OpenGL Extensions Guide

Graphics & Animation: 3D Drawing

2010-02-24

Contents

OpenGL Extensions Guide

Companion guide OpenGL Programming Guide for Mac OS X

Overview

OpenGL Extensions Guide is a reference for the extensions that Mac OS X supports. The guide provides a short summary of each extension, a link to the official specification, availability information, and hardware renderer support. **Availability** refers to the version of the Mac OS that introduces the extension. **Renderers** is a list of the renderers that, at the time of the extension's introduction in Mac OS X, supported the extension. To get complete availability and renderer information for a particular extension, you should also consult OpenGL Capabilities. This multidimensional table lists extensions and parameter values by the Mac OS X versions, renderers (graphics adaptors), and CPU hardware that supports each.

As the OpenGL specification evolves, it introduces more extensions that can you use to optimize performance or add cool effects to your OpenGL application. In addition to looking at the specification for an extension, you may want to consult the appropriate OpenGL specification that introduced that extension. Keep in mind that each version of the OpenGL specification modifies the previous one.

- OpenGL 1.1 Specification adds vertex arrays, polygon offset, logical operations, texture image formats, texture replace environment, texture proxies, copy texture and subtexture, and a number of other minor changes to the base OpenGL 1.0 specification.

  Availability: Available in Mac OS X v10.0 and later.

  Renderers: All

- OpenGL 1.2.1 Specification (PDF) adds 3D texturing, BGRA pixel formats, packed pixel formats, normal rescaling, separate specular color, texture coordinate edge clamping, texture LOD control, vertex array draw element range, and the imaging subset to the OpenGL 1.1 specification. (The imaging subset is optional. See `GL_ARB_imaging` (page 19).)

  Availability: Available in Mac OS X v10.0 and later.

  Renderers: Radeon, Radeon Mobility, Radeon 7500 Mobility, Radeon 8500, Radeon 9000, Radeon 9200, Radeon 9600, Radeon 9800, GeForce 3, GeForce 4Ti, GeForce F

- OpenGL 1.3 Specification (PDF) adds Compressed Textures, Cube Map Textures, Multisample, Multitexture, Texture Add Environment Mode, Texture Combine Environment Mode, Texture Dot3 Environment Mode, Texture Border Clamp, and Transpose Matrix to the OpenGL 1.2 specification.

  Availability: Available in Mac OS X v10.1 and later.

  Renderers: Radeon, Radeon Mobility, Radeon 7500 Mobility, Radeon 8500, Radeon 9000, Radeon 9200, Radeon 9600, Radeon 9800, GeForce 3, GeForce 4Ti, GeForce FX

- OpenGL 1.4 Specification (PDF) adds automatic Mipmap Generation, Blend Squaring, Imaging Subset changes, Depth and Shadow textures, Fog Coordinates, Multiple Draw Arrays, Point Parameters, Secondary Color, Separate Blend Functions, Stencil Wrap, Texture Crossbar Environment Mode, Texture level of detail (LOD) bias, Texture Mirrored Repeat, and Window Raster Position.

  Availability: Available in Mac OS X v10.2.3 and later.

  Renderers:

- OpenGL 1.5 Specification (PDF) adds Buffer Objects, Occlusion Queries, Shadow Functions, as well as introducing the ARB_shader_objects, ARB_vertex_shader and ARB_fragment_shader extensions.

  Availability: Available in Mac OS X v10.3.9 and later.

  Renderers: GeForce 6800, GeForce FX, Radeon X800, Radeon 9600/9700/9800

- OpenGL 2.0 Specification (PDF) adds Shader Objects, Shader Programs, and the OpenGL Shading Language as core features. It also adds Multiple Render Targets, Non-Power-Of-Two Textures, Point Sprites, Separate Blend Equations and Separate Stencils.

  Availability: Available in Mac OS X v10.4.4 and later.

  Renderers: Radeon X1600/X1900, Quadro FX 4500, GeForce 6800 or better.

- OpenGL 2.1 Specification (PDF) adds Shader Language 1.20, Non-Square Matrices, Pixel Buffer Objects, and sRGB Textures.

  Availability: Available in Mac OS X v10.5 and later.

  Renderers: Apple Software Renderer

OpenGL Extensions by Task

Working With Vertex Data

`GL_ARB_vertex_buffer_object` (page 25)
: Increases data transfer rate by caching data in high-performance graphics memory on the server.

`GL_EXT_transform_feedback` (page 34)
: Defines a transform feedback mode for recording vertex attributes for each primitive that OpenGL processes.

`GL_ATI_array_rev_comps_in_4_bytes` (page 26)
: Provides an optimized data transfer path for rendering vertex array data on certain ATI hardware when individual components are smaller than 4 bytes per component.

`GL_ARB_vertex_blend` (page 25)
: Provides the ability to replace the single modelview transformation with a set of *n* vertex units.

`GL_APPLE_vertex_array_object` (page 19)
: Handles multiple vertex arrays as array objects, similar to texture objects.

`GL_EXT_multi_draw_arrays` (page 31)
: Provides function for handling multiple lists of vertices in one call.

`GL_APPLE_vertex_array_range` (page 19)
: Specifies to use client memory for vertex arrays.

`GL_APPLE_vertex_program_evaluators` (page 19)
: Supports the use of one- and two-dimensional evaluators with vertex program attributes.

`GL_EXT_draw_range_elements` (page 29)
: Adds a vertex array rendering command (`glDrawRangeElementsEXT`) that is a restricted form of the `glDrawElements` command.

`GL_APPLE_element_array` (page 17)
: Improve draw-elements style vertex indices submission performance by allowing index arrays.

`GL_EXT_compiled_vertex_array` (page 29)
: Allows caching or precompiling static vertex array for more efficient rendering.

Working With Textures

`GL_ARB_texture_float` (page 24)
: Extends internal formats for textures that have 16- and 32-bit floating-point components.

`GL_ARB_texture_non_power_of_two` (page 24)
: Relaxes the size restrictions for the 1D, 2D, cube map, and 3D texture targets.

`GL_ARB_texture_rectangle` (page 24)
: Adds a new texture target that supports 2D textures without requiring power-of-two dimensions.

`GL_EXT_texture_compression_dxt1` (page 32)
: Provides compressed textures that allow for significantly reduced texture storage.

`GL_APPLE_texture_range` (page 19)
: Allows application to provide hints for texture storage and to specify a memory range for texture data.

