderbufferStorage(GL_RENDERBUFFER, GL_STENCIL_INDEX8_OES, 1024, 1024);
```

3.54. GL_OES_texture_cube_map

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x

Extension Description

The standard way to render a cube of textures, such as a skybox, is to render from six individual 2D textures, one for each face. This means that the render has to be split, however, so that each face can be rendered with its own texture. This extension adds a new texture layout scheme for efficient rendering of things like skyboxes, by storing the 6 faces of a cubemap in one texture object. As well as this, texture lookups are performed by using a vector that points at the cube, rather than using explicit x,y coordinates.

This extension also adds texture coordinate generation functions to automatically create texture coordinates based on the eye direction of the user, via one of two modes:

* Reflection map mode: Generates the coordinates matching the eye-space reflection vector. This is used for standard skybox or other cubemapping.

* Normal map mode: Generates the coordinates matching the vertex's transformed eye-space normal. This is often useful for more advanced techniques and diffuse lighting models.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_texture_cube_map.txt

Example Code

```
// Create a cubemap texture - have to specify all faces individually.
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X_OES, 0, GL_RGBA 1024, 0, GL_RGBA,
GL_UNSIGNED_BYTE, pixelData[0]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y_OES, 0, GL_RGBA 1024, 0, GL_RGBA,
GL_UNSIGNED_BYTE, pixelData[1]);
glTexImage2D(GL TEXTURE CUBE MAP POSITIVE Z OES, 0, GL RGBA 1024, 0, GL RGBA,
GL_UNSIGNED_BYTE, pixelData[2]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X_OES, 0, GL_RGBA 1024, 0, GL_RGBA,
GL_UNSIGNED_BYTE, pixelData[3]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_OES, 0, GL_RGBA 1024, 0, GL_RGBA,
GL UNSIGNED BYTE, pixelData[4]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_OES, 0, GL_RGBA 1024, 0, GL_RGBA,
GL_UNSIGNED_BYTE, pixelData[5]);
```

3.55. GL_OES_texture_env_crossbar

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension extends the combining functionality in OpenGL ES 1 to allow application developers to choose textures from specific active texture units as the source, rather than just the current texture.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_texture_env_crossbar.txt

Example Code

```
// Set the first source alpha texture environment variable to sample from the third texture
// unit
glTexEnvi(GL_TEXTURE_2D, GL_SOURCE0_ALPHA, GL_TEXTURE2);
```

3.56. GL_OES_texture_float

Supported Hardware

Series5/5XT, Series6 (ES2/3 Only)

Valid APIs

OpenGL ES 1.x, OpenGL ES 2.0

Extension Description

This extension adds the ability to use 32-bit floating point type (GL_FLOAT) representations of texture data as a readable texture format when shading an object. This extension does not add linear sampling from this texture type however due to the high cost of such an operation, which is instead enabled by GL_OES_texture_float_linear. This extension also does not allow these textures to be used as a colour attachment for a framebuffer.

The float data format allows greater range than an integer representation would and is typically used for High Dynamic Range (HDR) colour data.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_texture_float.txt

Example Code

```
// Create a floating point texture
glTexImage2D(GL TEXTURE 2D, 0, GL RGBA, 1024, 1024, 0, GL RGBA, GL FLOAT, pixelData);

// Only nearest filtering is supported for these textures for this extension. Linear filtering
// is enabled by separate extensions.
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

3.57. GL_OES_texture_half_float

Supported Hardware

Series5/5XT, Series6 (ES2/3 Only)

Valid APIs

OpenGL ES 1.x, OpenGL ES 2.0

Extension Description

This extension adds the ability to use 16-bit floating point type (GL_HALF_FLOAT) representations of texture data as a readable texture format when shading an object. This extension does not add linear sampling from this texture type however due to the high cost of such an operation, which is instead enabled by GL_OES_texture_half_float_linear. This extension also does not allow these textures to be used as a colour attachment for a framebuffer. The extension GL_EXT_color_buffer_half_float allows half float textures to be attached to a framebuffer.

