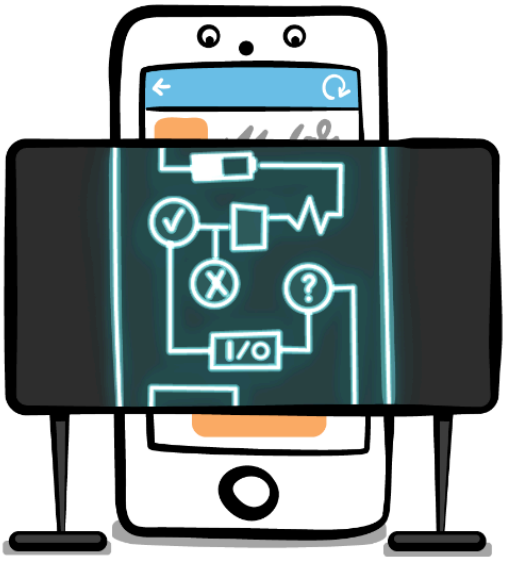
PRACTICAL INSTRUMENTS



HANDS-ON CHALLENGES

Practical Instruments

Luke Parham

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

Table of Contents: Extended

Challenge #3: Tracking Frame Drops

By Luke Parham

Run Loops

As you saw in the last video, your app renders to the screen 60 times a second. That means you have 1/60th of a second, or 16.67 milliseconds to do any given task.

Stepping back for a moment, all iOS apps have the concept of run loops, the most prominent one being the ***Main Run Loop***.

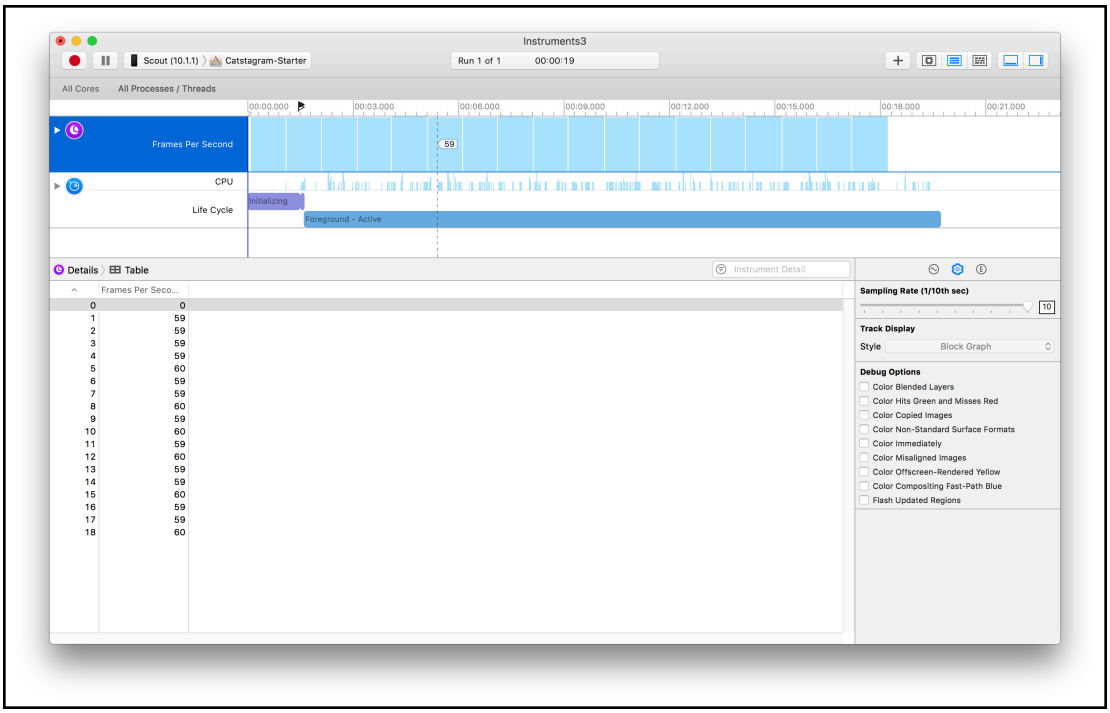
You can think of this as a big while loop. During the course of your app being run, events, as well as blocks of work, get collected and dispatched onto the main dispatch queue. These events and tasks can be things like touch events, notifications, timers and **Core Animation** transactions.

Each thread actually has its own run loop, and the main run loop is responsible for running all the work on the main thread.

