Windows Bridge for iOS

[CoreGraphics / Direct2D]

**DEV DESIGN** specification

Contents

[Overview 1](#_Toc462739571)

[Approach 1](#_Toc462739572)

[Technical Details 1](#_Toc462739573)

[Current CoreGraphics Implementation 2](#_Toc462739574)

[Implementing CoreGraphics with Direct2D 2](#_Toc462739575)

[Drawing and Rendering 2](#_Toc462739576)

[Drawing - Colors 7](#_Toc462739577)

[Drawing – Advanced Colors 7](#_Toc462739578)

[Drawing – Arcs, Curves, Points, Lines 7](#_Toc462739579)

[Image Loading 8](#_Toc462739580)

[Strategy 10](#_Toc462739581)

# Overview

This document outlines the long-term strategy in providing a platform-integrated graphics layer to the Islandwood project by leveraging Windows Direct2D and its associated technologies.

# Approach

As in our parallel effort to convert CoreText to use DirectWrite, we will incrementally reimplement the CoreGraphics API surface in terms of platform-provided Direct2D APIs.

Priority will be given to CoreGraphics APIs in use by our current enterprise partners.

# Technical Details

CoreGraphics is a full-featured framework that offers support for drawing images at runtime, loading images from disk, color space conversions, hardware-assisted rendering and PDF rendering. Herein, we will refer to the following terms:

**Context**

The Context, in CoreGraphics, is the destination for all drawing operations. It holds a path in progress, a rendering destination, and its various drawing parameters in a stateful stack.

**Path**

A Path represents the series of points, lines and arcs required to draw a shape.

**Bitmap, Image**

A Bitmap is a set of pixels in a certain, implementation-specific backing format. During drawing, all of a context’s draw commands will cause pixels to be rasterized into a bitmap, which can be written to disk or displayed to the screen.

## Current CoreGraphics Implementation

Today, CoreGraphics is implemented using Cairo. Cairo is a cross-platform 2D graphics library that provides support for pluggable output targets. Of the wide variety of Cairo output targets, we use only one: the bitmap target.

Currently, to interoperate Cairo with XAML we request that a given control provide a writable bitmap buffer. That buffer is given to Cairo as the backing store for the bitmap output target, and all draw commands are executed into that buffer immediately.

## Implementing CoreGraphics with Direct2D

### Rationale

Reimplementing parts of CoreGraphics to leverage existing Windows APIs allows us to springboard off the hard work done by other windows teams in in-box components. CG/Cairo is a costly framework with its own 2D rendering engine; to wit, it is not composable, brings myriad unnecessary dependencies, and does not render its graphics in line with XAML and other Windows platform technologies. CG/D2D, on the other hand, takes advantage of the native Windows graphics pipeline, including but not limited to its ability to offload intensive rendering tasks to the GPU.

While this prospective implementation will render into the same XAML Writable Bitmap as our current implementation, we may eventually be able to replace the lower-level graphics components in WinObjC with accelerated targets. When that happens, CG/D2D’s reliance on the generic D2D render target model will pay dividends.

### Details

Windows Direct2D supports drawing shapes (in D2D parlance, Geometries) and images to rendering targets, which can include bitmap images, printing contexts, and hardware-backed surfaces. Additionally, the Windows Image Codec library provides support for storing and loading images in common container formats to streams and files on disk.

In each of the below sections, the high-level CoreGraphics verticals and their constituent low-level APIs are mapped to their counterparts in Direct2D. A table captures the recommended mappings between the two API surfaces and calls out specific gaps, if any. The below coloring scheme will be used throughout the document.

|  |  |  |
| --- | --- | --- |
|  | **CoreGraphics** | **Direct2D** |
|  | Interface1  Currently Interoperable | No functionally compatible interface is available |
|  | Interface2  Supported currently with Caveat | Interface is not fully compatible functionally |
|  | Interface3  Not currently supported | A functionally compatible interface is available |
|  | Interface4  Unknown status | No compatible interface is required; can be implemented in terms of CoreGraphics intrinsics. |

### Drawing and Rendering

#### CGContext

CGContext represents an abstract rendering backend into which colors, states, and paths are pushed. It is the primary entry point for all the CoreGraphics rendering APIs. As there may be multiple backend types, CGContext intentionally leaves many of its parameters vague and implementation-defined.

