Advanced Graphics and Animations for iOS Apps

Session 419
Axel Wefers
iOS Software Engineer

Michael Ingrassia
iOS Software Engineer
What You Will Learn

Core Animation pipeline
Rendering concepts
UIBlurEffect
UIVibrancyEffect
Profiling tools
Case studies
Technology Framework

UIKit

Core Animation
OpenGL ES  Core Graphics
Graphics Hardware
Core Animation Pipeline

Axel Wefers
iOS Software Engineer
Core Animation Pipeline
Core Animation Pipeline

Application

Core Animation
Core Animation Pipeline
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU

Display
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU

Display

16.67 ms
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU

Display

Handle Events

16.67 ms
Core Animation Pipeline

Application → Core Animation
Render Server → Core Animation
GPU
Display

Commit Transaction

16.67 ms
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU

Display

Commit Transaction

Decode

16.67 ms
Core Animation Pipeline

- **Application**
  - Core Animation
- **Render Server**
  - Core Animation
- **GPU**
- **Display**

- Commit Transaction
- Decode
- Draw Calls

16.67 ms
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU

Display

Commit Transaction
Decode
Draw Calls
Render

16.67 ms
Core Animation Pipeline

1. Application
   Core Animation
   - Commit Transaction

2. Render Server
   Core Animation
   - Decode
   - Draw Calls

3. GPU
   - Render

4. Display
   - Display
   - 16.67 ms
Core Animation Pipeline

Application
   Core Animation
   Render Server
      Core Animation
GPU
Display

16.67 ms
Core Animation Pipeline

Application
  Core Animation

Render Server
  Core Animation

GPU

Display

Commit Transaction

16.67 ms
Commit Transaction
Commit Transaction

Set up the views
Commit Transaction

Set up the views
Draw the views

Layout Display
Commit Transaction

Set up the views
Draw the views
Additional Core Animation work
Commit Transaction

Set up the views
Draw the views
Additional Core Animation work
Package up layers and send them to render server
Commit Transaction

- Layout
- Display
- Prepare
- Commit
layoutSubviews overrides are invoked

View creation, **addSubview:**

Populate content, database lookups

Usually CPU bound or I/O bound
Display

Draw contents via `drawRect`: if it is overridden

String drawing

Usually CPU or memory bound
Prepare Commit

Image decoding
Image conversion
Commit

Package up layers and send to render server
Recursive
Expensive if layer tree is complex
Animation

Three-stage process

Application | Render Server
Animation

Three-stage process

1. Create animation and update view hierarchy (animateWithDuration:animations:)
Animation

Three-stage process

1. Create animation and update view hierarchy (`animateWithDuration:animations:`)
2. Prepare and commit animation (`layoutSubviews,drawRect:`)
Animation

Three-stage process

1. Create animation and update view hierarchy (`animateWithDuration:animations:`)
2. Layout, Display, Prepare, Commit

Diagram:
- Application
- Render Server

Steps:
1. Create animation and update view hierarchy
2. Layout, Display, Prepare, Commit
Animation

Three-stage process

1. Create animation and update view hierarchy
   (animateWithDuration:animations:)

2. Prepare and commit animation
   (layoutSubviews, drawRect:)

3. Render each frame
Rendering Concepts

Tile based rendering
Render passes
Example masking
Tile Based Rendering

Screen is split into tiles of NxN pixels
Each tile fits into the SoC cache
Geometry is split in tile buckets
Rasterization can begin after all geometry is submitted
Tile Based Rendering

Screen is split into tiles of NxN pixels
Each tile fits into the SoC cache
Geometry is split in tile buckets
Rasterization can begin after all geometry is submitted
Tile Based Rendering

Screen is split into tiles of N\times N pixels
Each tile fits into the SoC cache
Geometry is split in tile buckets
Rasterization can begin after all geometry is submitted
Tile Based Rendering

Screen is split into tiles of NxN pixels
Each tile fits into the SoC cache
Geometry is split in tile buckets
Rasterization can begin after all geometry is submitted
Tile Based Rendering

Screen is split into tiles of NxN pixels
Each tile fits into the SoC cache
Geometry is split in tile buckets
Rasterization can begin after all geometry is submitted
Tile Based Rendering

