Stanford CS193p

Developing Applications for iOS
Fall 2017-18

Today

- View Controller Lifecycle
  Keeping track of what's happening in your Controller as it goes through its lifetime
- Scroll View
  A UIView that lets you scroll around and zoom on other UIViews

View Controller Lifecycle

- View Controllers have a "Lifecycle"
  A sequence of messages is sent to a View Controller as it progresses through its "lifetime".
- Why does this matter?
  You very commonly override these methods to do certain work.
- The start of the lifecycle …
  Creation.
  MVCs are most often instantiated out of a storyboard (as you've seen).
  There are ways to do it in code (rare) as well which we will cover later in the quarter.
- What then?
  Preparation if being segued to.
  Outlet setting.
  Appearing and disappearing.
  Geometry changes.
  Low-memory situations.

View Controller Lifecycle

- Primary Setup

You already know about viewDidLoad …

```
override func viewDidLoad() {
     super.viewDidLoad() // always let super have a chance in lifecycle methods
     // do the primary setup of my MVC here
     // good time to update my View using my Model, for example, because my outlets are set
}
```

Do not do geometry-related setup here! Your bounds are not yet set!

View Controller Lifecycle

Will Appear

This method will be sent just before your MVC appears (or re-appears) on screen …

```
override func viewWillAppear(_ animated: Bool) {
     super.viewWillAppear(animated)
    // catch my View up to date with what went on while I was off-screen
}
```

Note that this method can be called repeatedly (vs. `viewDidLoad` which is only called once).

View Controller Lifecycle

- Did Appear

  You also find out after your MVC has finished appearing on screen …

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
   // maybe start a timer or an animation or start observing something (e.g. GPS position)?
}
```

  This is also a good place to start something expensive (e.g. network fetch) going.
  Why kick off expensive things here instead of in `viewDidLoad`?
  Because we know we're on screen so it won't be a waste.
  By "expensive" we usually mean "time consuming" but could also mean battery or storage.

  We must never block our UI from user interaction (thus background fetching, etc.).
  Our UI might need to come up incomplete and later fill in when expensive operation is done.
  We use "spinning wheels" and such to let the user know we're fetching something expensive.

View Controller Lifecycle

- Will Disappear

  Your MVC is still on screen, but it's about to go off screen.
  Maybe the user hit "back" in a UINavigationController?
  Or they switched to another tab in a UITabBarController?

```
override func viewWillDisappear(_ animated: Bool) {
     super.viewWillDisappear(animated)
     // often you undo what you did in viewDidAppear
     // for example, stop a timer that you started there or stop observing something
}
```

View Controller Lifecycle

- Did Disappear

  Your MVC went off screen.

  Somewhat rare to do something here, but occasionally you might want to “clean up” your MVC.

  For example, you could save some state or release some large, recreatable resource.

```
override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    // clean up MVC
}
```

View Controller Lifecycle

- Geometry

  You get notified when your top-level `view`'s bounds change (or otherwise needs a re-layout).
  In other words, when it receives `layoutSubviews`, you get to find out (before and after).

```
override func viewWillLayoutSubviews()
override func viewDidLayoutSubviews()
```

  Usually you don't need to do anything here because of Autolayout.
  But if you do have geometry-related setup to do, this is the place to do it (not in `viewDidLoad`!).

  These can be called often (just as `layoutSubviews()` in `UIView` can be called often).
  Be prepared for that.
  Don't do anything here that can't be properly (and efficiently) done repeatedly.
  It doesn't always mean your `view`'s bounds actually changed.

View Controller Lifecycle

- Autorotation

  When your device rotates, there's a lot going on.
  Of course your view's bounds change (and thus you'll get viewWill/DidLayoutSubviews).
  But the resultant changes are also automatically animated.
  You get to find out about that and even participate in the animation if you want …

```
override func viewWillTransition(
    to size: CGSize,
    with coordinator: UIViewControllerTransitionCoordinator
)
```

  You join in using the coordinator's animate(alongsideTransition:) methods.
  We don't have time to talk about how to do this, unfortunately!
  Check the documentation 🙁 .

View Controller Lifecycle

- Low Memory

It is rare, but occasionally your device will run low on memory.
This usually means a buildup of very large videos, images or maybe sounds.
If your app keeps strong pointers to these things in your heap, you might be able to help!
When a low-memory situation occurs, iOS will call this method in your Controller …

```
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // stop pointing to any large-memory things (i.e. let them go from my heap)
    //    that I am not currently using (e.g. displaying on screen or processing somehow)
    //    and that I can recreate as needed (by refetching from network, for example)
}
```

If your application persists in using an unfair amount of memory, you can get killed by iOS.

View Controller Lifecycle

- Waking up from an storyboard

This method is sent to all objects that come out of a storyboard (including your Controller) ...

