Technology and Refinements
iOS  macOS  watchOS  tvOS
High Sierra will be the last macOS release to support 32-bit apps without compromises.
64-bit Requirements

January 2018: New apps
June 2018: All apps and updates
Swift Playgrounds
1 Million
Users of Swift Playgrounds
Users Outside U.S.:

- United States
- China
- International

65%
Users Outside U.S. 65%

- China
- United States
- International
Line-by-line highlighting
Content update notifications
Add new Playground pages
Improved error messages
MapKit support

Goal: Use a `for` loop to repeat a sequence of commands.

In this puzzle, you must collect four gems that are located in the same relative locations around a square. You’ll create a `loop` that repeats the code below for each of the sides to solve the entire puzzle.

1. Drag a `for` loop from the code library, then drop it above the existing code.
2. Tap the bottom curly brace to select the loop.
3. Tap and hold that curly brace, then drag it downward to pull the existing code into the loop.

```
for i in 1 ... 4 {
    moveForward()
    collectGem()
    moveForward()
    moveForward()
    moveForward()
    moveForward()
    turnRight()
}
```
Line-by-line highlighting
Content update notifications
Add new Playground pages
Improved error messages
MapKit support
Swift Playgrounds 1.5
Program

Using the turtle prototype you have just built, create a function named `moveTurtle` that:

1. Moves your turtle forward for a distance of 40 cm and stops at the "fruit" without using a sensor.

Example

To control motors, remember the `motorOn` function that was covered in the previous chapters. To control two motors simultaneously, you can explore the `move` function. For example:

```plaintext
move(forRotations: 1.0, leftPort: .b, rightPort: .c, leftPower: 100.0, rightPower: 100.0)
```

Analyze

Use the information found in Port View and Data View to determine the speed of your turtle.

```swift
func TurtleMove() {
    ev3.move(forSeconds: 23, leftPort: .b, rightPort: .c, leftPower: 50, rightPower: 50)
}
TurtleMove()
```
The best way to control robots and drones with code you write yourself
Available Today
In the App Store
Available

This Fall

Swift Playgrounds 2
Playground Feeds
Author-hosted content
Subscription model
Update notifications
Localized content
Available This Fall

Integrated API documentation
Swift 4 and Swift 3.2
iOS 11 SDK
Camera and Augmented Reality
Available This Fall

Eight additional localizations
Source Editor
// A model of a solar system, including a central #Star with orbiting #Planets.
public class SolarSystem {

    // Create the sun.
    let sun = Star(name: "Sun", color: ▒)

    // Create the planets.
    let mercury = Planet(name: "Mercury", color: ▒)
    let venus = Planet(name: "Venus", color: ▒)
    let earth = Planet(name: "Earth", color: ▒)
    let mars = Planet(name: "Mars", color: ▒)
    let jupiter = Planet(name: "Jupiter", color: ▒)
    let saturn = Planet(name: "Saturn", color: ▒)
    let uranus = Planet(name: "Uranus", color: ▒)
    let neptune = Planet(name: "Neptune", color: ▒)

    // Create arrays for the objects in our SolarSystem.
    let planets: [Planet]
    let distantObjects: [TransNeptunianObject]

    public init()
    {
        // create the planets and distant objects.
        planets = [mercury, venus, earth, mars, jupiter, saturn, uranus, neptune]
        distantObjects = SolarSystem.loadAndCreateDistantObjects(parentStar: sun)

        // add Earth's moon.
        earth.addMoon(Moon(name: "Moon", color: ▒))
    }
}
SolarSystem App - Overview

About the app

“SolarSystem” provides a way for users to explore our solar system in a fun and interactive way. All the technical information is derived from various NASA sites, including [Planets](https://solarsystem.nasa.gov/planets) and [Moons](https://solarsystem.nasa.gov/planets/solarsystem/moons).

The SolarSystem app is provided under the [Apache License 2.0](https://github.com/apple/swift/blob/master/LICENSE.txt).

