OpenGL and IRIS GL

The first step in porting a RealityEngine IRIS GL application to OpenGL is to consult the *OpenGL Porting Guide*. It covers all the core IRIS GL and OpenGL functionality, window and event handling, and OpenGL extensions up to and including those in IRIX 5.3.

This appendix provides some additional information about porting IRIS GL to OpenGL, pointing to the extensions discussed in earlier chapters of this book where appropriate. In order to make it as short and comprehensible as possible, it won't repeat any information from the *OpenGL Porting Guide*.

Some IRIS GL Functionality and OpenGL Equivalents

This section provides an alphabetical list of IRIS GL functions and some other functionality and either a pointer to related OpenGL functions or an explanation of how to implement similar functionality in OpenGL.

backbuffer, frontbuffer

The framebuffer update semantics for rendering into multiple color buffers are different in IRIS GL and OpenGL. OpenGL on RealityEngine systems actually implements the IRIS GL semantics (computing one color value and writing it to all buffers) rather than the correct OpenGL semantics (computing a separate color value for each buffer). This can cause unexpected results if blending is used.

blendcolor

See "Blending Extensions" on page 140.

blendfunction

See "Blending Extensions" on page 140.

convolve

See “The Convolution Extension” on page 129.

displacepolygon

The OpenGL equivalent, **glPolygonOffsetEXT()**, is more general than **displacepolygon()**. You may need to tweak the parameter values to get the proper results. See “The Polygon Offset Extension” on page 145.

dither

OpenGL provides no control over video dithering. (This is also the case for IRIS GL in IRIX 5.3, unless overridden by an environment variable.)

fbsubtexload

See “The Subtexture Extension” on page 96.

gamma

Use the XSGIvc extension. See “Stereo Rendering” on page 74.

glcompat GLC_SET_VSYNC, GLC_GET_VSYNC, GLC_VSYNC_SLEEP

For GLC_GET_VSYNC, use **glXGetVideoSyncSGI()**. For GLC_VSYNC_SLEEP, use **glXWaitVideoSyncSGI()**. See “The Video Synchronization Extension” on page 167.

GLC_SET_VSYNC has no equivalent in OpenGL. To replace it, maintain a sync counter offset in a static variable.

glcompat SIR_VEN_INTERFACE, SIR_VEN_VIDEOCOPY

(This function copies Sirius video to the framebuffer)

Supported, with some constraints. Use **glXCreateGLXVideoSourceSGIX()** to create a video source context, **glXMakeCurrentReadSGI()** to set up the video source for a data transfer, and **glCopyPixels()** to copy the video data to the framebuffer.

hgram, gethgram (histogram)

Supported for:

```
glDrawPixels(lrectwrite), glCopyPixels(rectcopy),
glReadPixels(lrectread)
```

Use **glGetHistogramEXT()** and **glHistogramEXT()**; see “The Histogram and Minmax Extensions” on page 132.

ilbuffer, ildraw, readsource(SRC_ILBUFFER)

(This function provides accelerated drawing to offscreen framebuffer memory.)

See “The Pixel Buffer Extension” on page 179.

istexloaded

Use **glAreTexturesResidentEXT()**; see “Texture Priorities and Residency” on page 95 for texture management.

leftbuffer, rightbuffer

Use **glXChooseVisual()** and **glDrawBuffer()** for stereo in a window. For old-style stereo, see **XSGISetStereoMode()**.

libsphere—sphdraw, sphgnpolys, sphfree, sphmode, sphobj, sphrotmatrix, sphbgnbitmap, sphendbitmap, sphcolor

gluSphere() provides polygonal spheres. Only bilinear tessellation is supported; octahedral, icosahedral, barycentric, and cubic tessellations are not supported.

There is no support for the canonical orientation capability (sphrotmatrix), hemispheres (SPH_HEMI), or bitmap spheres.

linesmooth

Antialiased lines are supported, with one caveat: it is not possible to draw blended antialiased lines in a multisampled window, even when multisampling is disabled. See **glHint()** and **glEnable()** with the GL_LINE_SMOOTH parameter.

minmax, getminmax

For minimum and maximum pixel values, use **glGetMinmaxEXT()** and **glMinmaxEXT()**; see "The Histogram and Minmax Extensions" on page 132.

mswapbuffers

Not supported.

