OpenGL® ES 1.1 Extension Pack Specification

Version 1.03 (Annotated)

Editor : Aaftab Munshi

Contents

Chapter 1

Overview

This specification describes the OpenGL ES 1.1 Extension Pack specification. The OpenGL ES 1.1 Extension Pack is a collection of optional extensions added to OpenGL ES 1.1 that include features that are in OpenGL 1.5 but not in OpenGL ES 1.1. The functionality implemented by this extension pack brings a significant improvement in image quality and performance that can be leveraged by handheld 3D applications. It is the intent of the OpenGL ES working group that OpenGL ES 1.2 will make the list of features / extensions defined by this extension pack mandatory.

In addition to the optional extensions, OpenGL ES implementations that plan to support the Extension Pack are recommended to support a stencil bit depth of four or higher and an EGL config with a depth and stencil buffer, where stencil bit- depth is four or higher. This recommendation will become a mandatory requirement in OpenGL ES 1.2.

The extension strings that identify the OpenGL ES 1.1 Extension Pack are given by the following table:

Extension Name
`GL_OES_texture_env_crossbar`
`GL_OES_textured_mirrored_repeat`
`GL_OES_texture_cube_map`
`GL_OES_blend_subtract`
`GL_OES_blend_func_separate`
`GL_OES_blend_equation_separate`
`GL_OES_stencil_wrap`
`GL_OES_extended_matrix_palette`
`GL_OES_framebuffer_object`

The OpenGL ES 1.1 specification is written against the OpenGL 1.5 specification. Since the `GL_OES_texture_env_crossbar`, `GL_OES_textured_mirrored_repeat`, `GL_OES_texture_cube_map`, `GL_OES_blend_subtract`, `GL_OES_blend_func_separate`, and `GL_OES_stencil_wrap` extensions describe functionality that is already part of the OpenGL 1.5 specification, the corresponding OES extensions will only give an overview, and describe any new tokens and/or functions added by these extensions. Please refer to the OpenGL 1.5 specification for detailed description of how these features work.

Chapter 2

Texture Environment Crossbar

The `OES_texture_env_crossbar` extension adds the capability to use the texture color from other texture units as sources to the COMBINE environment function. OpenGL ES 1.1 defined texture combine functions which could use the color from the current texture unit as a source. This extension adds the ability to use the color from any texture unit as a source.

The tables that define arguments for `COMBINE_RGB` and `COMBINE_ALPHA` functions are extended to include `TEXTURE`*n*

SRC*n*_RGB	OPERAND*n*_RGB	Argument
TEXTURE	SRC_COLOR	C_s
	ONE_MINUS_SRC_COLOR	$1 - C_s$
	SRC_ALPHA	A_s
	ONE_MINUS_SRC_ALPHA	$1 - A_s$
TEXTURE*n*	SRC_COLOR	${C_s}^n$
	ONE_MINUS_SRC_COLOR	$1 - {C_s}^n$
	SRC_ALPHA	${A_s}^n$
	ONE_MINUS_SRC_ALPHA	$1 - {A_s}^n$
CONSTANT	SRC_COLOR	C_c
	ONE_MINUS_SRC_COLOR	$1 - C_c$
	SRC_ALPHA	Ac
	ONE_MINUS_SRC_ALPHA	$1 - A_c$
PRIMARY_COLOR	SRC_COLOR	C_f
	ONE_MINUS_SRC_COLOR	$1 - C_f$
	SRC_ALPHA	A_f
	ONE_MINUS_SRC_ALPHA	$1 - A_f$
PREVIOUS	SRC_COLOR	C_p
	ONE_MINUS_SRC_COLOR	$1 - C_p$
	SRC_ALPHA	A_p
	ONE_MINUS_SRC_ALPHA	$1 - A_p$

Table 2.1: Arguments for `COMBINE_RGB` functions.

