Drawing Path

Pie(9.0)

recordOp->openGL

자세한 코드

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

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

```
322
323 void SkiaPipeline::renderFrame(const LayerUpdateQueue& layers, const SkRect& clip,
324                                 const std::vector<sp<RenderNode>>& nodes, bool opaque,
325                                 bool wideColorGamut, const Rect& contentDrawBounds,
326                                 sk_sp<SkSurface> surface) {
327     renderVectorDrawableCache();
328
329     // draw all layers up front
330     renderLayersImpl(layers, opaque, wideColorGamut);
331
332     // initialize the canvas for the current frame, that might be a recording canvas if SKP
333     // capture is enabled.
334     std::unique_ptr<SkPictureRecorder> recorder;
335     SkCanvas* canvas = tryCapture(surface.get());
336
337     renderFrameImpl(layers, clip, nodes, opaque, wideColorGamut, contentDrawBounds, canvas);
338
339     endCapture(surface.get());
340
341     if (CC_UNLIKELY(Properties::debugOverdraw)) {
342         renderOverdraw(layers, clip, nodes, contentDrawBounds, surface);
343     }
344
345     ATRACE_NAME("flush commands");
346     surface->getCanvas()->flush();
347 }
```

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

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

```
783
784 void SkCanvas::flush() {
785     this->onFlush();
786 }
787
788 void SkCanvas::onFlush() {
789     SkBaseDevice* device = this->getDevice();
790     if (device) {
791         device->flush();
792     }
793 }
794
```

자세한 코드

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

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

```
783
784 void SkCanvas::flush() {
785     this->onFlush();
786 }
787
788 void SkCanvas::onFlush() {
789     SkBaseDevice* device = this->getDevice();
790     if (device) {
791         device->flush();
792     }
793 }
794
```

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

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

```
1662 ////////////////////////////////////////////////////////////////////////////
1663
1664 void SkGpuDevice::flush() {
1665     this->flushAndSignalSemaphores(0, nullptr);
1666 }
1667
1668 GrSemaphoresSubmitted SkGpuDevice::flushAndSignalSemaphores(int numSemaphores,
1669                                                             GrBackendSemaphore signalSemaphores[]) {
1670     ASSERT_SINGLE_OWNER
1671
1672     return fRenderTargetContext->prepareForExternalIO(numSemaphores, signalSemaphores);
1673 }
```

자세한 코드

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

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

```
1382 GrSemaphoresSubmitted GrRenderTargetContext::prepareForExternalIO(
1383         int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
1384     ASSERT_SINGLE_OWNER
1385     if (this->drawingManager()->wasAbandoned()) { return GrSemaphoresSubmitted::kNo; }
1386     SkDEBUGCODE(this->validate();)
1387     GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "prepareForExternalIO", fContext);
1388
1389     return this->drawingManager()->prepareSurfaceForExternalIO(fRenderTargetProxy.get(),
1390                                                                numSemaphores,
1391                                                                backendSemaphores);
1392 }
```

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

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

```
1662 ////////////////////////////////////////////////////////////////////////////
1663
1664 void SkGpuDevice::flush() {
1665     this->flushAndSignalSemaphores(0, nullptr);
1666 }
1667
1668 GrSemaphoresSubmitted SkGpuDevice::flushAndSignalSemaphores(int numSemaphores,
1669                                                             GrBackendSemaphore signalSemaphores[]) {
1670     ASSERT_SINGLE_OWNER
1671
1672     return fRenderTargetContext->prepareForExternalIO(numSemaphores, signalSemaphores);
1673 }
```

자세한 코드

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

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

```
326 GrSemaphoresSubmitted GrDrawingManager::prepareSurfaceForExternalIO(
327         GrSurfaceProxy* proxy, int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
328     if (this->wasAbandoned()) {
329         return GrSemaphoresSubmitted::kNo;
330     }
331     SkASSERT(proxy);
332
333     GrSemaphoresSubmitted result = GrSemaphoresSubmitted::kNo;
334     if (proxy->priv().hasPendingIO() || numSemaphores) {
335         result = this->flush(proxy, numSemaphores, backendSemaphores);
336     }
337
338     if (!proxy->instantiate(fContext->contextPriv().resourceProvider())) {
339         return result;
340     }
341
342     GrGpu* gpu = fContext->contextPriv().getGpu();
343     GrSurface* surface = proxy->priv().peekSurface();
344
345     if (gpu && surface->asRenderTarget()) {
        gpu->resolveRenderTarget(surface->asRenderTarget());

    urn result;
```