---

## Background

### Definition

The Solar System is the gravitationally bound system comprising [the Sun](#the-sun) and the objects that orbit it, either directly or indirectly. Of those objects that orbit [the Sun](#the-sun) directly, the largest eight are the [planets](#planets), with the remainder being significantly smaller objects, such as dwarf planets and small Solar System bodies. Of the objects that orbit [the Sun](#the-sun) indirectly, the moons, two are larger than the smallest planet, [Mercury](#mercury).

### History

The Solar System formed 4.6 billion years ago from the gravitational collapse of a giant interstellar cloud.
public init() {

    // create the planets and distant objects.
    planets = [mercury, venus, earth, mars, jupiter, saturn, uranus, venus]
    distantObjects = SolarSystem.loadAndCreateDistantObjects(parentStar: sun)

    // add Earth's moon.
    earth.addMoon(Moon(name: "Moon", color: ))

    // add Jupiter's 67 moons 😊!
    if let path = Bundle.main.path(forResource: "MoonsOfJupiter", ofType: "txt") {
        do {
            // grab all the names of Jupiter's moons, which are separated by newlines.
            let data = try String(contentsOfFile: path, encoding: .utf8)
            let moonNames = data.components(separatedBy: .newlines)

            // loop over all of Jupiter's moon names...
        }
    }
}
300+ new diagnostics, analyzers, and fix-its
3x FASTER
File opening

60 FPS
Scrolling
3x FASTER File opening
60 FPS Scrolling
50x FASTER Jump to line
// Create arrays for the objects in our Solar System.
let planets: [Planet]
let distantObjects: [TransNeptunianObject]

public init()
{

    // create the planets and distant objects.
    planets = [mercury, venus, earth, mars, jupiter, saturn, uranus, venus]
distantObjects =SolarSystem.loadAndCreateDistantObjects(parentStar: sun)

    // add Earth's moon.
    earth.addMoon(name: "Moon", color: [])

    // add Jupiter's 67 moons 😊!
    if let path = Bundle.main.path(forResource: "MoonsOfJupiter", ofType: "txt") {
        do {
            // grab all the names of Jupiter's moons, which are separated by newlines.
            let data = try String(contentsOfFile: path, encoding: .utf8)
            let moonNames = data.components(separatedBy: newline)

            // loop over all of Jupiter's moon names.
            for moonName in moonNames {
                jupiter.addMoon(moonName: moonName, color: [])
            }
        } catch {
            /* shouldn't end up here. */
        }
    }

    // Calculates the coordinate of a given #Planet, at a particular date (which includes time),
    relative to the Sun.
// Create arrays for the objects in our SolarSystem.
let planets: [Planet]
let distantObjects: [TransNeptunianObject]

public init()
{
    
    // create the planets and distant objects.
    planets = [mercury, venus, earth, mars, Jupiter, saturn, uranus, venus]
    distantObjects = SolarSystem.loadAndCreateDistantObjects(parentStar: sun)

    // add Earth's moon.
    earth.addMoon(name: "Moon", color: [])

    // add Jupiter's 67 moons 😳
    if let path = Bundle.main.path(forResource: "MoonsOfJupiter", ofType: "txt") {
        do {
            // grab all the names of Jupiter's moons, which are separated by newlines.
            let data = try String(contentsOfFile: path, encoding: .utf8)
            let moonNames = data.components(separatedBy: newline)

            // loop over all of Jupiter's moon names.
            for moonName in moonNames {
                jupiter.addMoon(name: moonName, color: [])
            }
        } catch { /* shouldn't end up here. */ }
    }

    // Calculates the coordinate of a given #Planet, at a particular date (which includes time),
    // relative to the Sun.
Refactoring

