Advanced Tips and Tricks for High Resolution on OS X

Subtitle for Rent

Session 245

Patrick Heynen

Introduction

- Diving deeper into high resolution for OS X
- Taking full advantage of new APIs to achieve pixel precision
- Leveraging advanced Quartz technologies under high resolution
- Optimizing visual quality and performance

What You Will Learn

What You Will Learn

- How to work with OpenGL contexts

What You Will Learn

- How to work with OpenGL contexts
- Managing custom Core Animation layer trees

What You Will Learn

- How to work with OpenGL contexts
- Managing custom Core Animation layer trees
- Drawing into off-screen bitmaps

What You Will Learn

- How to work with OpenGL contexts
- Managing custom Core Animation layer trees
- Drawing into off-screen bitmaps
- Cooperating with dynamic display resolution changes

What You Will Learn

- How to work with OpenGL contexts
- Managing custom Core Animation layer trees
- Drawing into off-screen bitmaps
- Cooperating with dynamic display resolution changes
- Working with screen fonts and text rendering

What You Will Learn

- How to work with OpenGL contexts
- Managing custom Core Animation layer trees
- Drawing into off-screen bitmaps
- Cooperating with dynamic display resolution changes
- Working with screen fonts and text rendering
- Best practices for achieving quality and performance under high resolution

Technology Overview

Technology Overview

- New high-resolution display modes for Retina displays

Technology Overview

- New high-resolution display modes for Retina displays
- Screens and windows have a 2:1 pixel per point density ratio

Technology Overview

- New high-resolution display modes for Retina displays
- Screens and windows have a 2:1 pixel per point density ratio
- Frameworks provide automatic scaling to ensure consistent coordinate systems between 1x and 2x operation

Technology Overview

- New high-resolution display modes for Retina displays
- Screens and windows have a 2:1 pixel per point density ratio
- Frameworks provide automatic scaling to ensure consistent coordinate systems between 1x and 2x operation
- Quartz Window Manager ensures consistent presentation across multiple displays

NSImage and High Resolution

Chris Dreessen

NSImageRep Selection

- NSImage can contain many NSImageReps

NSImageRep Selection

- NSImage can contain many NSImageReps
- There is no notion of high or low resolution—It's all about pixels

NSImageRep Selection

- NSImage can contain many NSImageReps
- There is no notion of high or low resolution—It's all about pixels
- NSImage will prefer the smallest bitmap representation that has more pixels than the destination

NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

NSImage
Image Representations

cake.png (1x)

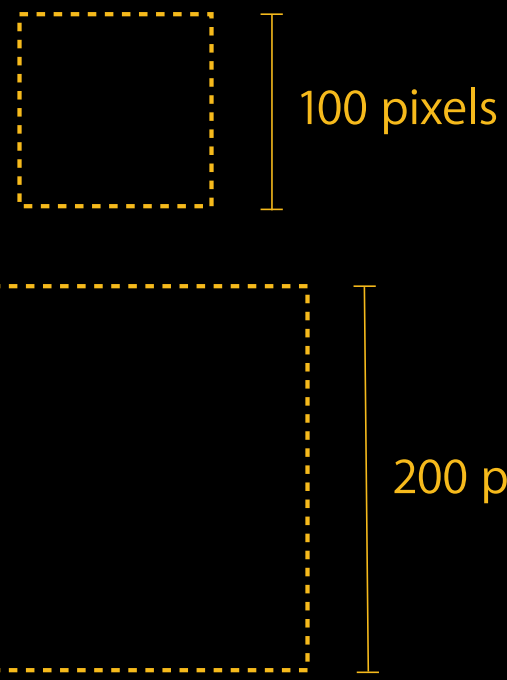
cake@2x.png (2x)

NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

Destination



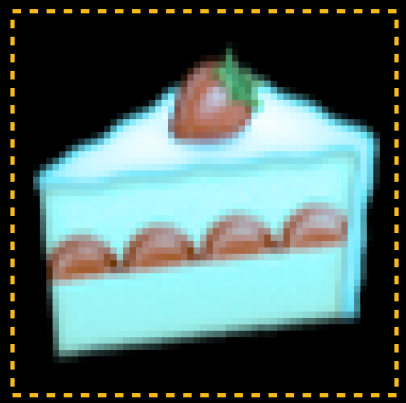
NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

Destination

100 pixels



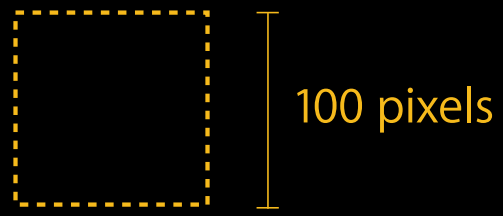
200 pixels

NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

Destination



NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

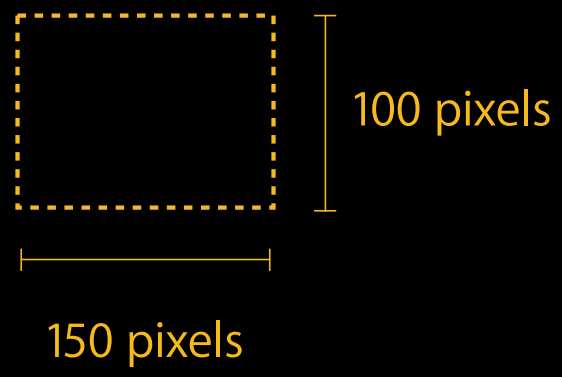
Destination

NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

Destination

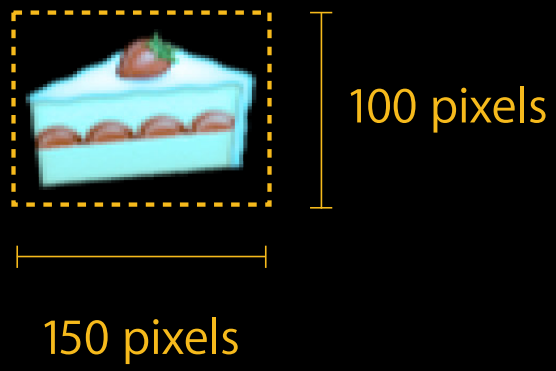


NSImage
Image Representations

cake.png (1x)

cake@2x.png (2x)

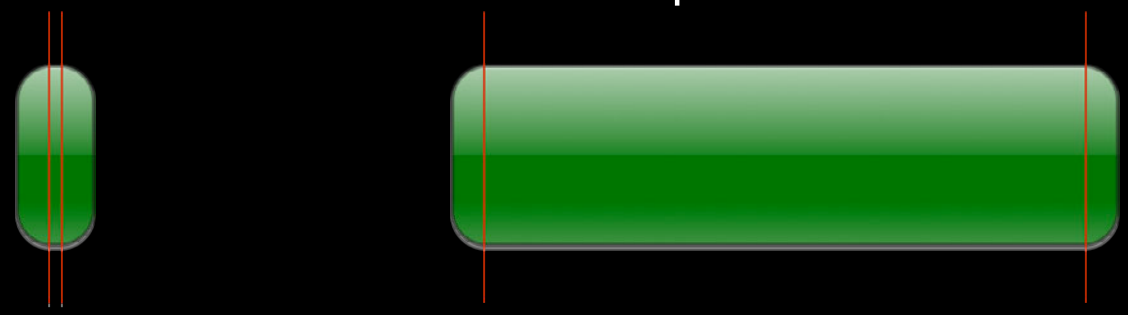
Destination



Stretching Images

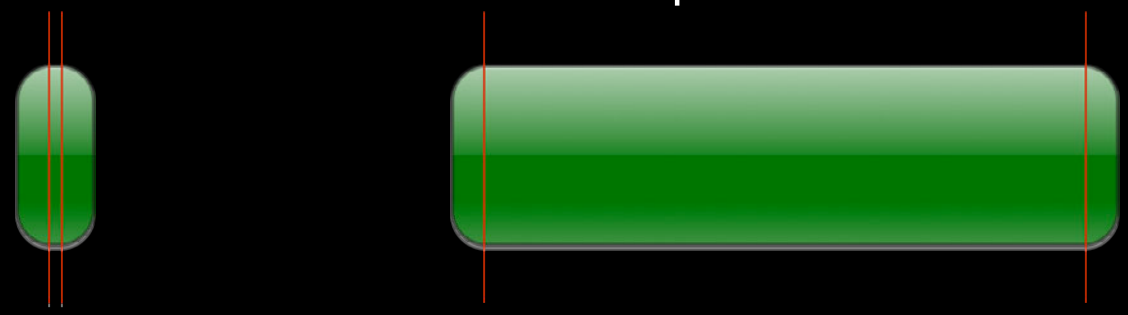
Stretching Images

- Consider a button made of two end caps and a resizable center

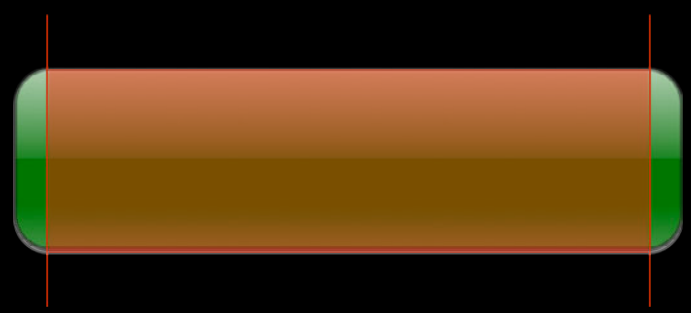


Stretching Images

- Consider a button made of two end caps and a resizable center



- Because of the stretch, NSImage will choose a 2x representation



Stretching Images

- Use these functions instead

  `NSDrawThreePartImage`

  `NSDrawNinePartImage`

- These tile the middle image instead of stretching it

Stretching Images

- Use these functions instead

  `NSDrawThreePartImage`

  `NSDrawNinePartImage`

- These tile the middle image instead of stretching it

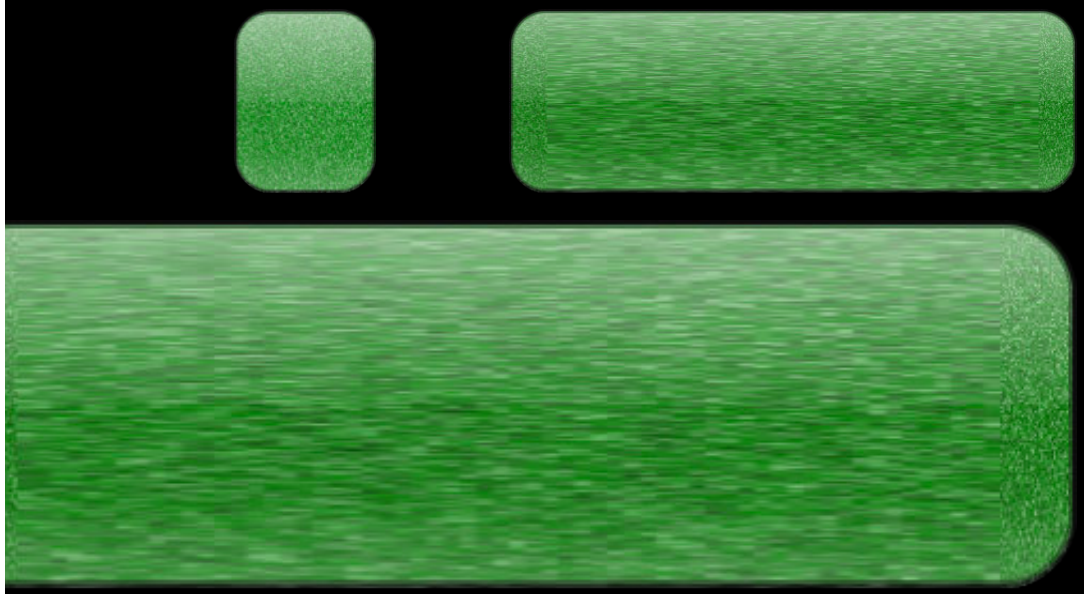
Stretching Images

- Use these functions instead

  `NSDrawThreePartImage`

  `NSDrawNinePartImage`

- These tile the middle image instead of stretching it



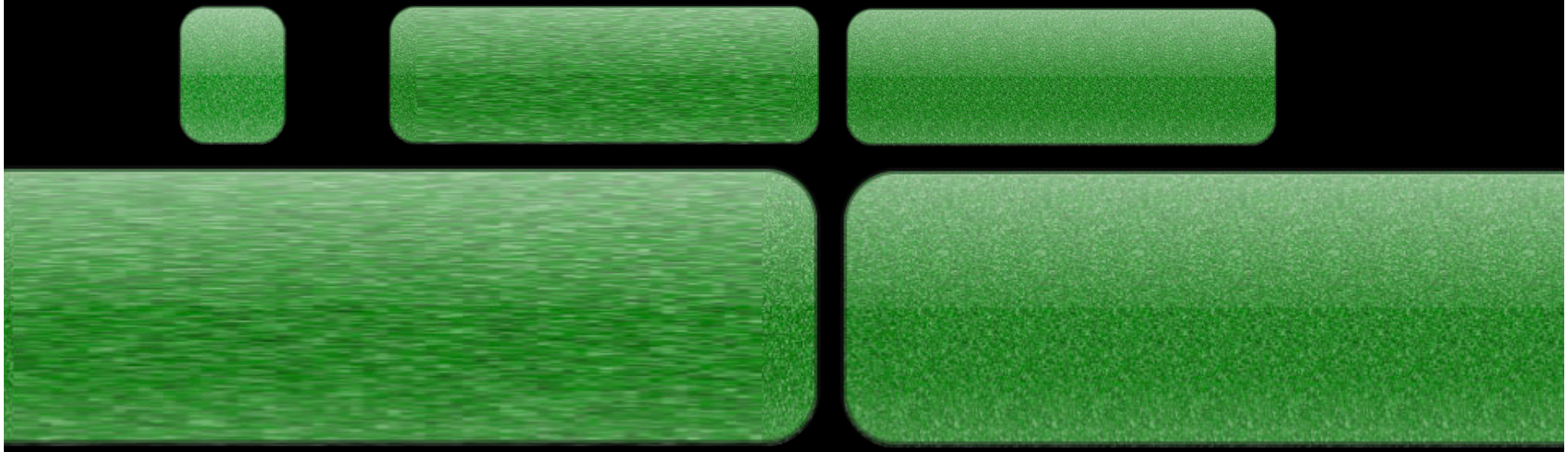
Stretching Images

- Use these functions instead

  `NSDrawThreePartImage`

  `NSDrawNinePartImage`

- These tile the middle image instead of stretching it



Stretching Images

- If you really can't tile but want to avoid the 2x rep being used:
  - `[NSImage setMatchesOnlyOnBestFittingAxis:YES]`