The half-float data format consists of 1 sign bit, 5 exponent bits and 10 mantissa bits, and allows greater range than an integer representation would, whilst reducing the memory footprint to that of a short data type.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_texture_float.txt

Example Code

```
// Create a floating point texture
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 1024, 1024, 0, GL_RGBA, GL_HALF_FLOAT_OES, pixelData);

// Only nearest filtering is supported for these textures for this extension. Linear filtering
// is enabled by separate extensions.
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

3.58. GL_OES_texture_mirrored_repeat

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x

Extension Description

This extension adds another wrap mode to OpenGL ES, similar to GL_REPEAT, but the image is flipped each time it is repeated. For horizontal repeats, the texture is flipped horizontally, and for vertical repeats it is flipped vertically. The benefit of this is that it means tiled textures can be created without having to worry about making the edges match up.

Note

This extension is part of the OpenGL ES 1.x Extension Pack Specification and is core functionality in OpenGL ES 2.0 and 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_texture_mirrored_repeat.txt

Example Code

```
// Set the wrap mode for a texture to mirrored repeat
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
```

3.59. GL_OES_texture_npot

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x, OpenGL ES 2.0

Extension Description

This extension removes almost all of the limitations surrounding non-power of two textures in OpenGL ES 1.x and 2.0. MIP Map specification is now allowed, and minification filters that include a MIP Map filter will now work as expected for Power of Two textures. All available wrap modes will now work as well, rather than just GL_CLAMP. The only restriction that this extension does not lift is that in OpenGL ES 1.x, glGenerateMIPMaps will not work unless GL_OES_framebuffer_object is also supported.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_texture_npot.txt

Example Code

```
// Upload a texture with dimensions of 15 by 47. Typically this isn't supported without
// extension support.
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 15, 47, 0, GL_RGBA, GL_UNSIGNED_BYTE, pixelData);

// This extension lifts any restrictions on non-power of two textures, and all filter/wrap
// modes can be used.
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

3.60. GL_OES_vertex_array_object

Supported Hardware

Series5/5XT, Series6

Valid APIs

OpenGL ES 1.x, OpenGL ES 2.0

Extension Description

This extension provides a level of encapsulation for bound vertex state such as that set by, for example 'glVertexPointer'. With this extension, any modifications to the vertex array state will bind specifically to the bound VAO, rather than being bound directly to the context. Changing between bound VAOs will modify the vertex state to whatever was last set within the newly bound VAO. This allows users to quickly switch between various vertex states, allowing them to, for example, switch between different model objects with far fewer API calls than are traditionally needed. To maintain compatibility with the Core ES APIs, this extension employs a "default VAO". What this means is that rather than the object name '0' being a special "nothing bound" name as in most OpenGL objects, it is instead a fully usable VAO.

More specifically, vertex array objects all the state set by the following methods:

ES1: gl*Pointer(), glEnableClientState(), glBindBuffer()*

ES2: glVertexAttribPointer(), glEnableVertexAttribArray(), glBindBuffer()*

*VAOs only store the bound GL_ELEMENT_ARRAY_BUFFER. Any binding to GL_ARRAY_BUFFER or other buffer types are separate and distinct from VAOs.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_vertex_array_object.txt

Example Code

```
// Generate a vertex array object
GLuint vertexArray;
glGenVertexArraysOES(1, &vertexArray);

// Bind the vertex array object
glBindVertexArrayOES(vertexArray);

// Set some vertex attribute pointers
glVertexAttribPointer(0, 4, GL_FLOAT, GL_TRUE, 0, dataPointer);

// Rebind the default VAO
glBindVertexArrayOES(0);
```

3.61. GL_OES_vertex_half_float

Supported Hardware

Series5/5XT, Series6 (ES2/3 Only)

Valid APIs

OpenGL ES 1.x, OpenGL ES 2.0

Extension Description

This extension adds the ability to use 16-bit floating point (GL_HALF_FLOAT) representations of vertex data when uploading it to GL. The half-float data format consists of 1 sign bit, 5 exponent bits and 10 mantissa bits, and allows greater range than an integer representation would whilst reducing the memory footprint to that of a short data type.

Note

This functionality is core to OpenGL ES 3.0.

Registry Link

http://www.khronos.org/registry/gles/extensions/OES/OES_vertex_half_float.txt

Example Code

```
// Set a vertex attribute to sample vertices as half float values.
glVertexAttribPointer(0, 4, GL_HALF_FLOAT_OES, GL_TRUE, 0, dataPointer);
```

4. Contact Details

For further support contact:

devtech@imgtec.com

PowerVR Developer Technology
Imagination Technologies Limited
Home Park Estate
Kings Langley
Herts, WD4 8LZ
United Kingdom

Tel: +44 (0) 1923 260511
Fax: +44 (0) 1923 277463

Alternatively, you can use the PowerVR Insider forums:

www.imgtec.com/forum

For more information about PowerVR or Imagination Technologies Limited visit our web pages at:

www.imgtec.com