All new engine and workflow for Swift, Objective-C, C, and C++
Refactoring
Demo
let planets: [Planet]
let distantObjects: [MinorMoon]

private static func loadAndCreateDistantObjects(parentStar: Star) -> [MinorMoon] {
    var distantObjects: [MinorMoon] = []
    if let path = Bundle.main.path(forResource: "DistantObjects", ofType: "txt") {
        let distantObjectNames = data.components(separatedBy: .newlines)
        let defaultShape = MinorMoon.shape.spherical

        // loop over all of distant object names.
        for distantObjectName in distantObjectNames {
            distantObjects.append(MinorMoon(name: distantObjectName, shape: defaultShape, parentStar: parentStar))
        }
    }
    return distantObjects
}

-(void)addDistantObject:MinorMoon*object {
    NSMutableArray<MinorMoon> mutableObjects = self.distantObjects.mutableCopy;
    [mutableObjects addObject:object];
    NSArray<MinorMoon> *newDistantObjects = [NSArray arrayWithArray:mutableObjects];
}
Transformations

- Add missing protocol requirements
- Generate missing implementation stubs
- Add missing overrides for abstract methods
- Extract to local variable
- Extract method / expression
- Expand ‘default’ in switch statements
- Convert if / else to / from switch statement
- Wrap string in NSLocalizedString macro
Open Source Transformation Engine
Swift
250,000
Swift apps on the App Store
Source compatibility

- Smaller binaries
- Improved Package Manager
- Memory exclusivity
- Smart keypaths

Dictionary enhancements

- Easier to use String
- Class-constrained protocols
- Encoding and decoding
- Numeric protocols
- Improved generics
String
String
Easier to use
More Unicode correct
Fast
var s = "Hello!"
for c in s.characters { print(c) }

s.characters.last == "!"

s.characters.index(of: "!")

String(s.characters.dropLast(1)) + " world!"
```swift
var s = "Hello!"
for c in s.characters { print(c) }

s.characters.last == "!"
s.characters.index(of: "!")

String(s.characters.dropLast(1)) + " world!"
```
var s = "Hello!"
for c in s.characters { print(c) }

s.characters.last == "!"

s.characters.index(of: "!")

String(s.characters.dropLast(1)) + " world!"
```swift
var s = "Hello!"
for c in s { print(c) }

s.last == "!"

s.index(of: "!")

s.dropLast(1) + " world!"
```
String simplified

Strings are now range-replaceable, bidirectional Collections

```swift
var s = "Hello!"
for c in s { print(c) }

s.last == "!"

s.index(of: "!")

s.dropLast(1) + " world!"
```
let poem = """'
'Twas brillig, and the slithy toves
  Did gyre and gimble in the wabe:
All mimsy were the borogoves,
  And the mome raths outgrabe.
"""
let index = poem.index(of: ",")!
let substring = poem[..<poem.index]  // 'Twas brillig

Simple and Expressive Slicing
Swift 3

"\n\n\n\n\n\n\n".count == 1
Unicode 9 Grapheme Breaking
"🇸🇸🇯🇯#

.count == 4

Unicode 9 Grapheme Breaking
3.5x FASTER

String processing

English
French
German
Spanish
2.5x FASTER

String processing

Simplified Chinese
Japanese
public struct Farm {
    public let name: String
    public let location: Location
    public let animals: [Animal]
}
public struct Farm {
    public let name: String
    public let location: Location
    public let animals: [Animal]
}
public struct Farm: Codable {
    public let name: String
    public let location: Location
    public let animals: [Animal]
}
```swift
public struct Farm: Codable {
    public let name: String
    public let location: Location
    public let animals: [Animal]
}

let farm = Farm(name: "Old MacDonald's Farm",
                 location: Location(latitude: 51.6216,
                                     longitude: 0.2692),
                 animals: [.cow, .dog, .chicken])
```
let farm = Farm(name: "Old MacDonal's Farm",
    location: Location(latitude: 51.6216,
                      longitude: 0.2692),
    animals: [.cow, .dog, .chicken])
let farm = Farm(name: "Old MacDonald's Farm",
    location: Location(latitude: 51.6216,
    longitude: 0.2692),
    animals: [.cow, .dog, .chicken])

let payload: Data = try JSONEncoder().encode(farm)
let payload: Data = try JSONEncoder().encode(farm)
let payload: Data = try JSONEncoder().encode(farm)

