Contents

List of Tables

Chapter 1

Overview

This document describes EGL, an interface between rendering APIs such as OpenGL ES or OpenVG (referred to collectively as *client APIs*) and an underlying native platform window system. It refers to concepts discussed in the OpenGL ES and OpenVG specifications, and should be read together with those documents.

EGL342(pro(v)1vid387(342(mechanism87(342(foter342(c9(rtering342(7(rendering342(suterf)10(a87(342(on6(to342

Ancillary buffers not meaningful to a client API do not affect compatibility;

Native rendering may be supported by window surfaces, but only if the native window system has a compatible rendering model allowing it to share the back color buffer, or if single buffered rendering to the window surface is being done.

When both native rendering APIs and client APIs are drawing into the same

revisions of client APIs where such types of state (for example, display lists) are defined and where such sharing makes sense.

2.4.1 OpenGL ES Texture Objects

Client API commands are not guaranteed to be atomic. Some such commands might otherwise impair interactive use of the windowing system by the user. For instance, rendering a large texture mapped polygon on a system with no graphics hardware, or drawing a large OpenGL ES vertex array, could prevent a user from popping up a menu soon enough to be usable.

Synchronization is in the hands of the client. It can be maintained at moderate cost with the judicious use of commands such as **glFinish**, **vgFinish**, **eglWaitClient**, and **eglWaitNative**, as well as (if they exist) synchronization commands present in native rendering APIs. Client API and native rendering can be done in parallel so long as the client does not preclude it with explicit synchronization calls.

Some specific error codes that may be generated by a failed EGL function, and their meanings, are described together with each function. However, not all possible errors are described with each function. Errors whose mean-

EGLDisplay. An EGL_NOT_INITIALIZED error is generated if EGL cannot be initialized for an otherwise valid *dpy*.

Initializing an already-initialized display is allowed, but the only effect of such a call is to return EGL_TRUE and update the EGL version numbers. An initialized display may be used from other threads in the same address space without being initalized again in those threads.

To release resources associated with use of EGL and client APIs on a display, call

```
EGLBoolean eglTerminate(EGLDisplay dpy);
```

Termination marks **all**

With the exception of the color buffer, most buffers are used only by one client API . The depth, multisample, and stencil buffers are specific to OpenGL ES , while the alpha mask buffer is specific to OpenVG . To conserve resources, implementations may delay creation of buffers until they are needed by EGL or a client API . For example, if an `EGLConfig` describes an alpha mask buffer with depth greater than zero, that buffer need not be allocated by a surface until an OpenVG context is bound to that surface.

The *color buffer* is shared by all client APIs rendering to a surface, and contains pixel color values.

EGL

OpenGL ES).

to get `EGLConfig`s that match a list of attributes. The return value and the meaning of *configs*, *config_size*, and *num_config*

Attribute	Default	Selection Criteria	Sort Order	Sort Priority
EGL				

surface. If its value is EGL

EGL_TEXTURE_FORMAT is EGL_TEXTURE_RGB or EGL_TEXTURE_RGBA), then the

3.5.4 Creating Native Pixmap Rendering Surfaces

EGL also supports rendering surfaces whose color buffers are stored in native pixmaps. Pixmaps differ from windows in that they are typically allocated in off-screen (non-visible) graphics or CPU memory. Pixmaps differ from pbuffers in that they do have an associated native pixmap and native pixmap type, and it may be possible to render to pixmaps using APIs other than client APIs .

EGL_BAD_MATCH error is generated. If *config* does not support the colorspace or alpha format attributes specified in *attrib_list* (as defined for **eglCreateWindowSurface**), an EGL_BAD_MATCH error is generated. If *config* is not a valid EGLConfig

Attribute	Type	

eglQuerySurface returns in *value*

Querying EGL_

Some of the functions described in this section make use of the *current rendering API*

If *share_context* is not EGL_NO_CONTEXT, then all shareable data, as defined by the client API (note that for OpenGL ES , shareable data excludes texture objects named 0) will be shared by *share_context*, all other contexts *share_context* already shares with, and the newly created context. An arbitrary number of EGLContexts can share data in this fashion. The OpenGL ES server context state for all shar-

- If *ctx*

3.8 Synchronization Primitives

To prevent native rendering API functions from executing until any outstanding client API rendering affecting the same surface is complete, call

```
EGLBoolean eglWaitClient();
```

engine denotes a particular *marking engine* (another drawing API, such as GDI or Xlib) to be waited on. Valid values of *engine* are defined by EGL extensions specific to implementations, but implementations will always recognize the symbolic constant EGL

3.9. POSTING THE.1100.81767.1233COLOR7.1233BUFFER

eglSwapInterval returns EGL

3.11. RELEASING THREAD STATE

Chapter 4

Extending EGL

Chapter 5

EGL Versions, Header Files, and Enumerants

Each version of EGL supports specified client API versions, and all prior versions

The state maintained by one rendering context is not affected by another except in case of state that may be explicitly shared at context creation time, such as textures.

Current Context an implicit context used by OpenGL ES and OpenVG , rather

Mark Callow, HI
Mark Tarlton, Motorola

Appendix B

Version 1.1

EGL version 1.1, approved on August 5, 2004, is the second release of EGL. It adds power management and swap control functionality based on vendor extensions from Imagination Technologies, and optional render-to-texture functionality based on the `WGL_ARB_render_texture` extension defined by the OpenGL ARB for desktop OpenGL.

B.1 Revision 1.1.2

EGL version 1.1.2 (revision 2 of EGL 1.1), approved on November 10, 2004, clarified

Appendix C

Tom Olson, TI

Appendix D

Version 1.3

EGL version 1.3 was voted out of the OpenKODE Working Group on December 4, 2006, and formally approval by the Khronos Board of Promoters on February 8, 2007. EGL 1.3 is the fourth release of EGL.

EGL 1.2 Token Name	EGL 1.3 Token Name

Sven Gothel, ATI
Teemu Rantalaiho, Hybrid Graphics
Tom Olson, TI

Index of EGL Commands