Its Direct2D analogue is the ID2D1RenderTarget interface, which abstracts rendering over several hardware and software backends.

##### Proposal

As CGContext maps to ID2DRenderTarget particularly well, it makes sense to provide a CGContext whose methods are implemented in terms of a D2D render target as per the mapping below. This will allow for any present and future D2D render targets to be fully represented in CoreGraphics.

###### Data Structure

struct \_CGD2DContext {

ComPtr<ID2D1RenderTarget> \_renderTarget;

std::stack<ComPtr<ID2D1DrawingStateBlock>> \_d2dDrawingStateStack;

std::stack<\_CGD2DDrawingState> \_cgDrawingStateStack;

CGPathRef \_currentPath;

// state control members that are documented as being **not part of the state snapshot**

bool \_allowsAntialiasing;

bool \_allowsFontSmoothing;

bool \_allowsFontSubpixelPositioning;

bool \_allowsFontSubpixelQuantization;

}

struct \_CGD2DDrawingState {

// contains state that is not captured in the D2D Drawing State block; this includes

// anything that is set at **draw time only**, or is otherwise not part of D2D’s global state.

ComPtr<ID2D1StrokeStyle> \_strokeStyle;

ComPtr<ID2D1Brush> \_fillBrush;

ComPtr<ID2D1Brush> \_strokeBrush;

CGFloat \_flatness;

CGPathRef \_currentClippingPath;

D2D1\_INTERPOLATION\_MODE \_bitmapInterpolationMode;

