DrawPath

SkLiteDL->record op

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

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

```
527 template <typename Fn, typename... Args>
528 inline void SkLiteDL::map(const Fn fns[], Args... args) const {
529     auto end = fBytes.get() + fUsed;
530     for (const uint8_t* ptr = fBytes.get(); ptr < end; ) {
531         auto op = (const Op*)ptr;
532         auto type = op->type;
533         auto skip = op->skip;
534         if (auto fn = fns[type]) {  // We replace no-op functions with nullptrs
535             fn(op, args...);        // to avoid the overhead of a pointless call.
536         }
537         ptr += skip;
538     }
539 }
540
```

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

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

```
181     };
182     struct DrawPath final : Op {
183         static const auto kType = Type::DrawPath;
184         DrawPath(const SkPath& path, const SkPaint& paint) : path(path), paint(paint) {}
185         SkPath  path;
186         SkPaint paint;
187         void draw(SkCanvas* c, const SkMatrix&) const { c->drawPath(path, paint); }
188     };
```

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

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

```
181     };
182     struct DrawPath final : Op {
183         static const auto kType = Type::DrawPath;
184         DrawPath(const SkPath& path, const SkPaint& paint) : path(path), paint(paint) {}
185         SkPath  path;
186         SkPaint paint;
187         void draw(SkCanvas* c, const SkMatrix&) const { c->drawPath(path, paint); }
188     };
```

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

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

```
1755
1756 void SkCanvas::drawPath(const SkPath& path, const SkPaint& paint) {
1757     TRACE_EVENT0("skia", TRACE_FUNC);
1758     this->onDrawPath(path, paint);
1759 }
1760
```

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

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

```
1755
1756 void SkCanvas::drawPath(const SkPath& path, const SkPaint& paint) {
1757     TRACE_EVENT0("skia", TRACE_FUNC);
1758     this->onDrawPath(path, paint);
1759 }
1760
```

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

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

```
2171
2172 void SkCanvas::onDrawPath(const SkPath& path, const SkPaint& paint) {
2173     if (!path.isFinite()) {
2174         return;
2175     }
2176
2177     const SkRect& pathBounds = path.getBounds();
2178     if (!path.isInverseFillType() && paint.canComputeFastBounds()) {
2179         SkRect storage;
2180         if (this->quickReject(paint.computeFastBounds(pathBounds, &storage))) {
2181             return;
2182         }
2183     }
2184
2185     if (pathBounds.width() <= 0 && pathBounds.height() <= 0) {
2186         if (path.isInverseFillType()) {
2187             this->internalDrawPaint(paint);
2188             return;
2189         }
2190     }
2191
2192     LOOPER_BEGIN(paint, SkDrawFilter::kPath_Type, &pathBounds)
2193
2194     while (iter.next()) {
2195         iter.fDevice->drawPath(path, looper.paint());
2196     }
2197
2198     LOOPER_END
2199 }
```

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

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

```
2171
2172 void SkCanvas::onDrawPath(const SkPath& path, const SkPaint& paint) {
2173     if (!path.isFinite()) {
2174         return;
2175     }
2176
2177     const SkRect& pathBounds = path.getBounds();
2178     if (!path.isInverseFillType() && paint.canComputeFastBounds()) {
2179         SkRect storage;
2180         if (this->quickReject(paint.computeFastBounds(pathBounds, &storage))) {
2181             return;
2182         }
2183     }
2184
2185     if (pathBounds.width() <= 0 && pathBounds.height() <= 0) {
2186         if (path.isInverseFillType()) {
2187             this->internalDrawPaint(paint);
2188             return;
2189         }
2190     }
2191
2192     LOOPER_BEGIN(paint, SkDrawFilter::kPath_Type, &pathBounds)
2193
2194     while (iter.next()) {
2195         iter.fDevice->drawPath(path, looper.paint());
2196     }
2197
2198     LOOPER_END
2199 }
```

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

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

```
605
606 void SkGpuDevice::drawPath(const SkPath& origSrcPath,
607                            const SkPaint& paint, const SkMatrix* prePathMatrix,
608                            bool pathIsMutable) {
609     ASSERT_SINGLE_OWNER
610     if (!origSrcPath.isInverseFillType() && !paint.getPathEffect() && !prePathMatrix) {
611         SkPoint points[2];
612         if (SkPaint::kStroke_Style == paint.getStyle() && paint.getStrokeWidth() > 0 &&
613             !paint.getMaskFilter() && SkPaint::kRound_Cap != paint.getStrokeCap() &&
614             this->ctm().preservesRightAngles() && origSrcPath.isLine(points)) {
615             // Path-based stroking looks better for thin rects
616             SkScalar strokeWidth = this->ctm().getMaxScale() * paint.getStrokeWidth();
617             if (strokeWidth >= 1.0f) {
618                 // Round capping support is currently disabled b.c. it would require a RRect
619                 // GrDrawOp that takes a localMatrix.
620                 this->drawStrokedLine(points, paint);
621                 return;
622             }
623         }
624     }
625
626     GR_CREATE_TRACE_MARKER_CONTEXT("SkGpuDevice", "drawPath", fContext.get());
627     if (!prePathMatrix && !paint.getMaskFilter()) {
628         GrPaint grPaint;
629         if (!SkPaintToGrPaint(this->context(), fRenderTargetContext->colorSpaceInfo(), paint,
630                               this->ctm(), &grPaint)) {
631             return;
632         }
633         fRenderTargetContext->drawPath(this->clip(), std::move(grPaint), GrAA(paint.isAntiAlias()),
634                                        this->ctm(), origSrcPath, GrStyle(paint));
635         return;
636     }
637     GrBlurUtils::drawPathWithMaskFilter(fContext.get(), fRenderTargetContext.get(), this->clip(),
638                                         origSrcPath, paint, this->ctm(), prePathMatrix,
639                                         this->devClipBounds(), pathIsMutable);
640 }
641
```