At the end of each "turn" of the main run loop, a Vsync event occurs, aka the current frame is drawn to the screen. If any work submitted to the main queue takes longer than 16ms, or really much longer than 5 to 10 milliseconds depending on what else is going on in the system, the run loop won't be available when it comes time to draw the current frame. This means the system will skip that frame and hopefully catch the run loop on the next go-round.

CADisplayLink

As you'll see in the Core Animation video, Instruments does offer an out of the box solution for tracking down frame drops.



This is fine, but it can be a bit course grained and, in my experience, a little misleading.

Luckily, the fine folks over at Facebook came up with a pretty elegant solution for seeing exactly when and by how big a margin you're dropping frames in your app.

If you've never used `CADisplayLink` before, its a built-in tool you can use to sync things up with the Vsync events in your app. Usually it's useful for things like animations, but it can also be and excellent aid in your quest for perfect performance.

To get started, head to ***AppDelegate.swift*** and add the following property to the top.

```
var lastTime: CFTimeInterval = 0.0
```

This will come in handy in a moment when you start tracking the timing of your Vsync events.

Next, add the following two lines to the end of `application(_:didFinishLaunchingWithOptions:)`, right before the return.

```
let link = CADisplayLink(target: self, selector:
#selector(AppDelegate.update(link:)))
link.add(to: RunLoop.main, forMode: .commonModes)
```

Here, you create a new `CADisplayLink` and add it to the main run loop.

Now, add the following method so the display link will have something to call on

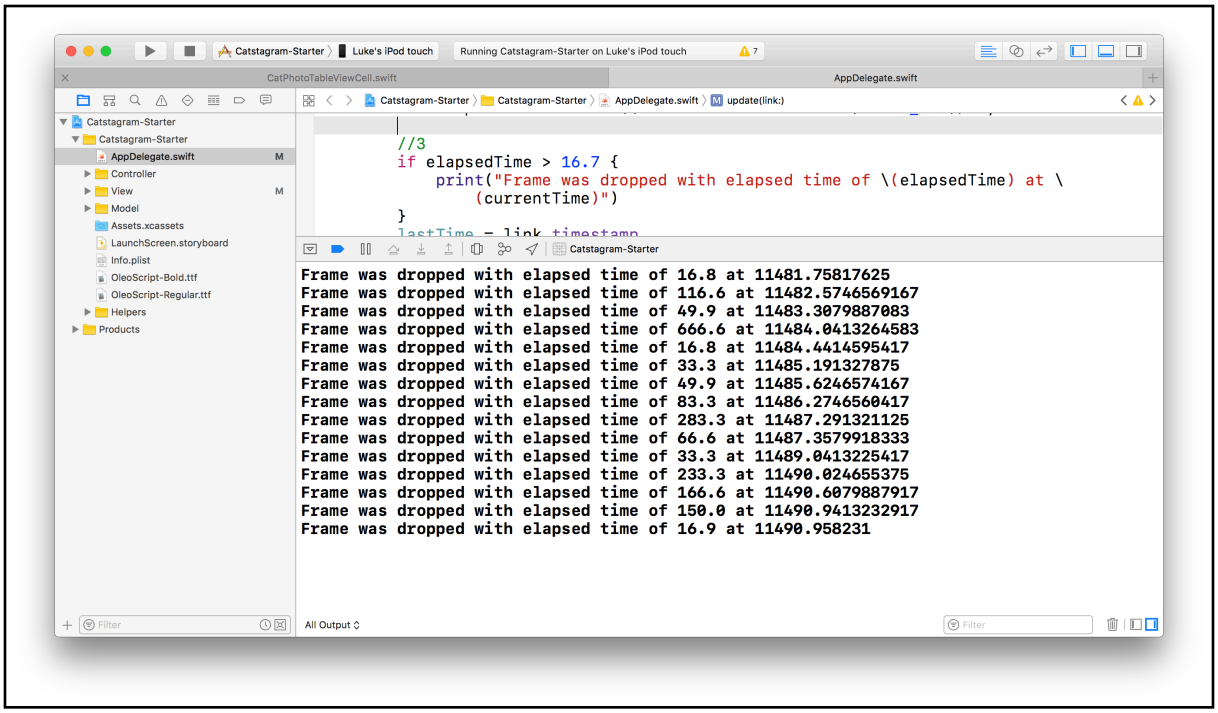
every turn of the run loop.

```
func update(link: CADisplayLink) {
    //1
    if lastTime == 0.0 {
        lastTime = link.timestamp
    }

    //2
    let currentTime = link.timestamp
    let elapsedTime = floor((currentTime - lastTime) * 10_000)/10;

    //3
    if elapsedTime > 16.7 {
        print("Frame was dropped with elapsed time of \(elapsedTime) at \
(currentTime)")
    }
    lastTime = link.timestamp
}
```

With this, go ahead and build and run to check it out. Here's what my console looks like running this on an iPod 5G. As you can see, there's a heckuva lot of frame drops up in there.