}

##### API Mappings

|  |  |  |
| --- | --- | --- |
|  | CoreGraphics | Direct2D |
|  | **Drawing Parameters** | |
|  | CGContextGetInterpolationQuality  CGContextSetInterpolationQuality | ID2D1RenderTarget::DrawBitmap(…, D2D1\_INTERPOLATION\_MODE) |
|  | CGContextSetFlatness | Not represented in Direct2D. |
|  | CGContextSetLineCap  CGContextSetLineDash CGContextSetLineJoin  CGContextSetLineWidth  CGContextSetMiterLimit | ID2D1RenderTarget::DrawGeometry(…, ID2D1StrokeStyle) |
|  | CGContextSetPatternPhase | Transform on bitmap brush. |
|  | CGContextSetFillPattern | ID2D1BitmapBrush |
|  | CGContextSetStrokePattern | ID2D1BitmapBrush |
|  | CGContextSetRenderingIntent | ID2D1DeviceContext::CreateEffect( …*ColorManagementEffect*… )  ID2D1DeviceContext::DrawImage(…*effect*…) |
|  | CGContextSetBlendMode | D2D1\_BLEND\_MODE |
|  | CGContextSetAllowsAntialiasing | Internal property on \_CGD2DContext. |
|  | CGContextSetShouldAntialias | ID2D1RenderTarget::SetAntialiasMode |
|  | CGContextSetAllowsFontSmoothing | Internal property on \_CGD2DContext. |
|  | CGContextSetShouldSmoothFonts | ID2D1RenderTarget::SetTextAntialiasMode +  NO : D2D1\_TEXT\_ANTIALIAS\_MODE\_ALIASED  YES: D2D1\_TEXT\_ANTIALIAS\_MODE\_GRAYSCALE |
|  | CGContextSetAllowsFontSubpixelPositioning | Internal property on \_CGD2DContext. |
|  | CGContextSetShouldSubpixelPositionFonts | ID2D1RenderTarget::SetTextAntialiasMode +  YES: D2D1\_TEXT\_ANTIALIAS\_MODE\_CLEARTYPE  NO : D2D1\_TEXT\_ANTIALIAS\_MODE\_GRAYSCALE |
|  | CGContextSetAllowsFontSubpixelQuantization | Internal property on \_CGD2DContext. |
|  | CGContextSetShouldSubpixelQuantizeFonts | YES: D2D1\_TEXT\_ANTIALIAS\_MODE\_CLEARTYPE |
|  | **State Management** | |
|  | CGContextSaveGState | ID2D1RenderTarget::SaveDrawingState  *Push onto \_drawingStateStack.*  *There are some parameters we must keep track of ourselves.* |
|  | CGContextRestoreGState | ID2D1RenderTarget::RestoreDrawingState  *Pop off \_drawingStateStack.*  *There are some parameters we must keep track of ourselves.* |
|  | **Paths** | |
|  | CGContextAdd…  Arc  ArcToPoint  CurveToPoint  Lines  LineToPoint  Path  QuadCurveToPoint  Rect  Rects  EllipseInRect | Implement using CGPath.  Operates on \_currentPath ivar. |
|  | CGContextCopyPath |
|  | CGContextBeginPath |
|  | CGContextClosePath |
|  | CGContextMoveToPoint |
|  | CGContextIsPathEmpty |
|  | CGContextGetPathCurrentPoint |
|  | CGContextGetPathBoundingBox |
|  | CGContextPathContainsPoint |
|  | **Clipping** | |
|  | CGContextClip | ID2D1RenderTarget::PushLayer( *geometry* )  EO is a property of the *Geometry*.  **If the Geometry represents a rectangle, we can perhaps use PushAxisAlignedClip instead.** |
|  | CGContextEOClip |
|  | CGContextClipToRect | CGContextClip |
|  | CGContextClipToRects |
|  | CGContextGetClipBoundingBox | Implement in terms of the CGPath used to create the clipping geometry. |
|  | CGContextClipToMask | ID2D1RenderTarget::PushLayer( *mask bitmap brush* ) |
|  | **Drawing** | |
|  | CGContextDrawPath | ID2D1::FillGeometry  ID2D1::StrokeGeometry  EO is a property of the *Geometry* |
|  | CGContextStrokePath | CGContextDrawPath(Stroke) |
|  | CGContextFillPath | CGContextDrawPath(Fill) |
|  | CGContextEOFillPath | CGContextDrawPath(EOFill) |
|  | CGContextFillRect | FillGeometry |
|  | CGContextFillRects |
|  | CGContextFillEllipseInRect |
|  | CGContextStrokeRect | DrawGeometry |
|  | CGContextStrokeRectWithWidth |
|  | CGContextStrokeEllipseInRect |
|  | CGContextStrokeLineSegments |
|  | **Colors** | |
|  | CGContextSetAlpha | Store internally.  **On Draw:**  ID2D1RenderTarget::PushLayer( *alpha* )  // draw geometry  ::PopLayer(…) |
|  | CGContextSetCMYKFillColor | Solid Color:  ID2D1RenderTarget::CreateSolidColorBrush  Pattern:  Draw pattern into CGBitmapContext  Capture CGBitmapContext’s Render Target  ID2D1RenderTarget::CreateBitmapBrush |
|  | CGContextSetFillColor |
|  | CGContextSetCMYKStrokeColor |
|  | CGContextSetFillColorWithColor |
|  | CGContextSetGrayFillColor |
|  | CGContextSetGrayStrokeColor |
|  | CGContextSetRGBFillColor |
|  | CGContextSetRGBStrokeColor |
|  | CGContextSetStrokeColor |
|  | CGContextSetStrokeColorWithColor |
|  | CGContextSetShadow | ID2D1DeviceContext::CreateEffect(  *Affine Transform Effect(*  *Shadow Effect*  *)*  )  ID2D1DeviceContext::DrawImage(…*effect*…) |
|  | CGContextSetShadowWithColor |
|  | **Context Transformation** | |
|  | CGContextConcatCTM  CGContextGetCTM  CGContextRotateCTM  CGContextScaleCTM  CGContextTranslateCTM | ID2D1RenderTarget::SetTransform  *Care must be taken to ensure that the CG transformation (native 1.0x-1.0 on iOS) is undone to match D2D’s coordinate system, which is top-left-anchored.*  *Rotations in D2D are backwards compared to those in CoreGraphics, so account for this as well.* |
|  | **Layering** | |
|  | CGContextBeginTransparencyLayer | ID2D1RenderTarget::PushLayer |
|  | CGContextBeginTransparencyLayerWithRect | ID2D1RenderTarget::PushLayer( *rect* ) |
|  | CGContextEndTransparencyLayer | ID2D1RenderTarget::PopLayer |
|  | **Drawing Images** | |
|  | CGContextDrawImage | ID2D1RenderTarget::DrawBitmap |
|  | CGContextDrawTiledImage | ID2D1RenderTarget::CreateBitmapBrush  *Use parameters for pattern repeat, and draw into full clipping region.* |
|  | **Drawing Gradients** | |
|  | CGContextDrawLinearGradient | ID2D1RenderTarget::CreateLinearGradientBrush |
|  | CGContextDrawRadialGradient | ID2D1RenderTarget::CreateRadialGradientBrush |
|  | **Drawing Text** | |
|  | *Text drawing will be relegated to CoreText, and is without the scope of this document.* | |