Antialias=AA

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

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

```
605
606 void SkGpuDevice::drawPath(const SkPath& origSrcPath,
607                             const SkPaint& paint, const SkMatrix* prePathMatrix,
608                             bool pathIsMutable) {
609     ASSERT_SINGLE_OWNER
610     if (!origSrcPath.isInverseFillType() && !paint.getPathEffect() && !prePathMatrix) {
611         SkPoint points[2];
612         if (SkPaint::kStroke_Style == paint.getStyle() && paint.getStrokeWidth() > 0 &&
613             !paint.getMaskFilter() && SkPaint::kRound_Cap != paint.getStrokeCap() &&
614             this->ctm().preservesRightAngles() && origSrcPath.isLine(points)) {
615             // Path-based stroking looks better for thin rects
616             SkScalar strokeWidth = this->ctm().getMaxScale() * paint.getStrokeWidth();
617             if (strokeWidth >= 1.0f) {
618                 // Round capping support is currently disabled b.c. it would require a RRect
619                 // GrDrawOp that takes a localMatrix.
620                 this->drawStrokedLine(points, paint);
621                 return;
622             }
623         }
624     }
625
626     GR_CREATE_TRACE_MARKER_CONTEXT("SkGpuDevice", "drawPath", fContext.get());
627     if (!prePathMatrix && !paint.getMaskFilter()) {
628         GrPaint grPaint;
629         if (!SkPaintToGrPaint(this->context(), fRenderTargetContext->colorSpaceInfo(), paint,
630                               this->ctm(), &grPaint)) {
631             return;
632         }
633         fRenderTargetContext->drawPath(this->clip(), std::move(grPaint), GrAA(paint.isAntiAlias()),
634                                        this->ctm(), origSrcPath, GrStyle(paint));
635         return;
636     }
637     GrBlurUtils::drawPathWithMaskFilter(fContext.get(), fRenderTargetContext.get(), this->clip(),
638                                         origSrcPath, paint, this->ctm(), prePathMatrix,
639                                         this->devClipBounds(), pathIsMutable);
640 }
641
```

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

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

```
1471
1472 void GrRenderTargetContext::drawPath(const GrClip& clip,
1473                                      GrPaint&& paint,
1474                                      GrAA aa,
1475                                      const SkMatrix& viewMatrix,
1476                                      const SkPath& path,
1477                                      const GrStyle& style) {
1478     ASSERT_SINGLE_OWNER
1479     RETURN_IF_ABANDONED
1480     SkDEBUGCODE(this->validate();)
1481             GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContextPriv", "drawPath", fContext);
1482
1483     GrShape shape(path, style);
1484     if (shape.isEmpty()) {
1485         if (shape.inverseFilled()) {
1486             this->drawPaint(clip, std::move(paint), viewMatrix);
1487         }
1488         return;
1489     }
1490
1491     AutoCheckFlush acf(this->drawingManager());
1492
```

●
●
●

```
1511     GrAAType aaType = this->chooseAAType(aa, GrAllowMixedSamples::kNo);
1512     if (GrAAType::kCoverage == aaType) {
1513         // TODO: Make GrShape check for nested rects.
1514         SkRect rects[2];
1515         if (shape.style().isSimpleFill() && fills_as_nested_rects(viewMatrix, path, rects)) {
1516             // Concave AA paths are expensive - try to avoid them for special cases
1517             SkRect rects[2];
1518
1519             if (fills_as_nested_rects(viewMatrix, path, rects)) {
1520                 std::unique_ptr<GrDrawOp> op =
1521                         GrRectOpFactory::MakeAAFillNestedRects(std::move(paint), viewMatrix, rects);
1522                 if (op) {
1523                     this->addDrawOp(clip, std::move(op));
1524                 }
1525                 // A null return indicates that there is nothing to draw in this case.
1526                 return;
1527             }
1528         }
1529     }
1530     this->drawShapeUsingPathRenderer(clip, std::move(paint), aa, viewMatrix, shape);
```

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

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

```
1471
1472 void GrRenderTargetContext::drawPath(const GrClip& clip,
1473                                       GrPaint&& paint,
1474                                       GrAA aa,
1475                                       const SkMatrix& viewMatrix,
1476                                       const SkPath& path,
1477                                       const GrStyle& style) {
1478     ASSERT_SINGLE_OWNER
1479     RETURN_IF_ABANDONED
1480     SkDEBUGCODE(this->validate();)
1481             GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContextPriv", "drawPath", fContext);
1482
1483     GrShape shape(path, style);
1484     if (shape.isEmpty()) {
1485         if (shape.inverseFilled()) {
1486             this->drawPaint(clip, std::move(paint), viewMatrix);
1487         }
1488         return;
1489     }
1490
1491     AutoCheckFlush acf(this->drawingManager());
1492
```

```
1511     GrAAType aaType = this->chooseAAType(aa, GrAllowMixedSamples::kNo);
1512     if (GrAAType::kCoverage == aaType) {
1513         // TODO: Make GrShape check for nested rects.
1514         SkRect rects[2];
1515         if (shape.style().isSimpleFill() && fills_as_nested_rects(viewMatrix, path, rects)) {
1516             // Concave AA paths are expensive - try to avoid them for special cases
1517             SkRect rects[2];
1518
1519             if (fills_as_nested_rects(viewMatrix, path, rects)) {
1520                 std::unique_ptr<GrDrawOp> op =
1521                         GrRectOpFactory::MakeAAFillNestedRects(std::move(paint), viewMatrix, rects);
1522                 if (op) {
1523                     this->addDrawOp(clip, std::move(op));
1524                 }
1525                 // A null return indicates that there is nothing to draw in this case.
1526                 return;
1527             }
1528         }
1529     }
1530     this->drawShapeUsingPathRenderer(clip, std::move(paint), aa, viewMatrix, shape);
```

