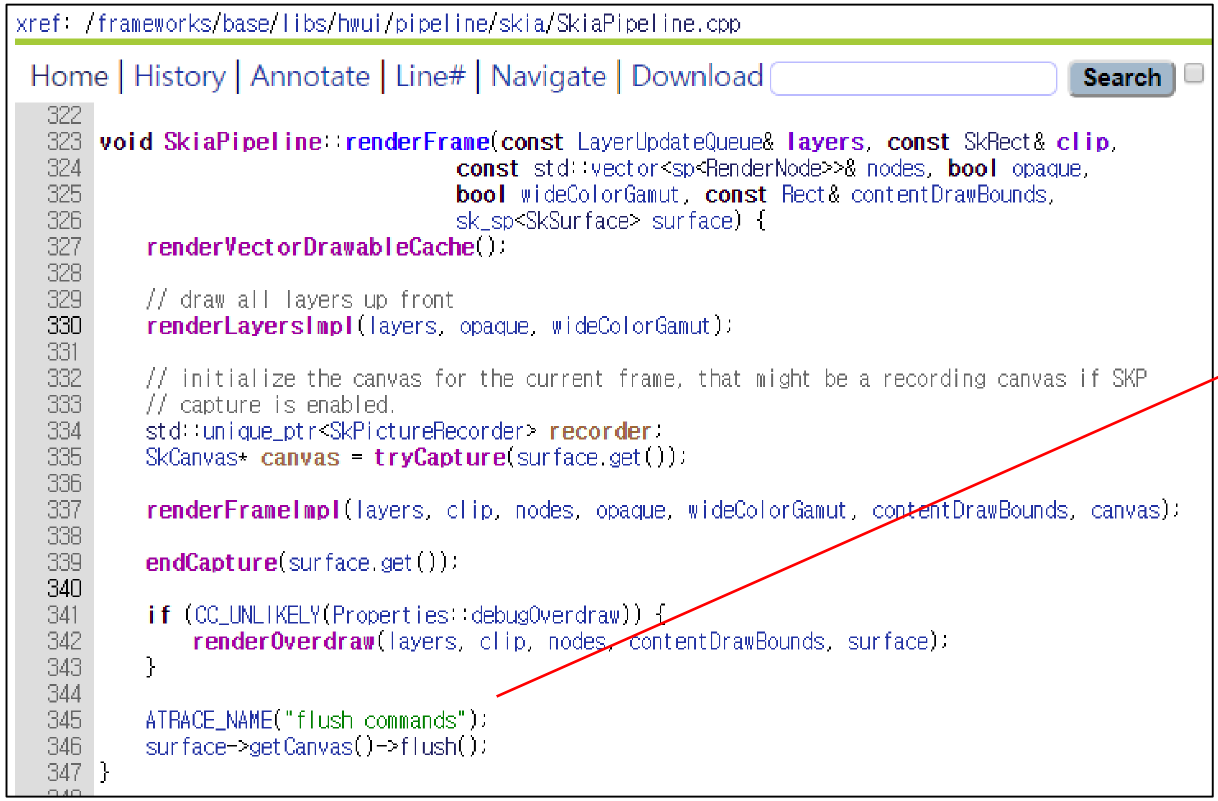
Drawing Path

Pie(9.0)

openGL->GPU(flush)

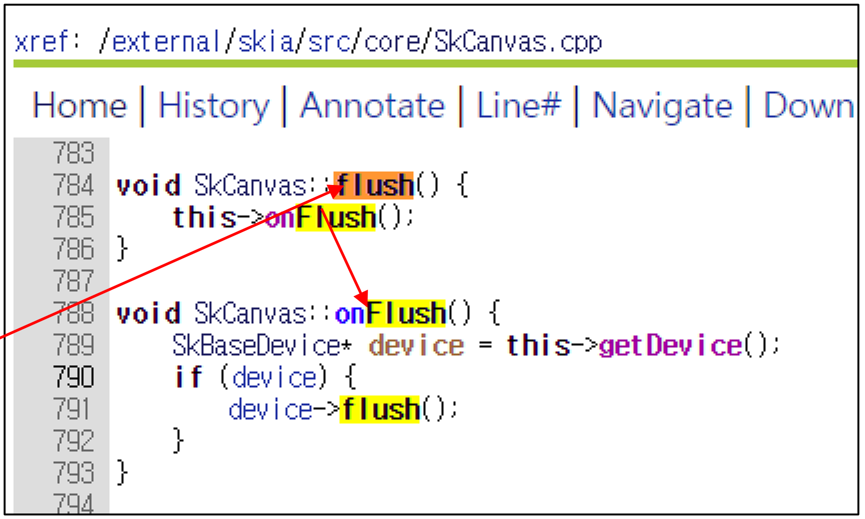
자세한 코드



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

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

```
void SkiaPipeline::renderFrame(const LayerUpdateQueue& layers, const SkRect& clip,
                               const std::vector<sp<RenderNode>>& nodes, bool opaque,
                               bool wideColorGamut, const Rect& contentDrawBounds,
                               sk_sp<SkSurface> surface) {
    renderVectorDrawableCache();

    // draw all layers up front
    renderLayersImpl(layers, opaque, wideColorGamut);

    // initialize the canvas for the current frame, that might be a recording canvas if SKP
    // capture is enabled.
    std::unique_ptr<SkPictureRecorder> recorder;
    SkCanvas* canvas = tryCapture(surface.get());

    renderFrameImpl(layers, clip, nodes, opaque, wideColorGamut, contentDrawBounds, canvas);

    endCapture(surface.get());

    if (CC_UNLIKELY(Properties::debugOverdraw)) {
        renderOverdraw(layers, clip, nodes, contentDrawBounds, surface);
    }

    ATRACE_NAME("flush commands");
    surface->getCanvas()->flush();
}
```

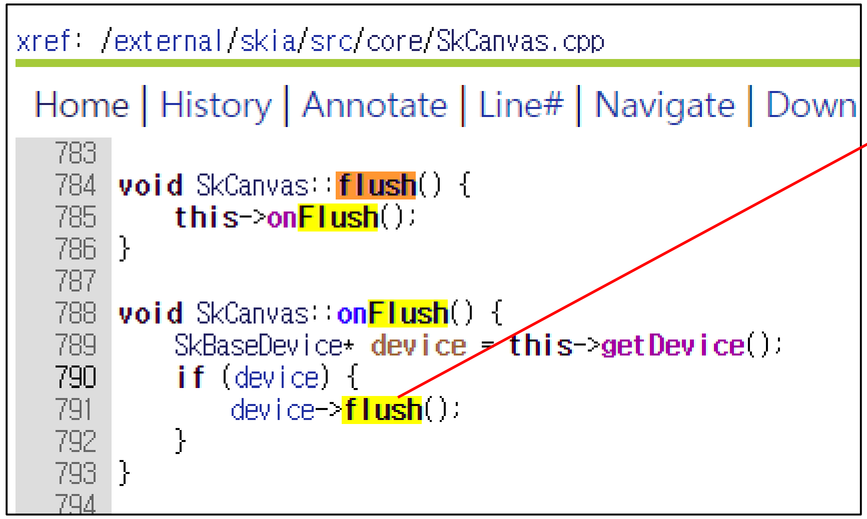


xref: /external/skia/src/core/SkCanvas.cpp

Home | History | Annotate | Line# | Navigate | Down

```

void SkCanvas::flush() {
    this->onFlush();
}

void SkCanvas::onFlush() {
    SkBaseDevice* device = this->getDevice();
    if (device) {
        device->flush();
    }
}
```