{
    "name": "Old MacDonald's Farm",
    "location": {
        "longitude": 0.2692,
        "latitude": 51.6216
    },
    "animals": [0, 1, 2]
}
let payload: Data = try JSONEncoder().encode(farm)

{
  "name":"Old MacDonald's Farm",
  "location": {
    "longitude": 0.2692, "latitude": 51.6216
  },
  "animals": [0, 1, 2]
}

let farm = try JSONDecoder().decode(Farm.self, from: payload)
let coordinates = farm.location
Easy to Adopt
Swift Language Version
Swift 3.2

Swift Language Version

Swift 4.0

✅ Swift 4.0
One compiler

Two Swift language modes

Mix-and-match 3.2 and 4.0 targets
Swift 3.2
Build projects with no modifications
Use almost all new Swift features
Use new APIs from SDK
No Swift-improvements to existing APIs
Swift 4.0
Use all the new language features
Improved performance
SDK improvements for Swift
Some migration required
Building Large Projects
40% faster
Building large mix-and-match Swift/Objective-C projects
2x faster

Building projects using multiple WMO targets
Core Technologies
Open Quickly first result

35x FASTER
35x FASTER
Open Quickly first result

50x FASTER
Large project search
Indexing While Building

Building ‘SolarSystem’ (and indexing) ...
Built in Swift
Layers atop llbuild
Modern architecture
Process separation
Unified dependency graph
Improved configuration analysis
Increased parallelism and caching
Incremental build
Incremental build
Incremental build

Up to 2.5x Faster build operations
Incremental build

Up to 40% Faster file compilation
Preview in Xcode 9
Opt-in via Workspace Settings
Default build system soon
Source Control
2/3 of all pull request related activity on GitHub happens from a Mac.
GitHub
Integrated into Xcode
3lvis/Networking

Easy HTTP Networking in Swift a URLSession wrapper with image caching support

- Swift
  - 76
  - 1007
  - README

- Done
- Clone
Networking was born out of the necessity of having a simple networking library that doesn’t have crazy programming abstractions or uses the latest reactive programming techniques, but just a plain, simple and convenient wrapper around URLSession that supports common needs such as faking requests and caching images out of the box. A library that is small enough to read in one go but useful enough to include in any project. That’s how Networking came to life, a fully tested library for iOS, tvOS, watchOS and OS X that will always be there for you.

- Super friendly API
Demo
View Controllers in View Debugger
View Controllers in View Debugger
SpriteKit Scenes in View Debugger
SpriteKit Scenes in View Debugger
SceneKit Scenes in View Debugger
SceneKit Scenes in View Debugger
Address Sanitizer

Thread Sanitizer

Address Sanitizer
Sanitizers

- Thread Sanitizer
- Address Sanitizer
- Undefined Behavior Sanitizer
- Main Thread API Checker
Undefined Behavior Sanitizer
import Cocoa

class ViewController: UIViewController {

  @IBAction func buttonClicked(_ sender: UIButton) {
    DispatchQueue.global().async {
      sender.removeFromSuperview()
    }
  }

  // NSView.removeFromSuperview() must be called from main thread only

Testing and Continuous Integration
Xcode Server Built In
Xcode Server Built In
UI Testing Multiple Applications
UI Testing Multiple Applications
3x FASTER

