Performance Optimization on iPhone OS

Erik Neuenschwander Manager, iOS Performance

Ben Weintraub iOS Performance Engineer

Introduction

- Performance is important!
- Key aspect of App Store reviews
- You have the tools and skills to improve performance
- Today: Cover common cases and strategies

What You'll Learn

- How to test and measure performance scenarios
- How to improve key scenarios
 - Speedy launches
 - Smooth scrolling
 - Slim memory footprint
- How to prioritize performance issues

Measuring Performance

Measuring Performance Techniques



- Measure first
- Manual testing
- Automated testing

Measuring Performance Techniques

- What numbers should you measure?
- Everything affects performance



Measuring Performance Techniques

- Guessing is overrated
- Focus on scenarios
 - Measure things in turn
 - Change
 - Re-test
- It has to feel right

Measuring Performance

Tools

- Logging
 - •NSLog(@"That took %g seconds\n", timeWeWaited);
 - Log to file
- Instruments
- Simulator?

Simulator or Device?

- Simulator is fast and easy
 - Uses Mac hardware
 - Unrealistic "iOS" performance profile
- Exception: memory footprint
 - Simulator is a good model
 - More speed and features in Instruments
- Device is the final arbiter
 - Use for all speed-related testing
 - Verify memory fixes perform as expected



Measuring Performance

- Measure early, measure often
- Record "good" results as a baseline
- Turn off or remove logging for submitted app!
- Test every device you will support
- At minimum test the oldest device (today: iPhone 3G)

Key Scenarios

Speedy Launches

Speedy Launches

Importance

- First thing a user experiences or demos
- Very common scenario
 - Non-multitasking "launch"
 - Multitasking "resume"
- Slow performance will cause OS to abort your app
 - Maintain system responsiveness

Speedy Launches Watchdog

- The OS is watching you
- "Wall clock" time to gate
- Values subject to change

	Maximum Time
Launch	20 sec
Resume	10 sec
Suspend	10 sec
Quit	бsec
Complete operation (iOS 4)	10 min

Speedy Launches

Measuring launch

- Test with realistic data sets
- Use Time Profiler Instrument

Speedy Launches Time Profiler

- Collects backtraces at regular intervals
- Useful to see where execution time goes
- Look for work to defer, do on demand
- Necessary work
 - sort by time
 - speed the slowest parts

000	-	Time Profiler Trace		\bigcirc
🔲 💿 🗭 🛃 Compositio	ns ‡	🕑 🕑 🕙 🔷 00:01) "
Record Target		Inspection Range Run 1	of 1 Search	
Instruments	20:00			-
Time Profiler ()				
I I				
lime Profiler	E Call Tree 🗢 Samples			=
Sample Perspective	Running Time Symbol	Name		
O All Sample Counts	2902.0ms 71.9% D		ALayerDelegate) _layoutSublayersOfLayer:] UIKit 💿	6
 Running Sample Times 	2824.0ms 70.0% 💽	▼-[UILayou	tContainerView layoutSubviews] UIKit	
Call Tree	2824.0ms 70.0% D		igationController viewWillLayoutSubviews] UIKit	
Separate by Thread	2824.0ms 70.0% D		avigationController _startDeferredTransitionIfNeeded	
Invert Call Tree	2824.0ms 70.0% D		NavigationController _startTransition:fromViewContro	
Hide Missing Symbols	2819.0ms 69.8% D		JINavigationController _layoutViewController:] UIKit	
Hide System Libraries	2819.0ms 69.8% 🖸		[UINavigationController _computeAndApplyScrollCon	ite
Show Obj-C Only	2819.0ms 69.8% 🖸		[UIViewController contentScrollView] UIKit	_
Flatten Recursion	2819.0ms 69.8%		▼-[UIViewController view] UIKit 🕤	
Call Tree Constraints	2313.0ms 57.3%		CGContextDrawImage CoreGraphics	
Specific Data Mining	181.0ms 4.4%		►+[UIImage imageWithContentsOfFile:] UIKit	
Active Thread	166.0ms 4.1%		►+[NSKeyedUnarchiver unarchiveObjectWithFile:]	
All Threads \$	88.0ms 2.1%		NSSearchPathForDirectoriesInDomains Foundati	lor
An micado V	14.0ms 0.3%		CGBitmapContextCreate CoreGraphics	A
	14.0ms 0.3% 🕒		CGBitmapContextCreateImage CoreGraphics	Y
	C)4	► //.