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

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

```
1610
1611 void GrRenderTargetContext::drawShapeUsingPathRenderer(const GrClip& clip,
1612                                                        GrPaint&& paint,
1613                                                        GrAA aa,
1614                                                        const SkMatrix& viewMatrix,
1615                                                        const GrShape& originalShape) {
1616     ASSERT_SINGLE_OWNER
1617     RETURN_IF_ABANDONED
1618     GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "internalDrawPath", fContext);
1619
1620     SkIRect clipConservativeBounds;
1621     clip.getConservativeBounds(this->width(), this->height(), &clipConservativeBounds, nullptr);
1622
1623     GrShape tempShape;
1624     // NVPR cannot handle hairlines, so this would get picked up by a different stencil and
1625     // cover path renderer (i.e. default path renderer). The hairline renderer produces much
1626     // smoother hairlines than MSAA.
1627     GrAllowMixedSamples allowMixedSamples = originalShape.style().isSimpleHairline()
```

```
}
// This time, allow SW renderer
pr = this->drawingManager()->getPathRenderer(canDrawArgs, true, kType);
```

```
1672     if (!pr) {
1673 #ifdef SK_DEBUG
1674         SkDebugf("Unable to find path renderer compatible with path.\n");
1675 #endif
1676         return;
1677     }
1678
1679     GrPathRenderer::DrawPathArgs args{this->drawingManager()->getContext(),
1680                                       std::move(paint),
1681                                       &GrUserStencilSettings::kUnused,
1682                                       this,
1683                                       &clip,
1684                                       &clipConservativeBounds,
1685                                       &viewMatrix,
1686                                       canDrawArgs.fShape,
1687                                       aaType,
1688                                       this->colorSpaceInfo().isGammaCorrect()};
1689     pr->drawPath(args);
1690 }
1691
```

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

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

```
1610
1611 void GrRenderTargetContext::drawShapeUsingPathRenderer(const GrClip& clip,
1612                                                         GrPaint&& paint,
1613                                                         GrAA aa,
1614                                                         const SkMatrix& viewMatrix,
1615                                                         const GrShape& originalShape) {
1616     ASSERT_SINGLE_OWNER
1617     RETURN_IF_ABANDONED
1618     GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "internalDrawPath", fContext);
1619
1620     SkIRect clipConservativeBounds;
1621     clip.getConservativeBounds(this->width(), this->height(), &clipConservativeBounds, nullptr);
1622
1623     GrShape tempShape;
1624     // NVPR cannot handle hairlines, so this would get picked up by a different stencil and
1625     // cover path renderer (i.e. default path renderer). The hairline renderer produces much
1626     // smoother hairlines than MSAA.
1627     GrAllowMixedSamples allowMixedSamples = originalShape.style().isSimpleHairline()
```

```
}
// This time, allow SW renderer
pr = this->drawingManager()->getPathRenderer(canDrawArgs, true, kType);
```

```
1672     if (!pr) {
1673 #ifdef SK_DEBUG
1674         SkDebugf("Unable to find path renderer compatible with path.\n");
1675 #endif
1676         return;
1677     }
1678
1679     GrPathRenderer::DrawPathArgs args{this->drawingManager()->getContext(),
1680                                       std::move(paint),
1681                                       &GrUserStencilSettings::kUnused,
1682                                       this,
1683                                       &clip,
1684                                       &clipConservativeBounds,
1685                                       &viewMatrix,
1686                                       canDrawArgs.fShape,
1687                                       aaType,
1688                                       this->colorSpaceInfo().isGammaCorrect()};
1689     pr->drawPath(args);
1690 }
1691
```

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

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

```
136     */
137     bool drawPath(const DrawPathArgs& args) {
138         SkDEBUGCODE(args.validate();)
139 #ifdef SK_DEBUG
140         CanDrawPathArgs canArgs;
141         canArgs.fCaps = args.fContext->caps();
142         canArgs.fClipConservativeBounds = args.fClipConservativeBounds;
143         canArgs.fViewMatrix = args.fViewMatrix;
144         canArgs.fShape = args.fShape;
145         canArgs.fAAType = args.fAAType;
146         canArgs.validate();
147
148         canArgs.fHasUserStencilSettings = !args.fUserStencilSettings->isUnused();
149         SkASSERT(!(canArgs.fAAType == GrAAType::kMSAA &&
150                    GrFSAAType::kUnifiedMSAA != args.fRenderTargetContext->fsaaType()));
151         SkASSERT(!(canArgs.fAAType == GrAAType::kMixedSamples &&
152                    GrFSAAType::kMixedSamples != args.fRenderTargetContext->fsaaType()));
153         SkASSERT(CanDrawPath::kNo != this->canDrawPath(canArgs));
154         if (!args.fUserStencilSettings->isUnused()) {
155             SkPath path;
156             args.fShape->asPath(&path);
157             SkASSERT(args.fShape->style().isSimpleFill());
158             SkASSERT(kNoRestriction_StencilSupport == this->getStencilSupport(*args.fShape));
159         }
160 #endif
161         return this->onDrawPath(args);
162     }
163
```

```
struct DrawPathArgs {
    GrContext*                  fContext;
    GrPaint&&                   fPaint;
    const GrUserStencilSettings* fUserStencilSettings;
    GrRenderTargetContext*      fRenderTargetContext;
    const GrClip*               fClip;
    const SkIRect*              fClipConservativeBounds;
    const SkMatrix*             fViewMatrix;
    const GrShape*              fShape;
    GrAAType                    fAAType;
    bool                        fGammaCorrect;
```