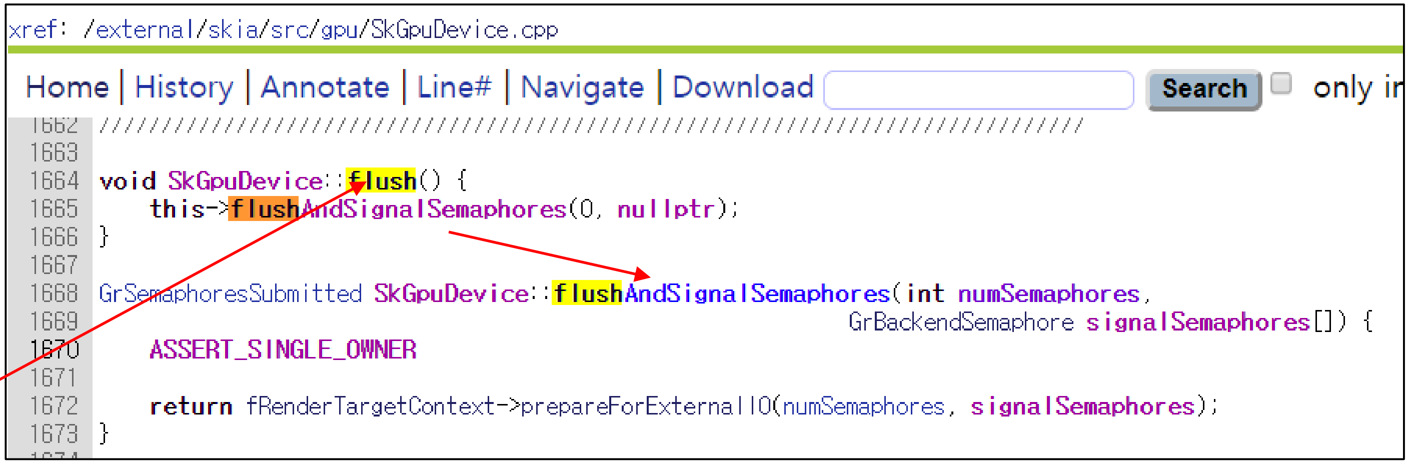
자세한 코드



xref: /external/skia/src/core/SkCanvas.cpp

Home | History | Annotate | Line# | Navigate | Down

```
void SkCanvas::flush() {
    this->onFlush();
}

void SkCanvas::onFlush() {
    SkBaseDevice* device = this->getDevice();
    if (device) {
        device->flush();
    }
}
```

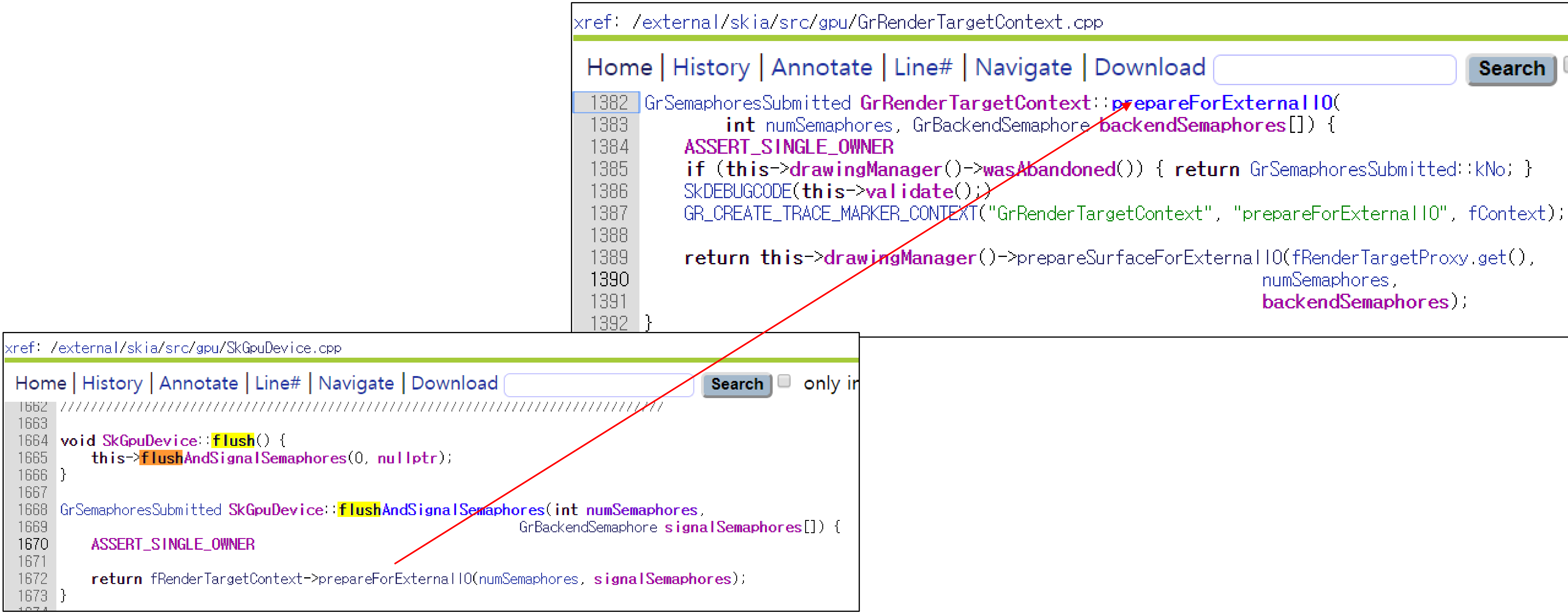


xref: /external/skia/src/gpu/SkGpuDevice.cpp

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

```
///////////////////////////////////////////////////////////////////////////////

void SkGpuDevice::flush() {
    this->flushAndSignalSemaphores(0, nullptr);
}

GrSemaphoresSubmitted SkGpuDevice::flushAndSignalSemaphores(int numSemaphores,
                                                            GrBackendSemaphore signalSemaphores[]) {
    ASSERT_SINGLE_OWNER

    return fRenderTargetContext->prepareForExternalIO(numSemaphores, signalSemaphores);
}
```

자세한 코드


Home | History | Annotate | Line# | Navigate | Download
Search
Home | History | Annotate | Line# | Navigate | Download
Search
only i