Swapping multiple windows (for example, main and overlay buffers, or windows on genlocked pipes) from multiple threads can be accomplished fairly reliably with semaphored calls to **glXSwapBuffers()**. The following code fragment outlines the approach:

```
/* Create some semaphores: */
       usptr_t* arena = usinit("/usr/tmp/our_arena");
       usema_t* pipe0ready = usnewsema(arena, 0);
       usema_t* pipe1ready = usnewsema(arena, 0);

/* After the process for pipe0 finishes its frame, it signals its
completion and waits for pipe1. When pipe1 is also ready, pipe0 swaps:
*/
       usvsema(pipe0ready);
       uspsema(pipe1ready);
       glXSwapBuffers(dpy, drawable);

/* The process for pipe 1 does the converse: */
       usvsema(pipe1ready);
       uspsema(pipe0ready);
       glXSwapBuffers(dpy, drawable);
```

multisample, getmultisample, msalpha, msmask, mspattern, mssize (multisample antialiasing)

Supported. See "The Multisample Extension" on page 154.

For msalpha, see **glEnable()** with arguments GL_SAMPLE_ALPHA_TO_MASK_SGIS and GL_SAMPLE_ALPHA_TO_ONE_SGIS.

For msmask, see **glSampleMaskSGIS()**.

For mspattern, see **glSamplePatternSGIS()**.

For mssize, see **glXChooseVisual()**.

For “light points,” use multisampling with

```
glHint(GL_POINT_SMOOTH_HINT,GL_NICEST)
```

The maximum point diameter is 3 (the same as IRIS GL).

For fast tag clear, see **glTagSampleBufferSGIX()**.

pixelmap

Differs from IRIS GL. The OpenGL function **glPixelMap()** specifies a lookup table for pixel transfer operations, just as pixelmap does in IRIS GL. However, the location of the lookup table in the pixel processing pipeline is different. The IRIS GL lookup table appears immediately after convolution, while the OpenGL lookup table appears almost at the beginning of the pipeline (immediately after the first scale and bias). The two pipelines are equivalent only when convolution is disabled.

Pixel mapping is supported for the following calls:

```
glDrawPixels(lrectwrite), glCopyPixels(rectcopy),
glReadPixels(lrectread), glTexImage(texdef),
```

It is not supported for

```
glTexSubImage(subtexload)
```

pixmode

Most of the functions of pixmode are supported, albeit in different ways:

- PM_SHIFT, PM_ADD24: Use the OpenGL color matrix extension to swizzle color components or to scale and bias pixel values. See **glPixelTransfer()**.
- PM_EXPAND, PM_C0, PM_C1: Use the standard OpenGL color lookup table to convert bitmap data to RGBA. See **glPixelTransfer()** and **glPixelMap()**.
- PM_TTOB, PM_RTOL: Use **glPixelZoom()** with negative zoom factors to reflect images when drawing or copying. Reflection during reading is not supported.
- PM_INPUT_FORMAT, PM_OUTPUT_FORMAT, PM_INPUT_TYPE, PM_OUTPUT_TYPE, PM_ZDATA: Use the **glReadPixels()**, **glDrawPixels()**, and **glCopyPixels()** type and format parameters.
- PM_OFFSET, PM_STRIDE, PM_SIZE: Use **glPixelStore()**.

pntsize

Supported. See comments under “multisample, getmultisample, msalpha, msmask, mspattern, mssize (multisample antialiasing)” on page 290.

polymode

OpenGL doesn’t support PYM_HOLLOW. **glPolygonMode(GL_LINE)** is the closest approximation. See also **glPolygonOffsetEXT()** and “The Polygon Offset Extension” on page 145.

polysmooth

OpenGL doesn’t support PYM_SHRINK.

popup planes

OpenGL doesn’t support drawing in the popup planes.

readcomponent

Use the color matrix extension (see **glPixelTransfer()**) to extract one or more color channels for image processing. The implementation is optimized for the case in which all channels but one are multiplied by zero, and all framebuffer channels but one are write-masked (see **glColorMask()**). See “Using the Color Matrix and the Color Writemask” on page 285.

The color matrix is supported for **glDrawPixels()** and **glCopyPixels()** but not for **glTexImage()** or **glReadPixels()**.

See “The Color Matrix Extension” on page 128 for more information.

RGBwritemask

OpenGL supports masking an entire RGBA color channel, but not arbitrary sets of bits within an RGBA color channel.

setvideo, setmonitor

OpenGL has no support for these routines.

Video output format should be changed with the *setmon* command (this is now recommended for IRIS GL as well as OpenGL).

OpenGL supports stereo-in-a-window; see **glXChooseVisual()** and **glDrawBuffer()**. For old-style stereo, see **XSGISetStereoMode()**.

Use the Video Library (VL) or the XSGIvc extension for other video device control tasks.

subtexload

See “The Subtexture Extension” on page 96.

tevdef, tevbind

TV_COMPONENT_SELECT (the ability to pack multiple shallow textures together, then unpack and select one of them during drawing) is not supported.

texbind

Texture definition and binding are combined into a single operation in standard OpenGL. However, the texture object extension makes them separate again (albeit in a manner that differs from IRIS GL). Use **glBindTextureEXT()**; see “The Texture Object Extension” on page 93.