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

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

```
136     */
137    bool drawPath(const DrawPathArgs& args) {
138        SkDEBUGCODE(args.validate();)
139 #ifdef SK_DEBUG
140        CanDrawPathArgs canArgs;
141        canArgs.fCaps = args.fContext->caps();
142        canArgs.fClipConservativeBounds = args.fClipConservativeBounds;
143        canArgs.fViewMatrix = args.fViewMatrix;
144        canArgs.fShape = args.fShape;
145        canArgs.fAAType = args.fAAType;
146        canArgs.validate();
147
148        canArgs.fHasUserStencilSettings = !args.fUserStencilSettings->isUnused();
149        SkASSERT(!(canArgs.fAAType == GrAAType::kMSAA &&
150                   GrFSAAType::kUnifiedMSAA != args.fRenderTargetContext->fsaaType()));
151        SkASSERT(!(canArgs.fAAType == GrAAType::kMixedSamples &&
152                   GrFSAAType::kMixedSamples != args.fRenderTargetContext->fsaaType()));
153        SkASSERT(CanDrawPath::kNo != this->canDrawPath(canArgs));
154        if (!args.fUserStencilSettings->isUnused()) {
155            SkPath path;
156            args.fShape->asPath(&path);
157            SkASSERT(args.fShape->style().isSimpleFill());
158            SkASSERT(kNoRestriction_StencilSupport == this->getStencilSupport(*args.fShape));
159        }
160 #endif
161        return this->onDrawPath(args);
162    }
163
```

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

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

```
230 bool GrSoftwarePathRenderer::onDrawPath(const DrawPathArgs& args) {
231     GR_AUDIT_TRAIL_AUTO_FRAME(args.fRenderTargetContext->auditTrail(),
232                               "GrSoftwarePathRenderer::onDrawPath");
233     if (!fProxyProvider) {
234         return false;
235     }
236
237     // We really need to know if the shape will be inverse filled or not
238     bool inverseFilled = false;
239     SkTLazy<GrShape> tmpShape;
240     SkASSERT(!args.fShape->style().applies());
241     // If the path is hairline, ignore inverse fill.
242     inverseFilled = args.fShape->inverseFilled() &&
243                     !IsStrokeHairlineOrEquivalent(args.fShape->style(), *args.fViewMatrix, nullptr);
244
245     SkIRect unclippedDevShapeBounds, clippedDevShapeBounds, devClipBounds;
246     // To prevent overloading the cache with entries during animations we limit the cache of masks
247     // to cases where the matrix preserves axis alignment.
```

```
361         if (!proxy) {
362             return false;
363         }
364         if (useCache) {
365             SkASSERT(proxy->origin() == kTopLeft_GrSurfaceOrigin);
366             fProxyProvider->assignUniqueKeyToProxy(maskKey, proxy.get());
367             args.fShape->addGenIDChangeListener(new PathInvalidator(maskKey));
368         }
369     }
370     if (inverseFilled) {
371         DrawAroundInvPath(args.fRenderTargetContext, GrPaint::Clone(args.fPaint),
372                           *args.fUserStencilSettings, *args.fClip, *args.fViewMatrix, devClipBounds,
373                           unclippedDevShapeBounds);
374     }
375     DrawToTargetWithShapeMask(
376             std::move(proxy), args.fRenderTargetContext, std::move(args.fPaint),
377             *args.fUserStencilSettings, *args.fClip, *args.fViewMatrix,
378             SkIPoint{boundsForMask->fLeft, boundsForMask->fTop}, *boundsForMask);
379
380     return true;
381 }
382
```

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

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

```
230 bool GrSoftwarePathRenderer::onDrawPath(const DrawPathArgs& args) {
231     GR_AUDIT_TRAIL_AUTO_FRAME(args.fRenderTargetContext->auditTrail(),
232                               "GrSoftwarePathRenderer::onDrawPath");
233     if (!fProxyProvider) {
234         return false;
235     }
236
237     // We really need to know if the shape will be inverse filled or not
238     bool inverseFilled = false;
239     SkTLazy<GrShape> tmpShape;
240     SkASSERT(!args.fShape->style().applies());
241     // If the path is hairline, ignore inverse fill.
242     inverseFilled = args.fShape->inverseFilled() &&
243                     !IsStrokeHairlineOrEquivalent(args.fShape->style(), *args.fViewMatrix, nullptr);
244
245     SkIRect unclippedDevShapeBounds, clippedDevShapeBounds, devClipBounds;
246     // To prevent overloading the cache with entries during animations we limit the cache of masks
247     // to cases where the matrix preserves axis alignment.
```

```
361         if (!proxy) {
362             return false;
363         }
364         if (useCache) {
365             SkASSERT(proxy->origin() == kTopLeft_GrSurfaceOrigin);
366             fProxyProvider->assignUniqueKeyToProxy(maskKey, proxy.get());
367             args.fShape->addGenIDChangeListener(new PathInvalidator(maskKey));
368         }
369     }
370     if (inverseFilled) {
371         DrawAroundInvPath(args.fRenderTargetContext, GrPaint::Clone(args.fPaint),
372                           *args.fUserStencilSettings, *args.fClip, *args.fViewMatrix, devClipBounds,
373                           unclippedDevShapeBounds);
374     }
375     DrawToTargetWithShapeMask(
376             std::move(proxy), args.fRenderTargetContext, std::move(args.fPaint),
377             *args.fUserStencilSettings, *args.fClip, *args.fViewMatrix,
378             SkIPoint{boundsForMask->fLeft, boundsForMask->fTop}, *boundsForMask);
379
380     return true;
381 }
382
```