Off-Screen Images

Off-Screen Images

Off-Screen Images

- Do not use –`[NSImage lockFocus]`
  - It will flatten your image into a single bitmap

Off-Screen Images

- Do not use –`[NSImage lockFocus]`
  - It will flatten your image into a single bitmap
- Use the new block-based API

```
+[NSImage imageWithSize:flipped:drawingHandler:]
```

Off-Screen Images

- Do not use –[NSImage lockFocus]
  - It will flatten your image into a single bitmap
- Use the new block-based API

```
+[NSImage imageWithSize:flipped:drawingHandler:]
```

- The code in the block is what you would normally call between -lockFocus and -unlockFocus

```
NSImage *image = [NSImage imageWithSize:NSMakeRect(0, 0, 10, 10)
                                flipped:NO
                         drawingHandler:^BOOL(NSRect dstRect)
                                        {
                                            [[NSColor redColor] set];
                                            NSRectFill(dstRect);
                                            return YES;
                                        }];
```

Off-Screen Images

- Do not use –[NSImage lockFocus]
  - It will flatten your image into a single bitmap
- Use the new block-based API

```
+[NSImage imageWithSize:flipped:drawingHandler:]
```

- The code in the block is what you would normally call between -lockFocus and -unlockFocus

```
NSImage *image = [NSImage imageWithSize:NSMakeRect(0, 0, 10, 10)
                                flipped:NO
                         drawingHandler:^BOOL(NSRect dstRect)
                                        {
                                            [[NSColor redColor] set];
                                            NSRectFill(dstRect);
                                            return YES;
                                        }];
```

Off-Screen Images

- Create and render bitmaps explicitly

```
NSBitmapImageRep *myRep = [[NSBitmapImageRep alloc]
                            initWithBitmapDataPlanes: NULL
                            pixelsWide: width * scaleFactor
                            pixelsHigh: height * scaleFactor
                            bitsPerSample: 8
                            samplesPerPixel: 4
                            hasAlpha: YES
                            isPlanar: NO
                            colorSpaceName: NSCalibratedRGBColorSpace
                            bytesPerRow: 0
                            bitsPerPixel: 0];
[myRep setSize: NSMakeSize(width, height)]; // Communicates DPI
```

Off-Screen Images

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
```

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
```

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
```

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
[[NSColor redColor] set];
```

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
[[NSColor redColor] set];
[NSBezierPath bezierPathWithRect:NSMakeRect(0, 0, 10, 10) fill];
```

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
[[NSColor redColor] set];
[NSBezierPath bezierPathWithRect:NSMakeRect(0, 0, 10, 10) fill];
[NSGraphicsContext restoreGraphicsState];
```

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
[[NSColor redColor] set];
[NSBezierPath bezierPathWithRect:NSMakeRect(0, 0, 10, 10) fill];
[NSGraphicsContext restoreGraphicsState];
```

- `NSGraphicsContext` automatically sets up the context transformation matrix based on the pixel dimensions and size of the destination image rep

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
[[NSColor redColor] set];
[NSBezierPath bezierPathWithRect:NSMakeRect(0, 0, 10, 10) fill];
[NSGraphicsContext restoreGraphicsState];
```

- `NSGraphicsContext` automatically sets up the context transformation matrix based on the pixel dimensions and size of the destination image rep
- You can create and render to multiple bitmap image reps—One for each scale factor or screen

Off-Screen Images

- Use `NSGraphicsContext` to render to a bitmap

```
[NSGraphicsContext saveGraphicsState];
[NSGraphicsContext setCurrentContext:
    [NSGraphicsContext graphicsContextWithBitmapImageRep: bitmapImageRep]];
[[NSColor redColor] set];
[NSBezierPath bezierPathWithRect:NSMakeRect(0, 0, 10, 10) fill];
[NSGraphicsContext restoreGraphicsState];
```

- `NSGraphicsContext` automatically sets up the context transformation matrix based on the pixel dimensions and size of the destination image rep
- You can create and render to multiple bitmap image reps—One for each scale factor or screen

Obsolete NSImage methods

```
-[NSImage compositeToPoint:]
-[NSImage dissolveToPoint:]
```

- These don’t fully respect the context transformation matrix

Obsolete NSImage methods

```
-[NSImage compositeToPoint:]
-[NSImage dissolveToPoint:]
```

- These don't fully respect the context transformation matrix
- If it doesn't begin with "draw" don't use it for drawing

Obsolete NSImage methods

```
-[NSImage compositeToPoint:]
-[NSImage dissolveToPoint:]
```

- These don't fully respect the context transformation matrix
- If it doesn't begin with "draw" don't use it for drawing
- Use `-[NSImage drawInRect:fromRect:operation:fraction:respectFlipped:hints:]` instead

Obsolete NSImage methods

```
-[NSImage compositeToPoint:]
-[NSImage dissolveToPoint:]
```

- These don't fully respect the context transformation matrix
- If it doesn't begin with "draw" don't use it for drawing
- Use `-[NSImage drawInRect:fromRect:operation:fraction:respectFlipped:hints:]` instead

```
[image drawInRect:NSMakeRect(0, 0, 10, 10) fromRect:NSZeroRect
operation:NSCompositeSourceOver fraction:1 respectFlipped:NO hints:nil]
```

OpenGL and High Resolution

OpenGL and High Resolution

OpenGL and High Resolution

- By default, all OpenGL surfaces are created at low resolution

OpenGL and High Resolution

- By default, all OpenGL surfaces are created at low resolution
- Surface resolution is identical to 1x operation for compatibility

OpenGL and High Resolution

- By default, all OpenGL surfaces are created at low resolution
- Surface resolution is identical to 1x operation for compatibility
- Applications must adapt their code to work properly under high resolution

OpenGL and High Resolution

OpenGL and High Resolution

- Request high resolution on a per view basis
  - `[NSView setWantsBestResolutionOpenGLSurface:YES]`

OpenGL and High Resolution

- Request high resolution on a per view basis

```
-[NSView setWantsBestResolutionOpenGLSurface:YES]
```

- Adjust glViewPort code to use correct pixel bounds

```
NSRect pixelBounds = [self convertRectToBacking:[self bounds]];
glViewPort( 0, 0, pixelBounds.size.width, pixelBounds.size.height );
```

OpenGL and High Resolution

- Request high resolution on a per view basis

```
-[NSView setWantsBestResolutionOpenGLSurface:YES]
```

- Adjust glViewPort code to use correct pixel bounds

```
NSRect pixelBounds = [self convertRectToBacking:[self bounds]];
glViewPort( 0, 0, pixelBounds.size.width, pixelBounds.size.height );
```

- Incorporate UI scale into model-view transform if necessary

OpenGL and High Resolution

- Request high resolution on a per view basis

```
-[NSView setWantsBestResolutionOpenGLSurface:YES]
```

- Adjust glViewPort code to use correct pixel bounds

```
NSRect pixelBounds = [self convertRectToBacking:[self bounds]];
glViewPort( 0, 0, pixelBounds.size.width, pixelBounds.size.height );
```

- Incorporate UI scale into model-view transform if necessary
- Update texture resources if needed

Demo

Adapting OpenGL for High Resolution

Chess

OpenGL and High Resolution

Special considerations

OpenGL and High Resolution

Special considerations

- Device dependent geometry

OpenGL and High Resolution

Special considerations

- Device dependent geometry
  - Viewport, scissor, and stencil rectangles are always in pixels

OpenGL and High Resolution

Special considerations

- Device dependent geometry
  - Viewport, scissor, and stencil rectangles are always in pixels
  - Convert inputs to view backing space for consistency across resolutions

OpenGL and High Resolution

Special considerations

- Device dependent geometry
  - Viewport, scissor, and stencil rectangles are always in pixels
  - Convert inputs to view backing space for consistency across resolutions
- Text and graphical user interaction elements may need to be generated and managed at multiple resolutions

OpenGL and High Resolution

Special considerations

- Device dependent geometry
  - Viewport, scissor, and stencil rectangles are always in pixels
  - Convert inputs to view backing space for consistency across resolutions
- Text and graphical user interaction elements may need to be generated and managed at multiple resolutions
- MSAA can be costly with marginal benefit at Retina resolutions

OpenGL and High Resolution

Full screen operation

OpenGL and High Resolution

Full screen operation

- Avoid changing the display mode of the system

OpenGL and High Resolution

Full screen operation

- Avoid changing the display mode of the system
- Create an application window covering the entire screen

OpenGL and High Resolution

Full screen operation

- Avoid changing the display mode of the system
- Create an application window covering the entire screen
  - System will provide optimized context performance

OpenGL and High Resolution

Full screen operation

- Avoid changing the display mode of the system
- Create an application window covering the entire screen
  - System will provide optimized context performance
  - Enables critical system dialogs to present above your content

Backing Coordinate Systems

Backing Coordinate Systems

Coordinate System Conversion	
`-convertRectToBacking:`	NSView, NSWindow, NSScreen
`-convertRectFromBacking:`	NSView, NSWindow, NSScreen

Backing Coordinate Systems

Backing Coordinate Systems

- Key principles about backing coordinate systems

Backing Coordinate Systems

- Key principles about backing coordinate systems
  - Units are pixels

Backing Coordinate Systems

- Key principles about backing coordinate systems
  - Units are pixels
  - Values ascend from lower left to upper right

Backing Coordinate Systems

- Key principles about backing coordinate systems
  - Units are pixels
  - Values ascend from lower left to upper right
  - Integral values are pixel aligned

Backing Coordinate Systems

- Key principles about backing coordinate systems
  - Units are pixels
  - Values ascend from lower left to upper right
  - Integral values are pixel aligned
- No guarantees regarding the relationship between local coordinates and backing coordinates

Backing Coordinate Systems

- Key principles about backing coordinate systems
  - Units are pixels
  - Values ascend from lower left to upper right
  - Integral values are pixel aligned
- No guarantees regarding the relationship between local coordinates and backing coordinates
- Unique for each view, window, and screen

Backing Coordinate Systems

- Key principles about backing coordinate systems
  - Units are pixels
  - Values ascend from lower left to upper right
  - Integral values are pixel aligned
- No guarantees regarding the relationship between local coordinates and backing coordinates
- Unique for each view, window, and screen
  - Always use the same object to convert to and from backing

Core Animation

Core Animation and High Resolution

Custom CALayer content

Core Animation and High Resolution

Custom CALayer content

- Layer bounds and position use points

Core Animation and High Resolution

Custom CALayer content

- Layer bounds and position use points
- Layer `contents` and `contentsScale` must be explicitly managed for sharp results

Core Animation and High Resolution

Custom CALayer content

- Layer bounds and position use points
- Layer `contents` and `contentsScale` must be explicitly managed for sharp results
- Layer `contentsGravity` also affects positioning

Core Animation and High Resolution

Custom CALayer content

- Layer bounds and position use points
- Layer `contents` and `contentsScale` must be explicitly managed for sharp results
- Layer `contentsGravity` also affects positioning
- CGContexts provided to layer delegates already include scaling

Core Animation and High Resolution

Custom CALayer content

- Layer bounds and position use points
- Layer `contents` and `contentsScale` must be explicitly managed for sharp results
- Layer `contentsGravity` also affects positioning
- CGContexts provided to layer delegates already include scaling
  - `layer.contentsScale` must be set correctly!

Core Animation and High Resolution

Custom CALayer content

- Layer bounds and position use points
- Layer `contents` and `contentsScale` must be explicitly managed for sharp results
- Layer `contentsGravity` also affects positioning
- CGContexts provided to layer delegates already include scaling
  - `layer.contentsScale` must be set correctly!
  - No need to change `–drawLayer:inContext:`

Core Animation and High Resolution

NSImage as layer.contents

Core Animation and High Resolution

NSImage as layer.contents

- You can set the contents of a layer to an NSImage

```
layer.contents = [NSImage imageNamed:@"MyImage"]
```

Core Animation and High Resolution

NSImage as layer.contents

- You can set the contents of a layer to an NSImage

```
layer.contents = [NSImage imageNamed:@"MyImage"]
```

- Automatically chooses the best representation for the screen the layer is on
  - Standard resolution screen will use the 1x rep
  - High resolution screen will use the 2x rep

Core Animation and High Resolution

NSImage as layer.contents

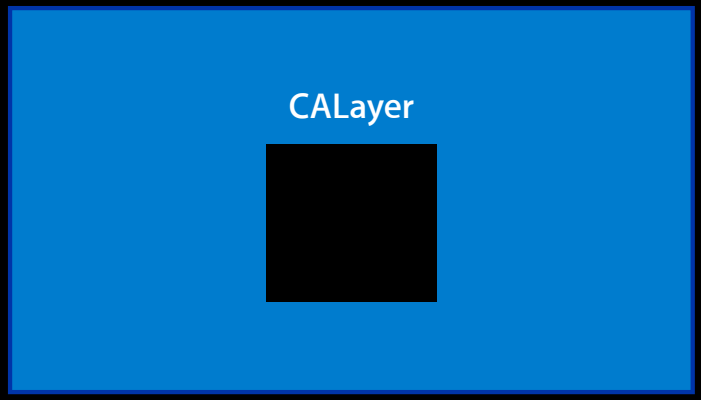
- You can set the contents of a layer to an NSImage

```
layer.contents = [NSImage imageNamed:@"MyImage"]
```

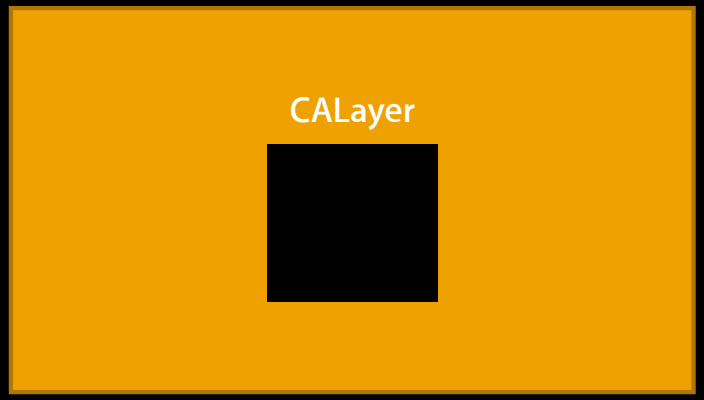
- Automatically chooses the best representation for the screen the layer is on
  - Standard resolution screen will use the 1x rep
  - High resolution screen will use the 2x rep
- There are some edge cases…