`GL_EXT_texture_mirror_clamp` (page 33)
: Extends texture wrapping to include three mirroring modes.

`GL_EXT_texture_sRGB` (page 34)
: Supports the sRGB color space for textures.

`GL_ARB_texture_mirrored_repeat` (page 24)
: Extends the set of texture wrap modes to include a mirrored repeat mode.

`GL_APPLE_client_storage` (page 17)
: Allows applications to cache textures locally for use by OpenGL.

`GL_ARB_texture_env_combine` (page 23)
: Adds a texture environment function that lets you combine texture operations.

`GL_ARB_texture_env_crossbar` (page 24)
: Adds the capability to use the texture color from other texture units as sources to the `GL_COMBINE_ARB` environment function.

`GL_ARB_texture_cube_map` (page 23)
: Provides a texture generation scheme for cube map textures, where the current texture is a set of six 2-dimensional images that represent the faces of a cube.

`GL_ARB_texture_border_clamp` (page 23)
: Adds a texture clamping algorithm for clamping texture coordinates at all mipmap levels such that the `GL_NEAREST` and `GL_LINEAR` filters return only the color of the border texels.

`GL_ARB_texture_env_add` (page 23)
: Adds support for the texture environment function `GL_ADD`.

`GL_ATI_texture_compression_3dc` (page 27)

`GL_ARB_texture_compression` (page 23)
: Provides a framework and formats for compressed textures.

`GL_EXT_texture_lod_bias` (page 33)
: Allows adding a bias value to the texture level-of-detail parameter.

`GL_EXT_texture_env_add` (page 33)
: Add a texture environment function for adding textures.

`GL_SGIS_texture_edge_clamp` (page 37)
: Defines an algorithm that clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel.

`GL_EXT_texture_compression_s3tc` (page 32)
: Adds texture compression functionality specific to the S3 S3TC format.

`GL_ARB_texture_compression_rgtc` (page 23)
: Provides new texture compression formats suitable for red and red-green textures.

`GL_ATI_texture_env_combine3` (page 27)
: Adds texture combination operations, including `GL_MODULATE_ADD`, `GL_MODULATE_SIGNED_ADD`, and `GL_MODULATE_SUBTRACT`.

`GL_EXT_paletted_texture` (page 31)
: Adds texture formats and calls that support paletted textures.

`GL_EXT_shared_texture_palette` (page 32)
: Defines a shared texture palette to use in place of the texture object palettes provided by the GL_EXT_paletted_texture extension.

`GL_ATI_texture_mirror_once` (page 27)
: Extends the set of texture wrap modes to include two modes that effectively use a texture map twice as large as the original image in which the additional half of the new image is a mirror image of the original image.

`GL_EXT_texture_rectangle` (page 34)
: Adds a texture target that supports 2D textures without requiring power-of-two dimensions.

`GL_ARB_texture_env_dot3` (page 24)
: Adds dot product operations for textures.

`GL_EXT_texture_filter_anisotropic` (page 33)
: Provides support for anisotropic texturing filtering schemes without specifying an anisotropic filtering formula.

`GL_SGIS_texture_lod` (page 37)
: Imposes two constraints related to the texture level of detail parameter.

`GL_ARB_multitexture` (page 21)
: Adds support for multiple texture units.

Using Pixel Buffers

`GL_ARB_pixel_buffer_object` (page 21)
: Allows you to use buffer objects with pixel data.

`GL_APPLE_pixel_buffer` (page 18)
: Supports offscreen buffers for accelerated rendering and texturing.

Using Framebuffer Objects

`GL_EXT_framebuffer_object` (page 30)
: Provides an offscreen buffer for rendering.

`GL_EXT_framebuffer_multisample` (page 30)
: Extends framebuffers to allow multisampling.

`GL_EXT_framebuffer_blit` (page 30)
: Uses separate framebuffer bindings for drawing and reading and defines a function for transferring data between them.

Optimizing Flushing

`GL_APPLE_fence` (page 17)
: Defines primitives that you can insert into the OpenGL command stream to monitor command completion.

`GL_APPLE_object_purgeable` (page 18)
: Supports marking storage for OpenGL objects as purgeable or nonpurgeable.

`GL_APPLE_flush_render` (page 17)
: Submits pending OpenGL commands but does not copy the results to the screen.

`GL_APPLE_flush_buffer_range` (page 17)
: Supports flushing a subrange of a buffer object.

Working With Depth and Stencil Buffers

`GL_EXT_packed_depth_stencil` (page 31)
: Supports interleaving the depth and stencil buffers into one buffer.

`GL_EXT_depth_bounds_test` (page 29)
: Adds a test for deciding whether to discard a fragment based on a user-defined minimum and maximum depth value.

`GL_APPLE_aux_depth_stencil` (page 16)
: Allocates a separate depth buffer for the color buffer and for each auxiliary buffer.

`GL_ARB_depth_texture` (page 20)
: Defines a depth texture format to use for shadow casting.

`GL_NV_depth_clamp` (page 35)
: Supports rasterizing line and polygon primitives without clipping the primitive to the near or far clip volume planes.

`GL_EXT_stencil_wrap` (page 32)
: Defines two stencil operations that wrap the result.

`GL_EXT_stencil_two_side` (page 32)
: Provides two-sided stencil testing.

`GL_ATI_separate_stencil` (page 26)
: Provides the ability to modify the stencil buffer based on the facing direction of the primitive that generates the fragment.

`GL_ARB_occlusion_query` (page 21)
: Supports querying the number of samples that a primitive or group of primitives draws.

`GL_NV_conditional_render` (page 34)
: Allows a program to conditionally execute rendering commands based on an occlusion query.

`GL_EXT_clip_volume_hint` (page 29)
: Defines hints that indicate whether the application requires volume clipping for primitives.

`GL_ATI_point_cull_mode` (page 26)

Working With Lighting, Fog, and Shadow Effects

`GL_EXT_separate_specular_color` (page 31)
: Adds a second color to rasterization only if you have enabled lighting.

`GL_APPLE_specular_vector` (page 18)
: Provides an alternative lighting model that produces results similar to, and often more realistic than, the existing lighting model.

`GL_NV_light_max_exponent` (page 35)
: Extends the maximum shininess and spot exponent beyond `128.0`.