Rendering pass

Render Server
  Core Animation

Application
  Core Animation
Tile Based Rendering

Rendering pass

Render Server
- Core Animation
  - OpenGL

Application
- Core Animation

GPU
Tile Based Rendering

Rendering pass

- Render Server
  - Core Animation
    - OpenGL
- Application
  - Core Animation
- Command Buffer
- GPU
Tile Based Rendering
Rendering pass

- Render Server
  - Core Animation
    - OpenGL
- Application
  - Core Animation
- Command Buffer
- Vertex Processing
  - "Vertex Shader"
- GPU
Tile Based Rendering

Rendering pass

Render Server
- Core Animation
  - OpenGL

Application
- Core Animation

GPU
- Command Buffer
  - Vertex Processing "Vertex Shader"
  - Tiling
Tile Based Rendering

Rendering pass

- Render Server
  - Core Animation
    - OpenGL
  - Command Buffer
- Application
  - Core Animation
  - Command Buffer
  - Tiler
    - Vertex Processing "Vertex Shader"
    - Tiling
- GPU
Tile Based Rendering

Rendering pass

- Render Server
  - Core Animation
  - OpenGL

- Application
  - Core Animation

- Tiler
  - Vertex Processing "Vertex Shader"
  - Tiling
  - Parameter Buffer

- Command Buffer

- GPU
Tile Based Rendering

Rendering pass

Render Server
- Core Animation
  - OpenGL

Application
- Core Animation
  - GPU

Tiler
- Command Buffer
  - Vertex Processing
    - "Vertex Shader"
  - Tiling

Parameter Buffer
- Pixel Processing
  - "Pixel Shader"
Tile Based Rendering

Rendering pass

- Render Server
  - Core Animation
    - OpenGL
  - Command Buffer
- Application
  - Core Animation
- Tiler
  - Vertex Processing
    - "Vertex Shader"
  - Tiling
  - Parameter Buffer
- Renderer
  - Pixel Processing
    - "Pixel Shader"
Tile Based Rendering

Rendering pass

- Render Server
  - Core Animation
  - OpenGL
- Application
  - Core Animation
- Tiler
  - Command Buffer
  - Vertex Processing "Vertex Shader"
  - Tiling
- Renderer
  - Parameter Buffer
  - Pixel Processing "Pixel Shader"
  - Render Buffer
- GPU
Masking

Rendering passes

- Render Server
  - Core Animation
    - OpenGL
- Application
  - Core Animation
- Command Buffer
- GPU
Masking

Rendering passes

Pass 1
Render layer mask to texture
Masking
Rendering passes

Pass 1
Render layer mask to texture

Pass 2
Render layer content to texture
Masking
Rendering passes

Pass 1
Render layer mask to texture

Pass 2
Render layer content to texture

Compositing pass
Apply mask to content texture
UIBlurEffect

Axel Wefers
iOS Software Engineer
UIVisualEffectView with UIBlurEffect

UIBlurEffect styles

- No effect
- Extra light
- Light
- Dark
UIVisualEffectView with UIBlurEffect
Rendering passes (best case)
UIVisualEffectView with UIBlurEffect

Rendering passes (best case)
UIVisualEffectView with UIBlurEffect

Rendering passes (best case)

Command Buffer → Tiler → Vertex Processing → Render → Pixel Processing → Content → Downscaled Content

Pass 1
Render content

Pass 2
Capture content
UIVisualEffectView with UILBlurEffect

Rendering passes (best case)

GPU

Pass 1
Render content

Pass 2
Capture content

Pass 3
Horizontal blur

Pass 4
Vertical blur
UIVisualEffectView with UIBlurEffect

Rendering passes (best case)

Pass 1
Render content

Pass 2
Capture content

Pass 3
Horizontal blur

Pass 4
Vertical blur

Compositing pass
Upscale and tint
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air
UIVisualEffectView with UIBlurEffect
GPU utilization, fullscreen, iPad Air

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air

Renderer
Pixel processing

Tiler
Vertex processing

VBlank interrupt
Every 16.67 ms

16.67 ms
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air

Pass 1

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air

Pass 1

Pass 2

VBlank interrupt
Every 16.67 ms

Tiler
Vertex processing

Renderer
Pixel processing
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air

Pass 1

Pass 2

Pass 3

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air

Pass 1

Pass 2

Pass 3

Pass 4

16.67 ms

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIBlurEffect

GPU utilization, fullscreen, iPad Air

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
UIVisualEffectView with UIBlurEffect
GPU utilization, fullscreen, iPad Air