SRC*n*_ALPHA	OPERAND*n*_ALPHA	Argument
TEXTURE	SRC_ALPHA	A_s
	ONE_MINUS_SRC_ALPHA	$1 - A_s$
TEXTURE*n*	SRC_ALPHA	${A_s}^n$
	ONE_MINUS_SRC_ALPHA	$1 - {A_s}^n$
CONSTANT	SRC_ALPHA	A_c
	ONE_MINUS_SRC_ALPHA	$1 - A_c$
PRIMARY_COLOR	SRC_ALPHA	A_f
	ONE_MINUS_SRC_ALPHA	$1 - A_f$
PREVIOUS	SRC_ALPHA	A_p
	ONE_MINUS_SRC_ALPHA	$1 - A_p$

Table 2.2: Arguments for COMBINE_ALPHA functions.

Chapter 3

Mirrored Texture Addressing

The `OES_texture_mirrored_repeat` extension extends the set of texture wrap modes to include a mode (GL_MIRRORED_REPEAT) that effectively uses a texture map twice as large as the original image in which the additional half, for each coordinate, of the new image is a mirror image of the original image.

This new mode relaxes the need to generate images whose opposite edges match by using the original image to generate a matching ”mirror image”.

Wrap modes `REPEAT`, `CLAMP_TO_EDGE` and `MIRRORED_REPEAT` are now supported.

Chapter 4

Cube Maps

The `OES_texture_cube_map` extension provides a new texture generation scheme for cube map textures. Instead of the current texture providing a 2D lookup into a 2D texture image, the texture is a set of six 2D images representing the faces of a cube. The (s,t,r) texture coordinates are treated as a direction vector emanating from the center of a cube. At texture generation time, the interpolated per-fragment (s,t,r) selects one cube face 2D image based on the largest magnitude coordinate (the major axis). A new 2D (s,t) is calculated by dividing the two other coordinates (the minor axes values) by the major axis value. Then the new (s,t) is used to lookup into the selected 2D texture image face of the cube map.

Unlike a standard 2D texture that have just one target, a cube map texture has six targets, one for each of its six 2D texture image cube faces. All these targets must be consistent, complete, and have equal width and height (ie, square dimensions).

This extension also provides two new texture coordinate generation modes for use in conjunction with cube map texturing. The reflection map mode generates texture coordinates (s,t,r) matching the vertex's eye-space reflection vector. The reflection map mode is useful for environment mapping without the singularity inherent in sphere mapping. The normal map mode generates texture coordinates (s,t,r) matching the vertex's transformed eye-space normal. The normal map mode is useful for sophisticated cube map texturing-based diffuse lighting models.

The intent of the new texgen functionality is that an application using cube map texturing can use the new texgen modes to automatically generate the reflection or normal vectors used to look up into the cube map texture.

The following texgen modes are supported: REFLECTION_MAP and NORMAL_MAP. SPHERE_MAP, OBJECT_LINEAR, and EYE_LINEAR texgen modes are not supported. Texgen supports a new *coord* value `STR`. This allows the application to specify the texgen mode for the appropriate coordinates in a single call. Texgen with coord values of `S`, `T`, `R` and `Q` are not supported.

4.1 Coordinate Transformations

OpenGL 1.5	Common	Common-Lite
TexGen{**ifx**}**[v]**`(enum coord, enum pname, T params)`		
`pname = TEXTURE_GEN_MODE, params = OBJECT_LINEAR`	–	–
`pname = TEXTURE_GEN_MODE, params = EYE_LINEAR`	–	–
`pname = TEXTURE_GEN_MODE, params = SPHERE_MAP`	–	–
`pname = TEXTURE_GEN_MODE, params = REFLECTION_MAP`	◊	†

OpenGL 1.5	**Common**	**Common-Lite**
`pname = TEXTURE_GEN_MODE, params = NORMAL_MAP`	◊	†
`pname = OBJECT_PLANE`	–	–
`pname = EYE_PLANE`	–	–
TexGen{**d**}**[v]**`(enum coord, enum pname, T param)`	–	–
GetTexGen{**d**}**v**`(enum coord, enum pname, T *params)`	–	–
GetTexGen{**ifx**}**v**`(enum coord, enum pname, T *params)`	✓	✓
Enable/Disable`(TEXTURE_GEN_{STR})`	✓	✓
Enable/Disable`(TEXTURE_GEN_S,T,R,Q)`	–	–

4.2 Texture Addressing Modes

For cubemaps, the only allowed texture addressing mode is `CLAMP_TO_EDGE`.

