Image and Graphics Best Practices

Session 219

Kyle Sluder, iOS System Experience
UIImage and UIImageView

Custom drawing with UIKit

Advanced CPU and GPU techniques
Memory
CPU
Image Rendering Pipeline

Load → Decode → Render
Buffers

Contiguous region of memory
Buffers

Contiguous region of memory

Often viewed as sequence of elements
Image Buffers

In-memory representation of an image

Each element describes color of a single pixel

Buffer size is proportional to image size
Image Buffers
The frame buffer

UIWindow

UIImageView

UIImage
Image Buffers
The frame buffer

UIWindow

UIImageView

UIImage

Render
Image Buffers
The frame buffer

UIWindow
UIImageView
UIImage

Render
Image Buffers
The frame buffer

UIWindow
UIImageView
UIImage

Render
Image Buffers
The frame buffer

UIKit

UIWindow

UIImageView

UIImage

UIImage

Render

UIWindow

UIImage

UIImage

Render

UIWindow

UIImage

UIImage

Render

UIWindow

UIImage

UIImage
Image Buffers
The frame buffer

UIImage
UIImageView
UIKitView
UIKitWindow

60–120 Hz

Render

60–120 Hz
Image Buffers
The frame buffer

UIImage
UIImageView
UIImageView
UIImage

UIWindow

Render

60–120 Hz
Image Buffers

The frame buffer

- UIWindow
- UIImageView
- UIImage

Render

60–120 Hz
Image Buffers

The frame buffer

UIWindow

UIImageView

UIImage

Render

60–120 Hz
Image Buffers
The frame buffer

- UIWindow
- UIImageView
- UIImage

Render

60–120 Hz
Image Buffers

The frame buffer

- UIWindow
- UIImageView
- UIImage

Render

60–120 Hz
Data Buffers

Store contents of an image file in memory

Metadata describing dimensions of image

Image itself encoded as JPEG, PNG, or other (usually compressed) form

Bytes do not directly describe pixels
Pipeline in Action

UIImage

UIImageView

UIImage

Data Buffer
Pipeline in Action

UIImage

UIImageView

UIImage

Data Buffer

Decode

Image Buffer
Pipeline in Action

UIImageView

UIImage

Data Buffer

Decode

Image Buffer
Pipeline in Action
Decoding Concerns

CPU-intensive process

Retained for repeat rendering

Persistent large memory allocation

Proportional to original image size, not view size
Consequences of Excessive Memory Usage

- Increased fragmentation
- Poor locality of reference
- System starts compressing memory
- Process termination
Proactively Saving Memory

Downsampling

Load (UIImage) → Decode → Render (UIImageView)
Proactively Saving Memory

Downsampling

Load (CGImage Source) → Thumbnail (CGImageRef) → Decode → UIImage → Render (UIImageView)
Proactively Saving Memory

Downsampling

UIImage → Render (UIImageView)
func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions = 
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage = 
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
// Downsampling large images for display at smaller size

```swift
func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions = 
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage = 
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
```
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage =
        CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
  let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
  let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

  let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
  let downsampleOptions = 
    [kCGImageSourceCreateThumbnailFromImageAlways: true,
     kCGImageSourceShouldCacheImmediately: true,
     kCGImageSourceCreateThumbnailWithTransform: true,
     kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

  let downsampledImage = 
    CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
  return UIImage(cgImage: downsampledImage)
}
// Downsampling large images for display at smaller size

func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions = [kCGImageSourceCreateThumbnailFromImageAlways: true,
                              kCGImageSourceShouldCacheImmediately: true,
                              kCGImageSourceCreateThumbnailWithTransform: true,
                              kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage = CGImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
func downsample(imageAt imageURL: URL, to pointSize: CGSize, scale: CGFloat) -> UIImage {
    let imageSourceOptions = [kCGImageSourceShouldCache: false] as CFDictionary
    let imageSource = CGImageSourceCreateWithURL(imageURL as CFURL, imageSourceOptions)!

    let maxDimensionInPixels = max(pointSize.width, pointSize.height) * scale
    let downsampleOptions =
        [kCGImageSourceCreateThumbnailFromImageAlways: true,
         kCGImageSourceShouldCacheImmediately: true,
         kCGImageSourceCreateThumbnailWithTransform: true,
         kCGImageSourceThumbnailMaxPixelSize: maxDimensionInPixels] as CFDictionary

    let downsampledImage = CGBitmapImageSourceCreateThumbnailAtIndex(imageSource, 0, downsampleOptions)!
    return UIImage(cgImage: downsampledImage)
}
Without downsampling