NSImage will pick the representation that matches the resolution…

Standard Resolution Display



High Resolution Display

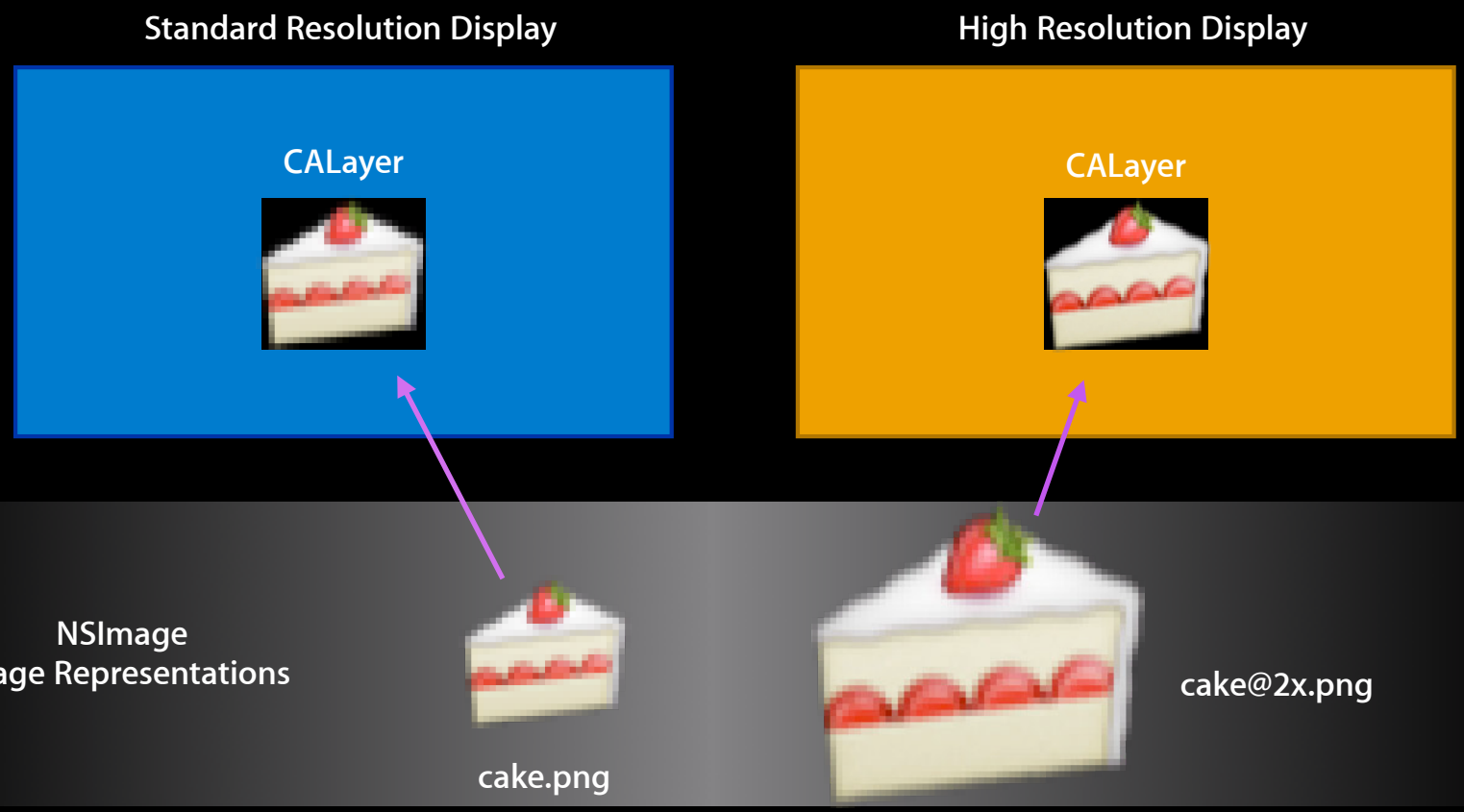


NSImage
Image Representations

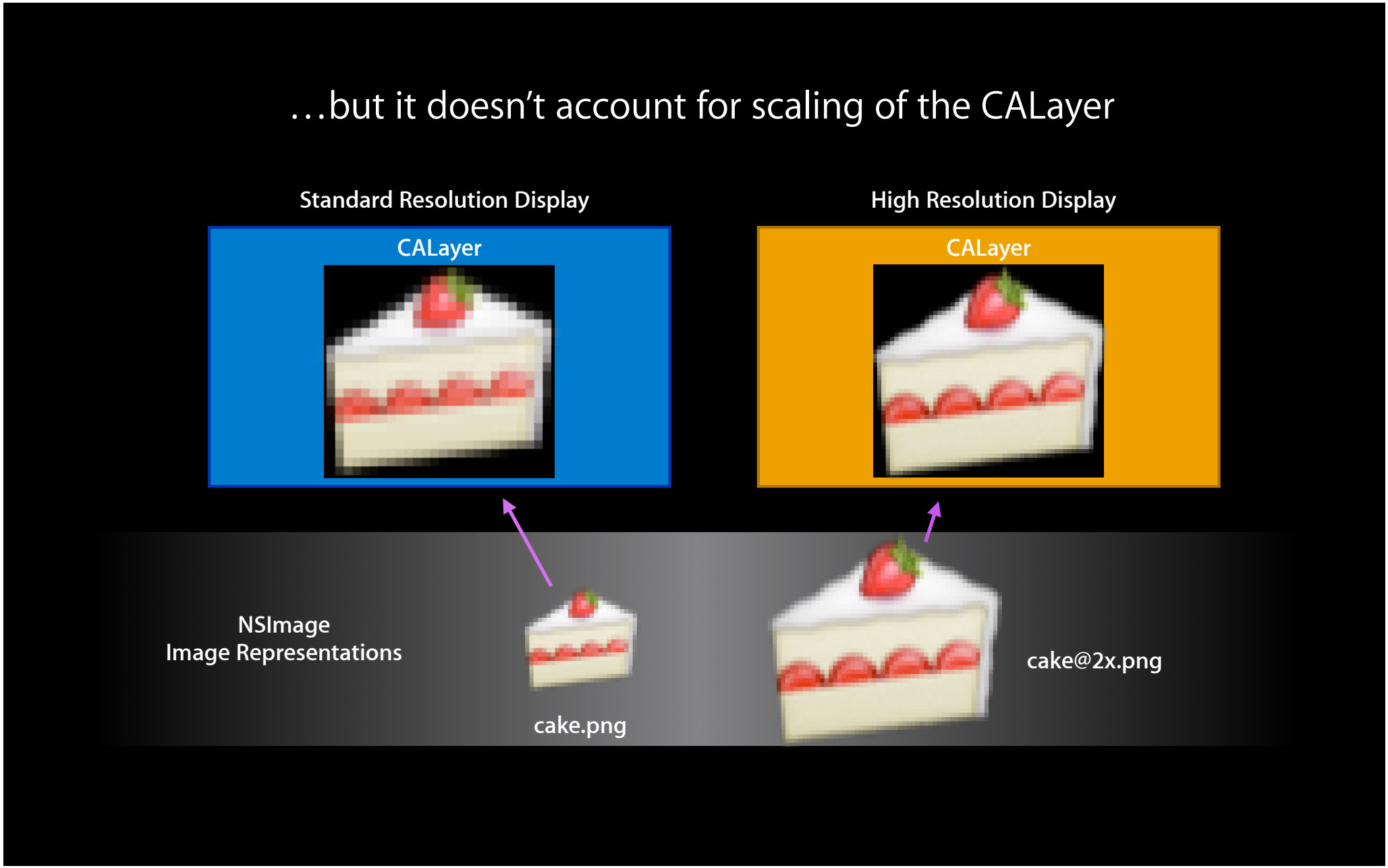
cake.png

cake@2x.png

NSImage will pick the representation that matches the resolution…

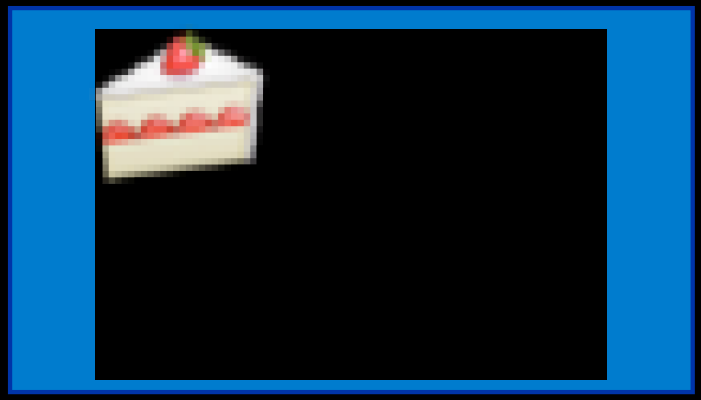


…but it doesn't account for scaling of the CALayer



Contents gravity values that don't resize will yield surprising results

Standard Resolution Display



High Resolution Display

```
layer.contentsGravity = kCAContentsGravityTopLeft
```

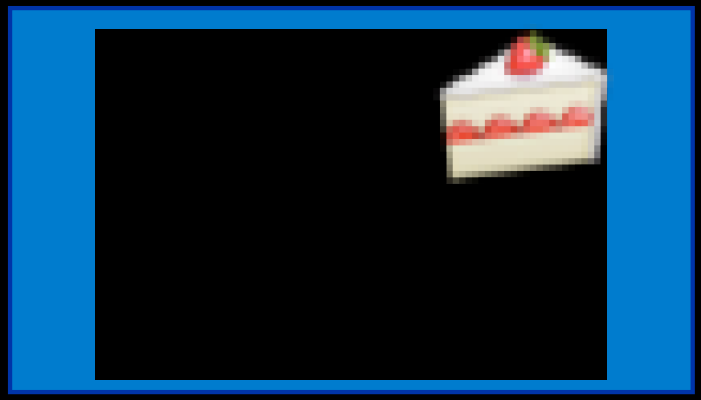
NSImage
Image Representations

cake.png

cake@2x.png

Contents gravity values that don’t resize will yield surprising results

Standard Resolution Display



High Resolution Display

```
layer.contentsGravity = kCAContentsGravityTopRight
```

NSImage
Image Representations

cake.png

cake@2x.png

Core Animation and High Resolution

NSImage as layer.contents

Core Animation and High Resolution

NSImage as layer.contents

- Scales added by layer bounds or transforms are not accounted for during representation selection

Core Animation and High Resolution

NSImage as layer.contents

- Scales added by layer bounds or transforms are not accounted for during representation selection
- Doesn't work for contents gravity other than `kCAContentsGravityResize`, `kCAContentsGravityResizeAspect`, and `kCAContentsGravityResizeFill`

Core Animation and High Resolution

NSImage as layer.contents

- Scales added by layer bounds or transforms are not accounted for during representation selection
- Doesn't work for contents gravity other than `kCAContentsGravityResize`, `kCAContentsGravityResizeAspect`, and `kCAContentsGravityResizeFill`
- For these cases use:
  - `-[NSImage recommendedLayerContentsScale:]`
  - `-[NSImage layerContentsForContentsScale:]`

Core Animation and High Resolution

Managing contents scale

Core Animation and High Resolution

Managing contents scale

- New convenience API (on the layer delegate)

```
-(BOOL)[id<CALayerDelegate> layer:(CALayer *)layer
       shouldInheritContentsScale:(CGFloat)scale
                        fromWindow:(NSWindow *)window]
```

Core Animation and High Resolution

Managing contents scale

- New convenience API (on the layer delegate)

```
-(BOOL)[id<CALayerDelegate> layer:(CALayer *)layer
       shouldInheritContentsScale:(CGFloat)scale
                        fromWindow:(NSWindow *)window]
```

- If you return YES from this, you also need to implement `displayLayer:` or `drawLayer:inContext:`

Core Animation and High Resolution

Managing contents scale

- New convenience API (on the layer delegate)

```
-(BOOL)[id<CALayerDelegate> layer:(CALayer *)layer
       shouldInheritContentsScale:(CGFloat)scale
                        fromWindow:(NSWindow *)window]
```

- If you return YES from this, you also need to implement `displayLayer:` or `drawLayer:inContext:`
- The delegate method is not invoked when a sublayer is added

Core Animation and High Resolution

Managing contents scale

- New convenience API (on the layer delegate)

```
-(BOOL)[id<CALayerDelegate> layer:(CALayer *)layer
       shouldInheritContentsScale:(CGFloat)scale
                        fromWindow:(NSWindow *)window]
```

- If you return YES from this, you also need to implement `displayLayer:` or `drawLayer:inContext:`
- The delegate method is not invoked when a sublayer is added
- When a `CALayer` is created, its `contentsScale` should be synchronized to backing scale of window it is contained in

Demo

Managing Custom CALayer Content

Responding to Resolution Changes

Responding to Resolution Changes

Responding to Resolution Changes

- Display size and resolution can change at any time!
  - Hot plug to external display
  - Mirroring to projector
  - Extended desktop

Responding to Resolution Changes

- Display size and resolution can change at any time!
  - Hot plug to external display
  - Mirroring to projector
  - Extended desktop
- Just because the machine has Retina doesn't mean software does not need to worry about 1x

Responding to Resolution Changes

- Display size and resolution can change at any time!
  - Hot plug to external display
  - Mirroring to projector
  - Extended desktop
- Just because the machine has Retina doesn't mean software does not need to worry about 1x
- Windows dragged between displays will change resolution automatically

Responding to Resolution Changes

Notifications

Responding to Resolution Changes

Notifications

- `NSWindow` will adjust backing resolution to match associated `NSScreen`

Responding to Resolution Changes

Notifications

- `NSWindow` will adjust backing resolution to match associated `NSScreen`
  - Off-screen windows get "highest" available resolution by default

Responding to Resolution Changes

Notifications

- `NSWindow` will adjust backing resolution to match associated `NSScreen`
  - Off-screen windows get "highest" available resolution by default
- Notification sent when window becomes associated with a new screen

Responding to Resolution Changes

Notifications

- `NSWindow` will adjust backing resolution to match associated `NSScreen`
  - Off-screen windows get "highest" available resolution by default
- Notification sent when window becomes associated with a new screen

```
NSWindowDidChangeBackingPropertiesNotification
```

Responding to Resolution Changes

Notifications

- `NSWindow` will adjust backing resolution to match associated `NSScreen`
  - Off-screen windows get "highest" available resolution by default
- Notification sent when window becomes associated with a new screen

```
NSWindowDidChangeBackingPropertiesNotification
-[id<NSWindowDelegate> windowDidChangeBackingProperties:]
```

Responding to Resolution Changes

Views

- Also new API on `NSView` (for subclassers)

```
-[NSView viewDidChangeBackingProperties]
```

- Called when the view is added to a window, or when the window scale factor or colorspace changes

```
- (void)viewDidChangeBackingProperties {
    [super viewDidChangeBackingProperties];
    self.layer.contentsScale =
        [image recommendedLayerContentsScale:self.window.backingScale];
    self.layer.contents =
        [image layerContentsForContentsScale:self.layer.contentsScale];
}
```

- Properties do not change when a view is removed from a window
- Initial properties for a view reflect the highest resolution screen

Text Rendering in High Resolution

Aki "I ⍰ Unicode" Inoue
Cocoa Engineer

What's New in 10.8

What's New in 10.8

No New API!