16.67 ms
UIVisualEffectView with UIBlurEffect

Fullscreen performance

- **UIBlurEffectStyleExtraLight**
  - iPad (3rd generation): 18.15 ms
  - iPad Air: 10.03 ms
- **UIBlurEffectStyleLight**
  - iPad (3rd generation): 14.69 ms
  - iPad Air: 6.88 ms
- **UIBlurEffectStyleDark**
  - iPad (3rd generation): 14.41 ms
  - iPad Air: 7.02 ms

GPU time in milliseconds (smaller is better)
UIVisualEffectView with UIBlurEffect

Fullscreen performance

- **UIBlurEffectStyleExtraLight**
  - iPad (3rd generation): 4.59 ms
  - iPad Air: 10.03 ms

- **UIBlurEffectStyleLight**
  - iPad (3rd generation): 4.59 ms
  - iPad Air: 6.88 ms

- **UIBlurEffectStyleDark**
  - iPad (3rd generation): 4.59 ms
  - iPad Air: 7.02 ms

GPU time in milliseconds (smaller is better)
# UIVisualEffectView with UIBlurEffect

## UIBlurEffect support

<table>
<thead>
<tr>
<th>Device</th>
<th>Blur</th>
<th>Tint</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPad 2</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>iPad (3rd generation)</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>iPad (4th generation)</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>iPad Air</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>iPad mini</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>iPad mini Retina display</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>All iPhones</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>iPod touch</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>
UIVisualEffectView with UIBlurEffect

Performance considerations

UIBlurEffect adds multiple offscreen passes depending on style.
Only dirty regions are redrawn.
Effect is very costly:
- UI can easily be GPU bound
- Keep bounds of view as small as possible
- Make sure to budget for effect
UIVibrancyEffect

Axel Wefers
iOS Software Engineer
UIVisualEffectView with UIVibrancyEffect

UIVibrancyEffect styles

Extra light

Light

Dark
UIVisualEffectView with UIVibrancyEffect

UIVibrancyEffect styles

Extra light	Light	Dark
UIVisualEffectView with UIVibrancyEffect
UIVibrancyEffect styles

Extra light

Light

Dark
UIVisualEffectView with UIVibrancyEffect

Rendering passes

- Render Server
- Core Animation
- OpenGL
- Command Buffer
- Application
- Core Animation

GPU
UIVisualEffectView with UIVibrancyEffect

Rendering passes

- GPU
  - Pass 1 to 5
    - Render blur effect
UIVisualEffectView with UIVibrancyEffect

Rendering passes

- **Pass 1 to 5**: Render blur effect
- **Pass 6**: Render layer content to texture
UIVisualEffectView with UIVibrancyEffect

Rendering passes

---

**Pass 1 to 5**
- Render blur effect

**Pass 6**
- Render layer content to texture

**Compositing pass**
- Apply filter to content texture
UIVisualEffectView with UIVibrancyEffect

GPU utilization, fullscreen, iPad Air

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIVibrancyEffect

GPU utilization, fullscreen, iPad Air

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Pass 6

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIVibrancyEffect

GPU utilization, fullscreen, iPad Air

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Pass 6
Pass 7

Tiler
Vertex processing

Renderer
Pixel processing

VBlank interrupt
Every 16.67 ms
UIVisualEffectView with UIVibrancyEffect

GPU utilization, fullscreen, iPad Air

- Tiler: Vertex processing
- Renderer: Pixel processing
- VBlank interrupt: Every 16.67 ms

16.67 ms
UIVisualEffectView with UIVibrancyEffect

Fullscreen performance

<table>
<thead>
<tr>
<th>Effect Style</th>
<th>GPU Time iPad (3rd generation)</th>
<th>GPU Time iPad Air</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIBlurEffectStyleExtraLight</td>
<td>4.59</td>
<td>10.03</td>
</tr>
<tr>
<td>UIBlurEffectStyleLight</td>
<td>4.59</td>
<td>6.88</td>
</tr>
<tr>
<td>UIBlurEffectStyleDark</td>
<td>4.59</td>
<td>7.02</td>
</tr>
</tbody>
</table>

GPU time in milliseconds (smaller is better)
UIVisualEffectView with UIVibrancyEffect