```
override func awakeFromNib() {
    super.awakeFromNib()
    // can initialize stuff here, but it's VERY early
    // it happens way before outlets are set and before you're prepared as part of a segue
}
```

This is pretty much a place of last resort.
Use the other View Controller Lifecycle methods first if at all possible.
It's primarily for situations where code has to be executed VERY EARLY in the lifecycle.

View Controller Lifecycle

- Summary

  Instantiated (from storyboard usually)

  `awakeFromNib` (only if instantiated from a storyboard)

  segue preparation happens

  outlets get set

  `viewDidLoad`

  These pairs will be called each time your Controller's `view` goes on/off screen …

    `viewWillAppear` and `viewDidAppear`

    `viewWillDisappear` and `viewDidDisappear`

  These "geometry changed" methods might be called at any time after `viewDidLoad` …

    `viewWillLayoutSubviews` and `viewDidLayoutSubviews`

  At any time, if memory gets low, you might get …

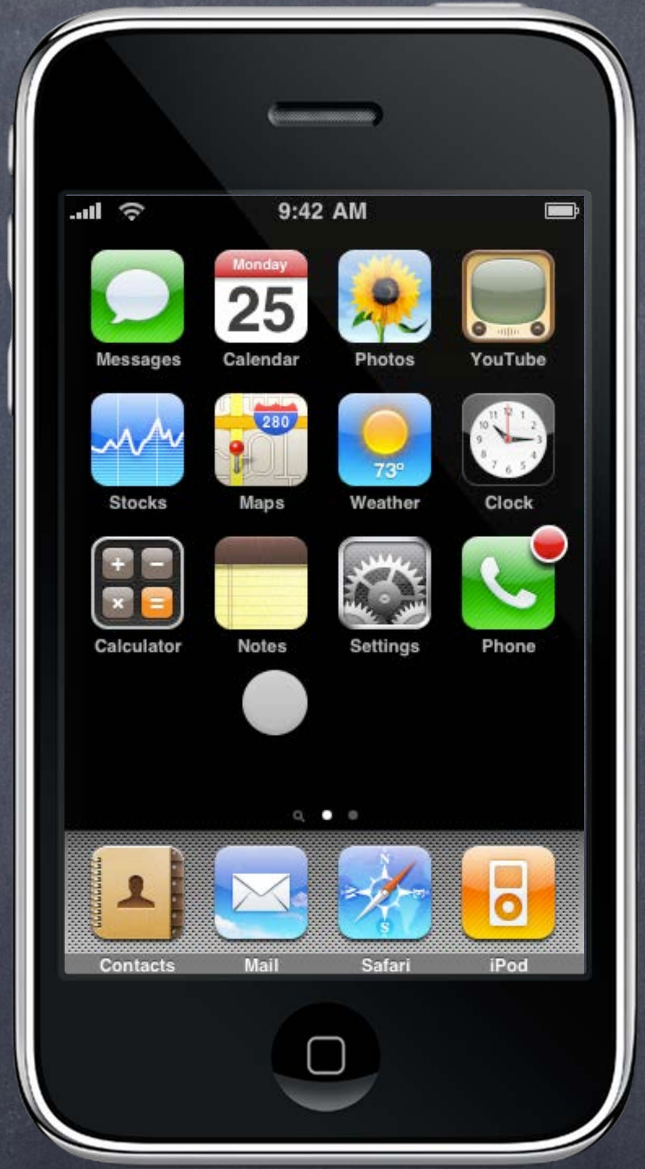
    `didReceiveMemoryWarning`

Demo

- View Controller Lifecycle

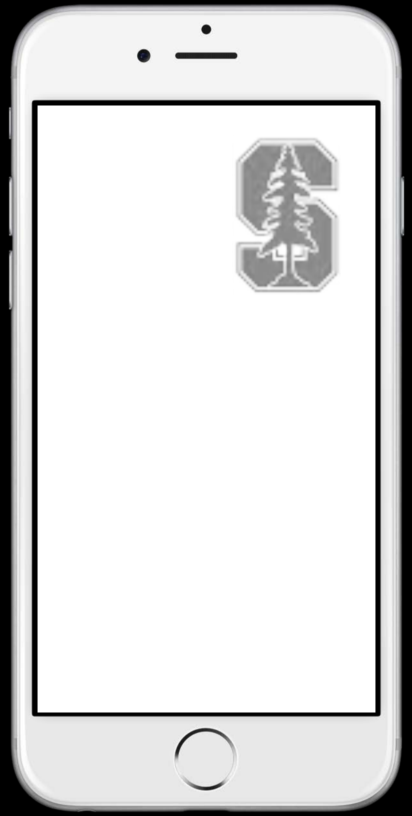
  Let's put some `print()`'s in our multiple-MVC version of Concentration

UIScrollView



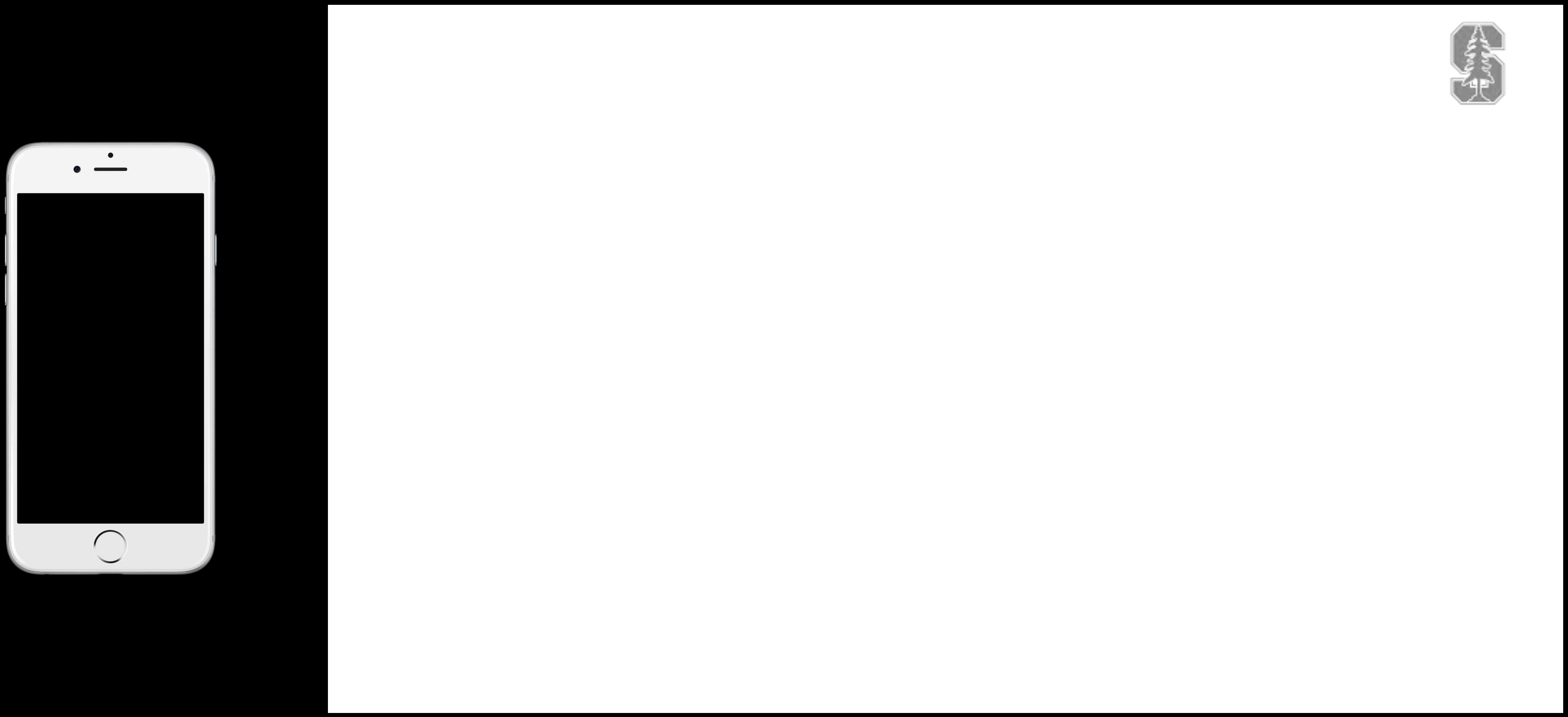
Adding subviews to a normal UIView …