What's New in 10.8

Screen Fonts

Deprecated

Screen Fonts

Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

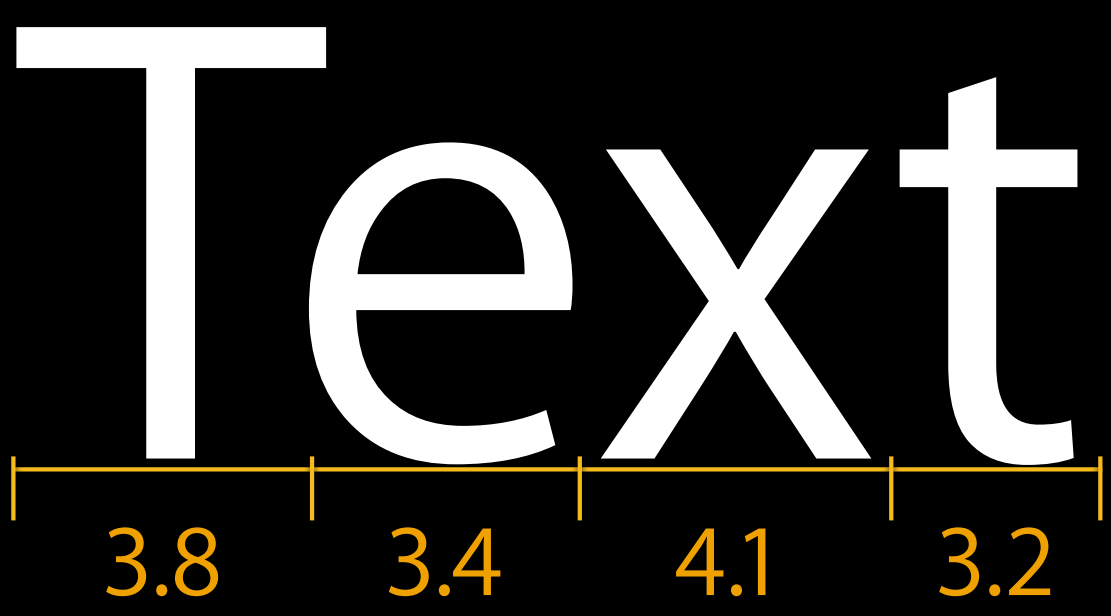
Screen Fonts Explained

“Base” font instance

Text

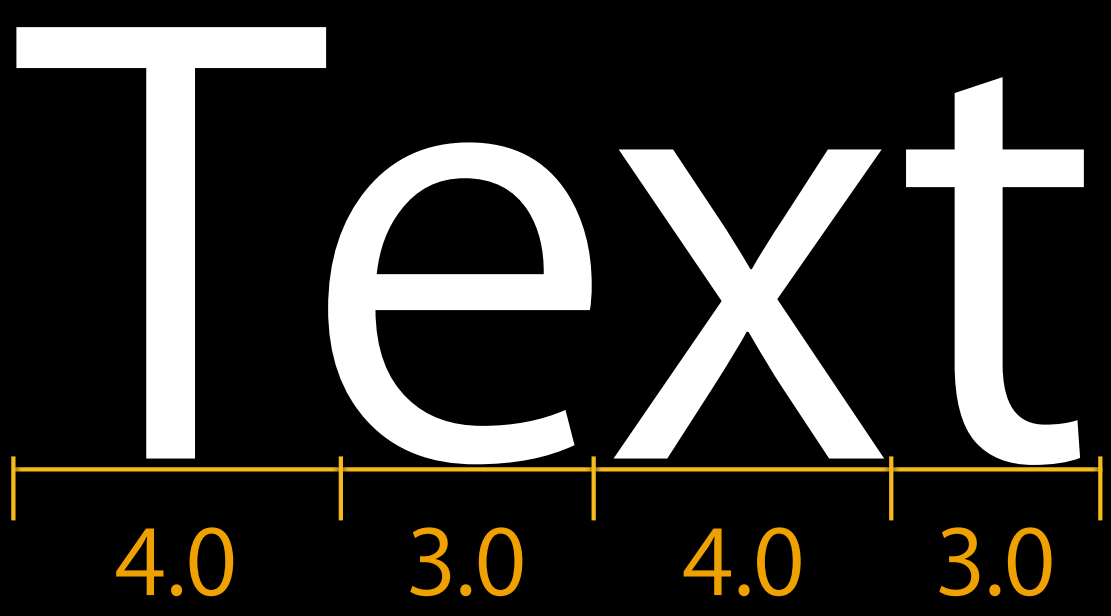
Screen Fonts Explained

“Base” font instance



Screen Fonts Explained

Screen font instance



Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

Screen Fonts Explained

- Integer glyph-spacing
- Sharp rendering with low resolution display
- Quartz rendering time bitmap caching
- Work nicely with hand-tuned bitmap fonts

- Uneven glyph-spacing
- Incompatible with kerning and ligature
- Inconsistent and non-linear width

Floating-Point Advancement Fonts

Ideal glyph advancement

Floating-Point Advancement Fonts

Ideal glyph advancement

- Utilizing the higher pixel density

Floating-Point Advancement Fonts

Ideal glyph advancement

- Utilizing the higher pixel density
- Allowing kerning and ligature in smaller point sizes

Floating-Point Advancement Fonts

Ideal glyph advancement

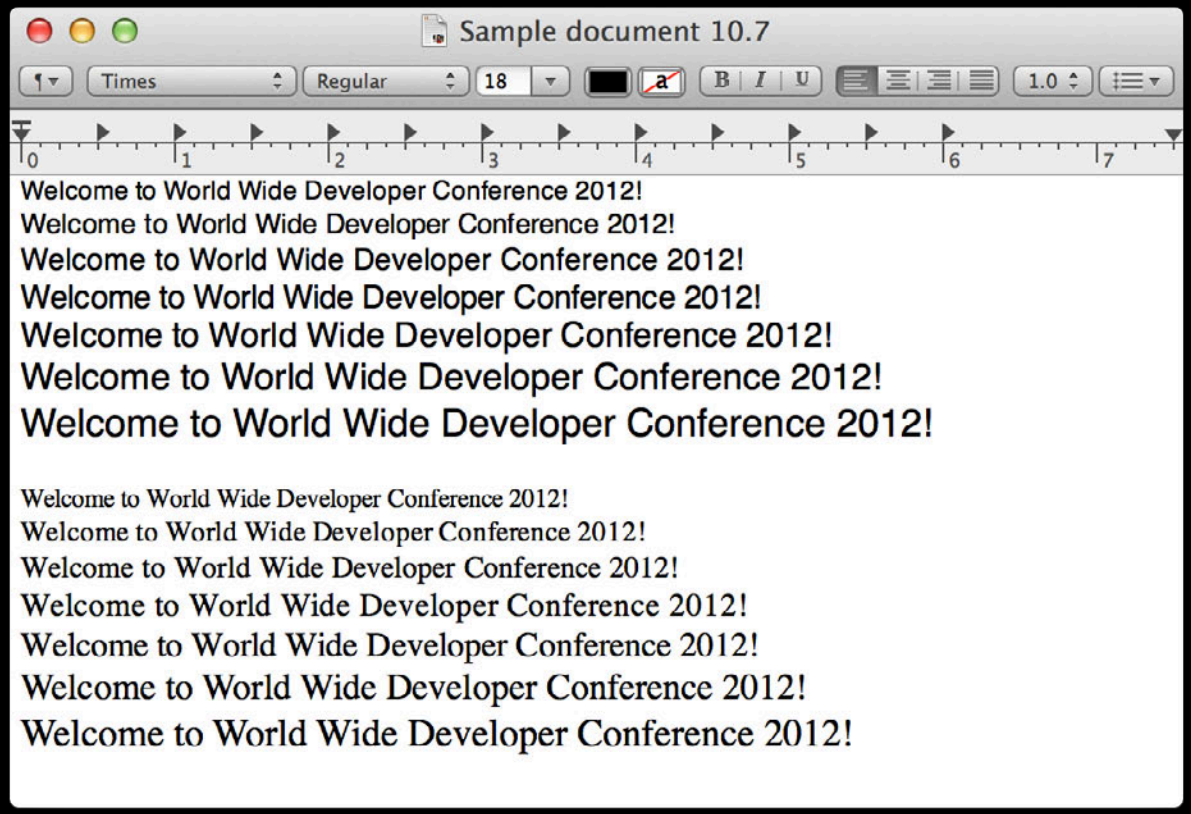
- Utilizing the higher pixel density
- Allowing kerning and ligature in smaller point sizes
- Layout as the font designer intended

Floating-Point Advancement Fonts

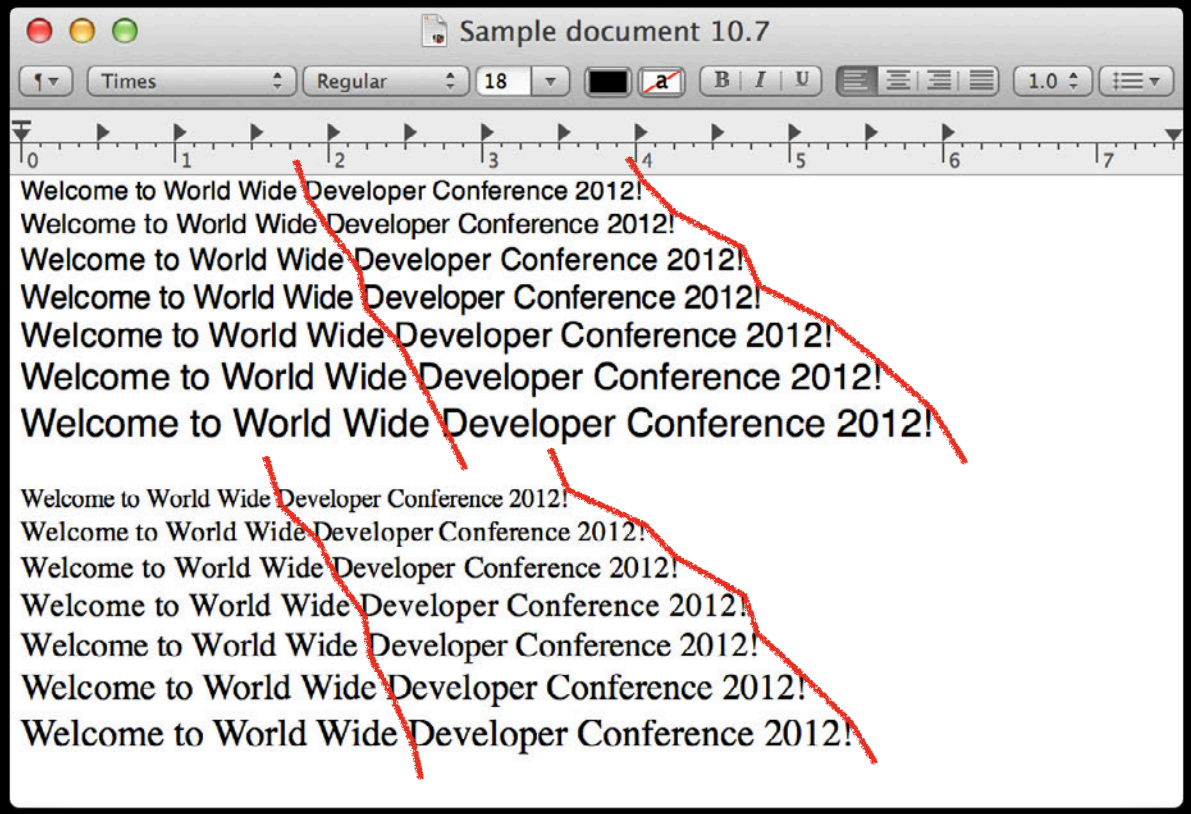
Ideal glyph advancement

- Utilizing the higher pixel density
- Allowing kerning and ligature in smaller point sizes
- Layout as the font designer intended
- Uniform transformation in scaled coordinates