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

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

```
1382 GrSemaphoresSubmitted GrRenderTargetContext::prepareForExternalIO(
1383         int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
1384     ASSERT_SINGLE_OWNER
1385     if (this->drawingManager()->wasAbandoned()) { return GrSemaphoresSubmitted::kNo; }
1386     SkDEBUGCODE(this->validate();)
1387     GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "prepareForExternalIO", fContext);
1388
1389     return this->drawingManager()->prepareSurfaceForExternalIO(fRenderTargetProxy.get(),
1390                                                                numSemaphores,
1391                                                                backendSemaphores);
1392 }
```

자세한 코드

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

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

```
326 GrSemaphoresSubmitted GrDrawingManager::prepareSurfaceForExternalIO(
327         GrSurfaceProxy* proxy, int numSemaphores, GrBackendSemaphore backendSemaphores[]) {
328     if (this->wasAbandoned()) {
329         return GrSemaphoresSubmitted::kNo;
330     }
331     SkASSERT(proxy);
332
333     GrSemaphoresSubmitted result = GrSemaphoresSubmitted::kNo;
334     if (proxy->priv().hasPendingIO() || numSemaphores) {
335         result = this->flush(proxy, numSemaphores, backendSemaphores);
336     }
337
338     if (!proxy->instantiate(fContext->contextPriv().resourceProvider())) {
339         return result;
340     }
341
342     GrGpu* gpu = fContext->contextPriv().getGpu();
343     GrSurface* surface = proxy->priv().peekSurface();
344
345     if (gpu && surface->asRenderTarget()) {
346         gpu->resolveRenderTarget(surface->asRenderTarget());
347     }
348     return result;
349 }
```

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

ome | History | Annotate | Line# | Navigate | Download

```
96     GrSemaphoresSubmitted flush(GrSurfaceProxy* proxy,
97                                 int numSemaphores = 0,
98                                 GrBackendSemaphore backendSemaphores[] = nullptr) {
99         return this->internalFlush(proxy, GrResourceCache::FlushType::kExternal,
00                                    numSemaphores, backendSemaphores);
01     }
```

자세한 코드

*이 다음부터 로그로 확인 못해봤습니다.

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

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

```
96      GrSemaphoresSubmitted flush(GrSurfaceProxy* proxy,
97                                  int numSemaphores = 0,
98                                  GrBackendSemaphore backendSemaphores[] = nullptr) {
99          return this->internalFlush(proxy, GrResourceCache::FlushType::kExternal,
100                                     numSemaphores, backendSemaphores);
101     }
```