Query performance
Query performance

3x FASTER

Using first match API

10x FASTER
Parallel Device and Simulator Testing

```bash
> xcodebuild test
  -destination "name=iPad Pro"
  -destination "name=iPhone7"
```
Parallel Device and Simulator Testing

```bash
> xcodebuild test
  -destination "name=iPad Pro"
  -destination "name=iPhone7"
```
Multiple Booted Simulator Devices
Simulator bezel
Hardware controls
Support for edge swipes
Fully resizable
Touch Bar controls
Wireless Development
Drag and Drop
Easy to adopt
Flexible and customizable
Fully multi-touch enabled
Secure by design
Automatic for Text and Web
Automatic for Text and Web
// Begin Drag
let dragData = self.data(at: sourceIndexPath)
let itemProvider = NSItemProvider(object: dragData)
return [UIDragItem(itemProvider: itemProvider)]
// Begin Drag
let dragData = self.data(at: sourceIndexPath)
let itemProvider = NSItemProvider(object: dragData)
return [UIDragItem(itemProvider: itemProvider)]

// Perform Drop
coordinator.session.loadObjects(ofClass: MyDataType.self)
{ (data) in
    self.insertData(data, at: destinationIndexPath)
collectionView.reloadData()
}
Gestures
Lift animations
Multi-item drags
Previews
Badges
Set-down animations
Customizable Appearance
Custom Data Types
Reordering Support
Reordering Support
Cross-Process
Cross-Process
In-App Multi-Touch
In-App Multi-Touch
System-Wide Multi-Touch
System-Wide Multi-Touch
Demo
Megan 8:30 AM
Hope you have a great flight! See you tonight!

Gram  Yesterday
Thinking about you a lot this week

Samantha  Saturday
How's the new place in San Francisco?

Mom  Friday
Looking forward to seeing you soon!
// Adopt Large Titles
navigationBar.prefersLargeTitles = true

// Automatically Choose Large Title
navigationItem.largeTitleDisplayMode = .automatic

// Adopt Unified Search Bar
navigationItem.searchController = searchController
Dynamic Type
Type filtering
Custom actions
Document creation
Appearance customization
Type filtering
Custom actions
Document creation
Appearance customization
In-app document sharing
Supports iOS and macOS
Leverages NSFileCoordinator
NSDocument support
Size classes
Auto layout
Default storyboards
App strip
Apps in the transcript
Direct send
40 million songs
Play songs and playlists
Listen to radio stations
Photos Project
Extensions
Extension UI
Hosted in Photos
Print Products
Wix.com

Web Photo Albums
Camera
Camera detects QR codes
Link into apps via Universal Links
HEVC and HEIF
H.264 – 2003
Complex Assets
High Dynamic Range
Wide Color
High Resolution
HEVC

2x

Compression

Performance

Scalability
HEVC

2x Compression

A10 Performance

Scalability
HEIF

Compound Assets

Extensible and Rich

Modern Container

Modern Container

Compound Assets

Extensible and Rich

Modern Container
```swift
var asset = AVAsset(url: URL(fileURLWithPath: "hevc.mov"))
if !asset.isPlayable {
    asset = AVAsset(url: URL(fileURLWithPath: "h264.mov"))
}
```
Ecosystem

HEIF/HEVC
Wide Camera

Tele Camera
CIIImage(contents:)

CIIImage(contents: options: [kCIIImageAuxiliaryDepth:YES])

HEIF
Use Depth to create custom filters
Use Depth to create custom filters
Use Depth to create custom filters
Use Depth to create custom filters
Streaming Depth API

- Camera Stream
- Hardware Depth Engine
- AVDepthData
Vision
Bring powerful computer vision capabilities to your apps.
Face and Landmark Detection
Rectangle Detection
Text Detection

Global Accessibility Awareness Day
Text Detection
Barcode Detection
Object Tracking
Object Tracking
Vision and Core ML
Demo
Architecture

Applications
Architecture

Applications

- Vision
- Natural Language Processing

Domain Specific Frameworks
Core ML
Performance

Graph Optimization
Performance

Automatic Hardware Selection

GPU

CPU
Simple
Fast
Energy efficient
iOS, macOS, watchOS, tvOS
Metal
Your Application

3D Graphics and GPU Compute

Metal

GPU
1.7 Million

iOS apps using system frameworks calling Metal
“Metal’s richer feature set and lower overhead have allowed us to bring cutting edge games to the Mac with frame rates and effects that simply weren’t possible before.”