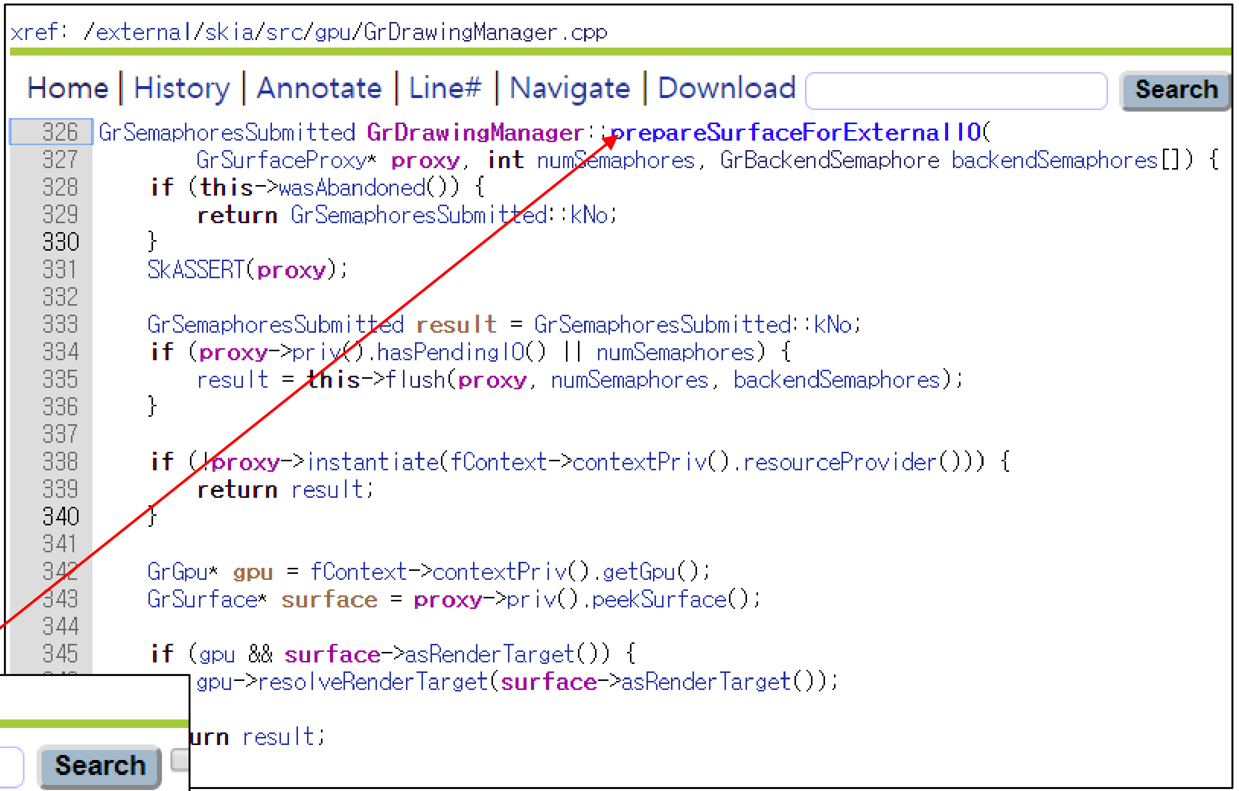
xref: /external/skia/src/gpu/GrRenderTargetContext.cpp

```
GrSemaphoresSubmitted GrRenderTargetContext::prepareForExternalIO(
        int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
    ASSERT_SINGLE_OWNER
    if (this->drawingManager()->wasAbandoned()) { return GrSemaphoresSubmitted::kNo; }
    SkDEBUGCODE(this->validate();)
    GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "prepareForExternalIO", fContext);

    return this->drawingManager()->prepareSurfaceForExternalIO(fRenderTargetProxy.get(),
                                                              numSemaphores,
                                                              backendSemaphores);
}
```

xref: /external/skia/src/gpu/SkGpuDevice.cpp

```
///////////////////////////////////////////////////////////////////////////////

void SkGpuDevice::flush() {
    this->flushAndSignalSemaphores(0, nullptr);
}

GrSemaphoresSubmitted SkGpuDevice::flushAndSignalSemaphores(int numSemaphores,
                                                           GrBackendSemaphore signalSemaphores[]) {
    ASSERT_SINGLE_OWNER

    return fRenderTargetContext->prepareForExternalIO(numSemaphores, signalSemaphores);
}
```

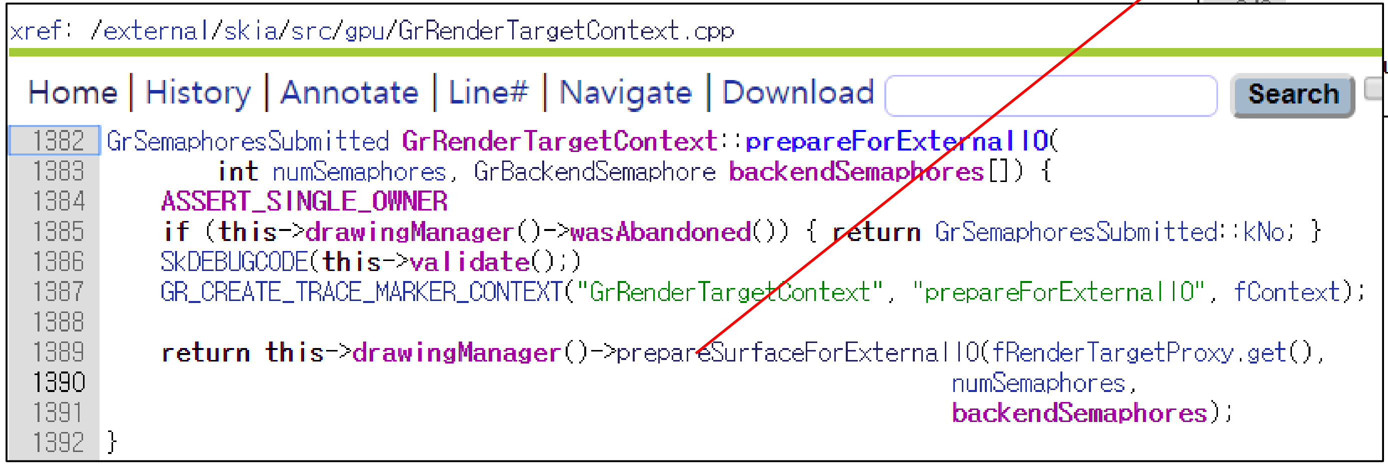
자세한 코드



xref: /external/skia/src/gpu/GrDrawingManager.cpp

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

```
GrSemaphoresSubmitted GrDrawingManager::prepareSurfaceForExternalIO(
        GrSurfaceProxy* proxy, int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
    if (this->wasAbandoned()) {
        return GrSemaphoresSubmitted::kNo;
    }
    SkASSERT(proxy);

    GrSemaphoresSubmitted result = GrSemaphoresSubmitted::kNo;
    if (proxy->priv().hasPendingIO() || numSemaphores) {
        result = this->flush(proxy, numSemaphores, backendSemaphores);
    }

    if (!proxy->instantiate(fContext->contextPriv().resourceProvider())) {
        return result;
    }

    GrGpu* gpu = fContext->contextPriv().getGpu();
    GrSurface* surface = proxy->priv().peekSurface();

    if (gpu && surface->asRenderTarget()) {
        gpu->resolveRenderTarget(surface->asRenderTarget());

urn result;
```