Detail texturing (TX_TEXTURE_DETAIL) is supported. Use **glDetailTexFuncSGIS()**; see “The Detail Texture Extension” on page 112.

Simple texture memory management (TX_TEXTURE_IDLE) is supported. Use **glPrioritizeTexturesEXT()**; see “Texture Priorities and Residency” on page 95.

texdef

1D, 2D, and 3D textures are supported. See **glTexImage1D()**, **glTexImage2D()**, and **glTexImage3DEXT()**; see “The 3D Texture Extension” on page 100.

TX_FAST_DEFINE is not supported. A copy of each texture is always placed in kernel memory for graphics context switching. (OpenGL semantics require the graphics library to read the texture exactly once; it cannot re-read the texture later to refresh texture memory after a graphics context switch). Loading subtextures is still possible, however; use **glTexSubImage2DEXT()** (see “The Subtexture Extension” on page 96).

The TX_BILINEAR_LEQUAL and TX_BILINEAR_GEQUAL filtering options, which are used to implement shadows, are not supported.

The TX_BICUBIC filter option, which is used for bicubic filtering and as a workaround for the lack of point sampling, is also not supported.

The TX_MINFILTER options for mipmapping are supported for 1D and 2D textures, but not for 3D textures. 3D textures must use GL_NEAREST (TX_POINT) or GL_LINEAR (TX_BILINEAR) filtering modes.

OpenGL differs from IRIS GL in that filtered versions of the texture image (when required by the current minification filter) are not generated automatically; the application must load them explicitly. Thus the TX_MIPMAP_FILTER_KERNEL token is not supported.

Separate magnification filters for color and alpha (TX_MAGFILTER_COLOR and TX_MAGFILTER_ALPHA) are not supported in the general case. However, it is possible to specify separate alpha and color magnification filters for detail and sharp texturing. See **glTexParameter()**.

Sharp texture filtering (TX_SHARPEN) is supported. Use **glTexParameter()** for setting the filtering mode, and **glSharpenTexFuncSGIS()** for setting the scaling function (TX_CONTROL_POINT, TX_CONTROL_CLAMP). See “The Sharpen Texture Extension” on page 107.

Detail texture (TX_ADD_DETAIL and TX_MODULATE_DETAIL) is supported for 2D. The parameters are specified differently from those in IRIS GL. See **glTexParameter()** for setting the filtering mode, and **glDetailTexFuncSGIS()** for setting the scaling function (TX_CONTROL_POINT, TX_CONTROL_CLAMP). See “The Detail Texture Extension” on page 112.

The TX_WRAP mode TX_SELECT is not supported. OpenGL provides GL_CLAMP (TX_CLAMP) and GL_REPEAT (TX_REPEAT).

TX_INTERNAL_FORMAT and all IRIS GL texel internal formats are supported. See the *components* parameter of **glTexImage2D()** for a list of the OpenGL internal formats. See also “The Texture Extension” on page 85 for new internal formats and how to use them.

TX_TILE (multipass rendering for high-resolution textures) is not supported directly. OpenGL border clamping can emulate tiling if you use the edges of neighboring tiles as the borders for the current tile.

tlutbind

In OpenGL, tlut definition and binding are combined into a single operation, and tluts apply to all texturing (rather than being bound to a particular texture target). See the comments under tlutdef.

tlutdef

Use **glTexColorTableParameterSGI()** for a description of the OpenGL texture color lookup process; see “The Texture Color Table Extension” on page 105. Use **glColorTableSGI()** for information about loading the lookup table; see “The Color Table Extension” on page 136.

The OpenGL lookup table semantics differ from those that IRIS GL used.

- The case described in the **tlutdef()** reference page and shown in the following table cannot be emulated in OpenGL. (nc stands for number of components, I for intensity, A for Alpha, R, G, and B for Red, Green, and Blue.)

tlut nc	texture nc	action
4	3	R, G, B, B looks up R, G, B, A

- The cases shown in the following table are supported directly, or can be supported with a judicious choice of table values and calls to **glEnable()**.

tlut nc	texture nc	action
2	1	I looks up I,A
	2	I,A looks up I,A
	3	R,G,B pass unchanged
	4	R,G,B,A pass unchanged
3	1	I looks up R,G,B
	2	I,A pass unchanged
	3	R,G,B looks up R,G,B
	4	R,G,B,A pass unchanged

tlut nc	texture nc	action
4	1	I looks up R,G,B,A
	2	I looks up RGB; A looks up A
	4	R,G,B,A looks up R,G,B,A

OpenGL supports cases that are not available under IRIS GL. See **glTexColorTableParameterSGI()** for more information.

underlays

There are no X11 Visuals for the underlay planes, so OpenGL rendering to underlays is not supported.