Working with Wide Color

Understanding and optimizing for Wide Color Gamut Displays

Session 712

Justin Stoyles Graphics and Media
Patrick Heynen Cocoa Frameworks
Steve Holt UIKit
Agenda

Core Color Concepts

What is wide color and why does it matter?

Adapting your content workflow

Implications on app colors

Optimizing your app drawing for wide color displays
Transforming color on macOS and iOS
Brief Overview

ColorSync
International Color Consortium (ICC)
Built in to OS X
Brief Overview

Retina Display
Brief Overview
Core Color Concepts
Core Color Concepts

color space
Color Channels
Color Primaries
Color Gamut
Core Color Concepts

color space
Color Channels
Color Primaries
Color Gamut
Core Color Concepts

color space
Color Channels
Color Primaries
Color Gamut
Core Color Concepts

color space
Color Channels
Color Primaries
Color Gamut
Core Color Concepts

color space
Color Channels
Color Primaries
Color Gamut

RGB {0.0, 0.0, 0.0} = Black
RGB {1.0, 1.0, 1.0} = White
RGB {1.0, 0.0, 0.0} = Red
Core Color Concepts

color space
Color Channels
Color Primaries
Color Gamut
What is wide color?
Standard RGB Color Space (sRGB)

Widely used
Based on ITU-R BT.709
Gamma $\approx 2.2$
Typical lighting conditions
Default color space for iOS
What do we do about this?
Introducing Display P3

Wide color space for today’s platforms
Based on the SMPTE DCI-P3
Gamma $\approx 2.2$
Typical lighting conditions
Display P3 in Action
Color Management on iOS 10
Introducing Extended Range sRGB

Same sRGB Primaries
Gamma $\approx 2.2$
Typical lighting conditions
Negative values and values greater than 1
Extended Range sRGB in Action

Display P3
{1.0, 0.0, 0.0}

Extended Range sRGB
{1.358, -0.074, -0.012}
Extended Range sRGB in Action

Display P3
\{1.0, 0.0, 0.0\}

Extended Range sRGB
\{1.358, -0.074, -0.012\}
Precision and Color
Precision and Color

2nd Floor

1st Floor

Appropriate height
Precision and Color

2nd Floor

1st Floor

Too tall
Precision and Color

8bit per color channel works well for sRGB
Precision and Color

2nd Floor

16bit per color channel for P3 and beyond

1st Floor

8bit per color channel works well for sRGB
We’ve Got You Covered

- Core Image
- WebKit
- ImageIO
- SpriteKit
- Core Graphics
- Retina Display
- SceneKit
- Core Animation
- UIKit
- AppKit
Wide Gamut Content

Adapting your content workflow

Patrick Heynen
Cocoa Frameworks
Where does Wide Color come from?
It comes from you!
Apps and their Content
Application Content Types
Application Content Types

Static image resources
Application Content Types

Static image resources
Document and network image resources
Application Content Types

Static image resources
Document and network image resources
Advanced Media

<table>
<thead>
<tr>
<th>Advances in iOS Photography</th>
<th>Mission</th>
<th>Tuesday 11:00AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing Live Photos and Raw on iOS</td>
<td>Marina</td>
<td>Wednesday 3:15PM</td>
</tr>
</tbody>
</table>
Application Content Types

Static image resources
Document and network image resources
Advanced Media
GPU Textures

What’s New in Metal, Part 2  Mission  Wednesday 11:00AM
Framing the Color Problem
Framing the Color Problem

App Content can come in a broad range of color richness from many sources
Framing the Color Problem

App Content can come in a broad range of color richness from many sources. Devices and Displays come in a broad range of color capabilities.
Framing the Color Problem

App Content can come in a broad range of color richness from many sources
Devices and Displays come in a broad range of color capabilities
How do we bridge the differences?
Solving the Color Problem

Color Management

Design → Tools → Deploy
Solving the Color Problem

Color Management

Design → Tools → Deploy
The job of color management is to ensure that an image looks the same on any output device no matter what color space it is encoded in.
How does it work?
Color Management
Color Management

Every image has an associated color space (color profile)
Every image has an associated color space (color profile)
Color matching maps image colors to output device
Color Management

Every image has an associated color space (color profile)