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

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

```
115
116 // MDB TODO: make use of the 'proxy' parameter.
117 GrSemaphoresSubmitted GrDrawingManager::internalFlush(GrSurfaceProxy*,
118                                                       GrResourceCache::FlushType type,
119                                                       int numSemaphores,
120                                                       GrBackendSemaphore backendSemaphores[]) {
121     GR_CREATE_TRACE_MARKER_CONTEXT("GrDrawingManager", "internalFlush", fContext);
122
123     if (fFlushing || this->wasAbandoned()) {
124         return GrSemaphoresSubmitted::kNo;
125     }
126     fFlushing = true;
127
128     for (int i = 0; i < fOpLists.count(); ++i) {
129         // Semi-usually the GrOpLists are already closed at this point, but sometimes Ganesh
130         // needs to flush mid-draw. In that case, the SkGpuDevice's GrOpLists won't be closed
131         // but need to be flushed anyway. Closing such GrOpLists here will mean new
132         // GrOpLists will be created to replace them if the SkGpuDevice(s) write to them again.
133         fOpLists[i]->makeClosed(*fContext->caps());
134     }
135
136 #ifdef SK_DEBUG
137     // This block checks for any unnecessary splits in the opLists. If two sequential opLists
138     // share the same backing GrSurfaceProxy it means the opList was artificially split.
139     if (fOpLists.count()) {
140         GrRenderTargetOpList* prevOpList = fOpLists[0]->asRenderTargetOpList();
141         for (int i = 1; i < fOpLists.count(); ++i) {
142             GrRenderTargetOpList* curOpList = fOpLists[i]->asRenderTargetOpList();
143
144             if (prevOpList && curOpList) {
145                 SkASSERT(prevOpList->fTarget.get() != curOpList->fTarget.get());
146             }
147
148             prevOpList = curOpList;
149         }
150     }
151 #endif
152
153     if (fSortRenderTargets) {
154         SkDEBUGCODE(bool result =) SkTTopoSort<GrOpList, GrOpList::TopoSortTraits>(&fOpLists);
155         SkASSERT(result);
156     }
157
158     GrGpu* gpu = fContext->contextPriv().getGpu();
159
160     GrOpFlushState flushState(gpu, fContext->contextPriv().resourceProvider(),
161                               &fTokenTracker);
```

```
117 GrSemaphoresSubmitted GrDrawingManager::internalFlush(GrSurfaceProxy*,
118                                                       GrResourceCache::FlushType type,
119                                                       int numSemaphores,
120                                                       GrBackendSemaphore backendSemaphores[]) {
121     GR_CREATE_TRACE_MARKER_CONTEXT("GrDrawingManager", "internalFlush", fContext);
122
123     if (fFlushing || this->wasAbandoned()) {
124         return GrSemaphoresSubmitted::kNo;
125     }
126     fFlushing = true;
127
128     for (int i = 0; i < fOpLists.count(); ++i) {
129         // Semi-usually the GrOpLists are already closed at this point, but sometimes Ganesh
130         // needs to flush mid-draw. In that case, the SkGpuDevice's GrOpLists won't be closed
131         // but need to be flushed anyway. Closing such GrOpLists here will mean new
132         // GrOpLists will be created to replace them if the SkGpuDevice(s) write to them again.
133         fOpLists[i]->makeClosed(*fContext->caps());
134     }
135
136 #ifdef SK_DEBUG
137     // This block checks for any unnecessary splits in the opLists. If two sequential opLists
138     // share the same backing GrSurfaceProxy it means the opList was artificially split.
139     if (fOpLists.count()) {
140         GrRenderTargetOpList* prevOpList = fOpLists[0]->asRenderTargetOpList();
141         for (int i = 1; i < fOpLists.count(); ++i) {
142             GrRenderTargetOpList* curOpList = fOpLists[i]->asRenderTargetOpList();
143
144             if (prevOpList && curOpList) {
145                 SkASSERT(prevOpList->fTarget.get() != curOpList->fTarget.get());
146             }
147
148             prevOpList = curOpList;
149         }
150     }
```

```
        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);
```



```
245     return result;
246 }
247
248 bool GrDrawingManager::executeOpLists(int startIndex, int stopIndex, GrOpFlushState* flushState) {
249     SkASSERT(startIndex <= stopIndex && stopIndex <= fOpLists.count());
250
251     GrResourceProvider* resourceProvider = fContext->contextPriv().resourceProvider();
252     bool anyOpListsExecuted = false;
253
254     for (int i = startIndex; i < stopIndex; ++i) {
255         if (!fOpLists[i]) {
256             continue;
257         }
258
259         if (resourceProvider->explicitlyAllocateGPUResources()) {
260             if (!fOpLists[i]->isInstantiated()) {
261                 // If the backing surface wasn't allocated drop the draw of the entire opList.
262                 fOpLists[i] = nullptr;
263                 continue;
264             }
265         } else {
266             if (!fOpLists[i]->instantiate(resourceProvider)) {
267                 SkDebugf("OpList failed to instantiate.\n");
268                 fOpLists[i] = nullptr;
269                 continue;
270             }
271         }
272
273         // TODO: handle this instantiation via lazy surface proxies?
274         // Instantiate all deferred proxies (being built on worker threads) so we can upload them
275         fOpLists[i]->instantiateDeferredProxies(fContext->contextPriv().resourceProvider());
276         fOpLists[i]->prepare(flushState);
277     }
278
279     // Upload all data to the GPU
280     flushState->preExecuteDraws();
281
282     // Execute the onFlush op lists first, if any.
283     for (sk_sp<GrOpList>& onFlushOpList : fOnFlushCBOpLists) {
284         if (!onFlushOpList->execute(flushState)) {
285             SkDebugf("WARNING: onFlushOpList failed to execute.\n");
286         }
287         SkASSERT(onFlushOpList->unique());
288         onFlushOpList = nullptr;
```

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

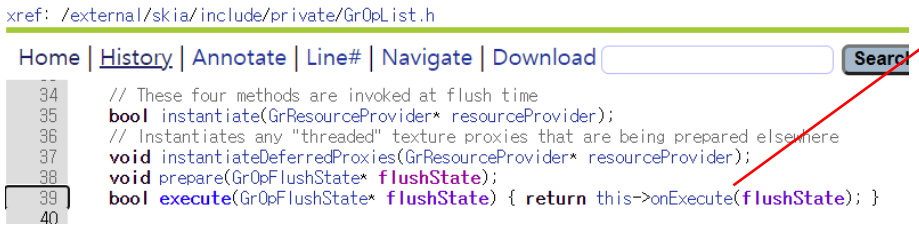
Home | History | Annotate | Line# | Navigate | Download Search only in GrDra