Fullscreen performance

<table>
<thead>
<tr>
<th>Effect Style</th>
<th>iPad (3rd generation)</th>
<th>iPad Air</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIBlurEffectStyleExtraLight</td>
<td>27.03</td>
<td>17.48</td>
</tr>
<tr>
<td>UIBlurEffectStyleLight</td>
<td>27.03</td>
<td>14.35</td>
</tr>
<tr>
<td>UIBlurEffectStyleDark</td>
<td>26.32</td>
<td>14.06</td>
</tr>
</tbody>
</table>
UIVisualEffectView with UIVibrancyEffect

Performance considerations

UIVibrancyEffect adds two offscreen passes
UIVibrancyEffect uses expensive compositing filter for content
Use UIVibrancyEffect on small regions
Only dirty regions are redrawn
UIVibrancyEffect is very costly on all devices
  → UI can easily be GPU bound
  → Keep bounds of view as small as possible
  → Make sure to budget for effects
Rasterization

Performance considerations

Use to composite to image once with GPU
Enable with `shouldRasterize` property on `CALayer`
Extra offscreen passes when updating content
Do not overuse, cache size is limited to 2.5x of screen size
Rasterized images evicted from cache if unused for more than 100ms
Rasterization

Typical use cases

Avoid redrawing expensive effects for static content
Avoid redrawing of complex view hierarchies
Group Opacity

Performance considerations

Disable with `allowsGroupOpacity` property on `CALayer`.

Will introduce offscreen passes:

- If layer is not opaque (`opacity != 1.0`)
- And if layer has nontrivial content (child layers or background image)

  ➞ Sub view hierarchy needs to be composited before being blended

Always turn it off if not needed.
Tools

Michael Ingrassia
iOS Software Engineer
Performance Investigation Mindset
## Performance Investigation Mindset

| What is the frame rate? | Goal is always 60 frames per second |
# Performance Investigation Mindset

<table>
<thead>
<tr>
<th>What is the frame rate?</th>
<th>Goal is always 60 frames per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
</tbody>
</table>
## Performance Investigation Mindset

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
</tbody>
</table>
## Performance Investigation Mindset

<table>
<thead>
<tr>
<th>Question</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Fewer is better</td>
</tr>
</tbody>
</table>
## Performance Investigation Mindset

<table>
<thead>
<tr>
<th>Question</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Fewer is better</td>
</tr>
<tr>
<td>Too much blending?</td>
<td>Less is better</td>
</tr>
</tbody>
</table>
## Performance Investigation Mindset

<table>
<thead>
<tr>
<th>Question</th>
<th>Guidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Fewer is better</td>
</tr>
<tr>
<td>Too much blending?</td>
<td>Less is better</td>
</tr>
<tr>
<td>Any strange image formats or sizes?</td>
<td>Avoid on-the-fly conversions or resizing</td>
</tr>
</tbody>
</table>
## Performance Investigation Mindset

<table>
<thead>
<tr>
<th>Question</th>
<th>Goals/Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Fewer is better</td>
</tr>
<tr>
<td>Too much blending?</td>
<td>Less is better</td>
</tr>
<tr>
<td>Any strange image formats or sizes?</td>
<td>Avoid on-the-fly conversions or resizing</td>
</tr>
<tr>
<td>Any expensive views or effects?</td>
<td>Understand the cost of what is in use</td>
</tr>
</tbody>
</table>
Performance Investigation Mindset

<table>
<thead>
<tr>
<th>Question</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Fewer is better</td>
</tr>
<tr>
<td>Too much blending?</td>
<td>Less is better</td>
</tr>
<tr>
<td>Any strange image formats or sizes?</td>
<td>Avoid on-the-fly conversions or resizing</td>
</tr>
<tr>
<td>Any expensive views or effects?</td>
<td>Understand the cost of what is in use</td>
</tr>
<tr>
<td>Anything unexpected in hierarchy?</td>
<td>Know the actual view hierarchy</td>
</tr>
</tbody>
</table>
Tools

Instruments
- Core Animation instrument
- OpenGL ES Driver instrument

Simulator
- Color debug options

Xcode
- View debugging
Instruments
Core Animation template

Choose a profiling template for: • iPhone5s (v8.0) > All Processes

- Blank
- Activity Monitor
- Allocations
- Automation
- Core Animation
- Counters
- Energy
-Leaks
- Network
- OpenGL ES
- System Trace