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

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

```
141
142 void GrSoftwarePathRenderer::DrawToTargetWithShapeMask(
143         sk_sp<GrTextureProxy> proxy,
144         GrRenderTargetContext* renderTargetContext,
145         GrPaint&& paint,
146         const GrUserStencilSettings& userStencilSettings,
147         const GrClip& clip,
148         const SkMatrix& viewMatrix,
149         const SkIPoint& textureOriginInDeviceSpace,
150         const SkIRect& deviceSpaceRectToDraw) {
151     SkMatrix invert;
152     if (!viewMatrix.invert(&invert)) {
153         return;
154     }
155
156     SkRect dstRect = SkRect::Make(deviceSpaceRectToDraw);
157
158     // We use device coords to compute the texture coordinates. We take the device coords and apply
159     // a translation so that the top-left of the device bounds maps to 0,0, and then a scaling
160     // matrix to normalized coords.
161     SkMatrix maskMatrix = SkMatrix::MakeTrans(SkIntToScalar(-textureOriginInDeviceSpace.fX),
162                                               SkIntToScalar(-textureOriginInDeviceSpace.fY));
163     maskMatrix.preConcat(viewMatrix);
164     paint.addCoverageFragmentProcessor(GrSimpleTextureEffect::Make(
165             std::move(proxy), maskMatrix, GrSamplerState::Filter::kNearest));
166     DrawNonAARect(renderTargetContext, std::move(paint), userStencilSettings, clip, SkMatrix::I(),
167                   dstRect, invert);
168 }
169
```

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

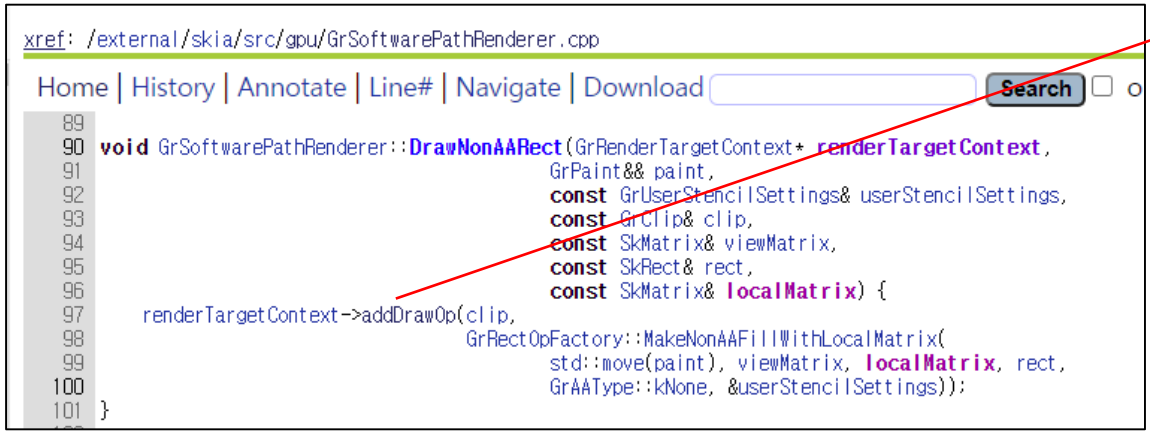
Home | History | Annotate | Line# | Navigate | Download Search only in Gr