`GL_NV_texgen_reflection` (page 36)
: Provides two new texture coordinate generation modes for texture-based lighting and environment mapping.

`GL_ARB_shadow` (page 22)
: Produces a Boolean texture value by comparing the texture *R* coordinate to a depth texture value.

`GL_ARB_shadow_ambient` (page 22)
: Supports ambient and shadow lighting without the need for multiple texture units.

`GL_ARB_point_parameters` (page 22)
: Supports additional geometric characteristics of points.

`GL_EXT_shadow_funcs` (page 32)
: Supports eight binary texture comparison functions

`GL_EXT_fog_coord` (page 29)
: Supports explicit per-vertex fog coordinates for fog computations.

`GL_NV_fog_distance` (page 35)
: Supports application-control of fog distance computations.

`GL_EXT_secondary_color` (page 31)
: Supports application-control of the RGB components of the secondary color used in the color-summation stage.

Using Vertex Programs

`GL_ARB_vertex_program` (page 25)
: Supports application-defined programs for computing vertex parameters.

`GL_NV_vertex_program2_option` (page 36)
: Extends the standard `ARB_vertex_program` language and execution environment.

`GL_NV_vertex_program3` (page 36)
: Provides additional vertex program functionality to extend the standard `ARB_vertex_program` language and execution environment.

Using Fragment Programs

`GL_ARB_fragment_program_shadow` (page 20)
: Removes the interaction between the `GL_ARB_fragment_program` (page 20) and `GL_ARB_shadow` (page 22) extensions.

`GL_NV_fragment_program2` (page 35)
: Provides additional fragment program functionality to extend the `GL_ARB_fragment_program` (page 20) specification.

`GL_NV_fragment_program_option` (page 35)
: Provides additional fragment program functionality to extend the standard `GL_ARB_fragment_program` (page 20) language and execution environment.

`GL_ARB_fragment_shader` (page 20)
: Defines fragment shader objects.

`GL_ARB_draw_buffers` (page 20)
: Extends fragment programs and shaders to allow multiple output colors, and for directing those outputs to multiple color buffers.

`GL_EXT_draw_buffers2` (page 29)
: Provides separate blend and write-masks for each color output.

`GL_ARB_fragment_program` (page 20)
: Supports programs that compute fragment parameters.

`GL_ATI_text_fragment_shader` (page 27)
: Defines a fragment processing model for expressing fragment color blending and dependent texture address modification.

Using Shader Objects

`GL_ARB_shading_language_100` (page 22)
: Indicates support for OpenGL Shading Language.

`GL_ARB_shader_objects` (page 22)
: Add support for shader and program objects.

`GL_ARB_shader_texture_lod` (page 22)
: Provides shader writers explicit control of level of detail for texture operations.

`GL_ARB_vertex_shader` (page 25)
: Adds programmable vertex-level processing.

`GL_EXT_gpu_shader4` (page 31)
: Extends the OpenGL Shading Language to support recently added hardware capabilities.

`GL_EXT_geometry_shader4` (page 30)
: Defines a shader for programmatically generating primitives.

`GL_NV_texture_shader` (page 36)
: Provides defines texture shader stage for mapping sets of texture coordinates to filtered colors.

`GL_NV_texture_shader2` (page 36)
: Adds texture_shader functionality to support texture shader operations for 3D textures.

`GL_NV_texture_shader3` (page 36)
: Extends the `GL_NV_texture_shader` functionality.

Supporting Data Types and Formats

`GL_APPLE_ycbcr_422` (page 19)
: Provides support for 2vuy and 2yuv texture formats.

`GL_APPLE_rgb_422` (page 18)
: Exposes a raw Y'CbCr format for pixel data.

`GL_APPLE_packed_pixels` (page 18)
: Supports packed pixels in host memory.

`GL_ARB_half_float_pixel` (page 20)
: Introduces a data type for half-precision (16-bit) floating-point quantities.

`GL_ARB_half_float_vertex` (page 21)
: Allows a half-precision (16-bit) floating-point quantity to be used in vertex calculations.

`GL_ARB_color_buffer_float` (page 19)
: Adds floating-point pixel formats.

`GL_ATI_texture_float` (page 27)
: Adds texture internal formats with 32- and 16-bit floating-point components.

`GL_APPLE_float_pixels` (page 17)
: Adds texture types, texture internal formats and color buffers composed of both 32 bit and 16 floating point numbers.

`GL_ARB_texture_rg` (page 24)
: Adds one and two channel texture formats.

`GL_EXT_texture_integer` (page 33)
: Allows for true integer formats to be used in textures.

`GL_EXT_abgr` (page 27)
: Extends the list of host-memory color formats, providing a reverse-order alternative to image format RGBA.

`GL_EXT_bgra` (page 27)
: Extends the list of host-memory color formats

`GL_EXT_framebuffer_sRGB` (page 30)
: Allows framebuffers to be created with non-linear sRGB formats.

`GL_SGI_color_matrix` (page 37)
: Adds a 4x4 matrix stack to the pixel transfer path.

`GL_ARB_transpose_matrix` (page 25)
: Supports the transfer of application matrices stored in row major order to the OpenGL implementation.

`GL_NV_register_combiners` (page 35)
: Provides an extremely configurable mechanism know as "register combiners" for computing fragment colors.

`GL_NV_register_combiners2` (page 36)
: Extends the register combiners functionality to support more color constant values that are unique for each general combiner stage.

`GL_EXT_rescale_normal` (page 31)
: Adds a normal rescaling to the transformation of the normal vector into eye coordinates.

`GL_APPLE_transform_hint` (page 19)
: Provides a hint which allows Open GL to choose to implement certain state dependent algebraic simplifications in the geometry transformation.

Imaging and Blending

`GL_ARB_imaging` (page 21)
: Provides support for color tables, convolution, color matrix, histogram, constant blend color, blend subtract and blend min/max.

`GL_EXT_blend_equation_separate` (page 28)
: Defines a blend equations for separating RGB and alpha blend factors and for combining source and destination blend terms.

`GL_EXT_blend_color` (page 28)
: Defines a constant color to include in blending equations.

`GL_EXT_blend_minmax` (page 28)
: Defines two equations that produce the minimum (or maximum) color components of the source and destination colors.

`GL_EXT_blend_subtract` (page 28)
: Defines two blending equations that produce an effect based on the difference of two input value.