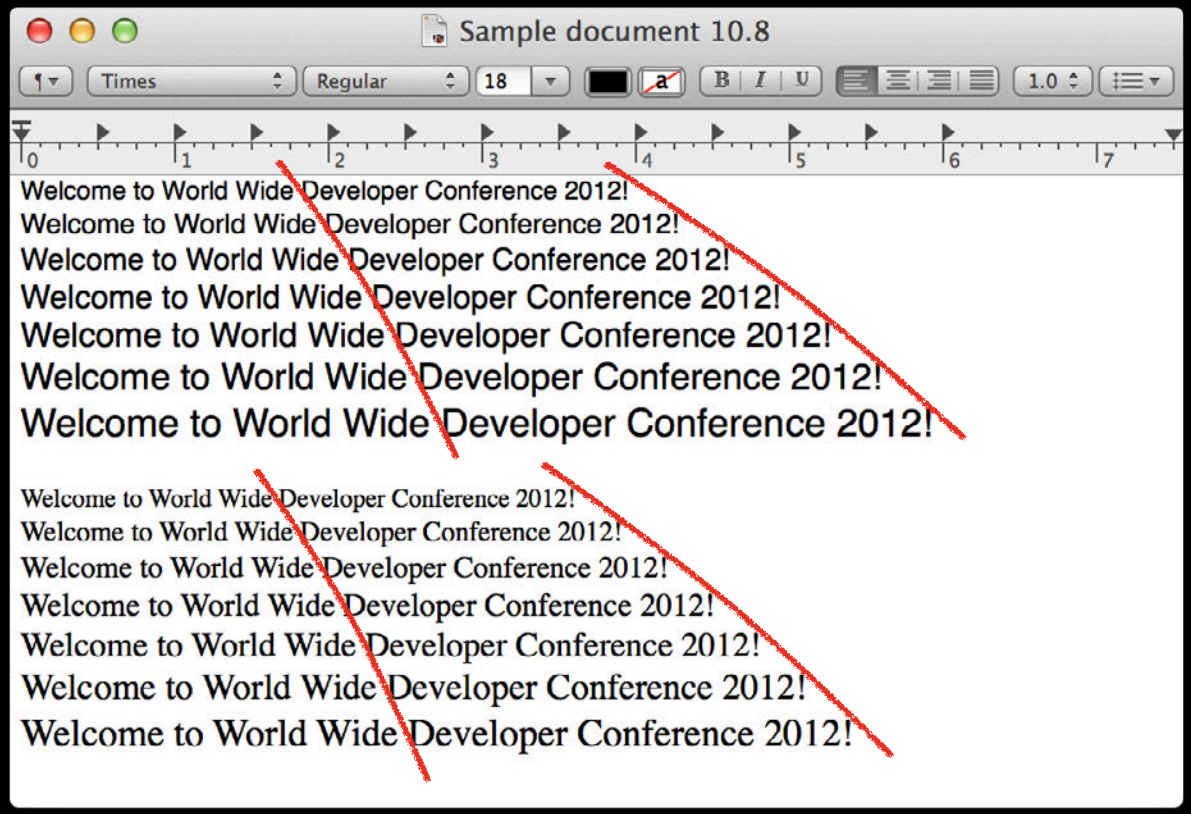
10.7 Text Layout



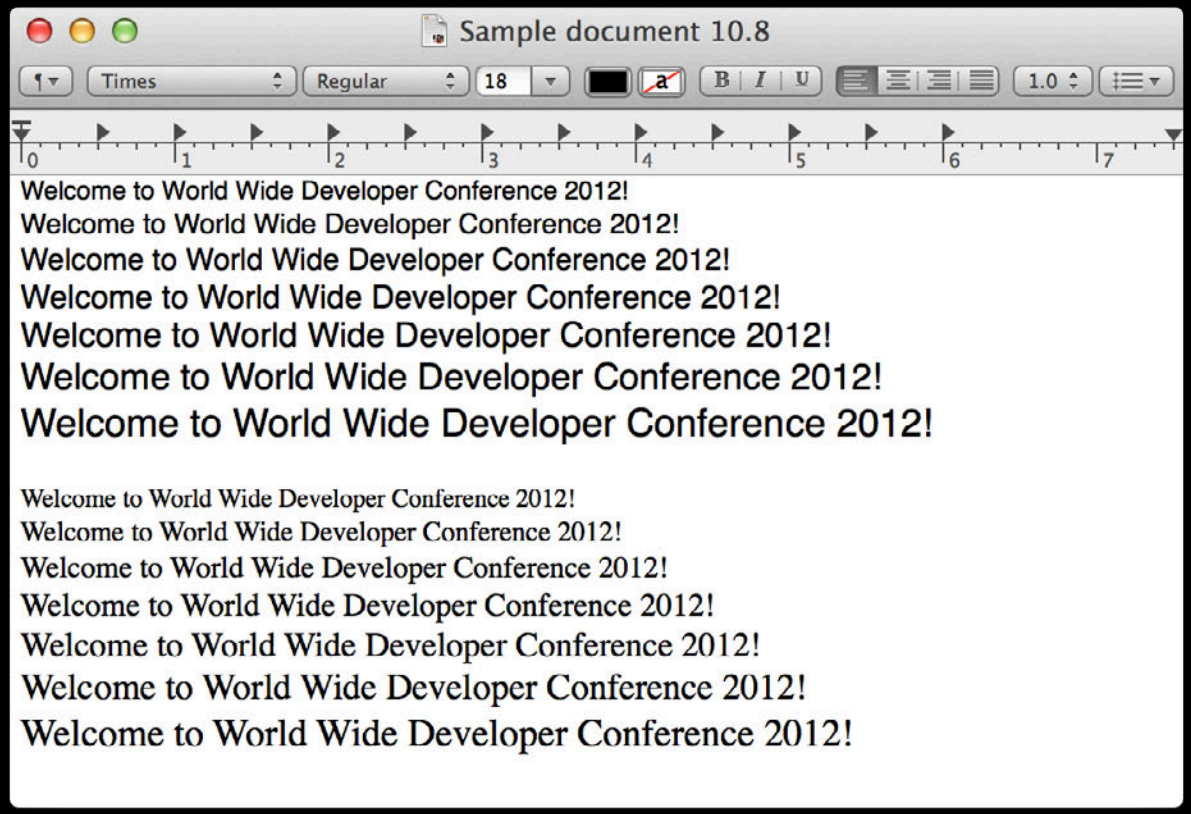
10.7 Text Layout



10.8 Text Layout



10.8 Text Layout



10.8 Text Layout

Layout Comparison

10.7

10.8

Layout Comparison

10.7

10.8

Text System API

- NSLayoutManager
- NSStringDrawing
- NSCell

Text System API

- NSLayoutManager

- NSStringDrawing
- NSCell

Text System API

- Document contents

  `NSLayoutManager`

- User Interface Elements

  `NSStringDrawing`
  `NSCell`

Screen Font API

- Document contents
  - `[NSLayoutManager usesScreenFonts]`
- User Interface Elements

  `NSStringDrawingDisableScreenFontSubstitution`

Screen Font API

- Document contents
  - `[NSLayoutManager usesScreenFonts]`
- User Interface Elements

  `NSStringDrawingDisableScreenFontSubstitution`

  for `-drawWithRect:options:` and `-boundingRectWithSize:options:`

Screen Font API

10.7 default setting

- Document contents

  `-[NSLayoutManager usesScreenFonts] = YES`

- User Interface Elements

  `NSStringDrawingDisableScreenFontSubstitution = NO`

Screen Font API

10.8 default setting

- Document contents

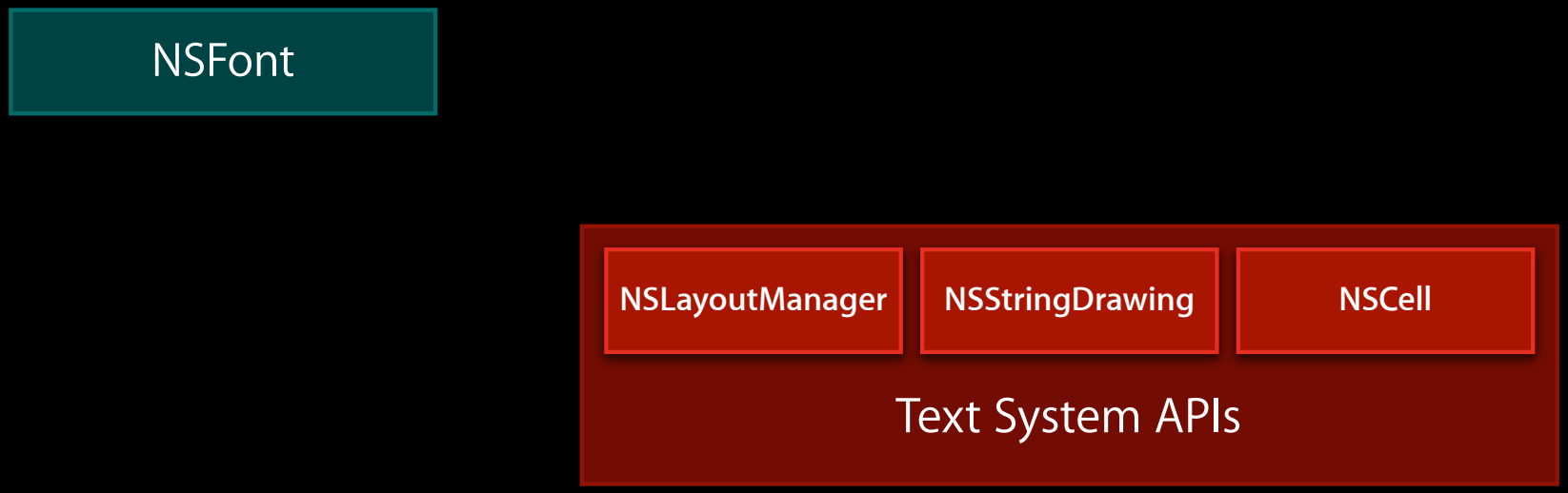
  `-[NSLayoutManager usesScreenFonts] = NO`

- User Interface Elements

  `NSStringDrawingDisableScreenFontSubstitution = YES`

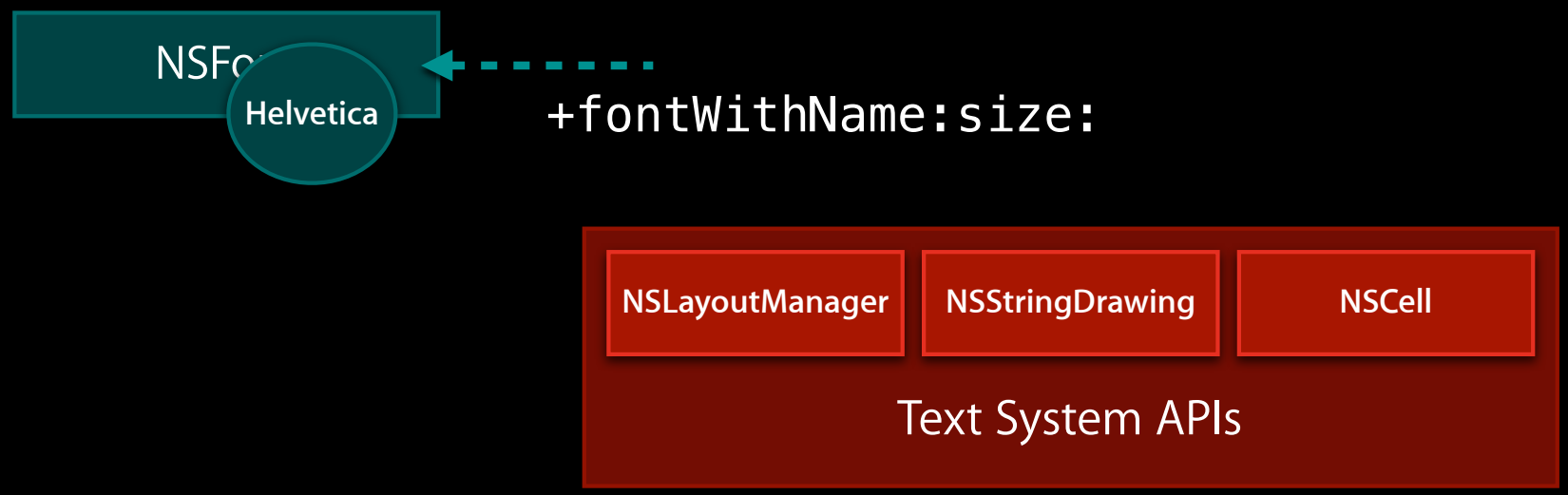
Screen Font Substitution

Anatomy of screen fonts



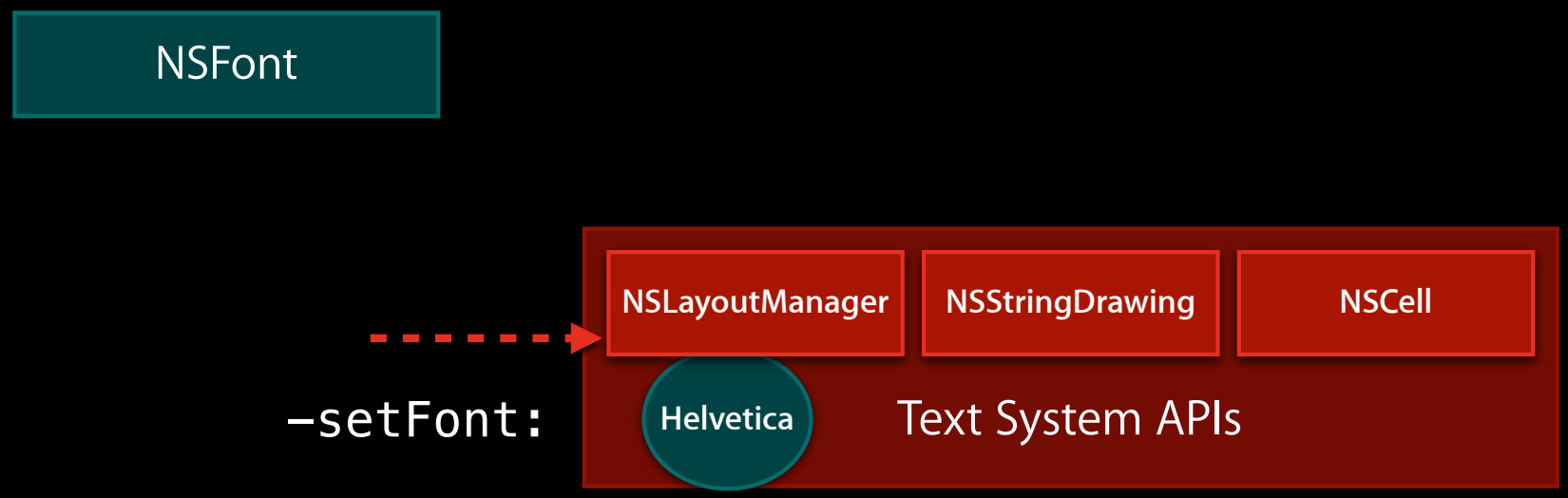
Screen Font Substitution

Anatomy of screen fonts



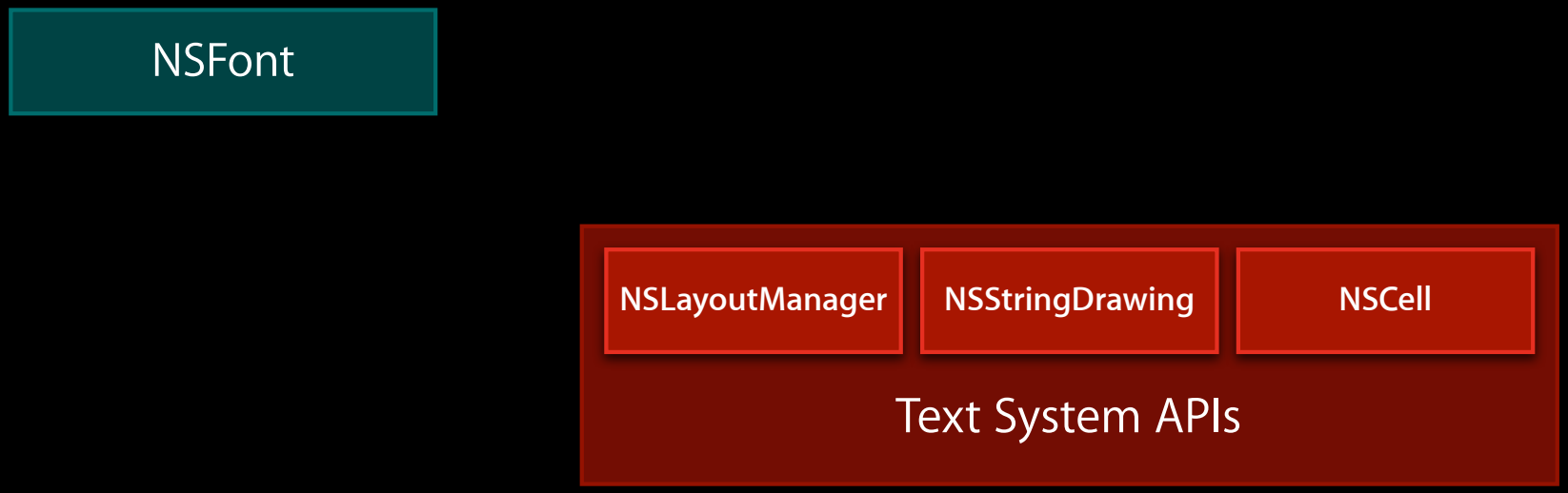
Screen Font Substitution

Anatomy of screen fonts



Screen Font Substitution

Anatomy of screen fonts



Screen Font Substitution

Anatomy of screen fonts

Screen Font Substitution

Anatomy of screen fonts

- Dynamically swaps with screen font behind the scene

Screen Font Substitution

Anatomy of screen fonts

- Dynamically swaps with screen font behind the scene
  - `[NSLayoutManager substituteFontForFont:]`
  - `[NSFont screenFont]`

