xref: /external/skia/src/gpu/gl/GrGLUtil.h

Home | History | Annotate | Line# | Navigate | Download

```
204 #endif
205
206 // internal macro to conditionally log the gl call using SkDebugf based on
207 // compile-time and run-time flags.
208 #if GR_GL_LOG_CALLS
209     extern bool gLogCallsGL;
210     #define GR_GL_LOG_CALLS_IMPL(X)                                 #
211         if (gLogCallsGL)                                            #
212             SkDebugf(GR_FILE_AND_LINE_STR "GL: " #X "\n")
213 #else
214     #define GR_GL_LOG_CALLS_IMPL(X)
215 #endif
216
217 // makes a GL call on the interface and does any error checking and logging
218 #define GR_GL_CALL(IFACE, X)                                        #
219     do {                                                            #
220         GR_GL_CALL_NOERRCHECK(IFACE, X);                            #
221         GR_GL_CHECK_ERROR_IMPL(IFACE, X);                           #
222     } while (false)
223
224 // Variant of above that always skips the error check. This is useful when
225 // the caller wants to do its own glGetError() call and examine the error value.
226 #define GR_GL_CALL_NOERRCHECK(IFACE, X)                             #
227     do {                                                            #
228         (IFACE)->fFunctions.f##X;                                   #
229         GR_GL_LOG_CALLS_IMPL(X);                                    #
230     } while (false)
231
232 // same as GR_GL_CALL but stores the return value of the gl call in RET
233 #define GR_GL_CALL_RET(IFACE, RET, X)                               #
234     do {                                                            #
235         GR_GL_CALL_RET_NOERRCHECK(IFACE, RET, X);                   #
236         GR_GL_CHECK_ERROR_IMPL(IFACE, X);                           #
237     } while (false)
238
239 // same as GR_GL_CALL_RET but always skips the error check.
240 #define GR_GL_CALL_RET_NOERRCHECK(IFACE, RET, X)                    #
241     do {                                                            #
242         (RET) = (IFACE)->fFunctions.f##X;                           #
243         GR_GL_LOG_CALLS_IMPL(X);                                    #
244     } while (false)
245
246 // call glGetError without doing a redundant error check or logging.
247 #define GR_GL_GET_ERROR(IFACE) (IFACE)->fFunctions.fGetError()
248
249 GrGLenum GrToGLStencilFunc(GrStencilTest test);
250
251 GrPixelConfig GrGLSizedFormatToPixelConfig(GrGLenum sizedFormat);
252
253 #endif
254
```