`GL_NV_blend_square` (page 34)
: Provides four additional blending factors.

`GL_EXT_blend_func_separate` (page 28)
: Adds a function that supports independent RGB and alpha blend factors.

`GL_ATI_blend_equation_separate` (page 26)

`GL_ATI_blend_weighted_minmax` (page 26)

Tessellation

`GL_ATI_pn_triangles` (page 26)
: Supports letting OpenGL internally tessellate input geometry internally into curved patches.

`GL_ATIX_pn_triangles` (page 26)

Binding and Loading Parameters

`GL_EXT_gpu_program_parameters` (page 30)
: Adds procedures that load multiple consecutive program environment parameters using a single call instead of multiple calls.

`GL_EXT_bindable_uniform` (page 28)
: Adds bindable uniform variables to the OpenGL shading language.

Multisampling and Using Mipmaps

`GL_SGIS_generate_mipmap` (page 37)
: Defines a mechanism by which OpenGL can derive the entire set of mipmap arrays when provided with only the base level array.

`GL_NV_multisample_filter_hint` (page 35)
: Provides a hint that permits implementations to provide an alternative method of resolving the color of multisampled pixels.

`GL_ARB_multisample` (page 21)
: Adds an antialiasing algorithm that samples multiple times at each pixel in a primitive.

Using Point Sprites

`GL_NV_point_sprite` (page 35)
: Allows applications to use points rather than quads for such things as particle systems.

`GL_ARB_point_sprite` (page 22)
: Provides support for point sprites.

Working With Raster Positions

`GL_ARB_window_pos` (page 26)
: Provides a set of functions to directly set the current raster position in window coordinates, bypassing the modelview matrix, the projection matrix and the viewport-to-window mapping.

`GL_IBM_rasterpos_clip` (page 34)
: Extends the semantics of the raster position functions.

OpenGL Extensions

GL_APPLE_aux_depth_stencil

Allocates a separate depth buffer for the color buffer and for each auxiliary buffer.

Discussion
When you use this extension, and the depth buffer size is nonzero, OpenGL automatically allocates a separate depth buffer for the color buffer and for each auxiliary buffer. Similarly, if the stencil buffer size is nonzero, OpenGL allocates a separate stencil buffer for the color buffer and each auxiliary buffer.

GL_APPLE_client_storage

Allows applications to cache textures locally for use by OpenGL.

Discussion
Provides facilities supplanting the OpenGL mechanism to update the working texture set, allowing and requiring applications to cache textures locally for use by OpenGL.

GL_APPLE_element_array

Improve draw-elements style vertex indices submission performance by allowing index arrays.

GL_APPLE_fence

Defines primitives that you can insert into the OpenGL command stream to monitor command completion.

Discussion
This extension lets you get fine-grained control over flushing. You can monitor whether OpenGL has executed a particular command and issue a finish command for a subset of the command stream. By using this extension, you can more efficiently synchronize the activity of the CPU and GPU, thereby avoiding unnecessary wait times.

GL_APPLE_float_pixels

Adds texture types, texture internal formats and color buffers composed of both 32 bit and 16 floating point numbers.

GL_APPLE_flush_buffer_range

Supports flushing a subrange of a buffer object.

Discussion
You can improve the performance of your application by using this extension when your application needs to write only to a subrange. This extension introduces two buffer object features: nonserialized buffer modification and explicit subrange flushing for buffer objects that are mapped.

GL_APPLE_flush_render

Submits pending OpenGL commands but does not copy the results to the screen.

Discussion

In single buffered mode the `glFlush` and `glFinish` functions submit the command stream and copy the resulting image to the screen. This extension defines two functions—`glFlushRenderAPPLE` and `glFinishRenderAPPLE`—that submit pending OpenGL commands but do *not* copy the results to the screen.

The extension also provides the function `glSwapAPPLE` which:

- Copies the rendered image for the current context to the screen without the need for context argument
- Works symmetrically in both single and double buffered modes.

GL_APPLE_object_purgeable

Supports marking storage for OpenGL objects as purgeable or nonpurgeable.

Discussion
Using this extension, you can eliminate unnecessary paging of resources. The typical options are to mark objects as `purgeable` and `released` or as `unpurgeable` and `retained`. OpenGL uses these settings as a guide when managing memory resources.

GL_APPLE_packed_pixels

Supports packed pixels in host memory.

Discussion
A packed pixel is represented entirely by one unsigned byte, one unsigned short, or one unsigned integer.

GL_APPLE_pixel_buffer

Supports offscreen buffers for accelerated rendering and texturing.

Discussion
You can use pixel buffers (also called pbuffers) to get accelerated performance when rendering to and texturing from a surface.

GL_APPLE_rgb_422

Exposes a raw Y'CbCr format for pixel data.

Discussion
It provides a new pixel format for unconverted video pixel data. Applications that use this are expected to provide a fragment shader that performs a color space transformation to RGB.

GL_APPLE_specular_vector

Provides an alternative lighting model that produces results similar to, and often more realistic than, the existing lighting model.

GL_APPLE_texture_range

Allows application to provide hints for texture storage and to specify a memory range for texture data.

GL_APPLE_transform_hint

Provides a hint which allows Open GL to choose to implement certain state dependent algebraic simplifications in the geometry transformation.

GL_APPLE_vertex_array_object

Handles multiple vertex arrays as array objects, similar to texture objects.

Discussion
This extension allows your application to change vertex array pointers more efficiently, by collecting them into an OpenGL object. Binding a vertex array object changes all of the vertex pointers with a single function call.

GL_APPLE_vertex_array_range

Specifies to use client memory for vertex arrays.

Discussion
When you use this extension, the GPU perform DMA transfers of your vertex data. Much of the benefit of this extension has been superseded by vertex buffer objects (`GL_ARB_vertex_buffer_object` (page 25)), which is part of the core OpenGL 1.5 specification.

GL_APPLE_vertex_program_evaluators

Supports the use of one- and two-dimensional evaluators with vertex program attributes.

Discussion
This extension operates comparable to normal evaluators.

GL_APPLE_ycbcr_422

Provides support for 2vuy and 2yuv texture formats.

GL_ARB_color_buffer_float

Adds floating-point pixel formats.

Discussion
This extension adds pixel formats with either 16-bit or 32-bit floating-point RGBA color components. It also provides a function to control how pixel data is clamped.

GL_ARB_depth_texture

Defines a depth texture format to use for shadow casting.