#### CGBitmapContext

CGBitmapContext is a CGContext that draws to an off-screen bitmap buffer. As such, it can be implemented in terms of **\_CGD2DContext**, above, with an ID2D1WicBitmapRenderTarget.

Consideration must be paid to the pixel format of a bitmap, and the bitmap’s context. Direct2D only supports a subset of its available pixel formats for a WIC Bitmap render target.

|  |  |
| --- | --- |
| Format | Alpha Mode |
| 32bpp BGRA | Premultiplied |
| Ignored |
| Unknown |
| 8bpp Alpha | Premultiplied |
| Straight |
| Unknown |
| Unknown | Premultiplied |
| Ignored |
| Unknown |

As such, the basic CoreGraphics Alpha and Bitmap modes do not perfectly map to those offered by Direct2D.

Images with mismatched alpha/bitmap modes, such as those loaded from a file, may require format translations. This is in line with our current implementation caveats.

Direct2D + DXGI rendering to a WIC Bitmap Context does not support Alpha-First formats.

Please see the *Bitmap Constants* section for a D2D/DXGI constant mapping.

|  |  |  |
| --- | --- | --- |
|  | CoreGraphics | Direct2D |
|  | CGBitmapContextCreate | Implement in terms of \_CGD2DContext + \_CGWICBitmapImage.  **Ideally**, we are in control of the backing formats for all bitmaps in use in our system. For places that need access to raw drawn-into byte data, concessions may have to be made.  *CG does not allow users to specify the actual pixel format in memory.* |
|  | CGBitmapContextCreateWithData  **NB**: poorly-named; both act on user-provided data.  The difference here is that a callback is emitted on Release. |
|  | CGBitmapContextCreateImage | Return a snapshot of our internal CGImage. |
|  | CGBitmapContextGetBitmapInfo | Local property on CGBitmapContext. |
|  | CGBitmapContextGetAlphaInfo |
|  | CGBitmapContextGetBitsPerComponent | Implement in terms of backing CGImage. |
|  | CGBitmapContextGetBitsPerPixel |
|  | CGBitmapContextGetBytesPerRow |
|  | CGBitmapContextGetColorSpace |
|  | CGBitmapContextGetData |
|  | CGBitmapContextGetHeight  CGBitmapContextGetWidth |

#### CGPDFContext

**No consideration is currently afforded to PDF rendering.**

#### CGLayer

##### Data Structure

struct \_\_CGLayer {

<unnamed-type> buffer;

CGContextRef context;

}

CGLayer is an offscreen buffer whose contents can be blitted to another context. As it approximates an image with a separate backing context, it can be implemented on top of other CoreGraphics and D2D primitives.

The reference platform documents that a CGLayer may be hardware-backed or copied to video memory, but the developer is not in control of whether this occurs.

A CGLayer contains a context that can be drawn into to manipulate the backing store.

ID2D1RenderTarget::CreateCompatibleRenderTarget can create a render target that we will use to back a CGContext.

### Drawing - Colors

#### CGColor

*[TBD]*

#### CGColorSpace

*[TBD]*

### Drawing – Advanced Colors

#### CGGradient

Struct \_\_CGGradient {

CGColor \_colors[];

CGFloat \_stops[];

}

CGGradient represents a set of colors and color stops, and can be modelled as a simple data carrier.

When a gradient is translated into a brush, the following APIs will be used.