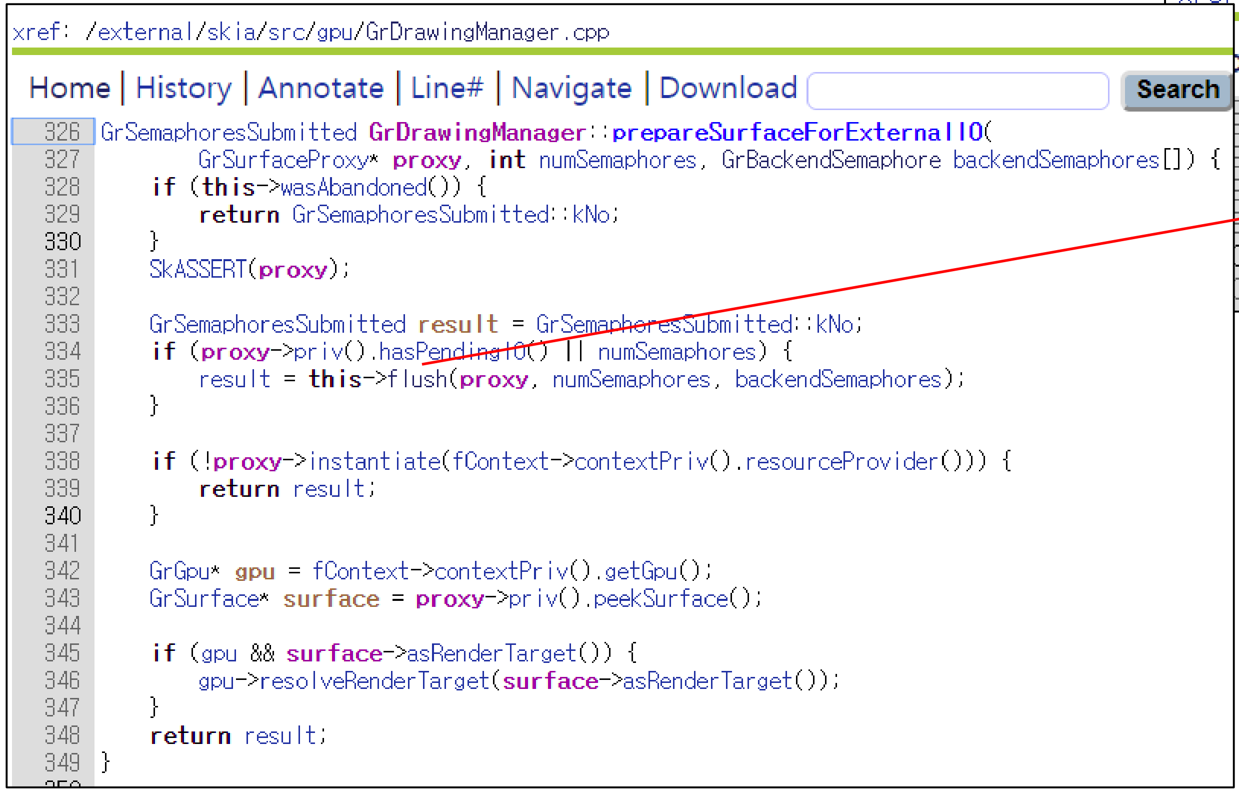


xref: /external/skia/src/gpu/GrRenderTargetContext.cpp

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

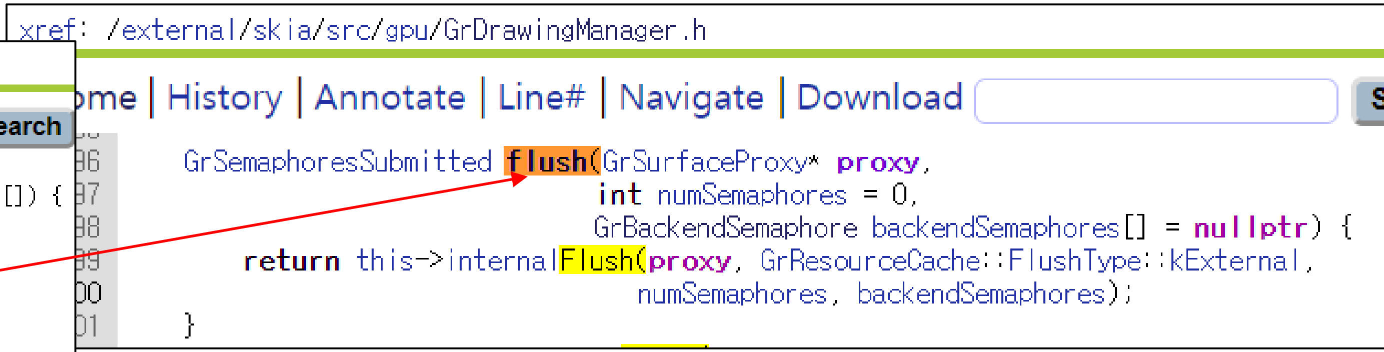
```
GrSemaphoresSubmitted GrRenderTargetContext::prepareForExternalIO(
        int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
    ASSERT_SINGLE_OWNER
    if (this->drawingManager()->wasAbandoned()) { return GrSemaphoresSubmitted::kNo; }
    SkDEBUGCODE(this->validate();)
    GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "prepareForExternalIO", fContext);

    return this->drawingManager()->prepareSurfaceForExternalIO(fRenderTargetProxy.get(),
                                                               numSemaphores,
                                                               backendSemaphores);
}
```