Discussion
You can also use this extension for image-based rendering or displacement mapping.

GL_ARB_draw_buffers

Extends fragment programs and shaders to allow multiple output colors, and for directing those outputs to multiple color buffers.

Discussion
This extension extends ARB_fragment_program and ARB_fragment_shader to allow multiple output colors, and provides a mechanism for directing those outputs to multiple color buffers.

For More Information
GL_ARB_fragment_shader
GL_ARB_fragment_program

GL_ARB_fragment_program

Supports programs that compute fragment parameters.

Discussion
A fragment program is a sequence of floating-point 4-component vector operations that determines how to transform a set of program parameters (not specific to an individual fragment) and an input set of per-fragment parameters to a set of per-fragment result parameters. You can write instruction sequences for fragment programs using the fragment programming model defined by this extension.

GL_ARB_fragment_program_shadow

Removes the interaction between the `GL_ARB_fragment_program` (page 20) and `GL_ARB_shadow` (page 22) extensions.

GL_ARB_fragment_shader

Defines fragment shader objects.

Discussion
A fragment shader object is a type of shader object that, when attached to a program object, you can compile and link to produce executable code that runs on the OpenGL fragment processor. See the ARB_shader_objects extension.

For More Information
GL_ARB_shader_objects

GL_ARB_half_float_pixel

Introduces a data type for half-precision (16-bit) floating-point quantities.

Discussion
The floating-point format is very similar to the IEEE single-precision floating-point standard, except that it has only 5 exponent bits and 10 mantissa bits.

GL_ARB_half_float_vertex

Allows a half-precision (16-bit) floating-point quantity to be used in vertex calculations.

Discussion
Allows the half-float pixel format introduced in the `GL_ARB_half_float_pixel` (page 20) extension to be used when specifying vertex data or calculations.

GL_ARB_imaging

Provides support for color tables, convolution, color matrix, histogram, constant blend color, blend subtract and blend min/max.

Discussion
Complete imaging subset providing: Color Tables, Convolution, Color Matrix, Histogram, Constant Blend Color, Blend Subtract and Blend Min/Max.

GL_ARB_multisample

Adds an antialiasing algorithm that samples multiple times at each pixel in a primitive.

Discussion
You can use multisampling on all Open GL primitives: points, lines, polygons, bitmaps, and images.

GL_ARB_multitexture

Adds support for multiple texture units.

GL_ARB_occlusion_query

Supports querying the number of samples that a primitive or group of primitives draws.

GL_ARB_pixel_buffer_object

Allows you to use buffer objects with pixel data.

Discussion
This extension expands on the interface provided by the GL_ARB_vertex_buffer_object extension (and later integrated into OpenGL 1.5). It lets you use buffer objects to pixel data as well as vertex data. By using this extension, increase the speed at which OpenGL executes pixel commands.

GL_ARB_point_parameters

Supports additional geometric characteristics of points.

Discussion
You can use this extension to render particles or tiny light sources, commonly referred to as **light points**.

GL_ARB_point_sprite

Provides support for point sprites.

Discussion
Applications such as particle systems have tended to use OpenGL quads rather than points to render their geometry, since they would like to use a custom-drawn texture for each particle, rather than the traditional OpenGL round antialiased points, and each fragment in a point has the same texture coordinates as every other fragment.

GL_ARB_shader_objects

Add support for shader and program objects.

Discussion
This extension adds calls that are necessary to manage shader objects and program objects as defined in the OpenGL 2.0 white papers by 3Dlabs.

GL_ARB_shader_texture_lod

Provides shader writers explicit control of level of detail for texture operations.

GL_ARB_shading_language_100

Indicates support for OpenGL Shading Language.

GL_ARB_shadow

Produces a Boolean texture value by comparing the texture *R* coordinate to a depth texture value.

Discussion
You can use this extension to implement shadow maps.

GL_ARB_shadow_ambient

Supports ambient and shadow lighting without the need for multiple texture units.

Discussion
When you use this extension, you need to specify the texture value to use when the texture comparison function fails. Normally this value is zero. By allowing an arbitrary value you can get functionality which otherwise requires you to use multiple texture units and an an advanced texture combine extension (such as that provided by the extension GL_NV_register_combiners).

GL_ARB_texture_border_clamp

Adds a texture clamping algorithm for clamping texture coordinates at all mipmap levels such that the `GL_NEAREST` and `GL_LINEAR` filters return only the color of the border texels.

Discussion
The clamping algorithm is `GL_CLAMP_TO_BORDER_ARB`.

GL_ARB_texture_compression

Provides a framework and formats for compressed textures.

Discussion
Allows OpenGL applications to use compressed texture images by providing both a framework upon which extensions providing specific compressed image formats can be built and a set of generic compressed internal formats that allow applications to specify that texture images should be stored in compressed form without needing to code for specific compression formats.

GL_ARB_texture_compression_rgtc

Provides new texture compression formats suitable for red and red-green textures.

Discussion
These formats are optimized to reduce the storage requirements of red or red-green textures.

GL_ARB_texture_cube_map

Provides a texture generation scheme for cube map textures, where the current texture is a set of six 2-dimensional images that represent the faces of a cube.

GL_ARB_texture_env_add

Adds support for the texture environment function `GL_ADD`.

Discussion
This extension implements the following equation:

```
Cv = Cf + Ct
```

GL_ARB_texture_env_combine

Adds a texture environment function that lets you combine texture operations.

Discussion
The function `GL_COMBINE_ARB` lets you program combine operations: `GL_REPLACE`, `GL_MODULATE`, `GL_ADD`, `GL_ADD_SIGNED_ARB`, `GL_SUBTRACT_ARB`, and `GL_INTERPOLATE_ARB`.

GL_ARB_texture_env_crossbar

Adds the capability to use the texture color from other texture units as sources to the `GL_COMBINE_ARB` environment function.

GL_ARB_texture_env_dot3

Adds dot product operations for textures.

Discussion
You can supply these operation to a texture combination function: `GL_DOT3_RGB_ARB` and `GL_DOT3_RGBA_ARB`.

GL_ARB_texture_float

Extends internal formats for textures that have 16- and 32-bit floating-point components.

Discussion
The 32-bit floating-point components are in the standard IEEE float format. The 16-bit floating-point components have 1 sign bit, 5 exponent bits, and 10 mantissa bits. The extension clamps floating-point components to the limits of the range represented by their format.