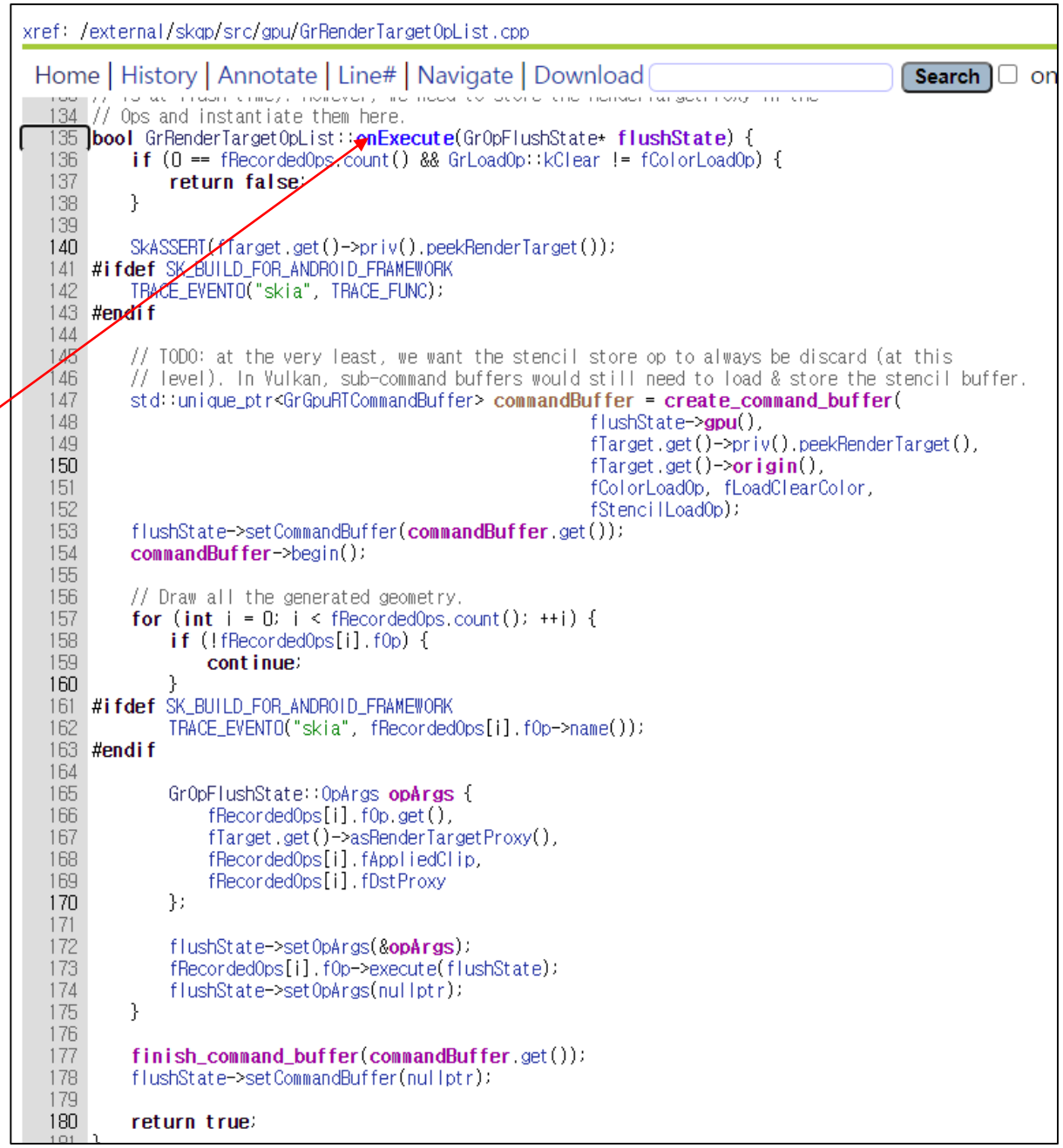
```
244
245     return result;
246 }
247
248 bool GrDrawingManager::executeOpLists(int startIndex, int stopIndex, GrOpFlushState* flushState) {
249     SkASSERT(startIndex <= stopIndex && stopIndex <= fOpLists.count());
250
251     GrResourceProvider* resourceProvider = fContext->contextPriv().resourceProvider();
252     bool anyOpListsExecuted = false;
253
254     for (int i = startIndex; i < stopIndex; ++i) {
255         if (!fOpLists[i]) {
256             continue;
257         }
258
259         if (resourceProvider->explicitlyAllocateGPUResources()) {
260             if (!fOpLists[i]->isInstantiated()) {
261                 // If the backing surface wasn't allocated drop the draw of the entire opList.
262                 fOpLists[i] = nullptr;
263                 continue;
264             }
265         } else {
266             if (!fOpLists[i]->instantiate(resourceProvider)) {
267                 SkDebugf("OpList failed to instantiate.\n");
268                 fOpLists[i] = nullptr;
269                 continue;
270             }
271         }
272
273         // TODO: handle this instantiation via lazy surface proxies?
274         // Instantiate all deferred proxies (being built on worker threads) so we can upload them
275         fOpLists[i]->instantiateDeferredProxies(fContext->contextPriv().resourceProvider());
276         fOpLists[i]->prepare(flushState);
277     }
278
279     // Upload all data to the GPU
280     flushState->preExecuteDraws();
281
282     // Execute the onFlush op lists first, if any.
283     for (sk_sp<GrOpList>& onFlushOpList : fOnFlushCBOpLists) {
284         if (!onFlushOpList->execute(flushState)) {
285             SkDebugf("WARNING: onFlushOpList failed to execute.\n");
286         }
287         SkASSERT(onFlushOpList->unique());
288         onFlushOpList = nullptr;
```