Screen Font Substitution

Anatomy of screen fonts

- Dynamically swaps with screen font behind the scene

```
-[NSLayoutManager substituteFontForFont:]
-[NSFont screenFont]
```

```
@implementation NSLayoutManager

- (NSFont *)substituteFontForFont:(NSFont)aFont {
  if ([self usesScreenFonts]) aFont = [aFont screenFont];

  return aFont;
}

@end
```

Screen Font Substitution

Anatomy of screen fonts

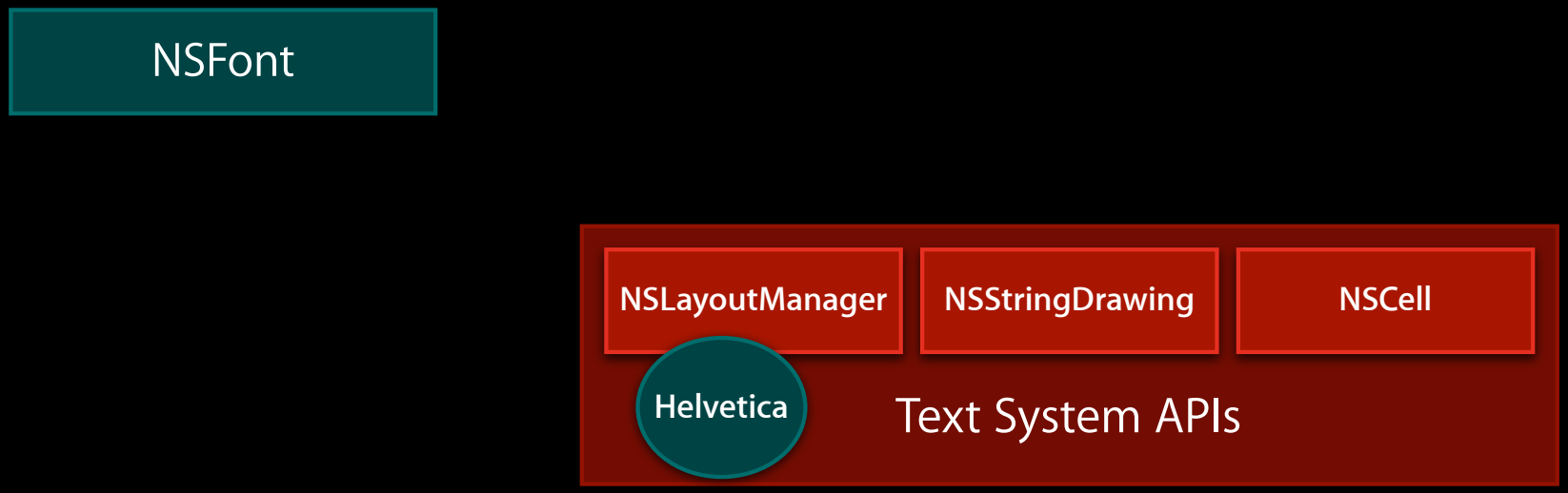
- Dynamically swaps with screen font behind the scene

```
-[NSLayoutManager substituteFontForFont:]
-[NSFont screenFont]
```

```
@implementation NSLayoutManager

- (NSFont *)substituteFontForFont:(NSFont)aFont {
  if ([self usesScreenFonts]) aFont = [aFont screenFont];

  return aFont;
}

@end
```

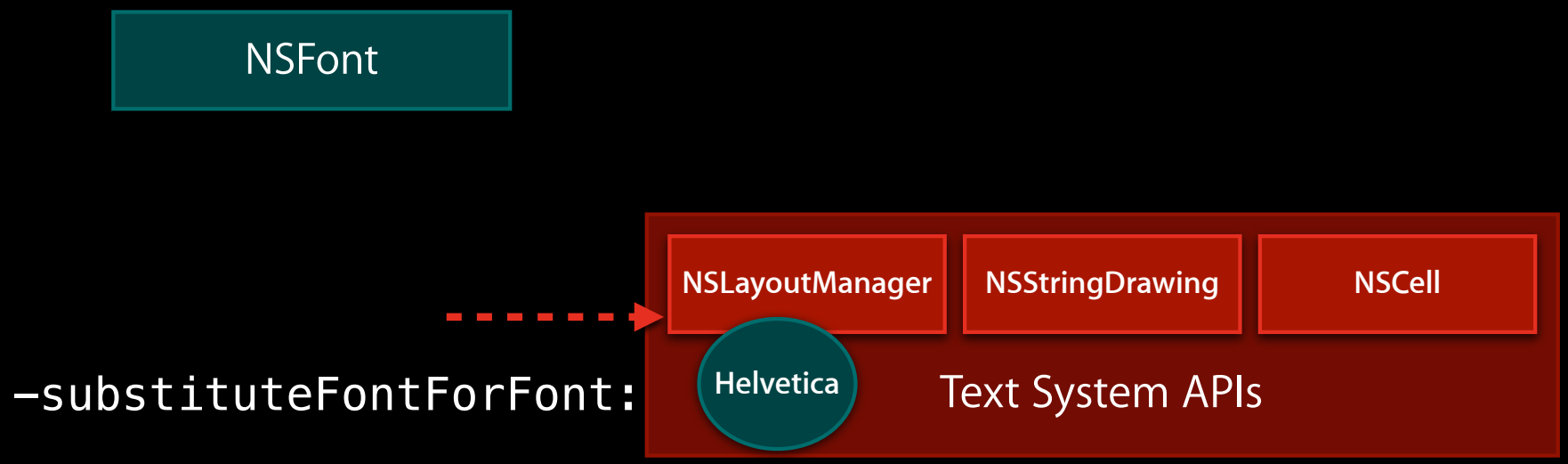
Screen Font Substitution

10.7 behavior



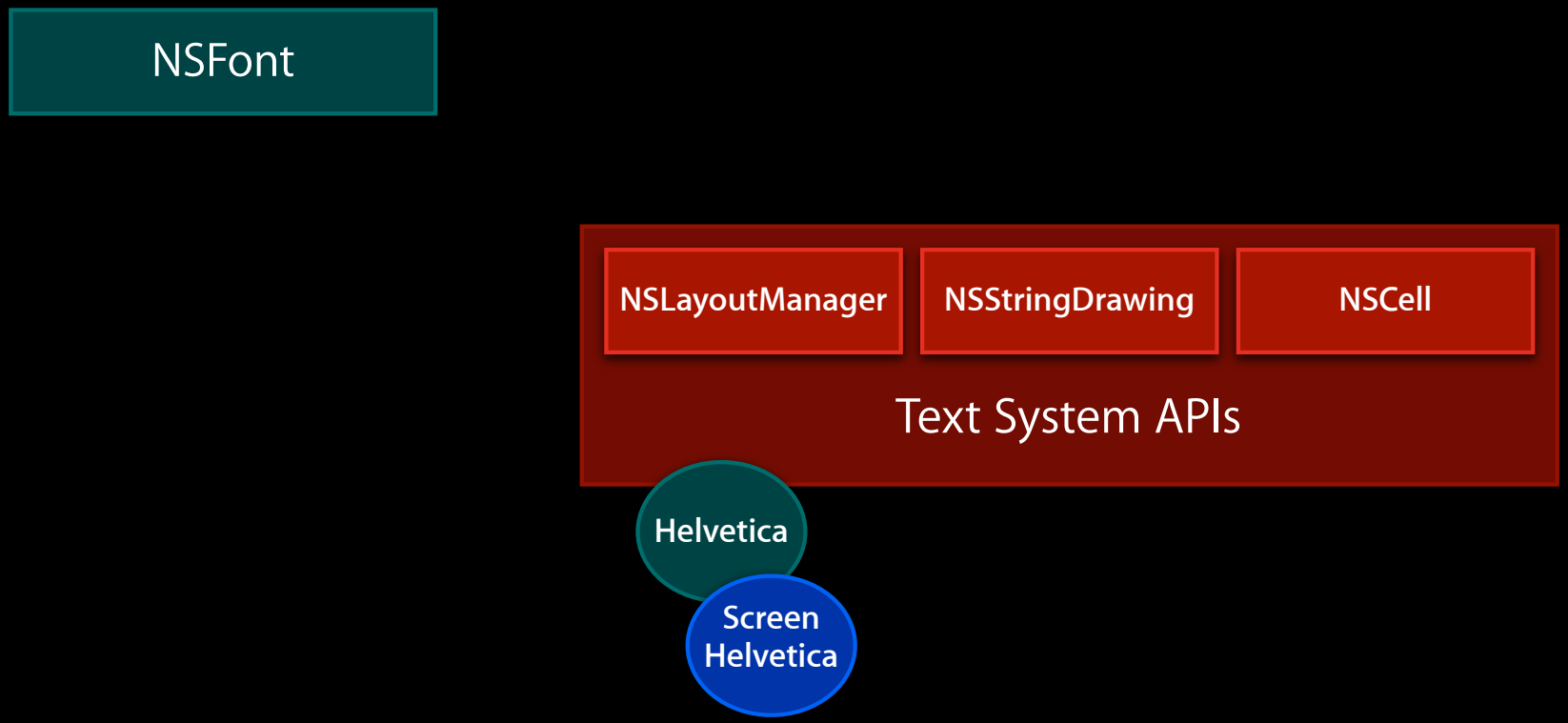
Screen Font Substitution

10.7 behavior



Screen Font Substitution

10.7 behavior

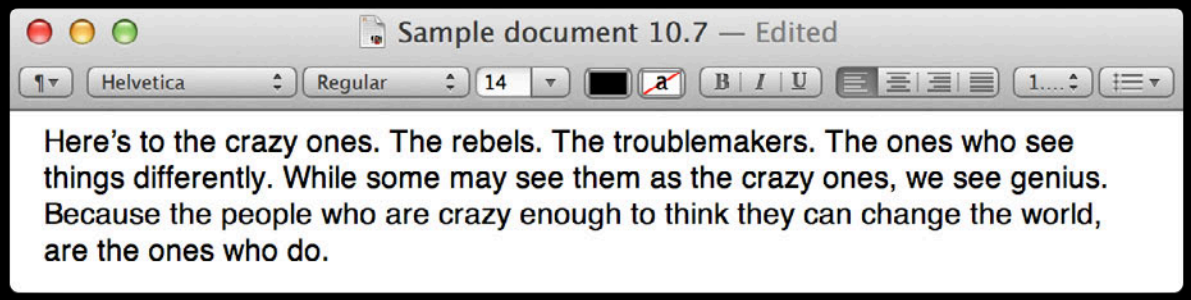


High Resolution Checklist

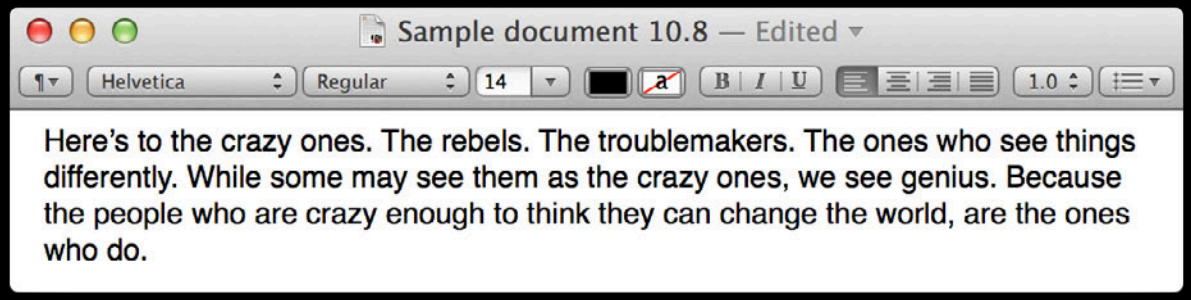
Anything to do?

High Resolution Checklist

Anything to do?



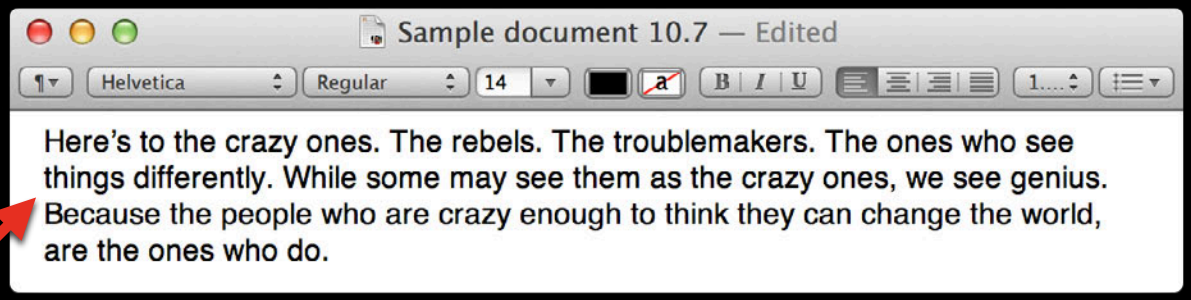
10.7



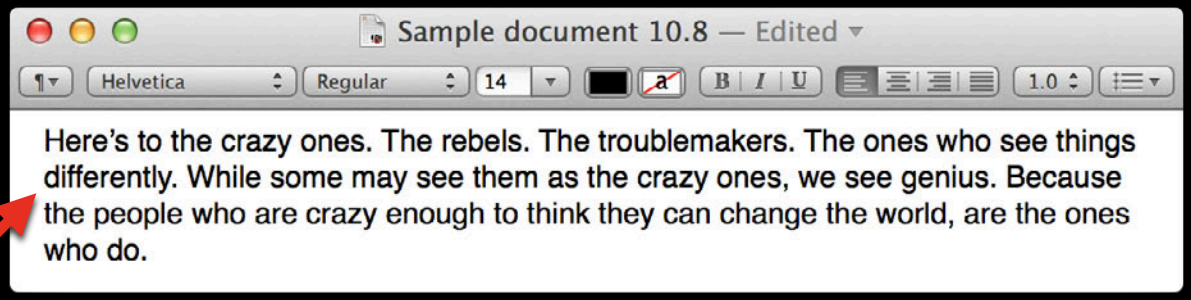
10.8

High Resolution Checklist

Anything to do?