4.3 Texture Completeness

For cube map textures, a texture is *cube complete* if the following conditions all hold true:

- the base level arrays of each of the six texture images making up the cube map have identical, positive, and square dimensions.
- the base level arrays were specified with the same type.

Finally, a cube map texture is *mipmap cube complete* if, in addition to being cube complete, each of the six texture images considered individually is complete.

OpenGL 1.5	**Common**	**Common-Lite**
TexImage2D`(enum target, int level, int internalFormat, sizei width, sizei height, int border, enum format, enum type, const void *pixels)`		
`target = TEXTURE_CUBE_MAP_POSITIVE_X, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_POSITIVE_Y, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_POSITIVE_Z, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_NEGATIVE_X, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_NEGATIVE_Y, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_NEGATIVE_Z, border = 0`	✓‡	✓‡
CompressedTexImage2D`(enum target, int level, enum internalformat, sizei width, sizei height, int border, sizei imageSize, const void *data)`		
`target = TEXTURE_CUBE_MAP_POSITIVE_X, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_POSITIVE_Y, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_POSITIVE_Z, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_NEGATIVE_X, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_NEGATIVE_Y, border = 0`	✓‡	✓‡
`target = TEXTURE_CUBE_MAP_NEGATIVE_Z, border = 0`	✓‡	✓‡
TexParameter{**if**}**[v]**`(enum target, enum pname, T param)`		
`target = TEXTURE_CUBE_MAP,`	✓	†

OpenGL 1.5	**Common**	**Common-Lite**
BindTexture(`enum target, uint texture`)		
`target = TEXTURE_CUBE_MAP`	✓	✓
Enable/Disable(`enum cap`)		
`cap = TEXTURE_CUBE_MAP`	✓	✓
GetTexGen{**ifx**}**v**(`enum env, enum pname, T *params`)	♢	†
GetTexGen{**d**}**v**(`enum env, enum pname, T *params`)	–	–

State	**Exposed**	**Queriable**	**Common Get**	**Common-Lite Get**
`TEXTURE_CUBE_MAP`	✓	✓	**IsEnabled**	**IsEnabled**
`TEXTURE_BINDING_CUBE_MAP`	✓	✓	**GetIntegerv**	**GetIntegerv**
`TEXTURE_CUBE_MAP_POSITIVE_X`	✓	–	–	–
`TEXTURE_CUBE_MAP_NEGATIVE_X`	✓	–	–	–
`TEXTURE_CUBE_MAP_POSITIVE_Y`	✓	–	–	–
`TEXTURE_CUBE_MAP_NEGATIVE_Y`	✓	–	–	–
`TEXTURE_CUBE_MAP_POSITIVE_Z`	✓	–	–	–
`TEXTURE_CUBE_MAP_NEGATIVE_Z`	✓	–	–	–

Table 4.3: Texture Objects

State	**Exposed**	**Queriable**	**Common Get**	**Common-Lite Get**
`MAX_CUBE_MAP_TEXTURE_SIZE`	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 4.4: Implementation Dependent Values

Chapter 5

Blending Extensions

The `OES_blend_subtract` extension adds two additional blending equations `FUNC_SUBTRACT` and `FUNC_REVERSE_SUBTRACT`

OpenGL 1.5	**Common**	**Common-Lite**
BlendEquation(`enum mode`)		
mode = `FUNC_SUBTRACT`	✓	✓
mode = `FUNC_REVERSE_SUBTRACT`	✓	✓

The `OES_blend_func_separate` extension extends the blending capability by defining a function that allows independent setting of the RGB and alpha blend factors for blend operations that require source and destination blend factors. It is not always desired that the blending used for RGB is also applied to alpha.