xref: /external/skia/include/gpu/gl/GrGLInterface.h

Home | History | Annotate | Line# | Navigate | Download

```
81  struct Functions {
82      GrGLFunction<GrGLActiveTextureProc> fActiveTexture;
83      GrGLFunction<GrGLAttachShaderProc> fAttachShader;
84      GrGLFunction<GrGLBeginQueryProc> fBeginQuery;
85      GrGLFunction<GrGLBindAttribLocationProc> fBindAttribLocation;
86      GrGLFunction<GrGLBindBufferProc> fBindBuffer;
87      GrGLFunction<GrGLBindFragDataLocationProc> fBindFragDataLocation;
88      GrGLFunction<GrGLBindFragDataLocationIndexedProc> fBindFragDataLocationIndexed;
89      GrGLFunction<GrGLBindFramebufferProc> fBindFramebuffer;
90      GrGLFunction<GrGLBindRenderbufferProc> fBindRenderbuffer;
91      GrGLFunction<GrGLBindTextureProc> fBindTexture;
92      GrGLFunction<GrGLBindVertexArrayProc> fBindVertexArray;
93      GrGLFunction<GrGLBlendBarrierProc> fBlendBarrier;
94      GrGLFunction<GrGLBlendColorProc> fBlendColor;
95      GrGLFunction<GrGLBlendEquationProc> fBlendEquation;
96      GrGLFunction<GrGLBlendFuncProc> fBlendFunc;
97      GrGLFunction<GrGLBlitFramebufferProc> fBlitFramebuffer;
98      GrGLFunction<GrGLBufferDataProc> fBufferData;
99      GrGLFunction<GrGLBufferSubDataProc> fBufferSubData;
100     GrGLFunction<GrGLCheckFramebufferStatusProc> fCheckFramebufferStatus;
101     GrGLFunction<GrGLClearProc> fClear;
102     GrGLFunction<GrGLClearColorProc> fClearColor;
103     GrGLFunction<GrGLClearStencilProc> fClearStencil;
104     GrGLFunction<GrGLClearTexImageProc> fClearTexImage;
105     GrGLFunction<GrGLClearTexSubImageProc> fClearTexSubImage;
106     GrGLFunction<GrGLColorMaskProc> fColorMask;
107     GrGLFunction<GrGLCompileShaderProc> fCompileShader;
108     GrGLFunction<GrGLCompressedTexImage2DProc> fCompressedTexImage2D;
109     GrGLFunction<GrGLCompressedTexSubImage2DProc> fCompressedTexSubImage2D;
110     GrGLFunction<GrGLCopyTexSubImage2DProc> fCopyTexSubImage2D;
111     GrGLFunction<GrGLCreateProgramProc> fCreateProgram;
112     GrGLFunction<GrGLCreateShaderProc> fCreateShader;
113     GrGLFunction<GrGLCullFaceProc> fCullFace;
114     GrGLFunction<GrGLDeleteBuffersProc> fDeleteBuffers;
115     GrGLFunction<GrGLDeleteFramebuffersProc> fDeleteFramebuffers;
116     GrGLFunction<GrGLDeleteProgramProc> fDeleteProgram;
117     GrGLFunction<GrGLDeleteQueriesProc> fDeleteQueries;
118     GrGLFunction<GrGLDeleteRenderbuffersProc> fDeleteRenderbuffers;
119     GrGLFunction<GrGLDeleteShaderProc> fDeleteShader;
120     GrGLFunction<GrGLDeleteTexturesProc> fDeleteTextures;
121     GrGLFunction<GrGLDeleteVertexArraysProc> fDeleteVertexArrays;
122     GrGLFunction<GrGLDepthMaskProc> fDepthMask;
123     GrGLFunction<GrGLDisableProc> fDisable;
124     GrGLFunction<GrGLDisableVertexAttribArrayProc> fDisableVertexAttribArray;
125     GrGLFunction<GrGLDrawArraysProc> fDrawArrays;
126     GrGLFunction<GrGLDrawArraysIndirectProc> fDrawArraysIndirect;
127     GrGLFunction<GrGLDrawArraysInstancedProc> fDrawArraysInstanced;
128     GrGLFunction<GrGLDrawBufferProc> fDrawBuffer;
129     GrGLFunction<GrGLDrawBuffersProc> fDrawBuffers;
130     GrGLFunction<GrGLDrawElementsProc> fDrawElements;
131     GrGLFunction<GrGLDrawElementsIndirectProc> fDrawElementsIndirect;
132     GrGLFunction<GrGLDrawElementsInstancedProc> fDrawElementsInstanced;
133     GrGLFunction<GrGLDrawRangeElementsProc> fDrawRangeElements;
134     GrGLFunction<GrGLEnableProc> fEnable;
135     GrGLFunction<GrGLEnableVertexAttribArrayProc> fEnableVertexAttribArray;
136     GrGLFunction<GrGLEndQueryProc> fEndQuery;
```

```
D: ----------ThreadedRenderer start---------------------
D: glCommand1: BufferData(target, srcSizeInBytes, src, fUsage)
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: BufferData(target, srcSizeInBytes, src, fUsage)
D: glCommand1: ActiveTexture(0x84C0 + lastUnitIdx)
D: glCommand1: BindTexture(glTex->target(), glTex->textureID())
D: glCommand1: PixelStorei(GR_GL_UNPACK_ALIGNMENT, config_alignment(texConfig))
D: glCommand1: TexSubImage2D(target, currentMipLevel, left, top, currentWidth, currentHeight, externalFormat, externalType, texelsShallowCopy[currentMipLevel].fPixels)
D: glCommand1: BindFramebuffer(0x8D40, target->renderFBOID())
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: EnableVertexAttribArray(i)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: ActiveTexture(0x84C0 + unit)
D: glCommand1: BindTexture(target, texture->textureID())
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: BindTexture(target, texture->textureID())
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawArrays(glPrimType, 0, vertexCount)
D: glCommand1: UseProgram(programID)
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: BindTexture(target, texture->textureID())
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribIPointer(index, layout.fCount, layout.fType, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: Enable(0x0C11)
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
D: glCommand1: UseProgram(programID)
D: glCommand1: Disable(0x0C11)
D: glCommand1: BindBuffer(bufferState.fGLTarget, glBuffer->bufferID())
D: glCommand1: DisableVertexAttribArray(i)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: VertexAttribPointer(index, layout.fCount, layout.fType, layout.fNormalized, stride, offsetAsPtr)
D: glCommand1: DrawRangeElements(glPrimType, minIndexValue, maxIndexValue, indexCount, 0x1403, indices)
```