GL_ARB_texture_mirrored_repeat

Extends the set of texture wrap modes to include a mirrored repeat mode.

Discussion
The `GL_MIRRORED_REPEAT_ARB` texture wrap mode effectively uses a texture map that is twice as large at the original image to accommodate the original image and its mirror image.

GL_ARB_texture_non_power_of_two

Relaxes the size restrictions for the 1D, 2D, cube map, and 3D texture targets.

Discussion
Conventional OpenGL texturing is limited to images with power-of-two dimensions and an optional 1-texel border.

GL_ARB_texture_rectangle

Adds a new texture target that supports 2D textures without requiring power-of-two dimensions.

Discussion
Without this extension, OpenGL limits textures to images that have power-of-two dimensions and an optional 1-texel border.

GL_ARB_texture_rg

Adds one and two channel texture formats.

Discussion
Adds red and red-green texture formats optimized for use in shaders. A typical use for these formats is for luminance or intensity values.

GL_ARB_transpose_matrix

Supports the transfer of application matrices stored in row major order to the OpenGL implementation.

GL_ARB_vertex_blend

Provides the ability to replace the single modelview transformation with a set of *n* vertex units.

Discussion
The number of vertex units is constrained to an implementation-defined maximum. Each unit having its own modelview transform matrix and weight that is used to scale and sum the final eye-space vertex.

GL_ARB_vertex_buffer_object

Increases data transfer rate by caching data in high-performance graphics memory on the server.

Discussion
Although this extension is typically used for vertex arrays, the same API is used by other extensions to allow you to use buffer objects to cache other types of data.

GL_ARB_vertex_program

Supports application-defined programs for computing vertex parameters.

Discussion
A vertex program is a sequence of floating-point 4-component vector operations that determines how a set of program parameters (defined outside of the OpenGL `glBegin` and `glEnd` command pair) and an input set of per-vertex parameters are transformed to a set of per-vertex result parameters.

GL_ARB_vertex_shader

Adds programmable vertex-level processing.

Discussion
Using this extension, you can write vertex shaders in a high level language as defined in the OpenGL Shading Language specification. The language itself is not discussed here. A vertex shader replaces the transformation, texture coordinate generation and lighting parts of OpenGL, and it also adds texture access at the vertex level. Furthermore, management of vertex shader objects and loading generic attributes are discussed. A vertex shader object, attached to a program object, can be compiled and linked to produce an executable that runs on the vertex processor in OpenGL. This extension also defines how such an executable interacts with the fixed functionality vertex processing of OpenGL 1.4.

GL_ARB_window_pos

Provides a set of functions to directly set the current raster position in window coordinates, bypassing the modelview matrix, the projection matrix and the viewport-to-window mapping.

Discussion
Furthermore, clip testing is not performed, so that the current raster position is always valid.

GL_ATIX_pn_triangles

GL_ATI_array_rev_comps_in_4_bytes

Provides an optimized data transfer path for rendering vertex array data on certain ATI hardware when individual components are smaller than 4 bytes per component.

GL_ATI_blend_equation_separate

GL_ATI_blend_weighted_minmax

GL_ATI_pn_triangles

Supports letting OpenGL internally tessellate input geometry internally into curved patches.

Discussion
Using this extension, you can produce smoother, more organic looking geometry. You can control the amount of tessellation to apply to each triangle using a global stat variable.

GL_ATI_point_cull_mode

GL_ATI_separate_stencil

Provides the ability to modify the stencil buffer based on the facing direction of the primitive that generates the fragment.

Discussion

GL_ATI_texture_compression_3dc

Discussion

GL_ATI_texture_env_combine3

Adds texture combination operations, including `GL_MODULATE_ADD`, `GL_MODULATE_SIGNED_ADD`, and `GL_MODULATE_SUBTRACT`.

Discussion
This extension requires the GL_ARB_texture_env_combine extension.

GL_ATI_texture_float

Adds texture internal formats with 32- and 16-bit floating-point components.

Discussion
The 32 bit floating-point components are in the standard IEEE float format. The 16 bit floating-point components have 1 sign bit, 5 exponent bits, and 10 mantissa bits. Floating-point components are clamped to the limits of the range representable by their format.

GL_ATI_texture_mirror_once

Extends the set of texture wrap modes to include two modes that effectively use a texture map twice as large as the original image in which the additional half of the new image is a mirror image of the original image.

Discussion
The modes are `GL_MIRROR_CLAMP_ATI` and `GL_MIRROR_CLAMP_TO_EDGE_ATI`. Using this extension, you can use images whose edges don't match.

GL_ATI_text_fragment_shader

Defines a fragment processing model for expressing fragment color blending and dependent texture address modification.

GL_EXT_abgr

Extends the list of host-memory color formats, providing a reverse-order alternative to image format RGBA.

GL_EXT_bgra

Extends the list of host-memory color formats

Discussion
This extension provides formats that, when reversed, match the memory layout of Mac OS CGrafPort and GWorld data types so that applications can use the same data in both Mac OS API calls and OpenGL pixel API calls.

GL_EXT_bindable_uniform

Adds bindable uniform variables to the OpenGL shading language.

Discussion
A bindable uniform variable uses storage that is not allocated by the compiler or linker, but is instead backed by a buffer object.

GL_EXT_blend_color

Defines a constant color to include in blending equations.

GL_EXT_blend_equation_separate

Defines a blend equations for separating RGB and alpha blend factors and for combining source and destination blend terms.

Discussion
`EXT_blend_func_separate` introduced separate RGB and alpha blend factors. `EXT_blend_minmax` introduced a distinct blend equation for combining source and destination blend terms. (`EXT_blend_subtract` and `EXT_blend_logic_op` added other blend equation modes.) OpenGL 1.4 integrated both functionalities into the core standard.

GL_EXT_blend_func_separate

Adds a function that supports independent RGB and alpha blend factors.

Discussion
Using this extension, you can independently set the RGB and alpha blend factors for blend operations that require source and destination blend factors.

GL_EXT_blend_minmax

Defines two equations that produce the minimum (or maximum) color components of the source and destination colors.

GL_EXT_blend_subtract

Defines two blending equations that produce an effect based on the difference of two input value.

Discussion
Using this extension, you can call the `BlendEquationEXT` function with either of these modes: `FUNC_SUBTRACT_EXT` or `FUNC_REVERSE_SUBTRACT_EXT`.