Making This Happen Less

That should be your goal when you're profiling your UI's performance. Technically, if you're seeing any drops in here that means you're not consistently hitting that 60 fps mark that sets apart the good from the great apps. That being said, this is really hard, especially on older devices!

That being said, it's good to continually be trying to see as few of these as you can.

Just to show you a contrast, go ahead and go to ***CatPhotoTableViewCell.swift***. Here, you'll see I put the `UIImageView`'s back that were slowing down the UI in the previous video. Swapping these out for the `AsyncImageView`s again will give you a good idea how many fewer frames you can drop with some optimizations.

First, go to property definitions and replace

```
var userAvatarImageView: UIImageView
var photoImageView: UIImageView
```

with

```
var userAvatarImageView: AsyncImageView
var photoImageView: AsyncImageView
```

And then, go down to `init(style:reuseIdentifier:)` and replace these two initilization lines:

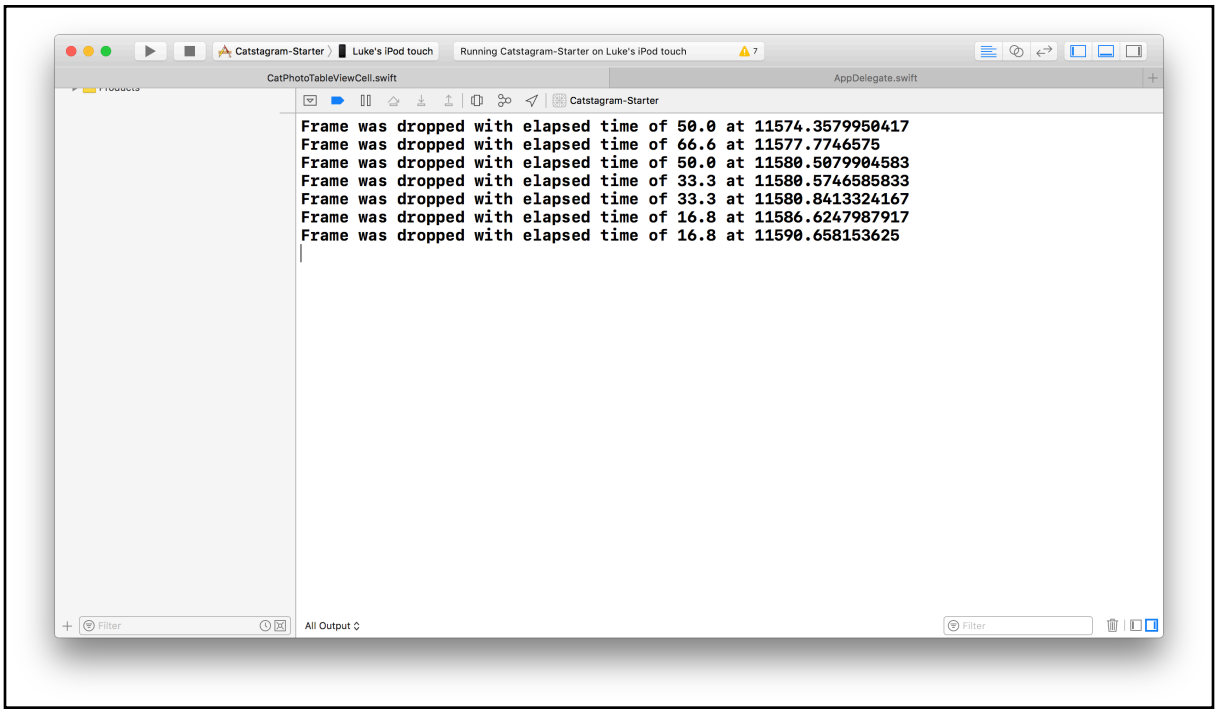
```
userAvatarImageView = UIImageView()
photoImageView      = UIImageView()
```

with

```
userAvatarImageView = AsyncImageView()
photoImageView      = AsyncImageView()
```

After that, go ahead and build and run again, this time you should see a lot less of these frame drop events!

Here's my iPod 5G again, dropping fewer frames.



If you look close, you'll notice the first list of drops had gaps as long as 666 (yikes) milliseconds. That means 41 frames were being dropped at a time before! Now most drops are in the 30~50ms range.