Color matching maps image colors to output device

Not for free: Every pixel needs to be color matched
Color Management

Every image has an associated color space (color profile)
Color matching maps image colors to output device
Not for free: Every pixel needs to be color matched
Potentially lossy: Color fidelity is lost when output has smaller gamut
Good News!
Color Management
Color Management

Color matching operations are easily hardware accelerated
Color Management

Color matching operations are easily hardware accelerated
Works automatically via Quartz 2D, ColorSync and Core Animation
Color Management

Color matching operations are easily hardware accelerated
Works automatically via Quartz 2D, ColorSync and Core Animation
Properly tagged content requires no code to display properly
Platform Color Management
Platform Color Management

macOS has been color managed since its inception!
macOS has been color managed since its inception!

iOS supports automatic color management on iOS 9.3 and later
Solving the Color Problem

Color Management

Design → Tools → Deploy
Design
Design Considerations for Wide Gamut
Design Considerations for Wide Gamut

Use wide gamut content where it makes sense
Design Considerations for Wide Gamut

Use wide gamut content where it makes sense
Use where vivid colors enhance the user experience
Design Considerations for Wide Gamut

Use wide gamut content where it makes sense
Use where vivid colors enhance the user experience
No need to change all content to P3
Use wide gamut content where it makes sense
Use where vivid colors enhance the user experience
No need to change all content to P3
Toolchain support makes gradual opt-in of wide gamut content possible
Upgrading Content to Wide Color

Be careful when promoting an existing design file to wide color!
Upgrading Content to Wide Color

Be careful when promoting an existing design file to wide color!

- Don’t “assign” P3 profile
Upgrading Content to Wide Color

Be careful when promoting an existing design file to wide color!

- Don’t “assign” P3 profile
Upgrading Content to Wide Color

Be careful when promoting an existing design file to wide color!

- Don’t “assign” P3 profile
- Convert to P3 instead
File Formats and Color Profiles
File Formats and Color Profiles

Use “Display P3” color profile for RGB working space
File Formats and Color Profiles

Use “Display P3” color profile for RGB working space

Use 16 bit per channel color mode
File Formats and Color Profiles

Use “Display P3” color profile for RGB working space
Use 16 bit per channel color mode
Use iMac (Late 2015) for accurate preview
File Formats and Color Profiles

Use “Display P3” color profile for RGB working space
Use 16 bit per channel color mode
Use iMac (Late 2015) for accurate preview
Export assets as 16bit PNG files with embedded “Display P3” ICC profile
File Formats and Color Profiles

Use “Display P3” color profile for RGB working space

Use 16 bit per channel color mode

Use iMac (Late 2015) for accurate preview

Export assets as 16bit PNG files with embedded “Display P3” ICC profile

Note: Save for Web and Export Assets do not support wide color!
File Formats and Color Profiles

Use “Display P3” color profile for RGB working space

Use 16 bit per channel color mode

Use iMac (Late 2015) for accurate preview

Export assets as 16bit PNG files with embedded “Display P3” ICC profile

Note: Save for Web and Export Assets do not support wide color!

Use Save As -> PNG format from 16bit source document as a workaround
Solving the Wide Color Problem

Design → Tools → Deploy
Solving the Wide Color Problem

Design → Tools → Deploy
Asset Catalogs
Asset Catalogs

Best deployment vehicle for static assets
Asset Catalogs

Best deployment vehicle for static assets

Automatic color correction
Asset Catalogs

- Best deployment vehicle for static assets
- Automatic color correction
- Automatic pixel format optimization
Asset Catalogs

- Best deployment vehicle for static assets
- Automatic color correction
- Automatic pixel format optimization
- App Slicing
Asset Catalog Enhancements
Asset Catalog Enhancements

Support for 16 bit source content
Asset Catalog Enhancements

Support for 16 bit source content
Cataloging by display gamut
Asset Catalog Enhancements

Support for 16 bit source content
Cataloging by display gamut
Three Easy Choices
Choice #1:
Choice #1: Do Nothing!
No Changes to Asset Catalog
No Changes to Asset Catalog
No Changes to Asset Catalog

Appearance will be preserved

8 bit sRGB
No Changes to Asset Catalog

Appearance will be preserved
No wide gamut colors

8 bit sRGB
Choice #2:
Choice #2: Upgrade to P3
Universal P3 Asset
Universal P3 Asset

Replace existing asset with upgraded asset

16 bit Display P3
Universal P3 Asset

Replace existing asset with upgraded asset
Automatic generation of sRGB derivative