Ian Bullock, Head of Technology
Feral Interactive
900 Million
Apple products supporting Metal
GPU-Driven Rendering
Significant Overhead Per Frame

Your Application

OpenGL

GPU

Time spent per frame
Dramatically Reduced CPU Costs
Even More Efficiency

Your Application

Time spent per frame

Metal 2

GPU
Indirect argument buffers
Raster order groups
Quad and SIMDGroup data exchange
Arrays of samplers
Uniform variables
Metal Render Pass

- Resource Arguments
- Graphics State
Metal Render Pass

- Resource Arguments
- Graphics State

Metal Resources
- material constants
- surface texture
- specular texture
- texture sampler

Setting all arguments every draw call
Material constants
Surface texture
Specular texture
Texture sampler
Initialize argument buffers once during set up
Set argument buffer per draw call

Metal Render Pass

Resource Arguments

Argument Buffer

Graphics State

Metal Resources

- material constants
- surface texture
- specular texture
- texture sampler
Initialize argument buffers once during set up
Set argument buffer per draw call
CPU Time Per Draw Call

Measured on iPhone 7 Plus

Time (microseconds)

Resources | 2 | 8 | 16
---|---|---|---
Without argument buffers | | | 8
With argument buffers | | |
CPU Time Per Draw Call

Measured on iPhone 7 Plus

Without argument buffers

With argument buffers
Unified API across macOS, iOS, and tvOS
Maintains compatibility wherever possible
Exposes key hardware features
Metal Performance Shaders
Resource Heaps
Linear Textures
Dual Source Blending
Allocate large buffers of memory
Ultra fast resource re-allocation
Abstract GPU hardware differences
Building block for MPS performance
Separate Allocations
Separate Allocations

- Memory Allocation for A: Texture A
- Memory Allocation for B: Texture B
- Memory Allocation for C: Texture C
Memory Allocation for Heap

Separate Allocations:
- Memory Allocation for A
  - Texture A
- Memory Allocation for B
  - Texture B
- Memory Allocation for C
  - Texture C

Using Heaps:
- Memory Allocation for Heap
Memory Allocation for Heap

Separate Allocations
- Memory Allocation for A: Texture A
- Memory Allocation for B: Texture B
- Memory Allocation for C: Texture C

Using Heaps
- Memory Allocation for Heap: Texture A
- Memory Allocation for Heap: Texture B
- Memory Allocation for Heap: Texture C
Memory Allocation for Heap

Separate Allocations
- Memory Allocation for A
  - Texture A
- Memory Allocation for B
  - Texture B
- Memory Allocation for C
  - Texture C

Using Heaps
- Memory Allocation for Heap
  - Texture A
  - Texture B
  - Texture C
Memory Allocation for Heap

Using Heaps

Separate Allocations

Memory Allocation for A
Texture A

Memory Allocation for B
Texture B

Memory Allocation for C
Texture C

Super fast re-allocation within heap
Memory Allocation for Heap

Using Heaps

Separate Allocations

Texture A

Texture B

Texture C

Super fast re-allocation within heap

Re-interpret existing memory

Texture A

Texture B

Texture C

Texture D
Memory Allocation for Heap

Separate Allocations
- Memory Allocation for A
  - Texture A
- Memory Allocation for B
  - Texture B
- Memory Allocation for C
  - Texture C

Using Heaps
- Memory Allocation for Heap
  - Texture C
Memory Allocation for Heap

Separate Allocations

Memory Allocation for A
Texture A

Memory Allocation for B
Texture B

Memory Allocation for C
Texture C

Using Heaps

Memory Allocation for Heap
Texture C

Memory savings when resources not needed simultaneously
Unified graphics and compute
MPS-optimized kernels and API
- Image processing
- Linear algebra
- Math operations
Machine Learning Acceleration
GPU-accelerated primitives
New MPS graph API for CNN
Efficient integration for existing Metal code
Applications