자세한 코드



xref: /external/skia/src/gpu/GrDrawingManager.cpp

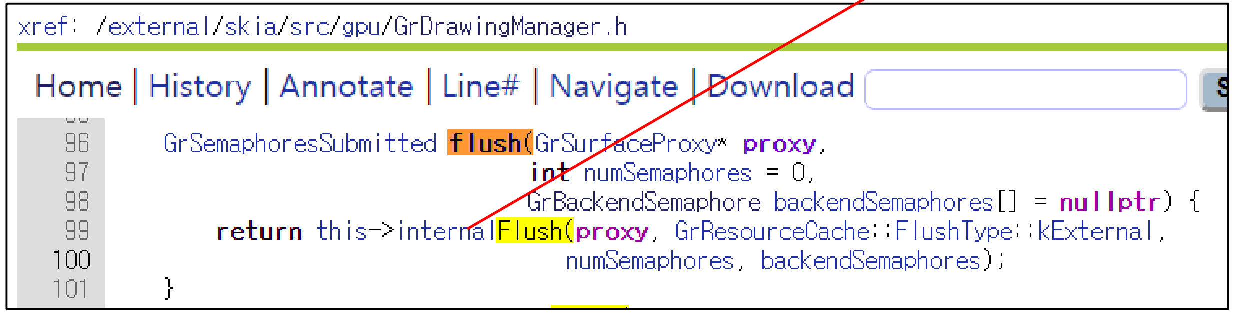
```
GrSemaphoresSubmitted GrDrawingManager::prepareSurfaceForExternalIO(
        GrSurfaceProxy* proxy, int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
    if (this->wasAbandoned()) {
        return GrSemaphoresSubmitted::kNo;
    }
    SkASSERT(proxy);

    GrSemaphoresSubmitted result = GrSemaphoresSubmitted::kNo;
    if (proxy->priv().hasPendingIO() || numSemaphores) {
        result = this->flush(proxy, numSemaphores, backendSemaphores);
    }

    if (!proxy->instantiate(fContext->contextPriv().resourceProvider())) {
        return result;
    }

    GrGpu* gpu = fContext->contextPriv().getGpu();
    GrSurface* surface = proxy->priv().peekSurface();

    if (gpu && surface->asRenderTarget()) {
        gpu->resolveRenderTarget(surface->asRenderTarget());
    }
    return result;
}
```



xref: /external/skia/src/gpu/GrDrawingManager.h

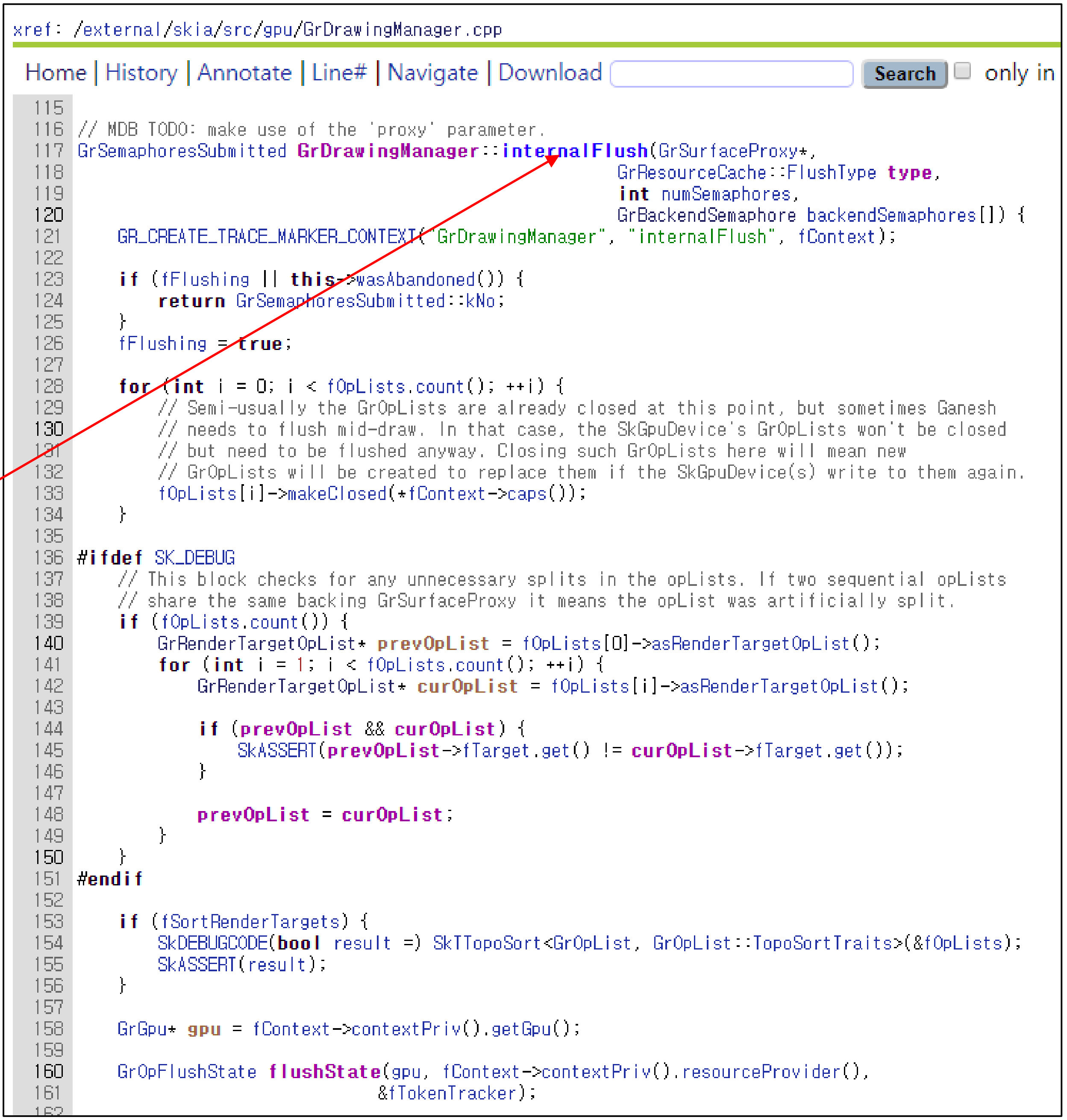
```
    GrSemaphoresSubmitted flush(GrSurfaceProxy* proxy,
                                int numSemaphores = 0,
                                GrBackendSemaphore backendSemaphores[] = nullptr) {
        return this->internalFlush(proxy, GrResourceCache::FlushType::kExternal,
                                   numSemaphores, backendSemaphores);
    }
```