Demo—Speedy Launches Time Profiler Instrument

Speedy Launches Tips

- System watchdog will terminate slow apps
- Collect trace with the Time Profiler Instrument
- Do less work
 - Defer work out of startup
- Do not block on slow operations
 - Never do networking on your main thread
- Optimize time-consuming activities
 - Launch-time data set needs to have a size limit
- Collect new traces to quantify results

Importance

- Table Views are popular
- Direct Manipulation UI demands responsiveness

Measuring scrolling

- Frames Per Second (FPS—"fips")
- 60 FPS == smooth
- Core Animation Instrument

Core Animation

- Measures FPS in realtime
 - Sub-second requires you to do math
 - Example: 18 frames in 0.3 s == 60 FPS
- Visual hints give insight into important rendering details
 - Color Blended Layers
 - Many others

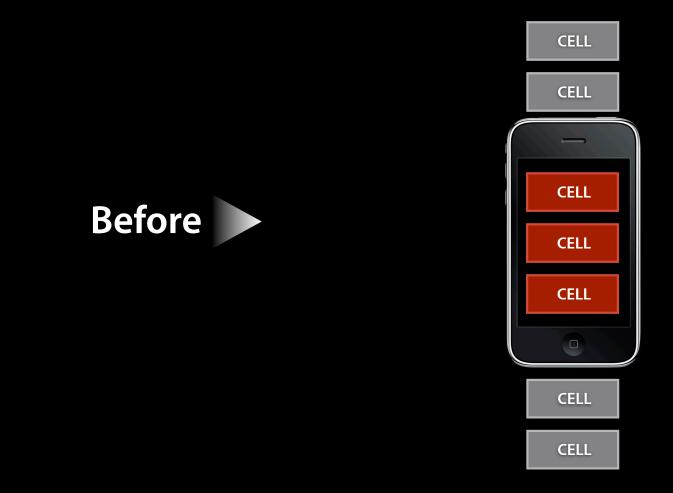
00		N	Core Animation Trace	\bigcirc
Record Target	t) (Image: Construction Range Image: Construction Range	»
Instruments	90:00			
Core Animation ()				
I I				
	1	• •)	<
Core Animation	Table	e ♦ Frames Per Second)	I ► (
Sampling Rate (1/10th sec)	Table	Frames Per Second 0 0)) 4 F =
Core Animation Sampling Rate (1/10th sec)	0 1 2	Frames Per Second 0 0 0)	=
Core Animation Sampling Rate (1/10th sec) Course of the sec of the	Table	Frames Per Second 0 0 0 0)) 4 F =
Core Animation Sampling Rate (1/10th sec) Debug Options Color Blended Layers Color Copied Images Color Copied Layers	Table 0 1 2 3 4	Frames Per Second 0 0 0 1)	↓ ► (
Core Animation Sampling Rate (1/10th sec) Color Body Clayers Color Indeed Layers Color Inmediately Color Inmediately Color Inmediately Color Inmediately	Table 0 1 2 3 4 5	Frames Per Second 0 0 0 0 1 24)	=
Core Animation Compliance (1/10th sec) Debug Options Color Riended Layers Color Copied images Color Misaligned images Color Misaligned images Color Misaligned mages		Frames Per Second 0 0 0 0 1 24 36)	=
Core Animation Sampling Rate (1/10th sec) Color Body Clayers Color Indeed Layers Color Inmediately Color Inmediately Color Inmediately Color Inmediately	Table Table 0 1 2 3 4 5 6 7	Frames Per Second 0 0 0 1 1 24 36 36		=
Core Animation Sampling Rate (1/10th sec) Debug Options Color Rended Layers Color Copied Images Color Copied Images Color Misaligned Images Color Offscreen-Rendered Yellow Color Offscreen-Rendered Yellow Color Offscreen-Rendered Yellow	Table Table Table Table 1 2 3 4 5 6 7 8 8 1 1 1 1 1 1 1 1	Frames Per Second 0 0 0 1 1 24 36 36 28		=
Core Animation Sampling Rate (1/10th sec) Debug Options Color Rended Layers Color Copied Images Color Copied Images Color Misaligned Images Color Offscreen-Rendered Yellow Color Offscreen-Rendered Yellow Color Offscreen-Rendered Yellow	Table 0 1 2 3 4 4 5 6 7 8 9	Frames Per Second 0 0 0 1 24 36 6 28 28)) 4 F
Core Animation Sampling Rate (1/10th sec) Debug Options Color Rended Layers Color Copied Images Color Copied Images Color Misaligned Images Color Offscreen-Rendered Yellow Color Offscreen-Rendered Yellow Color Offscreen-Rendered Yellow	■ Table 0 1 2 3 4 5 6 7 8 9 10	Frames Per Second 0 0 1 24 36 36 28 18 51		
Core Animation Sampling Rate (1/10th sec) Debug Options Color Binded Layers Color Copied Images Color Copied Images Color Misaligned Images Color Offscreen-Renderd Yellow Color Offscreen-Renderd Yellow Color Offscreen-Renderd Yellow	Table 0 1 2 3 4 4 5 6 7 8 9	Frames Per Second 0 0 0 1 24 36 6 28 28		