31.5MiB

Persistent memory use
Without downsampling

31.5MiB

Persistent memory use

With downsampling

18.4MiB
/ Downsampling large images for display at smaller size

func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "Cell", for: indexPath) as! MyCollectionViewCell
    cell.layoutIfNeeded() // Ensure imageView is its final size.
    let imageViewSize = cell.imageView.bounds.size
    let scale = collectionView.traitCollection.displayScale
    cell.imageView.image = downsample(imageAt: imageURLs[indexPath.item], to: imageViewSize, scale: scale)
    return cell
}
Decoding in Scrollable Views
Decoding in Scrollable Views
Decoding in Scrollable Views
Decoding in Scrollable Views
Decoding in Scrollable Views

CPU 1

CPU 2

Responsiveness

Battery Life
Decoding in Scrollable Views

Prefetching

CPU 1

CPU 2

A Tour of UICollectionView

Hall 3

Thursday 2:00PM
Decoding in Scrollable Views

Prefetching

Background decoding/downsampling
Decoding in Scrollable Views

Prefetching

Background decoding/downsampling

Responsiveness

Battery Life
Asynchronously downsampling on a global queue

```swift
func collectionView(_ collectionView: UICollectionView,
prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```
func collectionView(_ collectionView: UICollectionView,
prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
/ Asynchronously downsampling on a global queue

```swift
func collectionView(_ collectionView: UICollectionView, prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
```
// Asynchronously downsampling on a global queue

func collectionView(_ collectionView: UICollectionView,
prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        DispatchQueue.global(qos: .userInitiated).async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async {
                self.update(at: indexPath, with: downsampledImage)
            }
        }
    }
}
Thread Explosion

More images to decode than available CPUs

GCD continues creating threads as new work is enqueued

Each thread gets less time to actually decode images
let serialQueue = DispatchQueue(label: "Decode queue")

func collectionView(_ collectionView: UICollectionView, prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        serialQueue.async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
// Avoiding thread explosion when doing asynchronous work

let serialQueue = DispatchQueue(label: "Decode queue")

func collectionView(_ collectionView: UICollectionView, prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        serialQueue.async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
let serialQueue = DispatchQueue(label: "Decode queue")

func collectionView(_ collectionView: UICollectionView, prefetchItemsAt indexPaths: [IndexPath]) {
    // Asynchronously decode and downsample every image we are about to show
    for indexPath in indexPaths {
        serialQueue.async {
            let downsampledImage = downsample(images[indexPath.row])
            DispatchQueue.main.async { self.update(at: indexPath, with: downsampledImage) }
        }
    }
}
Image Sources

Image assets in asset catalog
Files in application/framework bundle
Files in Documents and Caches directories
Data downloaded from network
Image Sources

Image assets in asset catalog
Files in application/framework bundle
Files in Documents and Caches directories
Data downloaded from network
Prefer Image Assets
For artwork bundled with your app

Optimized name- and trait-based lookup

Smarter buffer caching

Per-device thinning

Vector artwork
Vector Artwork

Since iOS 11, image assets support "Preserve Vector Data"

Avoids blurriness and aliasing when drawn larger or smaller than natural size

Preserves legibility of icons in accessibility HUD
Vector Artwork Pipeline

Load (UIImage) → Rasterize → Render (UIImageView)

Resize
Vector Artwork Optimizations

Xcode rasterizes artwork for relevant scale factors while compiling.

Prerasterized artwork used when image is drawn at natural size.

If artwork has fixed sizes, use multiple image assets instead of relying on vector rasterization.
Custom Drawing with UIKit
UIView Subclass

UIButton

UIImageView
// Drawing a custom view by overriding UIView.draw(_:)

override func draw(_ rect: CGRect) {
    // Draw rounded rectangle background.
    let roundRectPath = UIBezierPath(roundedRect: self.bounds, cornerRadius: 4.0)
    UIColor.yellow.set()
    roundRectPath.fill()

    // Draw Live Photo icon.
    let image = UIImage(named: "LivePhotosIcon")
    image.draw(at: CGPoint(x: 2.0, y: 2.0))

    // Draw label.
    let text: NSAttributedString(string: "LIVE", attributes: ...)
    text.draw(at: CGPoint(x: 20.0, y: 2.0))
}
// Drawing a custom view by overriding UIView.draw(_:)

override func draw(_ rect: CGRect) {
    // Draw rounded rectangle background.
    let roundRectPath = UIBezierPath(roundedRect: self.bounds, cornerRadius: 4.0)
    UIColor.yellow.set()
    roundRectPath.fill()

    // Draw Live Photo icon.
    let image = UIImage(named: "LivePhotosIcon")
    image.draw(at: CGPoint(x: 2.0, y: 2.0))

    // Draw label.
    let text: NSAttributedString(string: "LIVE", attributes: ...)
    text.draw(at: CGPoint(x: 20.0, y: 2.0))
}
Custom Drawing Versus UIImageView

- UIImageView
- UIView Subclass
Custom Drawing Versus UIImageView
Custom Drawing Versus UIImageView

- UIImageView
- CALayer
- UIView Subclass
- CALayer

Decoded image
Custom Drawing Versus UIImageView

- UIImageView
  - CALayer
  - Decoded image

- UIView Subclass
  - CALayer
Custom Drawing Versus UIImageView
Custom Drawing Versus UIImageView

- UIImageView
- CALayer
  - contents
  - Decoded image
- UIView Subclass
- CALayer

The diagram illustrates the differences between custom drawing and using UIImageView. On the left side, direct custom drawing goes through CALayer, while on the right side, custom drawing through a UIView subclass also goes through CALayer.
Custom Drawing Versus UIImageView

- UIView Subclass
  - Decoded image
  - Backing store

- UIImageView
  - CALayer
    - contents
Custom Drawing Versus UIImageView

- UIImageView
  - CALayer
    - contents
    - Decoded image

- UIView Subclass
  - CALayer
    - contents
    - Backing store
Custom Drawing Versus UIImageView

UIView → CALayer

Decoded image

draw

UIView Subclass → CALayer

Backing store
Custom Drawing Versus UIImageView

- Custom Drawing: UIView Subclass → CALayer
- UIImageView: UIView → CALayer

Decoded image

Backing store
Custom Drawing Versus UIImageView

- ** UIImageView**
  - Subclass
  - CALayer
  - Frame Buffer

- **Custom Drawing**
  - CALayer
  - Frame Buffer
  - Decoded image
  - Backing store
Backing Store Memory Costs

Proportional to pixel size of view

Element size grows to accommodate color range used by drawing

Setting `CALayer.contentsFormat` opts out

Update `layerWillDraw(_:)` implementations
Reducing Backing Store Use

- Refactor larger views into subview hierarchies
- Reduce or eliminate overrides of `draw(_:)`
- Eliminate duplicate copies of image data
- Use optimized view properties and subclasses
Reducing Backing Store Use
Alternatives to custom drawing

Overriding `draw(_:)` opts into backing store

`UIView.backgroundColor` can render directly to frame buffer without a backing store

• ...except for pattern colors
• Use `UIImageView` with tiling image instead
Reducing Backing Store Use

Masking versus corner radius

`UIView.maskView` and `CALayer.maskLayer` render view hierarchy into temporary image buffer.

`CALayer.cornerRadius` does not require any image buffer.

Consider `UIImageView` with resizable image instead of masking for transparent backgrounds.
Reducing Backing Store Use
Eliminating duplicate image data

UIImageView can colorize monochrome images while rendering directly into frame buffer