- Vision
- Natural Language Processing

Core ML

Accelerate
- MPS

Domain Specific Frameworks
ML Framework
ML Performance Primitives

Applications
VR
Enabling VR development
360° video editing
3D content creation
Optimized low-latency support for HMD
Metal viewport arrays
System trace VR timelines
Frame debugger per-eye visualization
Available Today

Beta for macOS
HTC Vive headset and controllers
Traditional Rendering
60 frames / second

16.7 ms

GPU
Frame 0
Frame 1
Frame 2
Traditional Rendering

60 frames / second

16.7 ms

Available time for your app’s rendering
VR Rendering
90 frames / second

11.1 ms

Available time for your app's rendering
VR Rendering
90 frames / second

~10 ms
~1 ms

GPU
Available time for your app’s rendering

VR compositor
VR Rendering

~100 frames / second

~10 ms

~1 ms

Available time for your app’s rendering

GPU

VR compositor
External GPU Support
Metal device selection API
GPU connection notification API
Improved Metal driver robustness
macOS Developer Beta

Developer Kit
Available Today

Thunderbolt 3 enclosure
AMD Radeon RX 580
USB-C hub
HTC Vive discount
Developer Program members $599
Advanced Optimization Tools
GPU counter profiling
Pipeline metrics and remarks
Metal resource Quick Look
Pixel and vertex data inspection
Advanced filtering
New Metal capture APIs
### Potential Hotspots/Bottlenecks

- Texture cache miss rate is very high - 91%
- Texture unit is very stalled - 98%
- Fragment shader is texture sampling bound

### Counters

<table>
<thead>
<tr>
<th>Counters</th>
<th>Draw 284</th>
<th>Median</th>
<th>Max</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Time</td>
<td>1.76 ms</td>
<td>10.27 μs</td>
<td>1.76 ms</td>
<td>52.53 ms</td>
</tr>
<tr>
<td>Vertices</td>
<td>6</td>
<td>744</td>
<td>83,022</td>
<td>4,822,386</td>
</tr>
<tr>
<td>Vertices Reused</td>
<td>0%</td>
<td>33.34%</td>
<td>81.2%</td>
<td></td>
</tr>
<tr>
<td>Vertices Rendered</td>
<td>100%</td>
<td>66.67%</td>
<td>101.05%</td>
<td></td>
</tr>
<tr>
<td>Pixel per Vertex</td>
<td>175,104</td>
<td>20.95</td>
<td>692,160</td>
<td></td>
</tr>
<tr>
<td>Vertices per Second</td>
<td>539,921,000</td>
<td>15,731,200</td>
<td>1,839,880,000</td>
<td></td>
</tr>
<tr>
<td>Tessellation Busy</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td><strong>Vertex Shader</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex Shader Time</td>
<td>0.01%</td>
<td>37%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>VS Invocations</td>
<td>6</td>
<td>272</td>
<td>78,394</td>
<td>2,475,048</td>
</tr>
<tr>
<td>VS ALU Active Time</td>
<td>0.01%</td>
<td>0.35%</td>
<td>34.27%</td>
<td></td>
</tr>
<tr>
<td>VS Stall Time</td>
<td>0%</td>
<td>39.36%</td>
<td>90.57%</td>
<td></td>
</tr>
</tbody>
</table>
iPhone 6s and later
iPad Pro and later
Arkit
Hundreds of Millions

ARKit Capable Devices
Visual-Inertial Odometry
ARSessonConfiguration

ARWorldTrackingSessionConfiguration

ARSesson
ARSession

ARSessionDelegate or currentFrame

ARFrame
Application

ARKit

CoreMotion

Motion Sensors

Camera

AVFoundation

Metal

GPU

Rendering/Engine
Rendering/Engine
Metal 2
Hands On
Over 100 sessions
Over 200 labs
Over 1000 Apple engineers