OpenGL 1.5	**Common**	**Common-Lite**
BlendFuncSeparate(`enum srcRGB, enum dstRGB, enum srcAlpha, enum dstAlpha`)	✓	✓

State	**Exposed**	**Queriable**	**Common Get**	**Common-Lite Get**
`BLEND_SRC_RGB (v1.1 BLEND_SRC)`	✓	✓	**GetIntegerv**	**GetIntegerv**
`BLEND_DST_RGB (v1.1 BLEND_DST)`	✓	✓	**GetIntegerv**	**GetIntegerv**
`BLEND_SRC_ALPHA`	✓	✓	**GetIntegerv**	**GetIntegerv**
`BLEND_DST_ALPHA`	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 5.3: Pixel Operations

The `OES_blend_equation_separate` extension provides a separate blend equation for RGB and alpha to match the generality available for blend factors.

OpenGL 1.5	**Common**	**Common-Lite**
BlendEquationSeparate(`enum modeRGB, enum modeAlpha`)	✓	✓

State	**Exposed**	**Queriable**	**Common Get**	**Common-Lite Get**
`BLEND_EQUATION_RGB`	✓	✓	**GetIntegerv**	**GetIntegerv**
`BLEND_EQUATION_ALPHA`	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 5.5: Pixel Operations

Chapter 6

Stencil Extensions

The `OES_stencil_wrap` extension extends the StencilOp functions to support `INCR_WRAP` and `DECR_WRAP` modes.

OpenGL 1.5	**Common**	**Common-Lite**
StencilOp(`enum fail, enum zfail, enum zpass`)		
`fail, zfail, zpass = INCR_WRAP`	✓	✓
`fail, zfail, zpass = DECR_WRAP`	✓	✓

Chapter 7

Extended Matrix Palette

```
Name

    OES_extended_matrix_palette

Name Strings

    GL_OES_extended_matrix_palette

Contact

    Aaftab Munshi (amunshi@ati.com)

Status

    Ratified by the Khronos BOP, July 22, 2005.

Version


Number


Dependencies

    OES_matrix_palette is required
    OpenGL ES 1.1 is required.

Overview

    The OES_matrix_palette extension added the ability to support vertex skinning
    in OpenGL ES.  One issue with OES_matrix_palette is that the minimum size of
    the matrix palette is very small.  This leads to applications having to break
    geometry into smaller primitive sets called via. glDrawElements.  This has an
    impact on the overall performance of the OpenGL ES implementation.  In general,
    hardware implementations prefer primitive packets with as many triangles as
    possible.  The default minimum size defined in OES_matrix_palette is not
    sufficient to allow this.  The OES_extended_matrix_palette extension increases
```

    this minimum from 9 to 32.

Another issue is that it is very difficult for ISVs to handle different size matrix palettes as it affects how they store their geometry in the database - may require multiple representations which is not really feasible. So the minimum size is going to be what most ISVs will use.

By extending the minimum size of the matrix palette, we remove this fragmentation and allow applications to render geometry with minimal number of calls to glDrawElements or glDrawArrays. The OpenGL ES implementation can support this without requiring any additional hardware by breaking the primitive, plus it gives implementations the flexibility to accelerate with a bigger matrix palette if they choose to do so.

Additionally, feedback has also been received to increase the number of matrices that are blend per vertex from 3 to 4. The OES_extended_matrix_palette extension increases the minium number of matrices / vertex to 4.

IP Status

    None.

Issues

    None

New Procedures and Functions

    None

New Tokens

    No new tokens added except that the default values for MAX_PALETTE_MATRICES_OES and MAX_VERTEX_UNITS_OES are 32 and 4 respectively.

Additions to Chapter 2 of the OpenGL ES 1.0 Specification

    None

Errors

    None

New State

Get Value	Type	Command	Value	Description
MAX_PALETTE_MATRICES_OES	Z+	GetIntegerv	32	size of matrix palette
MAX_VERTEX_UNITS_OES	Z+	GetIntegerv	4	number of matrices per vertex

```
Revision History

    Feb 03, 2005   Aaftab Munshi    First draft of extension
```

Chapter 8

Framebuffer Objects

```
Name

    OES_framebuffer_object

Name Strings

    GL_OES_framebuffer_object

Contact

    Aaftab Munshi (amunshi@ati.com)

IP Status

    None.

Status

    Ratified by the Khronos BOP, July 22, 2005.

Version

    Last Modified Date:    July 18, 2005


Number


Dependencies

    OpenGL ES 1.0 is required.

    EXT_framebuffer_object is required.

Overview

    This extension defines a simple interface for drawing to rendering
```

destinations other than the buffers provided to the GL by the window-system. OES_framebuffer_object is a simplified version of EXT_framebuffer_object with modifications to match the needs of OpenGL ES.