쌓아둔 명령들은 대체적으로 GPU가 임의로 수행하지만, 강제로 GPU로 보내버리는 방법도 있다. 이때 명령을 보내는 것은 명시적이거나 암시적으로 수행할 수 있다. Rendering API인 OpenGL이 명시적으로 수행하는 방법은 두 가지인데 첫 번째로는 glFlush()를 사용하는 것이다. glFlush()를 호출하면, 현재 스레드에 eglMakeCurrent()를 통해 연결된 EGLContext에 아직 전달되지 않고 쌓여있는 GL 명령들을 모두 GPU로 보내버린다. glFlush()는 명령의 전달 완료 시점에서 바로 반환하는 함수이다. 해당 명령으로 인한 렌더링 결과 종료까지 CPU 코드의 진행을 멈추게 하고자 할 경우 이와 유사한 명시적 명령인 glFinish()를 사용한다. glFinish()는 완벽한 렌더링 결과를 얻고 나서야 함수가 반환될 것을 보장한다. 하지만 대체적으로 Window surface를 사용하는 경우엔 이미 eglSwapBuffers()를 이용하여 front/back 버퍼에 대해 병렬적으로 접근/렌더링을 하므로 이를 이용하는 편이 새로운 프레임을 더 빠르고 안정적으로 얻을 수 있어 보편적인 경우에 더 좋은 성능을 발휘한다. eglSwapBuffers()는 암시적으로 모든 GL 명령들을 현재 surface에 적용하라고 보낸 뒤에서야 이미지 버퍼를 swap 한다. 즉, glFlush()를 드라이버 차원에서 수행해준다고 볼 수 있다.

xref: /frameworks/base/libs/hwui/renderthread/CanvasContext.cpp

Home | History | Annotate | Line# | Navigate | Download Search only in Canva

```
438
439 void CanvasContext::draw() {
440     SkRect dirty;
441     mDamageAccumulator.finish(&dirty);
442
443     // TODO: Re-enable after figuring out cause of b/22592975
444     //     if (dirty.isEmpty() && Properties::skipEmptyFrames) {
445     //         mCurrentFrameInfo->addFlag(FrameInfoFlags::SkippedFrame);
446     //         return;
447     //     }
448
449     mCurrentFrameInfo->markIssueDrawCommandsStart();
450
451     Frame frame = mRenderPipeline->getFrame();
452
453     SkRect windowDirty = computeDirtyRect(frame, &dirty);
454
455     bool drew = mRenderPipeline->draw(frame, windowDirty, dirty, mLightGeometry, &mLayerUpdateQueue,
456                                       mContentDrawBounds, mOpaque, mWideColorGamut, mLightInfo,
457                                       mRenderNodes, &(profiler()));
458
459     int64_t frameCompleteNr = mFrameCompleteCallbacks.size() ? getFrameNumber() : -1;
460
461     waitOnFences();
462
463     bool requireSwap = false;
464     bool didSwap =
465             mRenderPipeline->swapBuffers(frame, drew, windowDirty, mCurrentFrameInfo, &requireSwap);
466
467     mIsDirty = false;
```