GL_EXT_clip_volume_hint

Defines hints that indicate whether the application requires volume clipping for primitives.

Discussion

GL_EXT_compiled_vertex_array

Allows caching or precompiling static vertex array for more efficient rendering.

GL_EXT_depth_bounds_test

Adds a test for deciding whether to discard a fragment based on a user-defined minimum and maximum depth value.

Discussion
OpenGL performs the depths bounds test on each fragment, after the scissor test and before the alpha test. The test compares the depth value stored at the location given by the coordinates of the incoming fragment coordinates (*xw*, `yw`) to a minimum and maximum depth value that you supply. If the stored depth value is outside the range (exclusive), OpenGL discards the incoming fragment.

GL_EXT_draw_buffers2

Provides separate blend and write-masks for each color output.

Discussion
This extension builds on the behavior of `GL_ARB_draw_buffers` (page 20) by allowing separate masks and blends to apply to each color output. The same blend operation is still applied to all outputs.

GL_EXT_draw_range_elements

Adds a vertex array rendering command (`glDrawRangeElementsEXT`) that is a restricted form of the `glDrawElements` command.

Discussion
Calling `glDrawRangeElementsEXT` requires your application to specify the range of possible indices that are referenced in the draw command. By reducing the range of possible indices, some OpenGL hardware renderers can process the vertex data more efficiently.

GL_EXT_fog_coord

Supports explicit per-vertex fog coordinates for fog computations.

Discussion
You can use this extension as an alternative to a using a fragment depth-based fog equation.

GL_EXT_framebuffer_blit

Uses separate framebuffer bindings for drawing and reading and defines a function for transferring data between them.

Discussion
This extension modifies the `GL_EXT_framebuffer_object` (page 30) extension by splitting the framebuffer object binding point into separate bindings for drawing and reading. Using this extension, you can copy directly from one framebuffer to another. It also adds the `BlitFramebufferEXT` function, which transfers a rectangular array of pixel values from one region of the source framebuffer to another in the destination framebuffer. This function can also perform data conversion where allowed.

GL_EXT_framebuffer_multisample

Extends framebuffers to allow multisampling.

Discussion
This extension alters `GL_EXT_framebuffer_object` (page 30) to allow multisample buffers. An application can use this extension to exercise more direct control over antialiasing operations in their programs.

GL_EXT_framebuffer_object

Provides an offscreen buffer for rendering.

Discussion
Framebuffers give you an alternative to using the buffers provided by the windowing system to OpenGL.

GL_EXT_framebuffer_sRGB

Allows framebuffers to be created with non-linear sRGB formats.

Discussion
The sRGB format is a standards-driven, non-linear color space that roughly corresponds to the 2.2 gamma correction. This extension allows framebuffer objects to be created and manipulated in the sRGB color space.

GL_EXT_geometry_shader4

Defines a shader for programmatically generating primitives.

Discussion
OpenGL executes geometry shaders after transforming vertices, but prior to color clamping, flat shading, and clipping.

GL_EXT_gpu_program_parameters

Adds procedures that load multiple consecutive program environment parameters using a single call instead of multiple calls.

Discussion
By using this extension, you can reduce the amount of CPU overhead involved in loading parameters.

GL_EXT_gpu_shader4

Extends the OpenGL Shading Language to support recently added hardware capabilities.

GL_EXT_multi_draw_arrays

Provides function for handling multiple lists of vertices in one call.

Discussion
These functions behave identically to the standard OpenGL 1.1 functions `glDrawArrays` and `glDrawElements` except that they handle multiple lists of vertices in one call. Using this extension you can use one function l to render more than one primitive such as triangle strip, triangle fan, and so on.

GL_EXT_packed_depth_stencil

Supports interleaving the depth and stencil buffers into one buffer.

Discussion
Typically this extension interleaves depth and stencil buffers with 24 bits of depth precision and 8 bits of stencil data.

GL_EXT_paletted_texture

Adds texture formats and calls that support paletted textures.

Discussion
A paletted texture consists of a palette of colors and image data that specifies indices into the color palette. Using this extension, you can reduce the amount a data needed to define a texture.

GL_EXT_rescale_normal

Adds a normal rescaling to the transformation of the normal vector into eye coordinates.

Discussion
The normal vector is rescaled after it is multiplied by the inverse modelview matrix and before it is normalized.

GL_EXT_secondary_color

Supports application-control of the RGB components of the secondary color used in the color-summation stage.

Discussion
The default color is `(0,0,0,0)`. You can use this extension only in RGBA mode and when the `GL_LIGHTING` parameter is disabled.

GL_EXT_separate_specular_color

Adds a second color to rasterization only if you have enabled lighting.

Discussion
This extension works only when you have RGBA lighting enabled. You use this extension to produce textured objects that have specular highlights that are the color of the lights.

GL_EXT_shadow_funcs

Supports eight binary texture comparison functions

Discussion
This extension generalizes the `GL_ARB_shadow` (page 22) extension to support all eight binary texture comparison functions rather than just the `GL_LEQUAL` and `GL_GEQUAL` functions.

GL_EXT_shared_texture_palette

Defines a shared texture palette to use in place of the texture object palettes provided by the GL_EXT_paletted_texture extension.

GL_EXT_stencil_two_side

Provides two-sided stencil testing.

Discussion
The stencil-related state (stencil operations, reference value, compare mask, and write mask) can be different for front- and back-facing polygons.

GL_EXT_stencil_wrap

Defines two stencil operations that wrap the result.

Discussion
The new operations are similar to what the `GL_INCR` and `GL_DECR` parameters specify, but they wrap the result instead of saturating it.

GL_EXT_texture_compression_dxt1

Provides compressed textures that allow for significantly reduced texture storage.

Discussion
Reducing texture storage is advantageous because of the smaller memory capacity of many embedded systems compared to desktop systems. Smaller textures also provide a welcome performance advantage since embedded platforms typically provide less performance than desktop systems. S3TC compressed textures are widely supported and used by applications. The DXT1 format is used in the vast majority of cases in which S3TC compressed textures are used.

GL_EXT_texture_compression_s3tc

Adds texture compression functionality specific to the S3 S3TC format.