자세한 코드


xref: /external/skia/src/gpu/GrDrawingManager.h

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

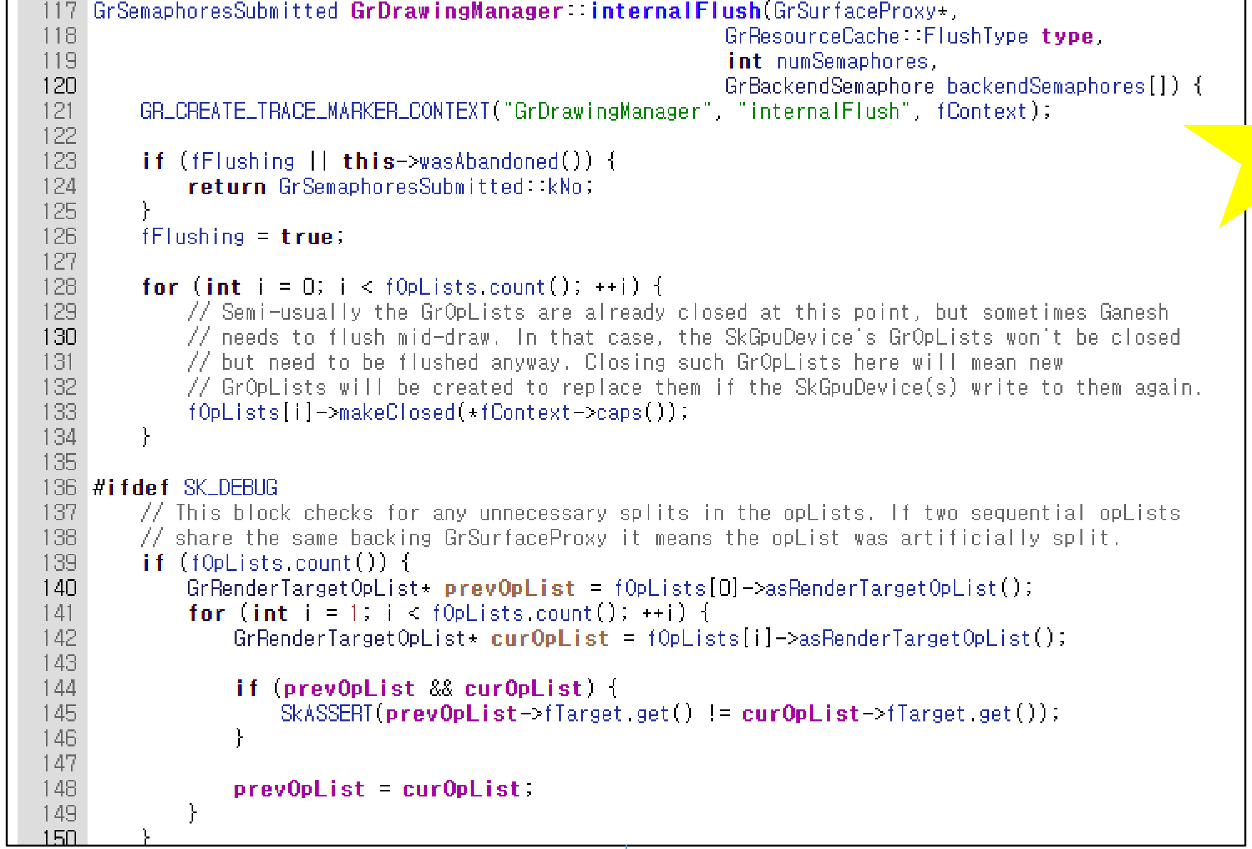
```
    GrSemaphoresSubmitted flush(GrSurfaceProxy* proxy,
                                int numSemaphores = 0,
                                GrBackendSemaphore backendSemaphores[] = nullptr) {
        return this->internalFlush(proxy, GrResourceCache::FlushType::kExternal,
                                   numSemaphores, backendSemaphores);
    }
```


xref: /external/skia/src/gpu/GrDrawingManager.cpp

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

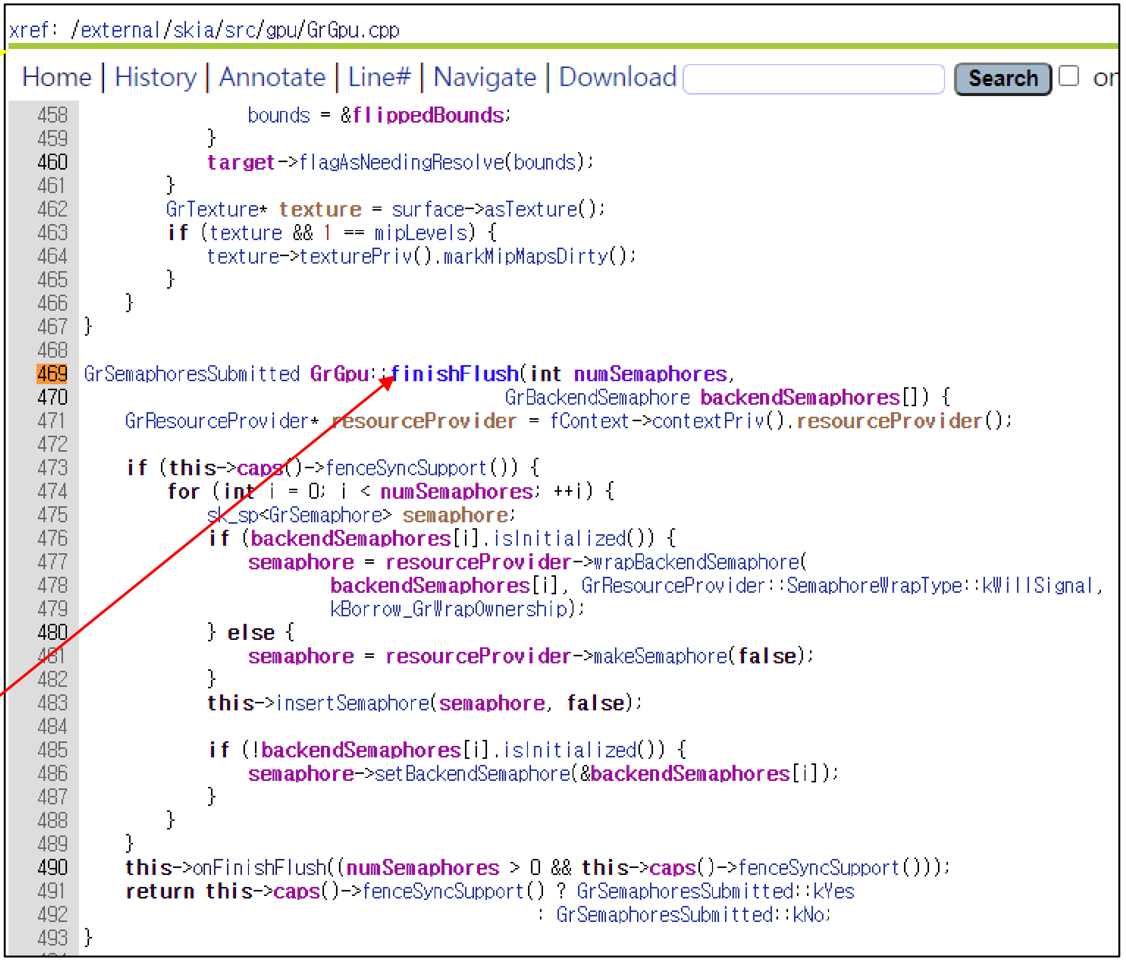
Search only in

```

// MDB TODO: make use of the 'proxy' parameter.
GrSemaphoresSubmitted GrDrawingManager::internalFlush(GrSurfaceProxy*,
                                                      GrResourceCache::FlushType type,
                                                      int numSemaphores,
                                                      GrBackendSemaphore backendSemaphores[]) {
    GR_CREATE_TRACE_MARKER_CONTEXT("GrDrawingManager", "internalFlush", fContext);

    if (fFlushing || this->wasAbandoned()) {
        return GrSemaphoresSubmitted::kNo;
    }
    fFlushing = true;

    for (int i = 0; i < fOpLists.count(); ++i) {
        // Semi-usually the GrOpLists are already closed at this point, but sometimes Ganesh
        // needs to flush mid-draw. In that case, the SkGpuDevice's GrOpLists won't be closed
        // but need to be flushed anyway. Closing such GrOpLists here will mean new
        // GrOpLists will be created to replace them if the SkGpuDevice(s) write to them again.
        fOpLists[i]->makeClosed(*fContext->caps());
    }

#ifdef SK_DEBUG
    // This block checks for any unnecessary splits in the opLists. If two sequential opLists
    // share the same backing GrSurfaceProxy it means the opList was artificially split.
    if (fOpLists.count()) {
        GrRenderTargetOpList* prevOpList = fOpLists[0]->asRenderTargetOpList();
        for (int i = 1; i < fOpLists.count(); ++i) {
            GrRenderTargetOpList* curOpList = fOpLists[i]->asRenderTargetOpList();