Core Animation
This template measures application graphics performance as well as CPU usage of a process via time profiling.
Instruments
Core Animation template

Choose a profiling template for: iPhone5s (v8.0) > All Processes

Standard  Custom  Recent

Blank  Activity Monitor  Allocations  Automation  Core Animation  Counters

Energy  Leaks  Network  OpenGL ES  OpenGL ES  System Trace

Core Animation
This template measures application graphics performance as well as CPU usage of a process via time profiling.
Core Animation Instrument
Measuring frame rate
Core Animation Instrument
Measuring frame rate
Core Animation Instrument
Measuring frame rate

The Core Animation Instrument is a tool used to measure the frame rate of animations in iOS applications. In the image, the instrument is shown with a chart that displays the frames per second (FPS) over time. The chart shows a consistent FPS of 60, indicating smooth animation performance. The lower part of the image shows a table with the number of frames per second for each second of the run, confirming the steady FPS of 60.
Core Animation Instrument
Measuring frame rate
Time Profiler Instrument
CPU utilization
Time Profiler Instrument

CPU utilization
Time Profiler Instrument

CPU utilization
Core Animation Instrument

Color debug options
Core Animation Instrument

Color debug options
Core Animation Instrument
Color debug options
Core Animation Instrument

Color blended layers
Core Animation Instrument
Color hits green and misses red
Core Animation Instrument
Color copied images
Core Animation Instrument

Color misaligned images
Core Animation Instrument
Color offscreen-rendered yellow
Core Animation Instrument
Color OpenGL fast path blue
Core Animation Instrument
Flash updated regions
## Performance Investigation Mindset

### Core Animation instrument summary

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU makes sense</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Fewer is better</td>
</tr>
<tr>
<td>Too much blending?</td>
<td>Less is better</td>
</tr>
<tr>
<td>Any strange image formats or sizes?</td>
<td>Avoid on-the-fly conversions or resizing</td>
</tr>
</tbody>
</table>
iOS Simulator Coloring Options
Instruments
OpenGL ES Driver template

Choose a profiling template for:  iPhone5s (v8.0)  All Processes

Standard  Custom  Recent

Energy Diagnostics Leaks Network OpenGL ES Analysis OpenGL ES Driver System Trace

System Usage Time Profiler Zombies

OpenGL ES Driver
This template measures OpenGL ES graphics performance as well as CPU usage of a process.

Open an Existing File...  Cancel  Choose
Instruments
OpenGL ES Driver template

Choose a profiling template for:  iPhone5s (v8.0)  All Processes

- Energy Diagnostics
-Leaks
-Network
-OpenGL ES Analysis
-OpenGL ES Driver
-System Trace

OpenGL ES Driver
This template measures OpenGL ES graphics performance as well as CPU usage of a process.
OpenGL ES Driver Instrument

Selecting statistics to list
OpenGL ES Driver Instrument

Selecting statistics to list
OpenGL ES Driver Instrument

Selecting statistics to list
OpenGL ES Driver Instrument

GPU utilization
OpenGL ES Driver Instrument

GPU utilization
OpenGL ES Driver Instrument

GPU utilization
Time Profiler Instrument

CPU utilization
Time Profiler Instrument

CPU utilization
Time Profiler Instrument

CPU utilization
## Performance Investigation Mindset

### OpenGL ES Driver instrument summary

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Goal is always 60 frames per second</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>Lower utilization is desired and saves battery</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>GPU is desirable but know when CPU make sense</td>
</tr>
</tbody>
</table>
Xcode

View debugging
Xcode

View debugging
Xcode

View debugging
Performance Investigation Mindset

Xcode view debugging summary

<table>
<thead>
<tr>
<th>Question</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any expensive views or effects?</td>
<td>Understand the cost of what is in use</td>
</tr>
<tr>
<td>Anything unexpected in hierarchy?</td>
<td>Know the actual view hierarchy</td>
</tr>
</tbody>
</table>
Case Studies

Michael Ingrassia
iOS Software Engineer
Case Studies

Explore several scenarios
Measure performance on different devices
Keep the same appearance with better performance
Fictitious Photo Application

Case study

Simple table view
Each cell shows a photo thumbnail and some text
Each photo has a small shadow
Measure Frame Rate on iPhone 5s
OpenGL ES Driver instrument
Measure Frame Rate on iPhone 5s
OpenGL ES Driver instrument
Awesome
Ship it?
Fictitious Photo Application
iPod touch scrolling performance

What about the performance on other devices?
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Analyzing View Hierarchy on iPod touch
Xcode view debugging
Analyzing View Hierarchy on iPod touch
Xcode view debugging
Color Offscreen-Rendered Yellow
Core Animation instrument
How Are We Setting up the Shadow?
How Are We Setting up the Shadow?