Discussion
This functionality is subject to all the requirements and limitations described by the extension GL_ARB_texture_compression extension, which supports DXT1, DXT3, and DXT5 texture compression formats. The S3 S3TC format is also known as DXTC other 3D API.)

GL_EXT_texture_env_add

Add a texture environment function for adding textures.

Discussion
The function `GL_ADD` implements the following equation:

```
Cv = Cf + Ct
```

GL_EXT_texture_filter_anisotropic

Provides support for anisotropic texturing filtering schemes without specifying an anisotropic filtering formula.

GL_EXT_texture_integer

Allows for true integer formats to be used in textures.

Discussion
While color components are normally stored as integers, these integer values are mapped to the `0.0` to `1.0` floating point range. This extension is used in conjunction with the `GL_EXT_gpu_shader4` (page 31) extension to allow true integer values to be stored in textures and used in shader programs.

GL_EXT_texture_lod_bias

Allows adding a bias value to the texture level-of-detail parameter.

Discussion
Provides a means to bias the lambda (a texture level-of-detail parameter that determines which mipmap levels and their relative mipmap weights for use in mipmapped texture filtering) by a constant (signed) value.

GL_EXT_texture_mirror_clamp

Extends texture wrapping to include three mirroring modes.

Discussion
The three mirroring modes are:

- `GL_MIRROR_CLAMP_EXT`
- `GL_MIRROR_CLAMP_TO_EDGE_EXT`
- `GL_MIRROR_CLAMP_TO_BORDER_EXT`

These modes effectively use a texture map that is twice as large as the original image to accommodate the original image and its mirror image.

GL_EXT_texture_rectangle

Adds a texture target that supports 2D textures without requiring power-of-two dimensions.

GL_EXT_texture_sRGB

Supports the sRGB color space for textures.

Discussion
The sRGB color space is based on typical (non-linear) monitor characteristics expected in a dimly lit office. It has been standardized by the International Electrotechnical Commission (IEC) as IEC 61966-2-1. The sRGB color space roughly corresponds to 2.2 gamma correction.

GL_EXT_transform_feedback

Defines a transform feedback mode for recording vertex attributes for each primitive that OpenGL processes.

Discussion
This extension is similar to the GL_NV_transform_feedback extension.

GL_IBM_rasterpos_clip

Extends the semantics of the raster position functions.

Discussion
It provides an enable that allows a raster position that would normally be clipped to be treated as a valid (albeit out-of-viewport) position.

GL_NV_blend_square

Provides four additional blending factors.

Discussion
This extension supports these blending factors:

- `GL_SRC_COLOR` and `GL_ONE_MINUS_SRC_COLOR` for source blending factors
- `GL_DST_COLOR` and `GL_ONE_MINUS_DST_COLOR` for destination blending factors

GL_NV_conditional_render

Allows a program to conditionally execute rendering commands based on an occlusion query.

Discussion
This extension builds on the `GL_ARB_occlusion_query` (page 21) extension by allowing a developer to submit drawing commands on an occlusion query that may not have completed. If the query has not completed, the application can choose either to stall until it does, or to submit the potentially occluded geometry anyway.

GL_NV_depth_clamp

Supports rasterizing line and polygon primitives without clipping the primitive to the near or far clip volume planes.

Discussion
Side clip volume planes still clip normally

GL_NV_fog_distance

Supports application-control of fog distance computations.

Discussion

GL_NV_fragment_program2

Provides additional fragment program functionality to extend the `GL_ARB_fragment_program` (page 20) specification.

Discussion
This extension is similar to the `GL_NV_fragment_program_option` (page 35) extension, in that it extends the standard `GL_ARB_fragment_program` language and execution environment.

GL_NV_fragment_program_option

Provides additional fragment program functionality to extend the standard `GL_ARB_fragment_program` (page 20) language and execution environment.

GL_NV_light_max_exponent

Extends the maximum shininess and spot exponent beyond `128.0`.

GL_NV_multisample_filter_hint

Provides a hint that permits implementations to provide an alternative method of resolving the color of multisampled pixels.

Discussion

GL_NV_point_sprite

Allows applications to use points rather than quads for such things as particle systems.

Discussion

GL_NV_register_combiners

Provides an extremely configurable mechanism know as "register combiners" for computing fragment colors.

GL_NV_register_combiners2

Extends the register combiners functionality to support more color constant values that are unique for each general combiner stage.

GL_NV_texgen_reflection

Provides two new texture coordinate generation modes for texture-based lighting and environment mapping.

GL_NV_texture_shader

Provides defines texture shader stage for mapping sets of texture coordinates to filtered colors.

Discussion
This extension provide you with more flexibility than standard OpenGL when you want to map texture coordinates to texture unit RGBA results. It introduce new texture formats and variations on a few existing ones.

GL_NV_texture_shader2

Adds texture_shader functionality to support texture shader operations for 3D textures.

GL_NV_texture_shader3

Extends the `GL_NV_texture_shader` functionality.

Discussion
This extension adds several texture shader operations, extending several existing texture shader operations, adding a new HILO8 internal format, and adding new and more flexible re-mapping modes for dot product and dependent texture shader operations.

GL_NV_vertex_program2_option

Extends the standard `ARB_vertex_program` language and execution environment.

Discussion
This extension provides additional vertex program functionality to extend the standard ARB_vertex_program language and execution environment.

GL_NV_vertex_program3

Provides additional vertex program functionality to extend the standard `ARB_vertex_program` language and execution environment.

Discussion
This extension, like the NV_vertex_program2_option extension, provides additional vertex program functionality to extend the standard ARB_vertex_program language and execution environment.

GL_SGIS_generate_mipmap

Defines a mechanism by which OpenGL can derive the entire set of mipmap arrays when provided with only the base level array.

Discussion

GL_SGIS_texture_edge_clamp

Defines an algorithm that clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel.

Discussion
The `GL_CLAMP_TO_EDGE_SGIS` algorithm clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel.

GL_SGIS_texture_lod

Imposes two constraints related to the texture level of detail parameter.

Discussion
The texture level of detail parameter (LOD) allows a large texture to be loaded and used initially at low resolution, and to have its resolution raised gradually as more resolution is desired or available.

GL_SGI_color_matrix

Adds a 4x4 matrix stack to the pixel transfer path.

Document Revision History

This table describes the changes to *OpenGL Extensions Guide*.

Date	Notes
2010-02-24	Updated for Mac OS X v10.6 and OpenGL 2.1.