16 bit Display P3
8 bit sRGB
Universal P3 Asset

Replace existing asset with upgraded asset
Automatic generation of sRGB derivative
High quality color match and dither to 8bit
Universal P3 Asset

Replace existing asset with upgraded asset
Automatic generation of sRGB derivative
High quality color match and dither to 8bit
Content selected according to display class
Choice #3:
Choice #3: Optimized Assets
Wide Color Optimized Assets
Wide Color Optimized Assets

16 bit Display P3

8 bit sRGB
Demo
Creating a wide color asset
Asset Catalog Deployment
Asset Catalog Deployment

App Slicing will ensure that appropriate variant is delivered to a given device
Asset Catalog Deployment

App Slicing will ensure that appropriate variant is delivered to a given device

No payload cost for wide gamut content on sRGB devices
Asset Catalog Deployment

NSImage automatically selects best representation for target display
Asset Catalog Deployment

NSImage automatically selects best representation for target display

Content is refreshed when display changes or properties change
Asset Catalog Storage
Asset Catalog Storage

High quality image conversions and efficient storage
Asset Catalog Storage

High quality image conversions and efficient storage

16 bit per component storage for wide color content
Asset Catalog Storage

High quality image conversions and efficient storage

16 bit per component storage for wide color content

Content dependent image compression
Lossy Compression

- Lossless (Inherited)
- Lossless
  - Automatic
- Lossy
  - Automatic
  - Basic
  - GPU Best Quality
  - GPU Smallest Size
Lossy Compression

Basic compression (JPEG + alpha)
Lossy Compression

Basic compression (JPEG + alpha)
Excellent performance across all devices
GPU Compression (ASTC)

Advanced Scalable Texture Compression
GPU Compression (ASTC)

Advanced Scalable Texture Compression
- Optimize for best quality or smallest size (4bpp or 1bpp)
GPU Compression (ASTC)

Advanced Scalable Texture Compression

- Optimize for best quality or smallest size (4bpp or 1bpp)
GPU Compression (ASTC)

Advanced Scalable Texture Compression
- Optimize for best quality or smallest size (4bpp or 1bpp)
- Automatic software fallback for devices lacking ASTC graphics capability
GPU Compression and Wide Color Assets

GPU compression uses ASTC LDR
texture compression
GPU Compression and Wide Color Assets

GPU compression uses ASTC LDR texture compression.

Wide content needs to be reduced to 8 bits before compression.
GPU Compression and Wide Color Assets

GPU compression uses ASTC LDR texture compression

Wide content needs to be reduced to 8 bits before compression

Xcode performs a high quality dither automatically
GPU Compression and Wide Color Assets

GPU compression uses ASTC LDR texture compression

Wide content needs to be reduced to 8bits before compression

Xcode performs a high quality dither automatically

Wide gamut colors preserved by encoding in Display P3 color space
What about Colors?
Colors in UI
Colors in UI

Most pixels on screen are solid colors
Colors in UI

Most pixels on screen are solid colors

Wide gamut colors present new challenges
Talking Colors
Color Specification

Colors are usually communicated with an assumed sRGB color space
Color Specification

Colors are usually communicated with an assumed sRGB color space

RGB (128, 45, 56)

#FF0456
This is no longer sufficient for Wide Gamut colors
Color Specification

Be specific about color space!
Color Specification

Be specific about color space!

Use Display P3 instead of sRGB when working with wide gamut designs
Color Specification

Be specific about color space!
Use Display P3 instead of sRGB when working with wide gamut designs
Use floating point for more precision
Color Specification

Be specific about color space!

Use Display P3 instead of sRGB when working with wide gamut designs

Use floating point for more precision

P3 (255, 128, 191)

P3 (1.0, 0.5, 0.75)
Picking Colors
Coding Colors
Constructing Wide Gamut Colors
Constructing Wide Gamut Colors

NSColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0)

UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0)
Constructing Extended Range sRGB Colors
Constructing Extended Range sRGB Colors

**NSColor** (red: 1.1, green: -.25, blue: 0.0, alpha: 1.0)

**UIColor** (red: 1.1, green: -.25, blue: 0.0, alpha: 1.0)
Storing Colors
Focus: Archiving Colors

Special care needs to be taken when storing wide gamut colors in document data.
Focus: Archiving Colors

Special care needs to be taken when storing wide gamut colors in document data.
Beware of assumed color spaces!
Focus: Archiving Colors

Special care needs to be taken when storing wide gamut colors in document data
Beware of assumed color spaces!
Consider encoding “compatible” sRGB color alongside new wide gamut colors
Focus: Archiving Colors

Special care needs to be taken when storing wide gamut colors in document data. Beware of assumed color spaces!