            if (prevOpList && curOpList) {
                SkASSERT(prevOpList->fTarget.get() != curOpList->fTarget.get());
            }

            prevOpList = curOpList;
        }
    }
#endif

    if (fSortRenderTargets) {
        SkDEBUGCODE(bool result =) SkTTopoSort<GrOpList, GrOpList::TopoSortTraits>(&fOpLists);
        SkASSERT(result);
    }

    GrGpu* gpu = fContext->contextPriv().getGpu();

    GrOpFlushState flushState(gpu, fContext->contextPriv().resourceProvider(),
                              &fTokenTracker);
```



```
GrSemaphoresSubmitted GrDrawingManager::internalFlush(GrSurfaceProxy*,
                                                       GrResourceCache::FlushType type,
                                                       int numSemaphores,
                                                       GrBackendSemaphore backendSemaphores[]) {
    GR_CREATE_TRACE_MARKER_CONTEXT("GrDrawingManager", "internalFlush", fContext);

    if (fFlushing || this->wasAbandoned()) {
        return GrSemaphoresSubmitted::kNo;
    }
    fFlushing = true;

    for (int i = 0; i < fOpLists.count(); ++i) {
        // Semi-usually the GrOpLists are already closed at this point, but sometimes Ganesh
        // needs to flush mid-draw. In that case, the SkGpuDevice's GrOpLists won't be closed
        // but need to be flushed anyway. Closing such GrOpLists here will mean new
        // GrOpLists will be created to replace them if the SkGpuDevice(s) write to them again.
        fOpLists[i]->makeClosed(*fContext->caps());
    }

#ifdef SK_DEBUG
    // This block checks for any unnecessary splits in the opLists. If two sequential opLists
    // share the same backing GrSurfaceProxy it means the opList was artificially split.
    if (fOpLists.count()) {
        GrRenderTargetOpList* prevOpList = fOpLists[0]->asRenderTargetOpList();
        for (int i = 1; i < fOpLists.count(); ++i) {
            GrRenderTargetOpList* curOpList = fOpLists[i]->asRenderTargetOpList();

            if (prevOpList && curOpList) {
                SkASSERT(prevOpList->fTarget.get() != curOpList->fTarget.get());
            }

            prevOpList = curOpList;
        }
    }
```