In this extension, these newly defined rendering destinations are known collectively as "framebuffer-attachable images". This extension provides a mechanism for attaching framebuffer-attachable images to the GL framebuffer as one of the standard GL logical buffers: color, depth, and stencil. When a framebuffer-attachable image is attached to the framebuffer, it is used as the source and destination of fragment operations as described in Chapter 4.

By allowing the use of a framebuffer-attachable image as a rendering destination, this extension enables a form of "offscreen" rendering. Furthermore, "render to texture" is supported by allowing the images of a texture to be used as framebuffer-attachable images. A particular image of a texture object is selected for use as a framebuffer-attachable image by specifying the mipmap level, cube map face (for a cube map texture) that identifies the image. The "render to texture" semantics of this extension are similar to performing traditional rendering to the framebuffer, followed immediately by a call to CopyTexSubImage. However, by using this extension instead, an application can achieve the same effect, but with the advantage that the GL can usually eliminate the data copy that would have been incurred by calling CopyTexSubImage.

This extension also defines a new GL object type, called a "renderbuffer", which encapsulates a single 2D pixel image. The image of renderbuffer can be used as a framebuffer-attachable image for generalized offscreen rendering and it also provides a means to support rendering to GL logical buffer types which have no corresponding texture format (stencil etc). A renderbuffer is similar to a texture in that both renderbuffers and textures can be independently allocated and shared among multiple contexts. The framework defined by this extension is general enough that support for attaching images from GL objects other than textures and renderbuffers could be added by layered extensions.

To facilitate efficient switching between collections of framebuffer-attachable images, this extension introduces another new GL object, called a framebuffer object. A framebuffer object contains the state that defines the traditional GL framebuffer, including its set of images. Prior to this extension, it was the window-system which defined and managed this collection of images, traditionally by grouping them into a "drawable". The window-system API’s would also provide a function (i.e., eglMakeCurrent) to bind a drawable with a GL context. In this extension however, this functionality is subsumed by the GL and the GL provides the function BindFramebufferOES to bind a framebuffer object to the current context. Later, the context can bind back to the window-system-provided framebuffer in order to display rendered content.

```
    Previous extensions that enabled rendering to a texture have been
    much more complicated.  One example is the combination of
    ARB_pbuffer and ARB_render_texture, both of which are window-system
    extensions.  This combination requires calling MakeCurrent, an
    operation that may be expensive, to switch between the window and
    the pbuffer drawables.  An application must create one pbuffer per
    renderable texture in order to portably use ARB_render_texture.  An
    application must maintain at least one GL context per texture
    format, because each context can only operate on a single
    pixelformat or FBConfig.  All of these characteristics make
    ARB_render_texture both inefficient and cumbersome to use.

    OES_framebuffer_object, on the other hand, is both simpler to use
    and more efficient than ARB_render_texture.  The
    OES_framebuffer_object API is contained wholly within the GL API and
    has no (non-portable) window-system components.  Under
    OES_framebuffer_object, it is not necessary to create a second GL
    context when rendering to a texture image whose format differs from
    that of the window.  Finally, unlike the pbuffers of
    ARB_render_texture, a single framebuffer object can facilitate
    rendering to an unlimited number of texture objects.

    Please refer to the EXT_framebuffer_object extension for a
    detailed explaination of how framebuffer objects are supposed to work,
    the issues and their resolution.  This extension can be found at
    http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt

New Tokens

    Accepted by the <internalformat> parameter of RenderbufferStorageOES

        RGB565_OES                0x8D62


New Procedures and Functions

    boolean IsRenderbufferOES(uint renderbuffer);
    void BindRenderbufferOES(enum target, uint renderbuffer);
    void DeleteRenderbuffersOES(sizei n, const uint *renderbuffers);
    void GenRenderbuffersOES(sizei n, uint *renderbuffers);

    void RenderbufferStorageOES(enum target, enum internalformat,
                                sizei width, sizei height);

    void GetRenderbufferParameterivOES(enum target, enum pname, int* params);

    boolean IsFramebufferOES(uint framebuffer);
    void BindFramebufferOES(enum target, uint framebuffer);
    void DeleteFramebuffersOES(sizei n, const uint *framebuffers);
    void GenFramebuffersOES(sizei n, uint *framebuffers);
    enum CheckFramebufferStatusOES(enum target);

    void FramebufferTexture2DOES(enum target, enum attachment,
                                 enum textarget, uint texture,
                                 int level);

    void FramebufferRenderbufferOES(enum target, enum attachment,
                                    enum renderbuffertarget, uint renderbuffer);

    void GetFramebufferAttachmentParameterivOES(enum target, enum attachment,
                                                enum pname, int *params);

    void GenerateMipmapOES(enum target);
```