Consider encoding “compatible” sRGB color alongside new wide gamut colors.

iOS: Use `CGColor.convert()` API to convert colors to sRGB.
Focus: Archiving Colors

Special care needs to be taken when storing wide gamut colors in document data. Beware of assumed color spaces!

Consider encoding “compatible” sRGB color alongside new wide gamut colors.

iOS: Use `CGColor.convert()` API to convert colors to sRGB.

macOS: Use `NSColor.usingColorSpace()` API to convert colors to sRGB.
Case Study: RTF / TextEdit
Case Study: RTF / TextEdit

Existing

{\colortbl;\red0\green0\blue255;\red0\green128\blue255;\red255\green14\blue12;}
Case Study: RTF / TextEdit

Existing

{\colortbl;\red0\green0\blue255;\red0\green128\blue255;\red255\green14\blue12;}

Case Study: RTF / TextEdit

Existing

{\colortbl;\red0\green0\blue255;\red0\green128\blue255;\red255\green14\blue12;}

New

{\*\expandedcolortbl;\cssrgb\c50000\c100000\c0;\cspthree\c100000\c50000\c0;\cssrgb\c-20000\c140000\c0;}
Case Study: RTF / TextEdit

**Existing**

{\colortbl;\red0\green0\blue255;\red0\green128\blue255;\red255\green14\blue12;}

**New**

{\*\expandedcolortbl;\cssrgb\c50000\c100000\c0;\cspthree\c100000\c50000\c0;\cssrgb\c-20000\c140000\c0;}
Surfing with Colors
Colors in the Web

All tagged images are color matched
Colors in the Web

All tagged images are color matched

Media Queries to resolve assets between P3 and sRGB capable devices

```
<picture>
  <source srcset="flower-p3.jpg" media="(color-gamut: p3)"/>
  <source srcset="flower-rgb.jpg"/>
</picture>
```
Colors in the Web

All tagged images are color matched

Media Queries to resolve assets between P3 and sRGB capable devices

```html
<picture>
  <source srcset="flower-p3.jpg" media="(color-gamut: p3)"

  <source srcset="flower-rgb.jpg">
</picture>
```

WebKit proposal for CSS colors in other color spaces
Wide Gamut Rendering

Optimizing your app’s drawing for wide gamut displays

Steve Holt UIKit
Drawing with Wide Color
Drawing with Wide Color

(Output simulated)
Drawing with Wide Color

Cocoa
Drawing with Wide Color
Cocoa

NSImage.init(size: NSSize, flipped drawingHandlerShouldBeCalledWithFlippedContext: Bool, drawingHandler: (NSRect) -> Bool)
Drawing with Wide Color

Cocoa

NSImage.init(size: NSSize, flipped drawingHandlerShouldBeCalledWithFlippedContext: Bool, drawingHandler: (NSRect) -> Bool)

• DrawingHandler called with current NSGraphicsContext instance
Cocoa Drawing with Wide Color

NSImage.init(size: NSSize, flipped drawingHandlerShouldBeCalledWithFlippedContext: Bool, drawingHandler: (NSRect) -> Bool)

- DrawingHandler called with current NSGraphicsContext instance
- Use anywhere you would NSImage
import Cocoa

let size = CGSize(width: 250, height: 250)
let image = NSImage(size: size, flipped: false) { drawRect -> Bool in
    let rects = drawRect.divide(drawRect.size.width/2, fromEdge: .minXEdge)

    NSColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.slice).fill()

    NSColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.remainder).fill()

    return true
}

import Cocoa

let size = CGSize(width: 250, height: 250)
let image = NSImage(size: size, flipped: false) { drawRect -> Bool in
    let rects = drawRect.divide(drawRect.size.width/2, fromEdge: .minXEdge)
    NSColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.slice).fill()
    NSColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.remainder).fill()
    return true
}

import Cocoa

let size = CGSize(width: 250, height: 250)
let image = NSImage(size: size, flipped: false) { drawRect -> Bool in
    let rects = drawRect.divide(drawRect.size.width/2, fromEdge: .minXEdge)

    NSColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.slice).fill()

    NSColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.remainder).fill()

    return true

}
import Cocoa

let size = CGSize(width: 250, height: 250)
let image = NSImage(size: size, flipped: false) { drawRect -> Bool in
    let rects = drawRect.divide(drawRect.size.width/2, fromEdge: .minXEdge)
    NSColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.slice).fill()
    NSColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.remainder).fill()
    return true
}
import Cocoa

let size = CGSize(width: 250, height: 250)
let image = NSImage(size: size, flipped: false) { drawRect -> Bool in
    let rects = drawRect.divide(drawRect.size.width/2, fromEdge: .minXEdge)

    NSColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.slice).fill()

    NSColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    NSBezierPath(rect: rects.remainder).fill()

    return true
}
Drawing with Wide Color

Cocoa

(Output simulated)
Drawing with Wide Color

CocoaTouch

UIGraphicsBeginImageContext(_ size: CGSize)
// Drawing with Wide Color
// CocoaTouch

import UIKit

let size = CGSize(width: 250, height: 250)
UIGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)

UIColor.red().set()
UIRectFill(rects.remainder)

let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

let image = UIGraphicsGetImageFromCurrentImageContext()
// Drawing with Wide Color
// CocoaTouch

import UIKit

let size = CGSize(width: 250, height: 250)
UIGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)

UIColor.red().set()
UIRectFill(rects.remainder)

let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

let image = UIGraphicsGetImageFromCurrentImageContext()
// Drawing with Wide Color
// CocoaTouch

import UIKit

let size = CGSize(width: 250, height: 250)
UIKitGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)
 UIColor.red().set()
UIRectFill(rects.remainder)

UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.remainder)

UIKitGraphicsEndImageContext()

let image = UIGraphicsGetImageFromCurrentImageContext()
// Drawing with Wide Color
// CocoaTouch

import UIKit

let size = CGSize(width: 250, height: 250)
UIGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)
UIColor.red().set()
UIRectFill(rects.remainder)

let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

let image = UIGraphicsGetImageFromCurrentImageContext()
import UIKit

let size = CGSize(width: 250, height: 250)
UIGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)
UIColor.red().set()
UIRectFill(rects.remainder)

let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

let image = UIGraphicsGetImageFromCurrentImageContext()
import UIKit

let size = CGSize(width: 250, height: 250)
UIGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)

UIColor.red().set()
UIRectFill(rects.remainder)

let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

let image = UIGraphicsGetImageFromCurrentImageContext()
import UIKit

let size = CGSize(width: 250, height: 250)
UIGraphicsBeginImageContext(size)

let rects = CGRect(origin: .zero, size: size).divide(size.width/2, fromEdge: .minXEdge)
UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
UIRectFill(rects.slice)
UIColor.red().set()
UIRectFill(rects.remainder)

let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
Drawing with Wide Color

(Output simulated)
“The format for the bitmap is a ARGB 32-bit integer pixel format using host-byte order.”

From the UIKit Function Reference
Drawing with Wide Color

CocoaTouch

UIGraphicsBeginImageContext(_ size: CGSize)
Drawing with Wide Color
CocoaTouch

```c
UIGraphicsBeginImageContext(_ size: CGSize)
```

- Cannot create contexts with more than 8 bits per color channel
Drawing with Wide Color

CocoaTouch

```swift
UIGraphicsBeginImageContext(_ size: CGSize)

• Cannot create contexts with more than 8 bits per color channel
• Cannot represent colors in extended range sRGB
```
Drawing with Wide Color
CocoaTouch

UIGraphicsBeginImageContext(_ size: CGSize)

- Cannot create contexts with more than 8 bits per color channel
- Cannot represent colors in extended range sRGB
- Existing interface has no ability to create a context in non-sRGB color space

UIGraphicsBeginImageContextWithOptions(_ size: CGSize, _ opaque: Bool, _ scale: CGFloat)
Drawing with Wide Color

CocoaTouch

UIGraphicsImageRenderer(size: CGSize)
// Drawing with Wide Color
// CocoaTouch

import UIKit

let renderer = UIGraphicsImageRenderer(size: CGSize(width: 250, height: 250))

let image = renderer.image { rendererContext in
    let bounds = rendererContext.format.bounds
    let rects = bounds.divide(bounds.size.width/2, fromEdge: .minXEdge)

    UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.slice)

    UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.remainder)
}

// Drawing with Wide Color
// CocoaTouch

import UIKit

let renderer = UIGraphicsImageRenderer(size: CGSize(width: 250, height: 250))

let image = renderer.image { rendererContext in
    let bounds = rendererContext.format.bounds
    let rects = bounds.divide(bounds.size.width/2, fromEdge: .minXEdge)

    UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.slice)

    UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.remainder)
}
import UIKit

let renderer = UIGraphicsImageRenderer(size: CGSize(width: 250, height: 250))

let image = renderer.image { rendererContext in
    let bounds = rendererContext.format.bounds
    let rects = bounds.divide(bounds.size.width/2, fromEdge: .minXEdge)

    UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.slice)

    UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.remainder)
}
// Drawing with Wide Color
// CocoaTouch

import UIKit

let renderer = UIGraphicsImageRenderer(size: CGSize(width: 250, height: 250))

let image = renderer.image { rendererContext in
    let bounds = rendererContext.format.bounds
    let rects = bounds.divide(bounds.size.width/2, fromEdge: .minXEdge)

    UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.slice)

    UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.remainder)
}

import UIKit

let renderer = UIGraphicsImageRenderer(size: CGSize(width: 250, height: 250))

let image = renderer.image { rendererContext in
    let bounds = rendererContext.format.bounds
    let rects = bounds.divide(bounds.size.width/2, fromEdge: .minXEdge)

    UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.slice)

    UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.remainder)
}

UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
rendererContext.fill(rects.slice)
import UIKit

let renderer = UIGraphicsImageRenderer(size: CGSize(width: 250, height: 250))

let image = renderer.image { rendererContext in
    let bounds = rendererContext.format.bounds
    let rects = bounds.divide(bounds.size.width/2, fromEdge: .minXEdge)

    UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.slice)

    UIColor(red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
    rendererContext.fill(rects.remainder)
}

UIColor(displayP3Red: 1.0, green: 0.0, blue: 0.0, alpha: 1.0).set()
Drawing with Wide Color

Cocoa

(Output simulated)
Drawing with Wide Color
CocoaTouch
Drawing with Wide Color

CocoaTouch

UIGraphicsImageRenderer(size: CGSize)
Drawing with Wide Color

CocoaTouch

UIGraphicsImageRenderer(size: CGSize)

• Fully color managed by default
Drawing with Wide Color

CocoaTouch

UIGraphicsImageRenderer(size: CGSize)

- Fully color managed by default
- Supports extended range sRGB color space
Drawing with Wide Color

CocoaTouch

UIKitGraphicsImageRenderer(size: CGSize)

- Fully color managed by default
- Supports extended range sRGB color space
- Manages CGContext lifetime
Drawing with Wide Color

CocoaTouch

UIGraphicsImageRenderer(size: CGSize)

• Fully color managed by default
• Supports extended range sRGB color space
• Manages CGContext lifetime
• Works with your existing code using UIGraphicsGetCurrentContext()
Wide Color on the Screen
Wide Color on the Screen
CocoaTouch
Wide Color on the Screen
CocoaTouch

UIView

draw(_ rect: CGRect) // called in the extended sRGB color space
Wide Color on the Screen

CocoaTouch

UIView

draw(_ rect: CGRect) // called in the extended sRGB color space

UIImageView

Color managed since iOS 9.3
Wide Color on the Screen
CocoaTouch
Wide Color on the Screen
CocoaTouch

UITraitCollection.displayGamut
Wide Color on the Screen
CocoaTouch

UITraitCollection.displayGamut

UIDisplayGamut
Wide Color on the Screen

CocoaTouch

UITraitCollection.displayGamut

UIDisplayGamut

• .P3
Wide Color on the Screen
CocoaTouch

UITraitCollection.displayGamut

UIDisplayGamut

• .P3
• .SRGB
Wide Color on the Screen

Controlling the depth—CocoaTouch

For UIView, use the view’s layer’s contentsFormat property
Wide Color on the Screen
Controlling the depth—CocoaTouch