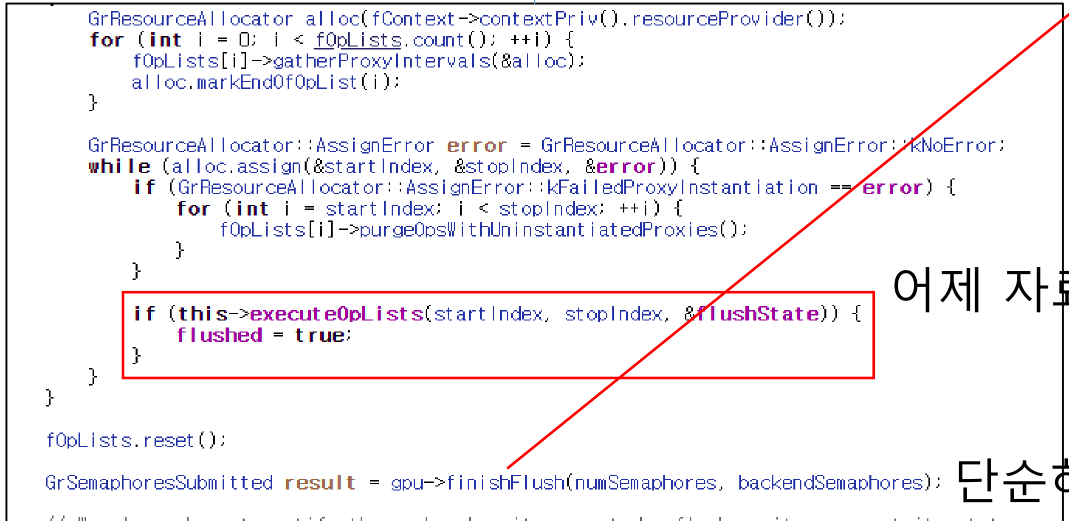
수
정



xref: /external/skia/src/gpu/GrGpu.cpp

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

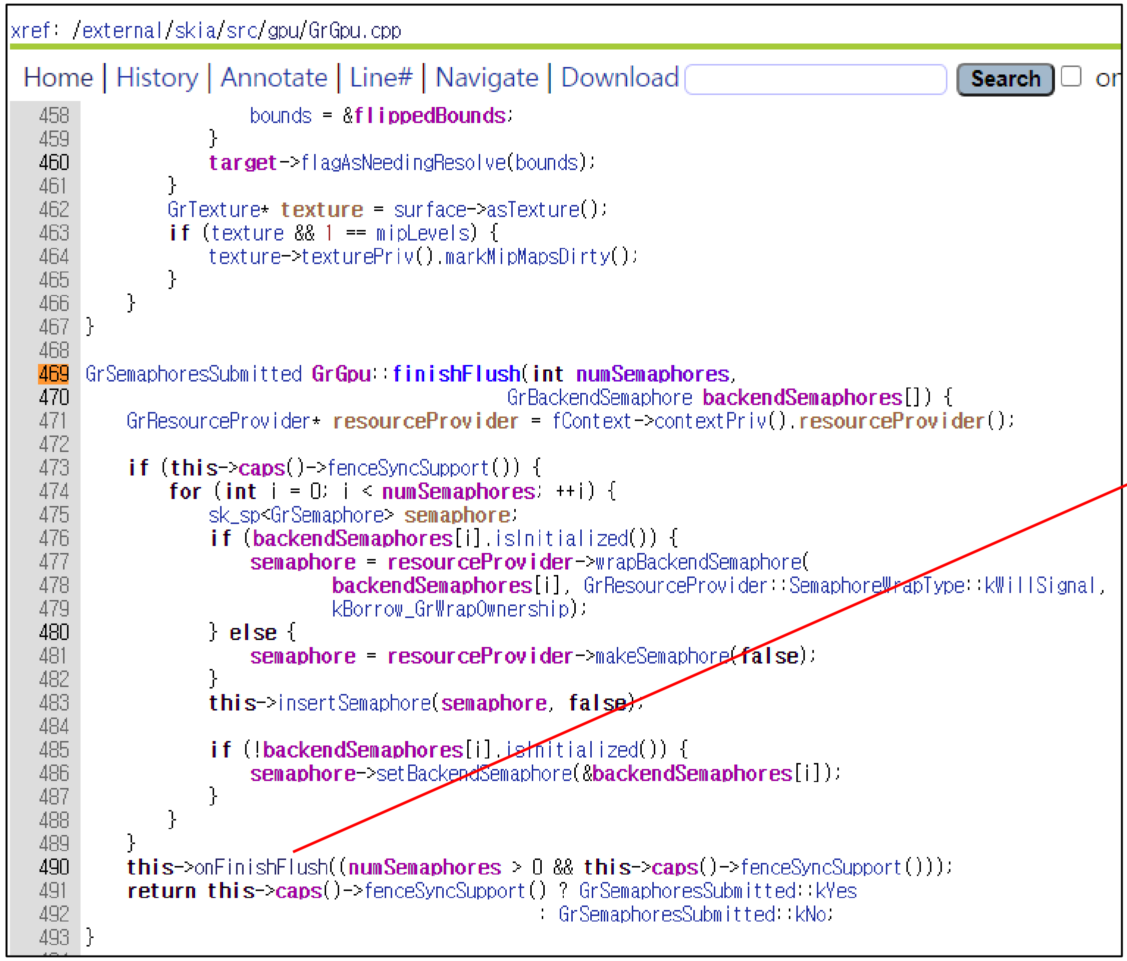
```
                bounds = &flippedBounds;
            }
            target->flagAsNeedingResolve(bounds);
        }
        GrTexture* texture = surface->asTexture();
        if (texture && 1 == mipLevels) {
            texture->texturePriv().markMipMapsDirty();
        }
    }
}

GrSemaphoresSubmitted GrGpu::finishFlush(int numSemaphores,
                                         GrBackendSemaphore backendSemaphores[]) {
    GrResourceProvider* resourceProvider = fContext->contextPriv().resourceProvider();

    if (this->caps()->fenceSyncSupport()) {
        for (int i = 0; i < numSemaphores; ++i) {
            sk_sp<GrSemaphore> semaphore;
            if (backendSemaphores[i].isInitialized()) {
                semaphore = resourceProvider->wrapBackendSemaphore(
                        backendSemaphores[i], GrResourceProvider::SemaphoreWrapType::kWillSignal,
                        kBorrow_GrWrapOwnership);
            } else {
                semaphore = resourceProvider->makeSemaphore(false);
            }
            this->insertSemaphore(semaphore, false);

            if (!backendSemaphores[i].isInitialized()) {
                semaphore->setBackendSemaphore(&backendSemaphores[i]);
            }
        }
    }
    this->onFinishFlush((numSemaphores > 0 && this->caps()->fenceSyncSupport()));
    return this->caps()->fenceSyncSupport() ? GrSemaphoresSubmitted::kYes
                                            : GrSemaphoresSubmitted::kNo;
}
```



```
        GrResourceAllocator alloc(fContext->contextPriv().resourceProvider());
        for (int i = 0; i < fOpLists.count(); ++i) {
            fOpLists[i]->gatherProxyIntervals(&alloc);
            alloc.markEndOfOpList(i);
        }

        GrResourceAllocator::AssignError error = GrResourceAllocator::AssignError::kNoError;
        while (alloc.assign(&startIndex, &stopIndex, &error)) {
            if (GrResourceAllocator::AssignError::kFailedProxyInstantiation == error) {
                for (int i = startIndex; i < stopIndex; ++i) {
                    fOpLists[i]->purgeOpsWithUninstantiatedProxies();
                }
            }

            if (this->executeOpLists(startIndex, stopIndex, &flushState)) {
                flushed = true;
            }
        }
    }

    fOpLists.reset();

    GrSemaphoresSubmitted result = gpu->finishFlush(numSemaphores, backendSemaphores);
```

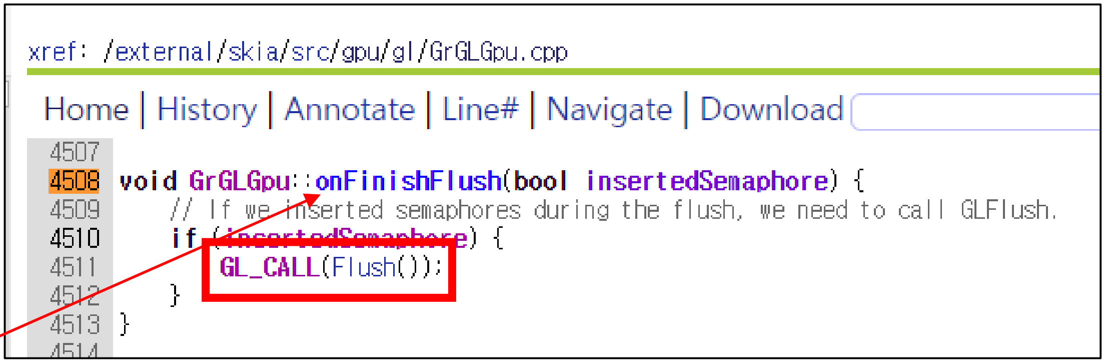
어제 자료는 executeOpLists함수에 관한 자료입니다.

단순히 flush는 이 함수를 따라 갑니다.



xref: /external/skia/src/gpu/GrGpu.cpp

```
                bounds = &flippedBounds;
            }
            target->flagAsNeedingResolve(bounds);
        }
        GrTexture* texture = surface->asTexture();
        if (texture && 1 == mipLevels) {
            texture->texturePriv().markMipMapsDirty();
        }
    }
}

GrSemaphoresSubmitted GrGpu::finishFlush(int numSemaphores,
                                         GrBackendSemaphore backendSemaphores[]) {
    GrResourceProvider* resourceProvider = fContext->contextPriv().resourceProvider();

    if (this->caps()->fenceSyncSupport()) {
        for (int i = 0; i < numSemaphores; ++i) {
            sk_sp<GrSemaphore> semaphore;
            if (backendSemaphores[i].isInitialized()) {
                semaphore = resourceProvider->wrapBackendSemaphore(
                        backendSemaphores[i], GrResourceProvider::SemaphoreWrapType::kWillSignal,
                        kBorrow_GrWrapOwnership);
            } else {
                semaphore = resourceProvider->makeSemaphore(false);
            }
            this->insertSemaphore(semaphore, false);

            if (!backendSemaphores[i].isInitialized()) {
                semaphore->setBackendSemaphore(&backendSemaphores[i]);
            }
        }
    }
    this->onFinishFlush((numSemaphores > 0 && this->caps()->fenceSyncSupport()));
    return this->caps()->fenceSyncSupport() ? GrSemaphoresSubmitted::kYes
                                            : GrSemaphoresSubmitted::kNo;
}
```



xref: /external/skia/src/gpu/gl/GrGLGpu.cpp

```
void GrGLGpu::onFinishFlush(bool insertedSemaphore) {
    // If we inserted semaphores during the flush, we need to call GLFlush.
    if (insertedSemaphore) {
        GL_CALL(Flush());
    }
}
```