OES_framebuffer_object implements the functionality defined by EXT_framebuffer_object with the following limitations:

- there is no support for DrawBuffer{s}, ReadBuffer{s}.

- FramebufferTexture2DOES can be used to render directly into the base level of a texture image only. Rendering to any mip-level other than the base level is not supported.

- FramebufferTexture3DOES is not supported as OpenGL ES 1.1 and 2.0 does not support 3D textures. Support for 3D textures in OpenGL ES 2.0 is provided by the OES_texture_3D optional extension. FramebufferTexture3DOES has been moved to this extension specification.

- section 4.4.2.1 of the EXT_framebuffer_object spec describes the function RenderbufferStorageEXT. This function establishes the data storage, format, and dimensions of a renderbuffer object's image. <target> must be RENDERBUFFER_EXT. <internalformat> must be one of the internal formats from table 3.16 or table 2.nnn which has a base internal format of RGB, RGBA, DEPTH_COMPONENT, or STENCIL_INDEX.

  The above paragraph is modified by OES_framebuffer_object and states thus:

  "This function establishes the data storage, format, and dimensions of a renderbuffer object's image. <target> must be RENDERBUFFER_OES. <internalformat> must be one of the sized internal formats from the following table which has a base internal format of RGB, RGBA, DEPTH_COMPONENT, or STENCIL_INDEX"

  The following formats are required:

Sized Internal Format	Base Internal format
RGB565_OES	RGB
RGBA4	RGBA
RGB5_A1	RGBA

```
        DEPTH_COMPONENT_16    DEPTH_COMPONENT

    The following formats are optional:

        Sized                 Base
        Internal Format       Internal format
        ---------------       ---------------
        RGBA8                 RGBA
        RGB8                  RGB
        DEPTH_COMPONENT_24    DEPTH_COMPONENT
        DEPTH_COMPONENT_32    DEPTH_COMPONENT
        STENCIL_INDEX1_OES    STENCIL_INDEX
        STENCIL_INDEX4_OES    STENCIL_INDEX
        STENCIL_INDEX8_OES    STENCIL_INDEX


    The optional formats are described by the OES_rgb8_rgba8, OES_depth24,
    OES_depth32, OES_stencil1, OES_stencil4, and OES_stencil8 extensions.
    Even though these formats are optional in this extension, the OpenGL ES
    APIs (1.x and 2.x versions) can mandate some or all of these optional formats.

    If RenderbufferStorageOES is called with an <internalformat> value that is
    not supported by the OpenGL ES implementation, an INVALID_ENUM error will
    be generated.
```

Revision History

02/25/2005	Aaftab Munshi	First draft of extension
04/27/2005	Aaftab Munshi	Added additional limitations to simplify OES_framebuffer_object implementations
07/06/2005	Aaftab Munshi	Added GetRenderbufferStorageFormatsOES removed limitations that were added to OES version of RenderbufferStorage, and FramebufferTexture2DOES.
07/07/2005	Aaftab Munshi	Removed GetRenderbufferStorageFormatsOES after discussions with Jeremy Sandmel, and added specific extensions for the optional renderbuffer storage foramts
07/18/2005	Aaftab Munshi	Added comment that optional formats can be mandated by OpenGL ES APIs.