Khronos Native Platform Graphics Interface

Inc.o

Contents

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

2.1.2 Displays

Most EGL calls include an `EGLDisplay` parameter. This represents the abstract

Single buffered rendering is used by pixmap surfaces. Memory for the color

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.

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

OpenGL ES texture state can be encapsulated in a named texture object. A texture object is created by binding an unused name to one of the supported texture targets (GL_TEXTURE_2D, GL_TEXTURE_3D, or GL_TEXTURE_CUBE_MAP) of an OpenGL ES context. When a texture object is bound, OpenGL ES operations on the target to which it is bound affect the bound texture object, and queries of the

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**, **eglWaitAPI**, 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

not all possible errors are described with each function. Errors whose meanings are identical across many functions (such as returning EGL_BAD_DISPLAY or EGL_

that API.

`EGL_NATIVE_RENDERABLE` is an

EGL_MAX_PBUFFER_WIDTH and EGL_

Client APIs may not be able to respect the requested rendering buffer. To determine the actual buffer being rendered to by a context, call **eglQueryContext** (see section 3.7.4).

EGL_COLORSPACE specifies the

respectively. If the pbuffer will be used as a OpenGL ES texture (i.e., the value of EGL_TEXTURE_TARGET is EGL_TEXTURE_2D, and the value of EGL_TEXTURE_FORMAT is EGL_TEXTURE_RGB or EGL_TEXTURE_RGBA), then the aspect ratio will be preserved and the new width and height will be valid sizes

- If the buffers contained in *buffer*

3.5.5 Destroying Rendering Surfaces

An EGLSurface of any type (window, pbuffer, or pixmap) is destroyed by calling

EGLBoolean **eglDestroySurface**(EGLDisplay *dpy*

Attribute

Texture Component	Size

- The EGL_MIPMAP_TEXTURE attribute of the pbuffer being bound is EGL

On failure **eglCreateContext** returns EGL_NO_CONTEXT. If the current rendering api is EGL_

Other errors may arise when the context state is inconsistent with the surface

3.7.4 Context Queries

is equivalent to

```
EGLenum api = eglQueryAPI();
eglBindAPI(EGL_OPENGL_ES_API);
eglWaitClient();
eglBindAPI(api);
```

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

```
EGLBoolean eglWaitNative(EGLint engine);
```

Native rendering calls made with the specified marking *engine*, and which affect

3.9.3 Posting Semantics

In EGL 1.1, *surface* must be bound to the current context. This restriction is expected to be lifted in future EGL revisions.

If *dpy* and *surface* are the display and surface for the calling thread's current context, **eglSwapBuffers** and **eglCopyBuffers** perform an implicit flush operation on the context (**glFlush** for an OpenGL ES context, **vgFlush** for an OpenVG con-

The default swap interval is 1.

3.9.4 Posting Errors

eglSwapBuffers and **eglCopyBuffers** return EGL_FALSE

Chapter 4

Extending EGL

EGL implementors may extend EGL by adding new commands or additional enu-

Chapter 5

Chapter 6

Glossary

Address Space the set of objects or memory locations accessible through a single name space. In other words, it is a data region that one or more threads may share through pointers.

Client an application, which communicates with the underlying EGL implemen-

The state maintained by one rendering context is not affected by another

Appendix A

Mark Callow, HI
Mark Tarlton, Motorola
Mike Olivarez, Motorola
Neil Trevett, 3Dlabs
Phil Huxley, Tao Group
Tom Olson, Texas Instruments
Ville Miettinen, Hybrid Graphics

Tom Olson, TI

Index of EGL Commands