Its analogues are **ID2D1LinearGradientBrush** and **ID2D1RadialGradientBrush.**

|  |  |  |
| --- | --- | --- |
|  | CoreGraphics | Direct2D |
|  |  | ID2D1RenderTarget::CreateLinearGradientBrush  ID2D1RenderTarget::CreateRadialGradientBrush |
|  | kCGGradientDrawsBeforeStartLocation  kCGGradientDrawsBeforeEndLocation | D2D1\_EXTEND\_MODE\_CLAMP (extent) |

#### CGPattern

CGPattern is a special type of user-drawn brush. The user provides a callback, and the callback is called with a CGContext. Into that context is rendered a small, tileable image.

When a pattern is translated into a brush, it will be an **ID2D1BitmapBrush** generated by drawing into a **CGBitmapContext**.

#### CGShading

### Drawing – Arcs, Curves, Points, Lines

#### CGPath / CGMutablePath

##### Data Structure

struct \_\_CGPath {

ComPtr<ID2D1PathGeometry> \_pathGeometry;

// mutable only

ComPtr<ID2D1GeometrySink> \_geometrySink;

}

CGPath represents a series of points connected by lines, arcs and square or quadratic curves. There is a direct mapping between CGPath and ID2D1PathGeometry, and CGMutablePath and ID2D1GeometrySink.

ID2DIPathGeometry implements the ID2D1Geometry interface, which provides for tessellation, widening, simplification, and stroking. However, ID2D1Geometry itself does not provide for generic path operations such as enumeration (relegated to path geometries). For this reason, ID2D1Geometry implementers such as ID2D1RectangleGeometry and ID2D1RoundedRectGeometry cannot be used to implement CGPath with full fidelity.

|  |  |  |
| --- | --- | --- |
|  | CoreGraphics | Direct2D |
|  | CGPathCreateMutable | ID2D1Factory::CreatePathGeometry()  ID2D1PathGeometry::Open(ID2DIGeometrySink\*\*) |
|  | CGPathCreateWithEllipseInRect  CGPathCreateWithRect | Implement in terms of other CoreGraphics primitives  CGPathCreateMutable  CGPathAdd{EllipseInRect,Rect,RoundedRect} |
|  | CGPathCreateWithRoundedRect |
|  | CGPathCreateCopyByTransformingPath  CGPathCreateCopyByDashingPath  CGPathCreateCopyByStrokingPath | ID2D1TransformedGeometry  ID2D1Geometry::Widen  Requires custom implementations of the *ID2D1GeometrySinks* to collect the new geometry elements. |
|  | CGPathCreateCopy CGPathCreateMutableCopy | Implement in terms of CoreGraphics primitives.  ID2D1Geometry::Stream can copy a geometry. |
|  | CGPathAdd\* | ID2D1GeometrySink::Add\*  ID2D1SimplifiedGeometrySink::Add\* |
|  | CGPathAddArcToPoint | ID2D1GeometrySink::BeginFigure(P1) (optional)  ID2D1GeometrySink::AddArc(P2) |
|  | CGPathAddQuadCurveToPoint | ID2D1GeometrySink::BeginFigure(P1) (optional)  ID2D1GeometrySink::AddQuadraticBezier(P2) |
|  | CGPathAddRelativeArc | There does not appear to be a Direct2D API that creates a relative arc. |
|  | CGPathAddRoundedRect | Implement in terms of AddLine and AddArcToPoint. |
|  | CGPathAddRects | Implement in terms of AddRect |
|  | CGPathApply | ID2D1PathGeometry::Stream(…custom sink…) |
|  | CGPathGetBoundingBox | ID2D1Geometry::GetBounds |
|  | CGPathGetPathBoundingBox | ID2D1Geometry::GetBounds (?)  This API should ignore Quad/Bezier control points. |
|  | CGPathGetCurrentPoint | We may have to track this ourselves. |
|  | CGPathIsEmpty | ID2D1Geometry::GetBounds will return an invalid rect for an empty Geometry. |
|  | CGPathIsRect | We must calculate this ourselves. |
|  | CGPathContainsPoint | ID2D1Geometry::FillContainsPoint  ID2D1Geometry::StrokeContainsPoint |
|  | CGPathEqualToPath | ID2D1Geometry::CompareWithGeometry |
|  | CGPathMoveToPoint | ID2D1GeometrySink::BeginFigure(point) |

### Image Loading

#### CGDataProvider

CGDataProvider provides an abstraction over a read-only stream or seekable buffer of data. It is not being considered for replacement as part of this initiative.