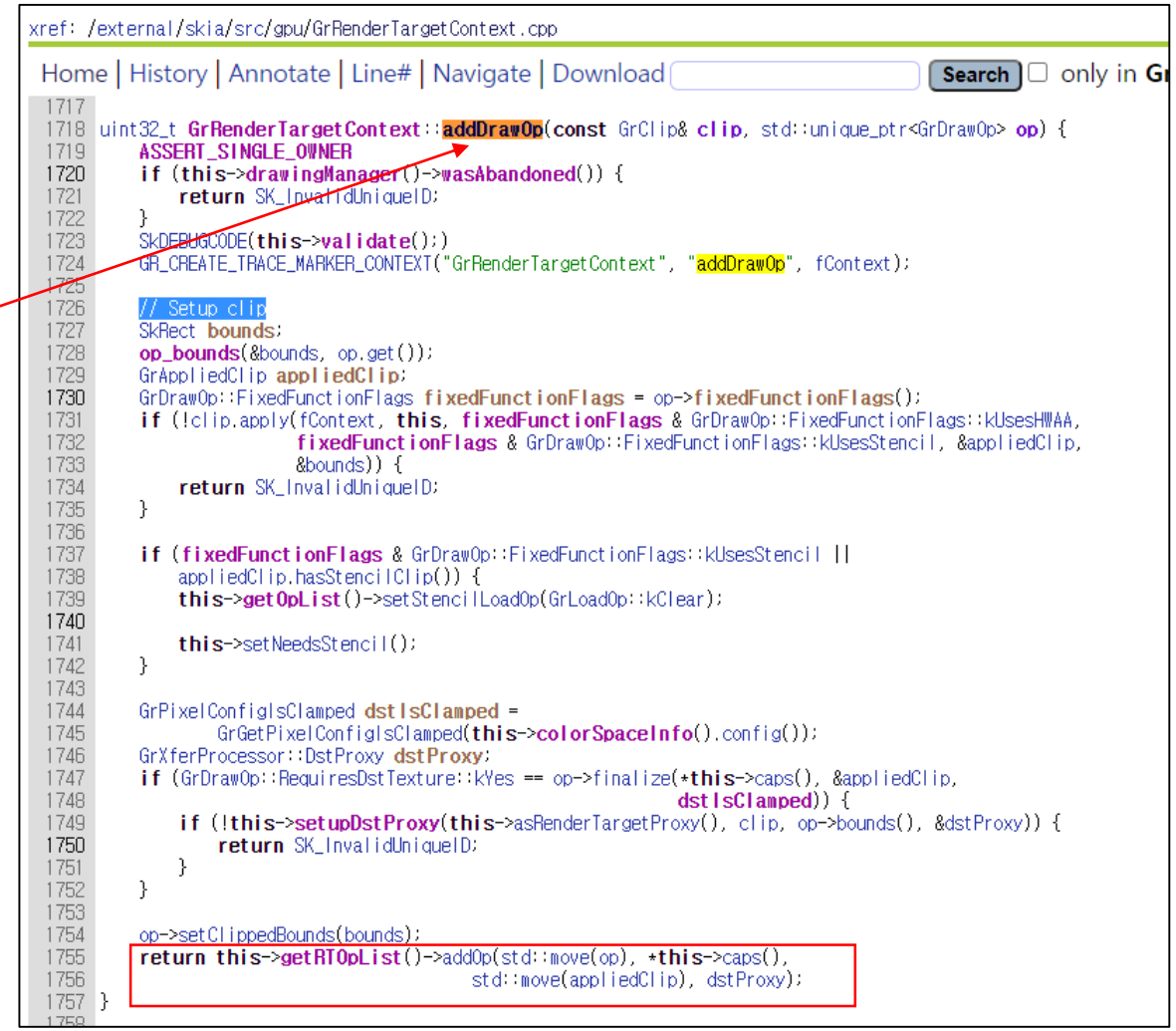
```
141
142 void GrSoftwarePathRenderer::DrawToTargetWithShapeMask(
143         sk_sp<GrTextureProxy> proxy,
144         GrRenderTargetContext* renderTargetContext,
145         GrPaint&& paint,
146         const GrUserStencilSettings& userStencilSettings,
147         const GrClip& clip,
148         const SkMatrix& viewMatrix,
149         const SkIPoint& textureOriginInDeviceSpace,
150         const SkIRect& deviceSpaceRectToDraw) {
151     SkMatrix invert;
152     if (!viewMatrix.invert(&invert)) {
153         return;
154     }
155
156     SkRect dstRect = SkRect::Make(deviceSpaceRectToDraw);
157
158     // We use device coords to compute the texture coordinates. We take the device coords and apply
159     // a translation so that the top-left of the device bounds maps to 0,0, and then a scaling
160     // matrix to normalized coords.
161     SkMatrix maskMatrix = SkMatrix::MakeTrans(SkIntToScalar(-textureOriginInDeviceSpace.fX),
162                                               SkIntToScalar(-textureOriginInDeviceSpace.fY));
163     maskMatrix.preConcat(viewMatrix);
164     paint.addCoverageFragmentProcessor(GrSimpleTextureEffect::Make(
165             std::move(proxy), maskMatrix, GrSamplerState::Filter::kNearest));
166     DrawNonAARect(renderTargetContext, std::move(paint), userStencilSettings, clip, SkMatrix::I(),
167                   dstRect, invert);
168 }
169
```

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

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

```
89
90 void GrSoftwarePathRenderer::DrawNonAARect(GrRenderTargetContext* renderTargetContext,
91                                             GrPaint&& paint,
92                                             const GrUserStencilSettings& userStencilSettings,
93                                             const GrClip& clip,
94                                             const SkMatrix& viewMatrix,
95                                             const SkRect& rect,
96                                             const SkMatrix& localMatrix) {
97     renderTargetContext->addDrawOp(clip,
98                                    GrRectOpFactory::MakeNonAAFillWithLocalMatrix(
99                                            std::move(paint), viewMatrix, localMatrix, rect,
100                                           GrAAType::kNone, &userStencilSettings));
101 }
```


xref: /external/skia/src/gpu/GrSoftwarePathRenderer.cpp
Home | History | Annotate | Line# | Navigate | Download
Search
89
90 void GrSoftwarePathRenderer::DrawNonAARect(GrRenderTargetContext* renderTargetContext,
91 GrPaint&& paint,
92 const GrUserStencilSettings& userStencilSettings,
93 const GrClip& clip,
94 const SkMatrix& viewMatrix,
95 const SkRect& rect,
96 const SkMatrix& localMatrix) {
97 renderTargetContext->addDrawOp(clip,
98 GrRectOpFactory::MakeNonAAFillWithLocalMatrix(
99 std::move(paint), viewMatrix, localMatrix, rect,
100 GrAAType::kNone, &userStencilSettings));
101 }