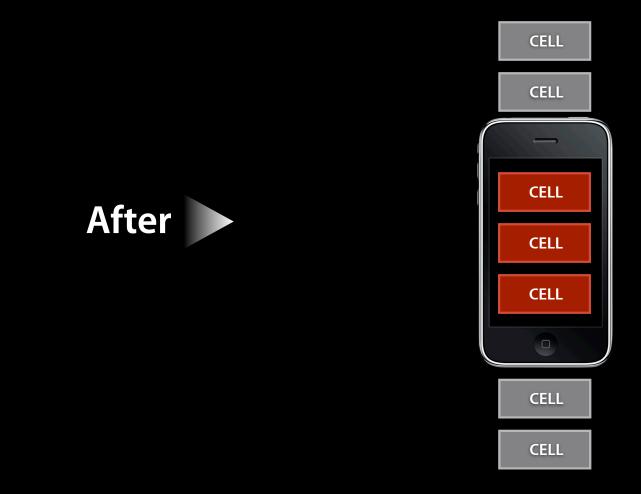
Demo—Smooth Scrolling

Testing and measurement

Smooth Scrolling—Cell Reuse



Smooth Scrolling—Cell Reuse



Demo—Smooth Scrolling Efficient drawing

- Tips
- Test scrolling scenarios
 - Easy to see via manual testing
 - Automated testing now supported
- Measure FPS with Core Animation Instrument
 - 60 FPS == good
- Reuse cells
 - --[UITableViewCell initWithStyle:reuseIdentifier:]
 - --[UITableView dequeueReusableCellWithIdentifier:]
- Stay opaque/green
 - Even for UILabel! OS will handle "blue background" for you

Slim Memory Footprint

Importance

- iOS has no swap
- Under memory pressure, OS will terminate apps
 - Maintain system stability

Jetsam

- Watches memory pressure
- Instant lightweight termination of applications
- More critical with multitasking
 - Apps with small footprint preserved longer
- Stay safe, stay low

Areas to focus on

- Avoidable spikes
- Leaks
- Abandoned memory

Avoidable spikes

- Definition: individual brief allocations all present simultaneously
- Processing large quantities of data
 - Break into independent batches
- Autoreleased objects
 - Reduce object lifetimes

Autorelease

• "Used to avoid worrying about retain/release"

Autorelease

- "Used to avoid worrying about retain/release"
- Used to return objects without retaining them
 Caller will retain if needed
- Objects added to an NSAutoreleasePool
- Pool calls release at the next turn of the runloop
- Deallocation may happen then, if retain count goes to zero
- In the meantime, memory can spike

Demo—Slim Memory Allocations Instrument

Autorelease

- A little more expensive than retain/release
- In designing your code
 - Only use at framework boundaries
 - Explicit retain/release when lifetime is managed by one entity
- When using API
 - Use Instruments to watch memory allocation pattern
 - Wrap intensive autorelease API uses in nested autorelease pool

Slim Memory Leaks

- Definition: allocated memory that is inaccessible
- Leaks Instrument examines the heap for leaked memory
- Most useful when app launched from the Instrument
- Identify moment of allocation
 - Not often the problem, but gives useful context
- Common mistakes
 - Unbalanced retain/release
 - Forget to release property's original value



Slim Memory

Abandoned memory

- Definition: leftover; accessible, but will never be used again
- Allocations Instrument offers Heapshot feature
- Two snapshots in time
- Look at (unexpected) differences



Demo—Slim Memory Allocations Instrument

Slim Memory Tips

- Jetsam will kill your app
- Leaks, abandonments, spikes
- Leaks Instrument
- Allocations Instrument with Heapshot
- Target autorelease use
- Nested autorelease pools

Prioritizing Performance Issues

Prioritization

Where performance fits

- There can be "show stopping" performance issues
- Establishing clear goals early can help consensus

Prioritization

Judging priority

- Frequency
 - Common scenario?
 - Consistently reproducible?
- Severity
 - Unresponsive for multiple seconds?
 - Single-digit FPS?
 - Cause a watchdog or Jetsam?