#### CGDataConsumer

CGDataConsumer provides an abstraction over a write-only stream of data. It is not being considered for replacement as part of this initiative.

#### CGImageSource

CGImageSource is an abstraction over CGDataProvider that allows for the loading of multi-frame images and their associated metadata.

**It is already implemented in terms of IWICBitmapDecoder.**

*Changes will be required to avoid unnecessary buffer copying as we move from Cairo to Direct2D.*

#### CGImageDestination

CGImageDestination is an abstraction over a CGDataConsumer which allows for the emission of multi-frame images and their associated metadata to a stream. It has a nearly perfect parallel in IWICBitmapEncoder.

**It is, in fact, already largely implemented in terms of IWICBitmapEncoder.**

|  |  |  |
| --- | --- | --- |
|  | CoreGraphics | Windows Imaging Codecs |
|  | CGImageDestinationCreate | IWICImagingFactory::CreateEncoder |
|  | CGImageDestinationAddImage | IWICBitmapEncoder::CreateNewFrame  Populate new IWICBitmapFrameEncode from IWICBitmap and various Sources from CGImage. |
|  | CGImageDestinationAddImageFromSource | IWICIBitmapEncoder::CreateNewFrame  Populate new IWICBitmapFrameEncode from IWICBitmapSource, via CGImageSource. |
|  | CGImageDestinationSetProperties | IWICIBitmapEncoder::GetMetadataQueryWriter |
|  | CGImageDestinationFinalize | IWICIBitmapEncoder::Commit |

#### CGImage

CGImage represents an arbitrary bucket of pixel data, possibly loaded from disk, which can be blitted into a context, drawn into, and rendered to the screen. CGImages can also be written to image streams such as CGImageDestination and CGDataConsumer.

It enjoys many parallels with IWICBitmap and IWICBitmapSource.

The current implementation caveats here revolve around pixel format support.

|  |  |  |
| --- | --- | --- |
|  | CoreGraphics | Windows Imaging Codecs |
|  | CGImageCreate | IWICImagingFactory::CreateImage |
|  | CGImageCreateCopy | IWICImagingFactory::CreateBitmapFromSource( *img* ) |
|  | CGImageCreateCopyWithColorSpace | Color Spaces are [TBD] |
|  | CGImageCreateWithJPEGDataProvider | IWICImagingFactory::CreateDecoder( *jpeg* ) |
|  | CGImageCreateWithPNGDataProvider | IWICImagingFactory::CreateDecoder( *png* ) |
|  | CGImageCreateWithImageInRect | IWICImagingFactory::CreateBitmapFromSourceRect( *rect* ) |
|  | CGImageCreateWithMask | IWICImagingFactory::CreateImage  *A mask is an image with only a black and white channel.* |
|  | CGImageCreateWithMaskingColors |
|  | CGImageMaskCreate | CGImageMask is a CGImage with a single channel, representing opacity in black and white. |
|  | CGImageGetAlphaInfo | IWICBitmapSource::GetPixelFormat  *Caveat: The pixel format returned by this method is not necessarily the pixel format the image is stored as. The codec may perform a format conversion from the storage pixel format to an output pixel format.*  <https://msdn.microsoft.com/en-us/library/windows/desktop/ee690181(v=vs.85).aspx> |
|  | CGImageGetBitmapInfo |
|  | CGImageGetBitsPerComponent |
|  | CGImageGetBitsPerPixel |
|  | CGImageGetBytesPerRow |
|  | CGImageGetColorSpace | Return internal property. |
|  | CGImageGetDataProvider | Return internal property. |
|  | CGImageGetDecode | Decode arrays may not be represented in Direct2D. |
|  | CGImageGetHeight | IWICBitmapSource::GetSize |
|  | CGImageGetWidth | IWICBitmapSource::GetSize |
|  | CGImageGetShouldInterpolate | Internal property.  Propagated to ID2D1RenderTarget::DrawBitmap |
|  | CGImageGetRenderingIntent | The rendering intent can bet set at draw time using an ID2D1Effect. |
|  | CGImageIsMask | Internal property. |