xref: /external/skia/src/gpu/GrRenderTargetContext.cpp
Home | History | Annotate | Line# | Navigate | Download
Search
only in Gr
1717
1718 uint32_t GrRenderTargetContext::addDrawOp(const GrClip& clip, std::unique_ptr<GrDrawOp> op) {
1719 ASSERT_SINGLE_OWNER
1720 if (this->drawingManager()->wasAbandoned()) {
1721 return SK_InvalidUniqueID;
1722 }
1723 SkDEBUGCODE(this->validate();)
1724 GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "addDrawOp", fContext);
1725
1726 // Setup clip
1727 SkRect bounds;
1728 op_bounds(&bounds, op.get());
1729 GrAppliedClip appliedClip;
1730 GrDrawOp::FixedFunctionFlags fixedFunctionFlags = op->fixedFunctionFlags();
1731 if (!clip.apply(fContext, this, fixedFunctionFlags & GrDrawOp::FixedFunctionFlags::kUsesHWAA,
1732 fixedFunctionFlags & GrDrawOp::FixedFunctionFlags::kUsesStencil, &appliedClip,
1733 &bounds)) {
1734 return SK_InvalidUniqueID;
1735 }
1736
1737 if (fixedFunctionFlags & GrDrawOp::FixedFunctionFlags::kUsesStencil ||
1738 appliedClip.hasStencilClip()) {
1739 this->getOpList()->setStencilLoadOp(GrLoadOp::kClear);
1740
1741 this->setNeedsStencil();
1742 }
1743
1744 GrPixelConfigIsClamped dstIsClamped =
1745 GrGetPixelConfigIsClamped(this->colorSpaceInfo().config());
1746 GrXferProcessor::DstProxy dstProxy;
1747 if (GrDrawOp::RequiresDstTexture::kYes == op->finalize(*this->caps(), &appliedClip,
1748 dstIsClamped)) {
1749 if (!this->setupDstProxy(this->asRenderTargetProxy(), clip, op->bounds(), &dstProxy)) {
1750 return SK_InvalidUniqueID;
1751 }
1752 }
1753
1754 op->setClippedBounds(bounds);
1755 return this->getRTOpList()->addOp(std::move(op), *this->caps(),
1756 std::move(appliedClip), dstProxy);
1757 }

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

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

```
1717
1718 uint32_t GrRenderTargetContext::addDrawOp(const GrClip& clip, std::unique_ptr<GrDrawOp> op) {
1719     ASSERT_SINGLE_OWNER
1720     if (this->drawingManager()->wasAbandoned()) {
1721         return SK_InvalidUniqueID;
1722     }
1723     SkDEBUGCODE(this->validate();)
1724     GR_CREATE_TRACE_MARKER_CONTEXT("GrRenderTargetContext", "addDrawOp", fContext);
1725
1726     // Setup clip
1727     SkRect bounds;
1728     op_bounds(&bounds, op.get());
1729     GrAppliedClip appliedClip;
1730     GrDrawOp::FixedFunctionFlags fixedFunctionFlags = op->fixedFunctionFlags();
1731     if (!clip.apply(fContext, this, fixedFunctionFlags & GrDrawOp::FixedFunctionFlags::kUsesHWAA,
1732                     fixedFunctionFlags & GrDrawOp::FixedFunctionFlags::kUsesStencil, &appliedClip,
1733                     &bounds)) {
1734         return SK_InvalidUniqueID;
1735     }
1736
1737     if (fixedFunctionFlags & GrDrawOp::FixedFunctionFlags::kUsesStencil ||
1738         appliedClip.hasStencilClip()) {
1739         this->getOpList()->setStencilLoadOp(GrLoadOp::kClear);
1740
1741         this->setNeedsStencil();
1742     }
1743
1744     GrPixelConfigIsClamped dstIsClamped =
1745             GrGetPixelConfigIsClamped(this->colorSpaceInfo().config());
1746     GrXferProcessor::DstProxy dstProxy;
1747     if (GrDrawOp::RequiresDstTexture::kYes == op->finalize(*this->caps(), &appliedClip,
1748                                                            dstIsClamped)) {
1749         if (!this->setupDstProxy(this->asRenderTargetProxy(), clip, op->bounds(), &dstProxy)) {
1750             return SK_InvalidUniqueID;
1751         }
1752     }
1753
1754     op->setClippedBounds(bounds);
1755     return this->getRTOpList()->addOp(std::move(op), *this->caps(),
1756                                       std::move(appliedClip), dstProxy);
1757 }
```

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

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

```
56     /**
57      * Together these two functions flush all queued up draws to GrCommandBuffer. The return value
58      * of executeOps() indicates whether any commands were actually issued to the GPU.
59      */
60     void onPrepare(GrOpFlushState* flushState) override;
61     bool onExecute(GrOpFlushState* flushState) override;
62
63     uint32_t addOp(std::unique_ptr<GrOp> op, const GrCaps& caps) {
64         auto addDependency = [ &caps, this ] (GrSurfaceProxy* p) {
65             this->addDependency(p, caps);
66         };
67
68         op->visitProxies(addDependency);
69
70         this->recordOp(std::move(op), caps);
71
72         return this->uniqueID();
73     }
74
75     uint32_t addOp(std::unique_ptr<GrOp> op, const GrCaps& caps,
76                    GrAppliedClip&& clip, const DstProxy& dstProxy) {
77         auto addDependency = [ &caps, this ] (GrSurfaceProxy* p) {
78             this->addDependency(p, caps);
79         };
80
81         op->visitProxies(addDependency);
82         clip.visitProxies(addDependency);
83
84         this->recordOp(std::move(op), caps, clip.doesClip() ? &clip : nullptr, &dstProxy);
85
86         return this->uniqueID();
87     }
88
```