We are asking Core Animation to generate the shadow

```objective-c
CALayer *imageViewLayer = cell.imageView.layer;
imageViewLayer.shadowColor = [UIColor blackColor].CGColor;
imageViewLayer.shadowOpacity = 1.0;
imageViewLayer.shadowRadius = 2.0;
imageViewLayer.shadowOffset = CGSizeMake(1.0, 1.0);
```
How Are We Setting up the Shadow?

We are asking Core Animation to generate the shadow

```objective-c
CALayer *imageViewLayer = cell.imageView.layer;
imageViewLayer.shadowColor = [UIColor blackColor].CGColor;
imageViewLayer.shadowOpacity = 1.0;
imageViewLayer.shadowRadius = 2.0;
imageViewLayer.shadowOffset = CGSizeMake(1.0, 1.0);
imageViewLayer.shadowPath = CGPathCreateWithRect(imageRect, NULL);
```

Perhaps there is a more efficient way
Color Offscreen-Rendered Yellow
Core Animation instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch

OpenGL ES Driver instrument
Measure Frame Rate on iPod touch

OpenGL ES Driver instrument
Awesome

Can we ship it now?
Measure Frame Rate on iPhone 4s
OpenGL ES Driver instrument
Measure Frame Rate on iPhone 4s
OpenGL ES Driver instrument
Measure Frame Rate on iPhone 4s
OpenGL ES Driver instrument
Fictitious Photo Application

Performance across devices

Frame Rate (target is 60 fps)

<table>
<thead>
<tr>
<th></th>
<th>CA Shadow</th>
<th>shadowPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone 5s</td>
<td>55</td>
<td>60</td>
</tr>
<tr>
<td>iPhone 5</td>
<td>45</td>
<td>60</td>
</tr>
<tr>
<td>iPhone 4s</td>
<td>33</td>
<td>60</td>
</tr>
<tr>
<td>iPod touch</td>
<td>33</td>
<td>60</td>
</tr>
</tbody>
</table>
Awesome

Ship it!
Offscreen passes are expensive
- Use Core Animation instrument to find them
- Know what you can do to avoid them

Measure performance across different devices
- Use OpenGL ES Driver instrument for GPU time
- Use Time Profiler instrument for CPU time

Know your view hierarchy and any hidden costs
- This is especially true for table cells and scrolling
Fictitious Contacts Application
Case study

Simple table view
Each cell shows a round thumbnail and some text
Fictitious Contacts Application
Performance across devices

- iPhone 5s: 60 fps
- iPhone 5: 60 fps
- iPhone 4s: 55 fps
- iPod touch: 46 fps

Frame Rate (target is 60 fps)
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Color Offscreen-Rendered Yellow

Core Animation instrument
How Are We Achieving Round Thumbnails?
How Are We Achieving Round Thumbnails?

We are asking Core Animation to mask the image

```cpp
CALayer *imageViewLayer = cell.imageView.layer;
imageViewLayer.cornerRadius = imageHeight / 2.0;
imageViewLayer.masksToBounds = YES;
```
We are asking Core Animation to mask the image

```objective-c
CALayer *imageViewLayer = cell.imageView.layer;
imageViewLayer.cornerRadius = imageHeight / 2.0;
imageViewLayer.masksToBounds = YES;
imageViewLayer.masksToBounds = YES;
```

Perhaps there is a more efficient way

- Don’t mask on the fly, pre-generate thumbnails as round, or
How Are We Achieving Round Thumbnails?