xref: /external/skia/include/private/GrOpList.h

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

```
34     // These four methods are invoked at flush time
35     bool instantiate(GrResourceProvider* resourceProvider);
36     // Instantiates any "threaded" texture proxies that are being prepared elsewhere
37     void instantiateDeferredProxies(GrResourceProvider* resourceProvider);
38     void prepare(GrOpFlushState* flushState);
39     bool execute(GrOpFlushState* flushState) { return this->onExecute(flushState); }
40
```


xref: /external/skia/include/private/GrOpList.h
Home | History | Annotate | Line# | Navigate | Download
Search
34 // These four methods are invoked at flush time
35 bool instantiate(GrResourceProvider* resourceProvider);
36 // Instantiates any "threaded" texture proxies that are being prepared elsewhere
37 void instantiateDeferredProxies(GrResourceProvider* resourceProvider);
38 void prepare(GrOpFlushState* flushState);
39 bool execute(GrOpFlushState* flushState) { return this->onExecute(flushState); }
40


xref: /external/skqp/src/gpu/GrRenderTargetOpList.cpp
Home | History | Annotate | Line# | Navigate | Download
Search
134 // Ops and instantiate them here.
135 bool GrRenderTargetOpList::onExecute(GrOpFlushState* flushState) {
136 if (0 == fRecordedOps.count() && GrLoadOp::kClear != fColorLoadOp) {
137 return false;
138 }
139
140 SkASSERT(fTarget.get()->priv().peekRenderTarget());
141 #ifdef SK_BUILD_FOR_ANDROID_FRAMEWORK
142 TRACE_EVENT0("skia", TRACE_FUNC);
143 #endif
144
145 // TODO: at the very least, we want the stencil store op to always be discard (at this
146 // level). In Vulkan, sub-command buffers would still need to load & store the stencil buffer.
147 std::unique_ptr<GrGpuRTCommandBuffer> commandBuffer = create_command_buffer(
148 flushState->gpu(),
149 fTarget.get()->priv().peekRenderTarget(),
150 fTarget.get()->origin(),
151 fColorLoadOp, fLoadClearColor,
152 fStencilLoadOp);
153 flushState->setCommandBuffer(commandBuffer.get());
154 commandBuffer->begin();
155
156 // Draw all the generated geometry.
157 for (int i = 0; i < fRecordedOps.count(); ++i) {
158 if (!fRecordedOps[i].fOp) {
159 continue;
160 }
161 #ifdef SK_BUILD_FOR_ANDROID_FRAMEWORK
162 TRACE_EVENT0("skia", fRecordedOps[i].fOp->name());
163 #endif
164
165 GrOpFlushState::OpArgs opArgs {
166 fRecordedOps[i].fOp.get(),
167 fTarget.get()->asRenderTargetProxy(),
168 fRecordedOps[i].fAppliedClip,
169 fRecordedOps[i].fDstProxy
170 };
171
172 flushState->setOpArgs(&opArgs);
173 fRecordedOps[i].fOp->execute(flushState);
174 flushState->setOpArgs(nullptr);
175 }
176
177 finish_command_buffer(commandBuffer.get());
178 flushState->setCommandBuffer(nullptr);
179
180 return true;