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

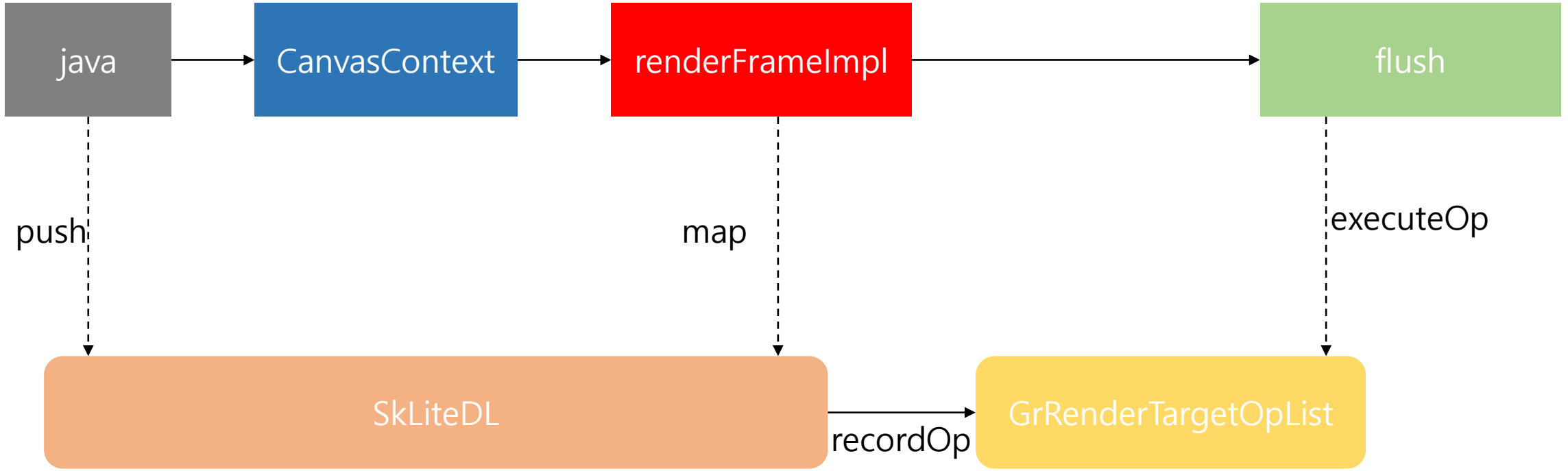
Home | History | Annotate | Line# | Navigate | Download Search only

```
56      /**
57       * Together these two functions flush all queued up draws to GrCommandBuffer. The return value
58       * of executeOps() indicates whether any commands were actually issued to the GPU.
59       */
60      void onPrepare(GrOpFlushState* flushState) override;
61      bool onExecute(GrOpFlushState* flushState) override;
62
63      uint32_t addOp(std::unique_ptr<GrOp> op, const GrCaps& caps) {
64          auto addDependency = [ &caps, this ] (GrSurfaceProxy* p) {
65              this->addDependency(p, caps);
66          };
67
68          op->visitProxies(addDependency);
69
70          this->recordOp(std::move(op), caps);
71
72          return this->uniqueID();
73      }
74
75      uint32_t addOp(std::unique_ptr<GrOp> op, const GrCaps& caps,
76                     GrAppliedClip&& clip, const DstProxy& dstProxy) {
77          auto addDependency = [ &caps, this ] (GrSurfaceProxy* p) {
78              this->addDependency(p, caps);
79          };
80
81          op->visitProxies(addDependency);
82          clip.visitProxies(addDependency);
83
84          this->recordOp(std::move(op), caps, clip.doesClip() ? &clip : nullptr, &dstProxy);
85
86          return this->uniqueID();
87      }
88
```

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

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

```
319
320 void GrRenderTargetOpList::recordOp(std::unique_ptr<GrOp> op,
321                                      const GrCaps& caps,
322                                      GrAppliedClip* clip,
323                                      const DstProxy* dstProxy) {
324     SkASSERT(fTarget.get());
325
326     // A closed GrOpList should never receive new/more ops
327     SkASSERT(!this->isClosed());
328
329     // Check if there is an op we can combine with by linearly searching back until we either
330     // 1) check every op
331     // 2) intersect with something
332     // 3) find a 'blocker'
333     GR_AUDIT_TRAIL_ADD_OP(fAuditTrail, op.get(), fTarget.get()->uniqueID());
334     GrOP_INFO("opList: %d Recording (%s, opID: %u)\n"
335               "\tBounds [L: %.2f, T: %.2f R: %.2f B: %.2f]\n",
336               this->uniqueID(),
337               op->name(),
338               op->uniqueID(),
339               op->bounds().fLeft, op->bounds().fTop,
340               op->bounds().fRight, op->bounds().fBottom);
341     GrOP_INFO(SkTabString(op->dumpInfo(), 1).c_str());
342     GrOP_INFO("\tOutcome:\n");
343     int maxCandidates = SkTMin(kMaxOpLookback, fRecordedOps.count());
344     // If we don't have a valid destination render target then we cannot reorder.
345     if (maxCandidates) {
346         int i = 0;
347         while (true) {
348             const RecordedOp& candidate = fRecordedOps.fromBack(i);
349
350             if (this->combineIfPossible(candidate, op.get(), clip, dstProxy, caps)) {
351                 GrOP_INFO("\t\tBackward: Combining with (%s, opID: %u)\n", candidate.fOp->name(),
352                           candidate.fOp->uniqueID());
353                 GrOP_INFO("\t\t\tBackward: Combined op info:\n");
354                 GrOP_INFO(SkTabString(candidate.fOp->dumpInfo(), 4).c_str());
355                 GR_AUDIT_TRAIL_OPS_RESULT_COMBINED(fAuditTrail, candidate.fOp.get(), op.get());
356                 return;
357             }
358             // Stop going backwards if we would cause a painter's order violation.
359             if (!can_reorder(fRecordedOps.fromBack(i).fOp->bounds(), op->bounds())) {
360                 GrOP_INFO("\t\tBackward: Intersects with (%s, opID: %u)\n", candidate.fOp->name(),
361                           candidate.fOp->uniqueID());
362                 break;
363             }
364             ++i;
365             if (i == maxCandidates) {
366                 GrOP_INFO("\t\tBackward: Reached max lookback or beginning of op array %d\n", i);
367                 break;
368             }
369         }
370     } else {
371         GrOP_INFO("\t\tBackward: FirstOp\n");
```


java
CanvasContext
renderFrameImpl
flush
push
map
executeOp
SkLiteDL
recordOp
GrRenderTargetOpList