xref: /frameworks/base/libs/hwui/pipeline/skia/SkiaOpenGLPipeline.cpp

Home | History | Annotate | Line# | Navigate | Download Search only in SkiaOpenG

```
107
108 bool SkiaOpenGLPipeline::swapBuffers(const Frame& frame, bool drew, const SkRect& screenDirty,
109                                       FrameInfo* currentFrameInfo, bool* requireSwap) {
110     GL_CHECKPOINT(LOW);
111
112     // Even if we decided to cancel the frame, from the perspective of jank
113     // metrics the frame was swapped at this point
114     currentFrameInfo->markSwapBuffers();
115
116     *requireSwap = drew || mEglManager.damageRequiresSwap();
117
118     if (*requireSwap && (CC_UNLIKELY(!mEglManager.swapBuffers(frame, screenDirty)))) {
119         return false;
120     }
121
122     return *requireSwap;
123 }
```

xref: /frameworks/base/libs/hwui/pipeline/skia/SkiaOpenGLPipeline.cpp

Home | History | Annotate | Line# | Navigate | Download Search only in SkiaOpenG

```
107
108 bool SkiaOpenGLPipeline::swapBuffers(const Frame& frame, bool drew, const SkRect& screenDirty,
109                                       FrameInfo* currentFrameInfo, bool* requireSwap) {
110     GL_CHECKPOINT(LOW);
111
112     // Even if we decided to cancel the frame, from the perspective of jank
113     // metrics the frame was swapped at this point
114     currentFrameInfo->markSwapBuffers();
115
116     *requireSwap = drew || mEglManager.damageRequiresSwap();
117
118     if (*requireSwap && (CC_UNLIKELY(!mEglManager.swapBuffers(frame, screenDirty)))) {
119         return false;
120     }
121
122     return *requireSwap;
123 }
```

xref: /frameworks/base/libs/hwui/renderthread/EglManager.cpp

Home | History | Annotate | Line# | Navigate | Download Search only in EglManager.cpp

```
433
434 bool EglManager::swapBuffers(const Frame& frame, const SkRect& screenDirty) {
435     if (CC_UNLIKELY(Properties::waitForGpuCompletion)) {
436         ATRACE_NAME("Finishing GPU work");
437         fence();
438     }
439
440     EGLint rects[4];
441     frame.map(screenDirty, rects);
442     eglSwapBuffersWithDamageKHR(mEglDisplay, frame.mSurface, rects, screenDirty.isEmpty() ? 0 : 1);
443
444     EGLint err = eglGetError();
445     if (CC_LIKELY(err == EGL_SUCCESS)) {
446         return true;
447     }
448     if (err == EGL_BAD_SURFACE || err == EGL_BAD_NATIVE_WINDOW) {
449         // For some reason our surface was destroyed out from under us
450         // This really shouldn't happen, but if it does we can recover easily
451         // by just not trying to use the surface anymore
452         ALOGW("swapBuffers encountered EGL error %d on %p, halting rendering...", err,
453               frame.mSurface);
454         return false;
455     }
456     LOG_ALWAYS_FATAL("Encountered EGL error %d %s during rendering", err, egl_error_str(err));
457     // Impossible to hit this, but the compiler doesn't know that
458     return false;
459 }
460
```