10.7



10.8

Document Compatibility Strategy

- Retain the screen font setting per document
  - `[NSLayoutManager usesScreenFonts]`

  `NSUsesScreenFontsDocumentAttribute`

- Deciding the screen font default setting

  `NSFontDefaultScreenFontSubstitutionEnabled`

Document Compatibility Strategy

- Retain the screen font setting per document
  - `[NSLayoutManager usesScreenFonts]`
  - `NSUsesScreenFontsDocumentAttribute`
- Deciding the screen font default setting
  - `NSFontDefaultScreenFontSubstitutionEnabled`

Document Compatibility Strategy

- Retain the screen font setting per document
  - `[NSLayoutManager usesScreenFonts]`
  - `NSUsesScreenFontsDocumentAttribute`
- Deciding the screen font default setting
  - `NSFontDefaultScreenFontSubstitutionEnabled`

Document Compatibility Strategy

- Retain the screen font setting per document
  - `[NSLayoutManager usesScreenFonts]`

  `NSUsesScreenFontsDocumentAttribute`

- Deciding the screen font default setting

  `NSFontDefaultScreenFontSubstitutionEnabled`
    - 10.8 SDK -> NO
    - Previous SDKs -> YES

Subtle Artwork Alignment Issues

Or, how math conspires against us

Dan Schimpf
Pixel Aligner

Alignment Issues

Alignment Issues

- Situations that worked perfectly well at 1x seem to fail at 2x

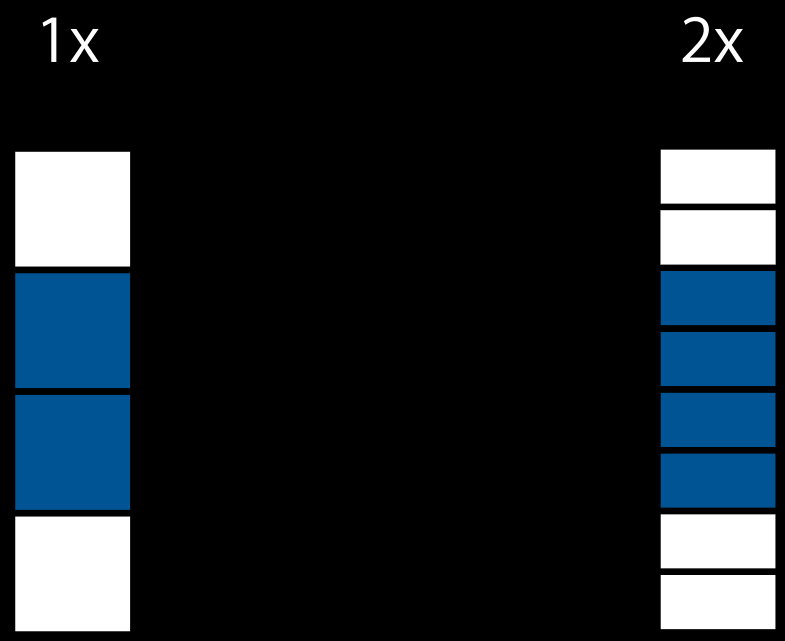
Alignment Issues

- Situations that worked perfectly well at 1x seem to fail at 2x
- Rounding differences cause unsuspecting layout to change at 2x

Alignment Issues

- Situations that worked perfectly well at 1x seem to fail at 2x
- Rounding differences cause unsuspecting layout to change at 2x
- There are no odd pixels at 2x!

Even Inside Even



y=1.0 pt → y=1.0 px

y=1.0 pt → y=2.0 px

Even Inside Even

1x

2x

y=1.0 pt → y=1.0 px

y=1.0 pt → y=2.0 px

Odd Inside Odd

1x

2x

y=1.0 pt → y=1.0 px

y=1.0 pt → y=2.0 px

Odd Inside Odd

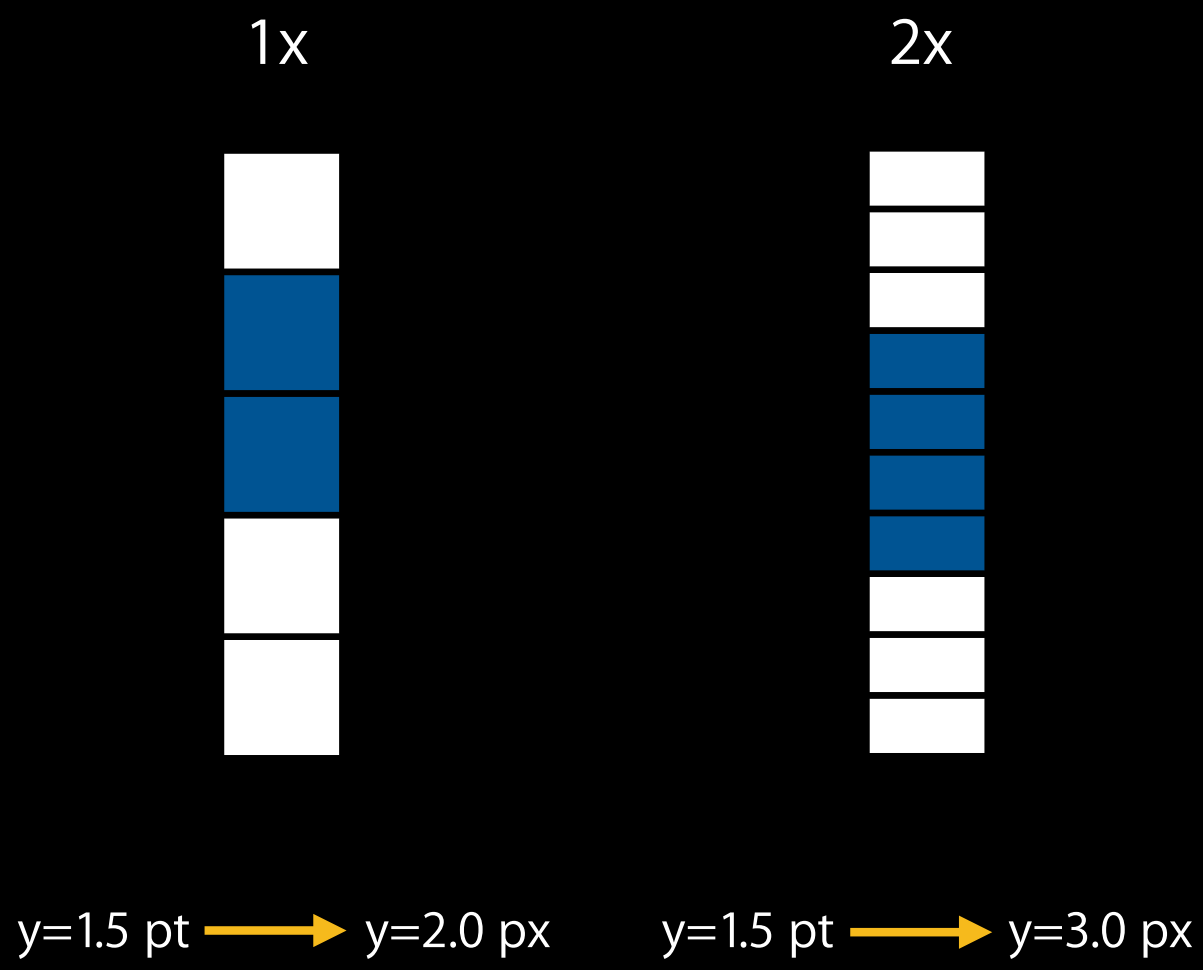
1x

2x

y=1.0 pt → y=1.0 px

y=1.0 pt → y=2.0 px

Even Inside Odd



Even Inside Odd

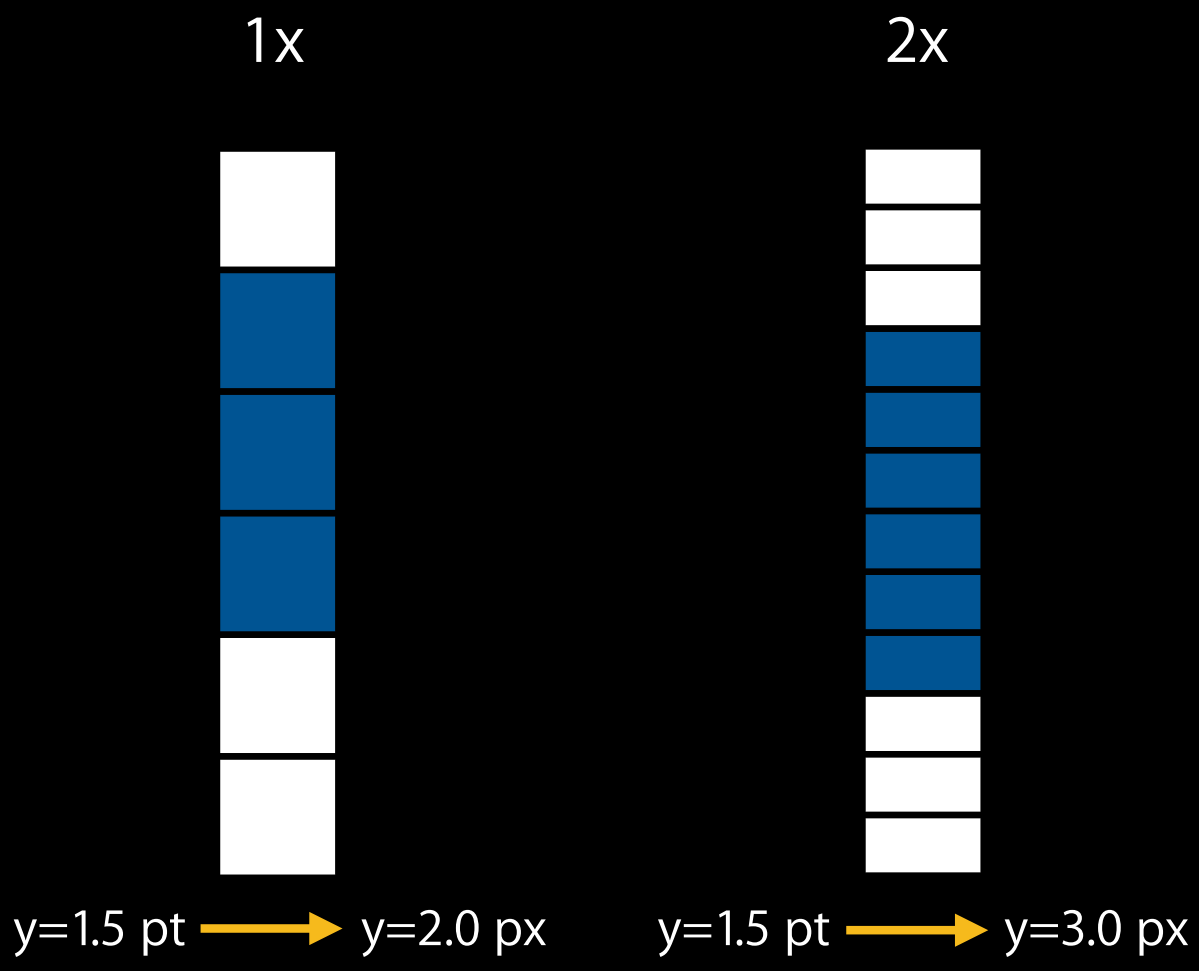
1x

2x

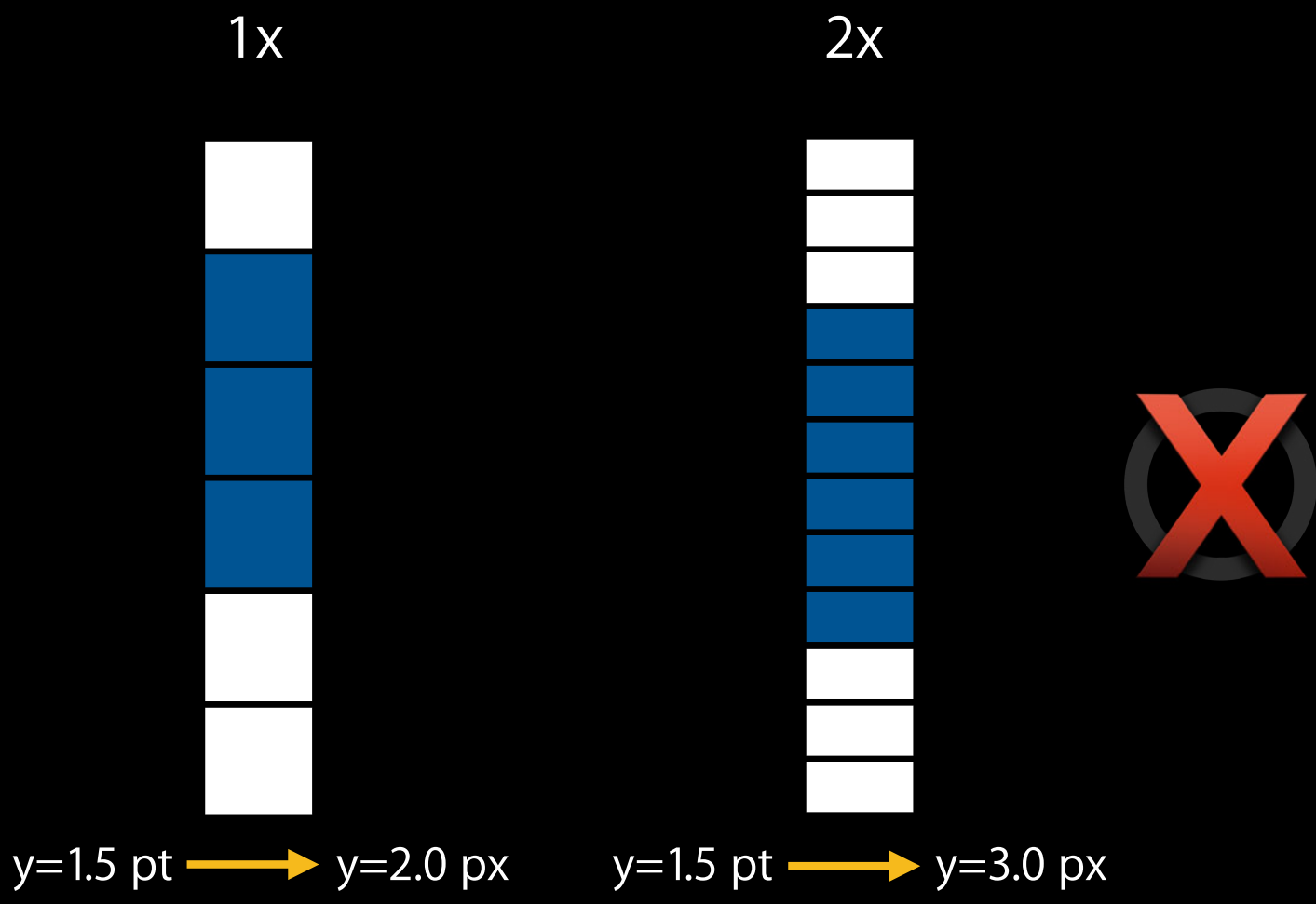
y=1.5 pt → y=2.0 px

y=1.5 pt → y=3.0 px

Odd Inside Even



Odd Inside Even



Recognizing Pixel Shifts

Squinting helps

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint
  - Observe any visual shifts