For UIView, use the view’s layer’s contentsFormat property

Valid CALayer contents formats:
- kCAContentsFormatRGBA8Uint
- kCAContentsFormatRGBA16Float
- kCAContentsFormatGray8Uint
Wide Color on the Screen
Cocoa
NSView
Wide Color on the Screen

Cocoa

NSView

· draw(_ dirtyRect: NSRect)
Wide Color on the Screen
Cocoa

NSView

- draw(_ dirtyRect: NSRect)
- viewDidLoad()
Wide Color on the Screen

Cocoa

NSView

- `draw(_ dirtyRect: NSRect)`
- `viewDidChangeBackingProperties()`
- `NSWindowDidChangeBackingPropertiesNotification //from your view’s window`
Wide Color on the Screen
Cocoa

**NSView**

- `draw(_ dirtyRect: NSRect)`
- `viewDidChangeBackingProperties()`
- `NSNotification` from your view’s window

- Backing properties include scale, color space and output display gamut
Wide Color on the Screen

Controlling the Depth—Cocoa

Use NSWindow’s depthLimit property to control how large the NSWindow’s backing store will be
Wide Color on the Screen

Controlling the Depth—Cocoa

Use NSWindow’s depthLimit property to control how large the NSWindow’s backing store will be.

NSWindowDepth
• NSWindowDepthTwentyfourBitRGB
• NSWindowDepthSixtyfourBitRGB
• NSWindowDepthOnehundredtwentyeightBitRGB
Wide Color is technology for the next generation of displays
Summary

Wide Color is technology for the next generation of displays
Color Gamuts and Color Management
Summary

Wide Color is technology for the next generation of displays
Color Gamuts and Color Management
Working with wide gamut content
Summary

Wide Color is technology for the next generation of displays

Color Gamuts and Color Management

Working with wide gamut content

Using colors in your application
Wide Color is technology for the next generation of displays
Color Gamuts and Color Management
Working with wide gamut content
Using colors in your application
Adapting your drawing code to wide color gamut
More Information

## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in iOS Photography</td>
<td>Pacific Heights</td>
<td>Tuesday 11:00AM</td>
</tr>
<tr>
<td>What's New in Cocoa</td>
<td>Nob Hill</td>
<td>Tuesday 11:00AM</td>
</tr>
<tr>
<td>What's New in Cocoa Touch</td>
<td>Presidio</td>
<td>Tuesday 1:40PM</td>
</tr>
<tr>
<td>What's New in Metal, Part 2</td>
<td>Pacific Heights</td>
<td>Wednesday 1:40PM</td>
</tr>
<tr>
<td>Live Photo Editing and RAW Processing with Core Image</td>
<td>Nob Hill</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>Advances in SceneKit Rendering</td>
<td>Mission</td>
<td>Thursday 11:00AM</td>
</tr>
</tbody>
</table>
# Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>What's New in SpriteKit</td>
<td>Presidio</td>
<td>Thursday 5:00PM</td>
</tr>
<tr>
<td>Crafting Modern Cocoa Apps</td>
<td>Pacific Heights</td>
<td>Friday 5:00PM</td>
</tr>
<tr>
<td>Labs</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Live Photo and Core Image Lab</td>
<td>Graphics, Games, and Media Lab C</td>
<td>Thursday 1:30PM</td>
</tr>
<tr>
<td>Cocoa Lab</td>
<td>Frameworks Lab D</td>
<td>Thursday 2:00PM</td>
</tr>
<tr>
<td>SceneKit Lab</td>
<td>Graphics, Games, and Media Lab A</td>
<td>Thursday 3:00PM</td>
</tr>
<tr>
<td>Live Photo and Core Image Lab</td>
<td>Graphics, Games, and Media Lab D</td>
<td>Friday 9:00AM</td>
</tr>
<tr>
<td>Cocoa Touch and 3D Touch Lab</td>
<td>Frameworks Lab C</td>
<td>Friday 10:30AM</td>
</tr>
<tr>
<td>Labs</td>
<td>Graphics, Games, and Media Lab</td>
<td>Date</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>SpriteKit Lab</td>
<td>B</td>
<td>Friday 12:00PM</td>
</tr>
<tr>
<td>Color Lab</td>
<td>C</td>
<td>Friday 4:00PM</td>
</tr>
<tr>
<td>Safari and WebKit Lab</td>
<td></td>
<td>Friday 4:30PM</td>
</tr>
</tbody>
</table>