xref: /external/skqp/src/gpu/GrRenderTargetOpList.cpp

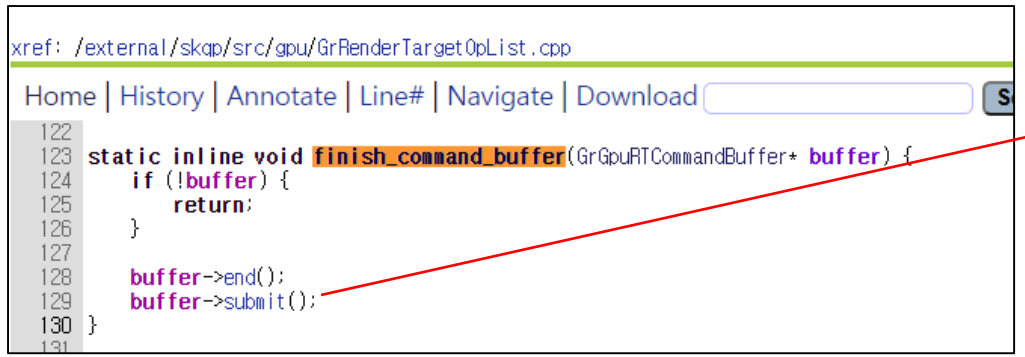
Home | History | Annotate | Line# | Navigate | Download Search

```
134 // Ops and instantiate them here.
135 bool GrRenderTargetOpList::onExecute(GrOpFlushState* flushState) {
136     if (0 == fRecordedOps.count() && GrLoadOp::kClear != fColorLoadOp) {
137         return false;
138     }
139
140     SkASSERT(fTarget.get()->priv().peekRenderTarget());
141 #ifdef SK_BUILD_FOR_ANDROID_FRAMEWORK
142     TRACE_EVENT0("skia", TRACE_FUNC);
143 #endif
144
145     // TODO: at the very least, we want the stencil store op to always be discard (at this
146     // level). In Vulkan, sub-command buffers would still need to load & store the stencil buffer.
147     std::unique_ptr<GrGpuRTCommandBuffer> commandBuffer = create_command_buffer(
148                                                 flushState->gpu(),
149                                                 fTarget.get()->priv().peekRenderTarget(),
150                                                 fTarget.get()->origin(),
151                                                 fColorLoadOp, fLoadClearColor,
152                                                 fStencilLoadOp);
153     flushState->setCommandBuffer(commandBuffer.get());
154     commandBuffer->begin();
155
156     // Draw all the generated geometry.
157     for (int i = 0; i < fRecordedOps.count(); ++i) {
158         if (!fRecordedOps[i].fOp) {
159             continue;
160         }
161 #ifdef SK_BUILD_FOR_ANDROID_FRAMEWORK
162         TRACE_EVENT0("skia", fRecordedOps[i].fOp->name());
163 #endif
164
165         GrOpFlushState::OpArgs opArgs {
166             fRecordedOps[i].fOp.get(),
167             fTarget.get()->asRenderTargetProxy(),
168             fRecordedOps[i].fAppliedClip,
169             fRecordedOps[i].fDstProxy
170         };
171
172         flushState->setOpArgs(&opArgs);
173         fRecordedOps[i].fOp->execute(flushState);
174         flushState->setOpArgs(nullptr);
175     }
176
177     finish_command_buffer(commandBuffer.get());
178     flushState->setCommandBuffer(nullptr);
179
180     return true;
```

xref: /external/skqp/src/gpu/GrRenderTargetOpList.cpp

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

```
122
123 static inline void finish_command_buffer(GrGpuRTCommandBuffer* buffer) {
124     if (!buffer) {
125         return;
126     }
127
128     buffer->end();
129     buffer->submit();
130 }
```


xref: /external/skqp/src/gpu/GrRenderTargetOpList.cpp
Home | History | Annotate | Line# | Navigate | Download
122
123 static inline void finish_command_buffer(GrGpuRTCommandBuffer* buffer) {
124 if (!buffer) {
125 return;
126 }
127
128 buffer->end();
129 buffer->submit();
130 }