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint
  - Observe any visual shifts
- If you have one display only:

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint
  - Observe any visual shifts
- If you have one display only:
  - Take screenshots in both modes

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint
  - Observe any visual shifts
- If you have one display only:
  - Take screenshots in both modes
  - Open both in Preview in same window

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint
  - Observe any visual shifts
- If you have one display only:
  - Take screenshots in both modes
  - Open both in Preview in same window
  - Scale the 1x screenshot to 200%

Recognizing Pixel Shifts

Squinting helps

- Easiest way to see issues is to test in both environments
- If you have two displays:
  - Set one to 2x and drag window back and forth
  - The window switches to 2x at the midpoint
  - Observe any visual shifts
- If you have one display only:
  - Take screenshots in both modes
  - Open both in Preview in same window
  - Scale the 1x screenshot to 200%
  - Flip back and forth

Pixel Shifts in the Wild

1x
1,000%

Pixel Shifts in the Wild

2x
500%

Where’s the Problem?

Where’s the Problem?

- This time, maybe it’s the design

Where's the Problem?

- This time, maybe it's the design
- See if you can redesign the 1x appearance to avoid this

Where's the Problem?

- This time, maybe it's the design
- See if you can redesign the 1x appearance to avoid this
- If you can't change, you'll have to tweak just 2x appearance

Fixing the Problem

Fixing the Problem

- Experiment with rounding direction

Fixing the Problem

- Experiment with rounding direction
  - Use `backingAlignedRect:options:`, which provides explicit control

Fixing the Problem

- Experiment with rounding direction
  - Use `backingAlignedRect:options:`, which provides explicit control
- You may have to add 0.5 points explicitly when running at 2x

Fixing the Problem

- Experiment with rounding direction
  - Use `backingAlignedRect:options:`, which provides explicit control
- You may have to add 0.5 points explicitly when running at 2x
  - Do this only if absolutely necessary

Scale Factors

Or, don’t use scale factors

Coordinate Spaces

Coordinate Spaces

- Cocoa is awash in coordinate spaces

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`
  - `NSView`

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`
  - `NSView`
  - `CALayer`

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`
  - `NSView`
  - `CALayer`
  - Bitmap contexts

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`
  - `NSView`
  - `CALayer`
  - Bitmap contexts
  - OpenGL contexts

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`
  - `NSView`
  - `CALayer`
  - Bitmap contexts
  - OpenGL contexts
- By dealing in the correct coordinate space, your code stays clean

Coordinate Spaces

- Cocoa is awash in coordinate spaces
  - `NSWindow`
  - `NSView`
  - `CALayer`
  - Bitmap contexts
  - OpenGL contexts
- By dealing in the correct coordinate space, your code stays clean
- Scale factor is already accounted for in these contexts

Converting Coordinates

Source	Destination	Method
NSView	NSView	convertRect:toView:

Converting Coordinates

Source	Destination	Method
NSView	NSView	convertRect:toView:
NSView	NSWindow	convertRect:toView: with nil view

Converting Coordinates

Source	Destination	Method
NSView	NSView	convertRect:toView:
NSView	NSWindow	convertRect:toView: with nil view
NSWindow	NSScreen	convertRectToScreen:

Converting Coordinates

Source	Destination	Method
NSView	NSView	convertRect:toView:
NSView	NSWindow	convertRect:toView: with nil view
NSWindow	NSScreen	convertRectToScreen:
NSView	CALayer	convertRectToLayer:

Converting Coordinates

Source	Destination	Method
NSView	NSView	convertRect:toView:
NSView	NSWindow	convertRect:toView: with nil view
NSWindow	NSScreen	convertRectToScreen:
NSView	CALayer	convertRectToLayer:
Anything	Backing	convertRectToBacking:

Using Scale Factors Incorrectly

```
- (void)drawRect:(NSRect)dirtyRect {
  CGFloat pixelX = self.frame.origin.x * self.window.backingScaleFactor;
```

Using Scale Factors Incorrectly

```
- (void)drawRect:(NSRect)dirtyRect {
  CGFloat pixelX = self.frame.origin.x * self.window.backingScaleFactor;
```

Recommended Coordinate Handling

```
- (void)drawRect:(NSRect)dirtyRect {
  NSRect pixelRect = [self convertRectToBacking:self.bounds];
  CGFloat pixelX = pixelRect.origin.x;
```

Recommended Coordinate Handling

```
- (void)drawRect:(NSRect)dirtyRect {
  NSRect pixelRect = [self convertRectToBacking:self.bounds];
  CGFloat pixelX = pixelRect.origin.x;
```

Tips

Tips

- Work in points wherever possible

Tips

- Work in points wherever possible
  - Be prepared for fractional points at 2x; they're okay!

Tips

- Work in points wherever possible
  - Be prepared for fractional points at 2x; they're okay!
- Convert coordinates to appropriate space before using them

Tips

- Work in points wherever possible
  - Be prepared for fractional points at 2x; they're okay!
- Convert coordinates to appropriate space before using them
- Never ask for the current scale factor

Tips

- Work in points wherever possible
  - Be prepared for fractional points at 2x; they're okay!
- Convert coordinates to appropriate space before using them
- Never ask for the current scale factor
  - If you absolutely need to, use your current window

Tips

- Work in points wherever possible
  - Be prepared for fractional points at 2x; they're okay!
- Convert coordinates to appropriate space before using them
- Never ask for the current scale factor
  - If you absolutely need to, use your current window
  - If you have no window or screen, it might be time to rethink your design

Capturing On-Screen Content

Patrick Heynen

Capturing On-Screen Content

Creating images of your App's user interface

Capturing On-Screen Content

Creating images of your App's user interface

- Typically used to temporarily cache expensive drawing
  - Animations
  - Drag images

Capturing On-Screen Content

Creating images of your App's user interface

- Typically used to temporarily cache expensive drawing
  - Animations
  - Drag images
- You are creating bitmaps (indirectly)

Capturing On-Screen Content

- Different techniques for different goals
  - Windows and Views vs. Displays

Capturing View Hierarchies

Capturing View Hierarchies

- Create bitmap to use as backing store for your capture

```
-(NSBitmapImageRep*)bitmapImageRepForCachingDisplayInRect:(NSRect)rect
```

Capturing View Hierarchies

- Create bitmap to use as backing store for your capture

```
-(NSBitmapImageRep*)bitmapImageRepForCachingDisplayInRect:(NSRect)rect
```

- Redraw view into new bitmap representation

```
-(void)cacheDisplayInRect:(NSRect)rect
          toBitmapImageRep:(NSBitmapImageRep*)bitmapImageRep
```

Capturing View Hierarchies

- Create bitmap to use as backing store for your capture

```
-(NSBitmapImageRep*)bitmapImageRepForCachingDisplayInRect:(NSRect)rect
```

- Redraw view into new bitmap representation

```
-(void)cacheDisplayInRect:(NSRect)rect
          toBitmapImageRep:(NSBitmapImageRep*)bitmapImageRep
```

- Create an NSImage

```
NSImage *captureImage = [[NSImage alloc] initWithSize:rect.size];
[captureImage addRepresentation:capturedBitmapRep];
```

Special Considerations

- Resolution of capture images will match original window

Special Considerations

- Resolution of capture images will match original window
- Resolution of off-screen view captures will match highest available screen

Special Considerations

- Resolution of capture images will match original window
- Resolution of off-screen view captures will match highest available screen
- Single resolution only!

Capturing Screen Content

Capturing Screen Content

- Need to use Quartz Display Services

```
CGImageRef CGDisplayCreateImageForRect(CGDirectDisplayID display,
  CGRect rect)
```

Capturing Screen Content

- Need to use Quartz Display Services

```
CGImageRef CGDisplayCreateImageForRect(CGDirectDisplayID display,
  CGRect rect)
```

- Calculate image size in points before constructing NSImage

```
NSSize sizeInPixels = NSMakeSize(CGImageGetWidth(cgImage),
                                 CGImageGetHeight(cgImage));

NSSize sizeInPoints = [screen convertSizeFromBacking:sizeInPixels;
NSImage *screenImage = [[NSImage alloc] initWithCGImage:cgImage
                                                 size:sizeInPoints];
```

Capturing Screen Content

- Use `NSScreen deviceDescription` to get Quartz Display ID

```
NSNumber *screenNumber = [[aScreen deviceDescription]
                                   objectForKey:@"NSScreenNumber"];
CGDirectDisplayID displayID = [screenNumber intValue];
```

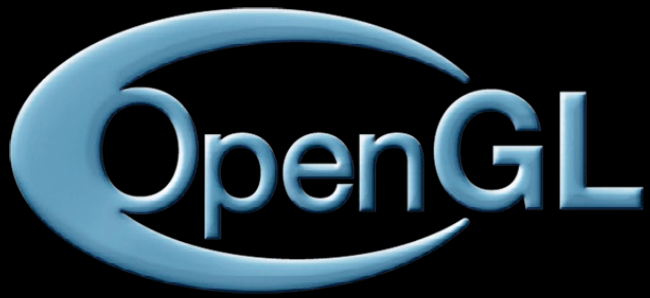
App Performance Under High Resolution

How to think about it

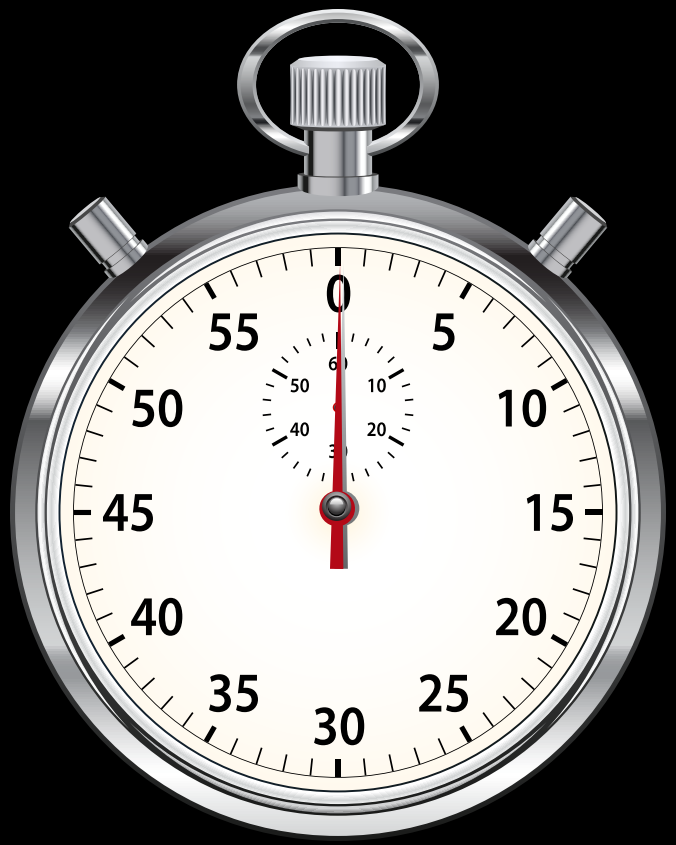
Your application will be processing 4–7x the amount of pixels under high resolution

Don’t despair, the hardware is there

Make sure your app leverages system graphics technologies as much as possible



Be aware of time/space trade-offs



More Information

Jake Behrens
UI Frameworks Evangelist
behrens@apple.com

Documentation
High Resolution Guidelines for OS X
http://developer.apple.com/

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Introduction to High Resolution on OS X	Presidio Wednesday 9:00AM
Delivering Web Content on High Resolution Displays	Nob Hill Wednesday 11:30AM
Advances in OpenGL and OpenGL ES	Pacific Heights Wednesday 2:00PM

Labs

Cocoa and XPC Lab	Essentials Lab A Friday 10:15AM

Summary

Summary

- Avoid using bitmaps when possible
  - Use the new `NSImage` block-based API

Summary

- Avoid using bitmaps when possible
  - Use the new `NSImage` block-based API
- Opt-in to high resolution OpenGL

  Use `setWantsBestResolutionOpenGLSurface:`

  Remember calls like `glViewPort` need to be in pixels!

Summary

- Avoid using bitmaps when possible
  - Use the new `NSImage` block-based API
- Opt-in to high resolution OpenGL

  Use `setWantsBestResolutionOpenGLSurface:`

  Remember calls like `glViewPort` need to be in pixels!
- Be aware of the contentsScale property when using CA layers

  Use the new `layer:shouldInheritContentsScale:fromWindow:` delegate method

Summary

- Avoid using bitmaps when possible
  - Use the new `NSImage` block-based API
- Opt-in to high resolution OpenGL

  Use `setWantsBestResolutionOpenGLSurface:`

  Remember calls like `glViewPort` need to be in pixels!
- Be aware of the contentsScale property when using CA layers

  Use the new `layer:shouldInheritContentsScale:fromWindow:` delegate method
- Screen fonts are deprecated

Summary

- Avoid using bitmaps when possible
  - Use the new `NSImage` block-based API
- Opt-in to high resolution OpenGL

  Use `setWantsBestResolutionOpenGLSurface:`

  Remember calls like `glViewPort` need to be in pixels!
- Be aware of the contentsScale property when using CA layers

  Use the new `layer:shouldInheritContentsScale:fromWindow:` delegate method
- Screen fonts are deprecated
- Pixel alignment may produce visually different results in high resolution

WWDC2012