xref: /frameworks/base/libs/hwui/renderthread/EglManager.cpp

Home | History | Annotate | Line# | Navigate | Download Search only in EglManager.cpp

```
433
434 bool EglManager::swapBuffers(const Frame& frame, const SkRect& screenDirty) {
435     if (CC_UNLIKELY(Properties::waitForGpuCompletion)) {
436         ATRACE_NAME("Finishing GPU work");
437         fence();
438     }
439
440     EGLint rects[4];
441     frame.map(screenDirty, rects);
442     eglSwapBuffersWithDamageKHR(mEglDisplay, frame.mSurface, rects, screenDirty.isEmpty() ? 0 : 1);
443
444     EGLint err = eglGetError();
445     if (CC_LIKELY(err == EGL_SUCCESS)) {
446         return true;
447     }
448     if (err == EGL_BAD_SURFACE || err == EGL_BAD_NATIVE_WINDOW) {
449         // For some reason our surface was destroyed out from under us
450         // This really shouldn't happen, but if it does we can recover easily
451         // by just not trying to use the surface anymore
452         ALOGW("swapBuffers encountered EGL error %d on %p, halting rendering...", err,
453               frame.mSurface);
454         return false;
455     }
456     LOG_ALWAYS_FATAL("Encountered EGL error %d %s during rendering", err, egl_error_str(err));
457     // Impossible to hit this, but the compiler doesn't know that
458     return false;
459 }
460
```

```
1373     if (!dp) return EGL_FALSE;
1374
1375     SurfaceRef _s(dp.get(), draw);
1376     if (!_s.get())
1377         return setError(EGL_BAD_SURFACE, (EGLBoolean)EGL_FALSE);
1378
1379     egl_surface_t* const s = get_surface(draw);
1380
1381     if (CC_UNLIKELY(dp->traceGpuCompletion)) {
1382         EGLSyncKHR sync = eglCreateSyncKHR(dpy, EGL_SYNC_FENCE_KHR, NULL);
1383         if (sync != EGL_NO_SYNC_KHR) {
1384             FrameCompletionThread::queueSync(sync);
1385         }
1386     }
1387
1388     if (CC_UNLIKELY(dp->finishOnSwap)) {
1389         uint32_t pixel;
1390         egl_context_t * const c = get_context( egl_tls_t::getContext() );
1391         if (c) {
1392             // glReadPixels() ensures that the frame is complete
1393             s->cnx->hooks[c->version]->gl.glReadPixels(0,0,1,1,
1394                     GL_RGBA,GL_UNSIGNED_BYTE,&pixel);
1395         }
1396     }
1397
1398     if (!sendSurfaceMetadata(s)) {
1399         native_window_api_disconnect(s->getNativeWindow(), NATIVE_WINDOW_API_EGL);
1400         return setError(EGL_BAD_NATIVE_WINDOW, (EGLBoolean)EGL_FALSE);
1401     }
1402
1403     if (n_rects == 0) {
1404         return s->cnx->egl.eglSwapBuffers(dp->disp.dpy, s->surface);
1405     }
1406
1407     std::vector<android_native_rect_t> androidRects((size_t)n_rects);
1408     for (int r = 0; r < n_rects; ++r) {
1409         int offset = r * 4;
1410         int x = rects[offset];
1411         int y = rects[offset + 1];
1412         int width = rects[offset + 2];
1413         int height = rects[offset + 3];
1414         android_native_rect_t androidRect;
1415         androidRect.left = x;
1416         androidRect.top = y + height;
1417         androidRect.right = x + width;
1418         androidRect.bottom = y;
1419         androidRects.push_back(androidRect);
1420     }
1421     native_window_set_surface_damage(s->getNativeWindow(), androidRects.data(), androidRects.size());
1422
1423     if (s->cnx->egl.eglSwapBuffersWithDamageKHR) {
1424         return s->cnx->egl.eglSwapBuffersWithDamageKHR(dp->disp.dpy, s->surface,
1425                 rects, n_rects);
1426     } else {
1427         return s->cnx->egl.eglSwapBuffers(dp->disp.dpy, s->surface);
```

```
1398    if (!sendSurfaceMetadata(s)) {
1399        native_window_api_disconnect(s->getNativeWindow(), NATIVE_WINDOW_API_EGL);
1400        return setError(EGL_BAD_NATIVE_WINDOW, (EGLBoolean)EGL_FALSE);
1401    }
1402
1403    if (n_rects == 0) {
1404        return s->cnx->egl.eglSwapBuffers(dp->disp.dpy, s->surface);
1405    }
1406
1407    std::vector<android_native_rect_t> androidRects((size_t)n_rects);
1408    for (int r = 0; r < n_rects; ++r) {
1409        int offset = r * 4;
1410        int x = rects[offset];
1411        int y = rects[offset + 1];
1412        int width = rects[offset + 2];
1413        int height = rects[offset + 3];
1414        android_native_rect_t androidRect;
1415        androidRect.left = x;
1416        androidRect.top = y + height;
1417        androidRect.right = x + width;
1418        androidRect.bottom = y;
1419        androidRects.push_back(androidRect);
1420    }
1421    native_window_set_surface_damage(s->getNativeWindow(), androidRects.data(), androidRects.size());
1422
1423    if (s->cnx->egl.eglSwapBuffersWithDamageKHR) {
1424        return s->cnx->egl.eglSwapBuffersWithDamageKHR(dp->disp.dpy, s->surface,
1425                rects, n_rects);
1426    } else {
1427        return s->cnx->egl.eglSwapBuffers(dp->disp.dpy, s->surface);
1428    }
1429 }
```