```
UIImage.withRenderingMode(_:)
```

or Rendering Mode popup in asset inspector

Set `tintColor` of image view to any solid color
Reducing Backing Store Use
UILabel optimizations for rendering text

UILabel is optimized for monochrome strings
Uses 75% smaller backing store when possible
Automatically upgrades to larger backing store for multicolor strings, emoji
Drawing Off-Screen

Use UIGraphicsImageRenderer to create and draw to an image buffer

Supports Wide Color, unlike UIGraphicsBeginImageContext()

Combine with UIImageView for efficient offscreen rendering
Drawing Off-Screen
Optimizing image buffers

Similar automatic Wide Color support as backing stores

Prior to iOS 12 and tvOS 12, defaults to Wide Color support based on hardware

Set `prefersExtendedRange` on `UIGraphicsImageRendererFormat` for direct control

`UIImage.imageRendererFormat` may offer an intermediate representation
Advanced CPU and GPU Techniques
Advanced Image Effects

Core Image

Consider Core Image for realtime effects

Executes on GPU, freeing up CPU

UIImageView renders CIImages efficiently

UIImage.init(ciImage:)
Advanced Image Processing
Interfacing with other frameworks

Use CVPixelBuffer to move data to frameworks like Metal, Vision, and Accelerate

Use the best initializer—don’t unwind work that’s already been done

Guard against moving work between GPU and CPU

Ensure buffers are correct format for Accelerate
Summary

Implement prefetching to prepare asynchronously

Reduce backing store usage by using UIImageView and UILabel

Don’t accidentally disable new optimizations for custom drawing

Prefer image assets for bundled artwork

Avoid over-reliance on Preserve Vector Data
## More Information

https://developer.apple.com/wwdc18/219

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Practical Approaches to Great App Performance</td>
<td>Hall 1</td>
<td>Wednesday 5:00PM</td>
</tr>
<tr>
<td>UIKit and Layout</td>
<td>Technology Lab 2</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>UIKit and Collection View Lab</td>
<td>Technology Lab 11</td>
<td>Friday 9:00AM</td>
</tr>
</tbody>
</table>