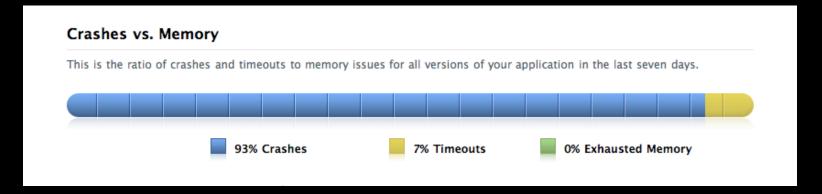
Prioritization

Judging priority

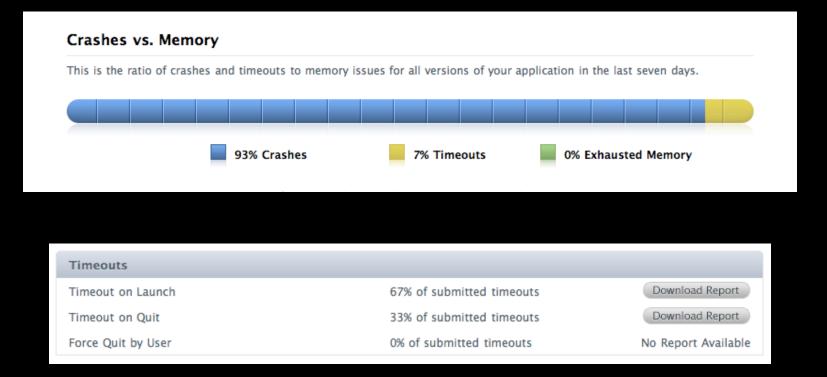
- Watchdog and Jetsam terminations look like a crash
- Top "crash" may not be a crash!

iTunes Connect

iTunes Connect Weighting issues



iTunes Connect Watchdog reports

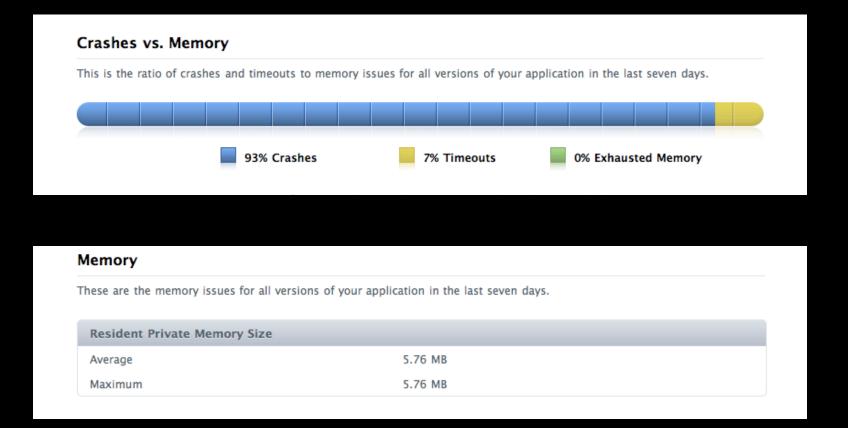


Memory

These are the memory issues for all versions of your application in the last seven days.

Resident Private Memory Size		
Average	5.76 MB	
Maximum	5.76 MB	

iTunes Connect Jetsam reports



Wrapping Up

Review

- Performance is critical
- Measure with Instruments
- Performance testing is best done on an older device
- Key Areas
 - Launch
 - Scrolling
 - Memory
- Develop clear performance goals
- Visit iTunes Connect for performance-related reports

More Information

Bill Dudney Application Frameworks Evangelist dudney@apple.com

Michael Jurewitz Developer Tools and Performance Evangelist jurewitz@apple.com

Documentation iPhone OS Performance Overview http://developer.apple.com/iphone/library/documentation/Performance/Conceptual/ PerformanceOverview/

Apple Developer Forums

http://devforums.apple.com/

Related Sessions

Advanced Performance Optimization on iPhone OS, Part 1	Mission Thursday 3:15PM
Advanced Performance Optimization on iPhone OS, Part 2	Mission Friday 11:30AM
Optimizing Core Data Performance on iPhone OS	Presidio Thursday 4:30PM
Advanced Memory Analysis with Instruments	On Video
Advanced Performance Analysis with Instruments	On Video
Building Animation Driven Interfaces	On Video

Labs

iPhone OS Performance Lab	Developer Tools Lab A Thursday, 4:30PM
Animation Lab	Application Frameworks Lab C Thursday, 4:30PM
OpenGL ES Lab	Graphics and Media Lab A Thursday, 9:00AM
iPhone OS Performance Lab	Developer Tools Lab A Friday, 9:00AM