xref: /frameworks/native/opengl/libs/EGL/egl_entries.in

Home | History | Annotate | Line# | Navigate | Download Search only in egl_entries.in

```
1  EGL_ENTRY(EGLDisplay, eglGetDisplay, NativeDisplayType)
2  EGL_ENTRY(EGLBoolean, eglInitialize, EGLDisplay, EGLint*, EGLint*)
3  EGL_ENTRY(EGLBoolean, eglTerminate, EGLDisplay)
4  EGL_ENTRY(EGLBoolean, eglGetConfigs, EGLDisplay, EGLConfig*, EGLint, EGLint*)
5  EGL_ENTRY(EGLBoolean, eglChooseConfig, EGLDisplay, const EGLint *, EGLConfig *, EGLint, EGLint *)
6
7  EGL_ENTRY(EGLBoolean, eglGetConfigAttrib, EGLDisplay, EGLConfig, EGLint, EGLint *)
8  EGL_ENTRY(EGLSurface, eglCreateWindowSurface, EGLDisplay, EGLConfig, NativeWindowType, const EGLint *)
9  EGL_ENTRY(EGLSurface, eglCreatePixmapSurface, EGLDisplay, EGLConfig, NativePixmapType, const EGLint *)
10 EGL_ENTRY(EGLSurface, eglCreatePbufferSurface,  EGLDisplay, EGLConfig, const EGLint *)
11 EGL_ENTRY(EGLBoolean, eglDestroySurface, EGLDisplay, EGLSurface)
12 EGL_ENTRY(EGLBoolean, eglQuerySurface,  EGLDisplay, EGLSurface, EGLint, EGLint *)
13 EGL_ENTRY(EGLContext, eglCreateContext, EGLDisplay, EGLConfig, EGLContext, const EGLint *)
14 EGL_ENTRY(EGLBoolean, eglDestroyContext, EGLDisplay, EGLContext)
15 EGL_ENTRY(EGLBoolean, eglMakeCurrent, EGLDisplay, EGLSurface, EGLSurface, EGLContext)
16 EGL_ENTRY(EGLContext, eglGetCurrentContext, void)
17 EGL_ENTRY(EGLSurface, eglGetCurrentSurface, EGLint)
18 EGL_ENTRY(EGLDisplay, eglGetCurrentDisplay, void)
19 EGL_ENTRY(EGLBoolean, eglQueryContext,  EGLDisplay, EGLContext, EGLint, EGLint *)
20 EGL_ENTRY(EGLBoolean, eglWaitGL, void)
21 EGL_ENTRY(EGLBoolean, eglWaitNative, EGLint)
22 EGL_ENTRY(EGLBoolean, eglSwapBuffers, EGLDisplay, EGLSurface)
23 EGL_ENTRY(EGLBoolean, eglCopyBuffers, EGLDisplay, EGLSurface, NativePixmapType)
24 EGL_ENTRY(EGLint, eglGetError, void)
25 EGL_ENTRY(const char*, eglQueryString, EGLDisplay, EGLint)
26 EGL_ENTRY(__eglMustCastToProperFunctionPointerType, eglGetProcAddress, const char *)
27
```