We are asking Core Animation to mask the image

```objc
CALayer *imageViewLayer = cell.imageView.layer;
imageViewLayer.cornerRadius = imageHeight / 2.0;
imageViewLayer.masksToBounds = YES;
```

Perhaps there is a more efficient way

- Don’t mask on the fly, pre-generate thumbnails as round, or
- If that is not possible, fake it
  - Table background is solid white
  - Render a white inverted circle on top of square thumbnail asset
  - Reducing offscreen passes but increasing blending, still a net performance win
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Measure Frame Rate on iPod touch
OpenGL ES Driver instrument
Fictitious Contacts Application
Performance across devices

Frame Rate (target is 60 fps)

- **before**
  - iPhone 5s: 60
  - iPhone 5: 60
  - iPhone 4s: 46
  - iPod touch: 55

- **after**
  - iPhone 5s: 60
  - iPhone 5: 60
  - iPhone 4s: 60
  - iPod touch: 60
Fictitious Contacts Application

Summary

Offscreen passes are expensive
• Use Core Animation instrument to find them
• Know what you can do to avoid them

Measure performance across different devices
• Use OpenGL ES Driver instrument for GPU time
• Use Time Profiler instrument for CPU time

Know your view hierarchy and any hidden costs
• This is especially true for table cells and scrolling
## Performance Investigation Mindset

### Summary

<table>
<thead>
<tr>
<th>Question</th>
<th>Tool/Instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the frame rate?</td>
<td>Core Animation or OpenGL ES Driver instrument</td>
</tr>
<tr>
<td>CPU or GPU bound?</td>
<td>OpenGL ES Driver and Time Profiler instrument</td>
</tr>
<tr>
<td>Any unnecessary CPU rendering?</td>
<td>Time Profiler instrument</td>
</tr>
<tr>
<td>Too many offscreen passes?</td>
<td>Core Animation instrument</td>
</tr>
<tr>
<td>Too much blending?</td>
<td>Core Animation instrument</td>
</tr>
<tr>
<td>Any strange image formats or sizes?</td>
<td>Core Animation instrument</td>
</tr>
<tr>
<td>Any expensive views or effects?</td>
<td>Xcode View Debugger</td>
</tr>
<tr>
<td>Anything unexpected in hierarchy?</td>
<td>Xcode View Debugger</td>
</tr>
</tbody>
</table>
Summary

Core Animation pipeline
Rendering concepts
UIBlurEffect
UIVibrancyEffect
Profiling tools
Case studies
More Information

Jake Behrens
App Frameworks Evangelist
behrens@apple.com

Dave DeLong
Developer Tools Evangelist
delong@apple.com

Documentation
Core Animation

Apple Developer Forums
http://devforums.apple.com
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Speaker</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improving Your App with Instruments</td>
<td>Marina</td>
<td>Tuesday 4:30PM</td>
</tr>
<tr>
<td>Debugging in Xcode 6</td>
<td>Marina</td>
<td>Wednesday 10:15AM</td>
</tr>
<tr>
<td>Writing Energy Efficient Code, Part 1</td>
<td>Russian Hill</td>
<td>Wednesday 10:15AM</td>
</tr>
<tr>
<td>Writing Energy Efficient Code, Part 2</td>
<td>Russian Hill</td>
<td>Wednesday 11:30AM</td>
</tr>
<tr>
<td>Creating Custom iOS User Interfaces</td>
<td>Marina</td>
<td>Wednesday 3:15PM</td>
</tr>
<tr>
<td>Building Interruptible and Responsive Interactions</td>
<td>Presidio</td>
<td>Friday 11:30AM</td>
</tr>
</tbody>
</table>
# Labs

<table>
<thead>
<tr>
<th>Lab</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Animation and Quartz 2D Lab</td>
<td>Graphics and Games Lab A</td>
<td>Tuesday 2:00PM</td>
</tr>
<tr>
<td>Interface Builder and Live Views Lab</td>
<td>Tools Lab C</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>Power and Performance Lab</td>
<td>Core OS Lab B</td>
<td>Wednesday 2:00PM</td>
</tr>
<tr>
<td>Dynamics, View Animations, and Core Animation Lab</td>
<td>Frameworks Lab A</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>Power and Performance Lab</td>
<td>Core OS Lab A</td>
<td>Thursday 3:15PM</td>
</tr>
<tr>
<td>Visual Effects and Appearance Customization Lab</td>
<td>Frameworks Lab A</td>
<td>Friday 9:00AM</td>
</tr>
</tbody>
</table>

---

*Note: The document appears to have a black background with white text, making it difficult to read.*