##### Bitmap Constants

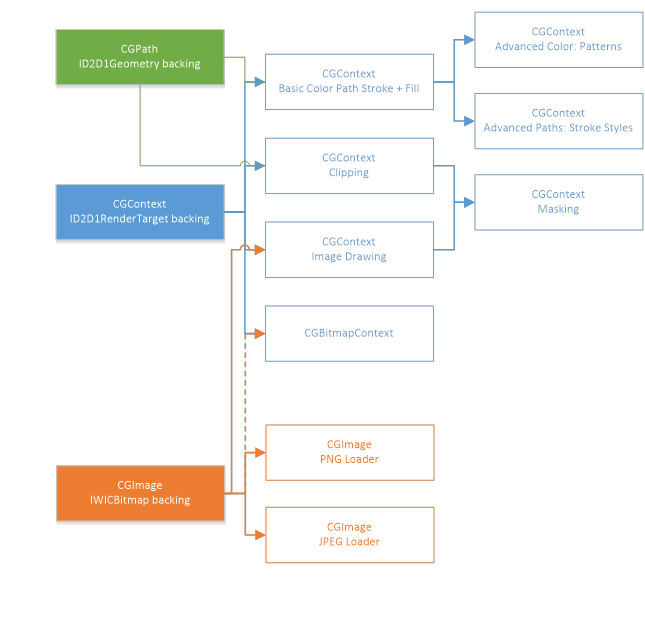
|  |  |
| --- | --- |
| CoreGraphics | Direct2D |
| **Alpha** | |
| kCGImageAlphaNone | D2D1\_ALPHA\_MODE\_IGNORE |
| kCGImageAlphaPremultipliedLast | DXGI\_FORMAT\_B8G8R8A8\_UNORM + D2D1\_ALPHA\_MODE\_PREMULTIPLIED |
| kCGImageAlphaPremultipliedFirst | No equivalent. |
| kCGImageAlphaLast | DXGI\_FORMAT\_B8G8R8A8\_UNORM + ???  D2D does not support ALPHA\_MODE\_STRAIGHT for BGRA8. |
| kCGImageAlphaFirst | No equivalent. |
| kCGImageAlphaNoneSkipLast | DXGI\_FORMAT\_B8G8R8A8\_UNORM + D2D1\_ALPHA\_MODE\_IGNORED |
| kCGImageAlphaNoneSkipFirst | No equivalent. |
| kCGImageAlphaOnly | DXGI\_FORMAT\_A8\_UNORM + D2D1\_ALPHA\_MODE\_STRAIGHT |
| **Format** | |
| kCGBitmapFloatComponents | *No direct equivalents; Direct2D and WIC offer format IDs, but they do not map to endianness.* |
| kCGBitmapByteOrderDefault |
| kCGBitmapByteOrder16Little |
| kCGBitmapByteOrder32Little |
| kCGByteOrder16Big |
| kCGByteOrder32Big |

# Strategy

Given we have a working CoreGraphics, it would be reasonable to perform an all-at-once migration in a separate working branch. By removing the Cairo code up front, we can identify what breakages we will encounter.

The CoreGraphics APIs derive from three major entry points: CGPath, CGContext and CGImage. While CGContext can draw a CGImage, and CGBitmapContext can draw into a CGImage, they can be worked on separately until they need to interoperate. Paths, likewise, are required only for drawing paths: image rendering can be completed in absence of path drawing. After the primary headline work on these three umbrella APIs, APIs that derive from and interoperate with them can be tackled in parallel.

This document proposes that work proceed in the following order.



**Figure 0** The hollow boxes represent parallel tasks, and the two filled boxes represent blocking tasks. The arrows radiating out from the tasks signify dependencies. For example, finishing CGImage(WIC) will unblock both the PNG loader and the JPEG loader, and finishing both CGContext(D2D) and CGImage(WIC) will unblock somebody to work on CGBitmapContext.