```
logo.frame = CGRect(x: 300, y: 50, width: 120, height: 180)
view.addSubview(logo)
```



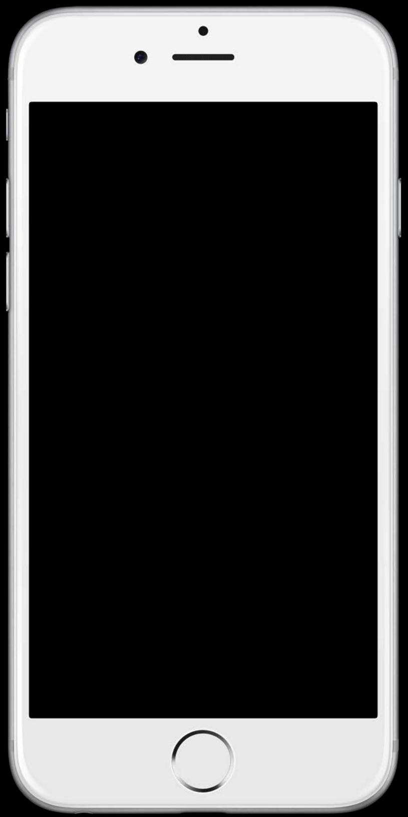
Adding subviews to a UIScrollView …

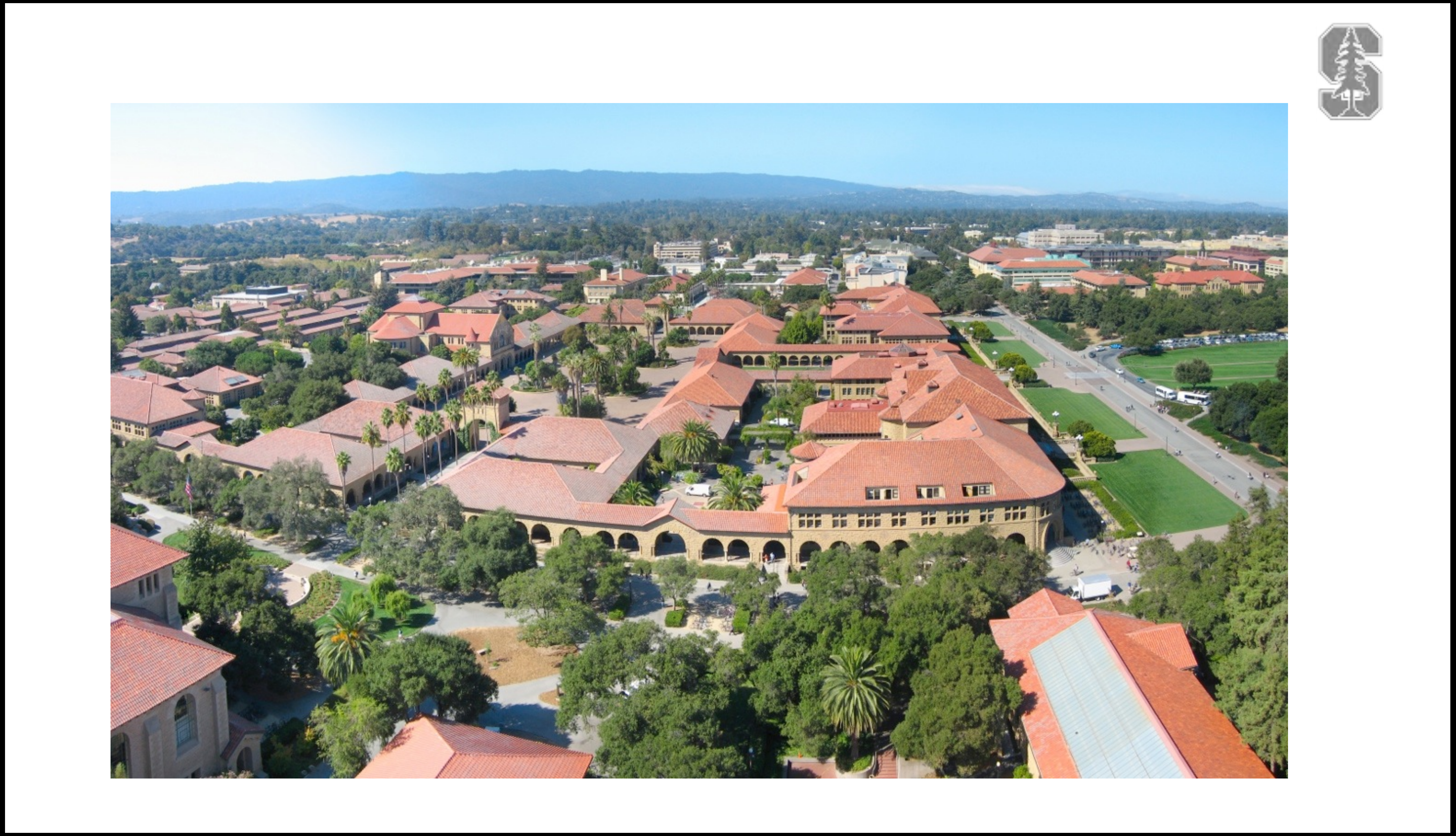
```
scrollView.contentSize = CGSize(width: 3000, height: 2000)
logo.frame = CGRect(x: 2700, y: 50, width: 120, height: 180)
scrollView.addSubview(logo)
```



Adding subviews to a UIScrollView …

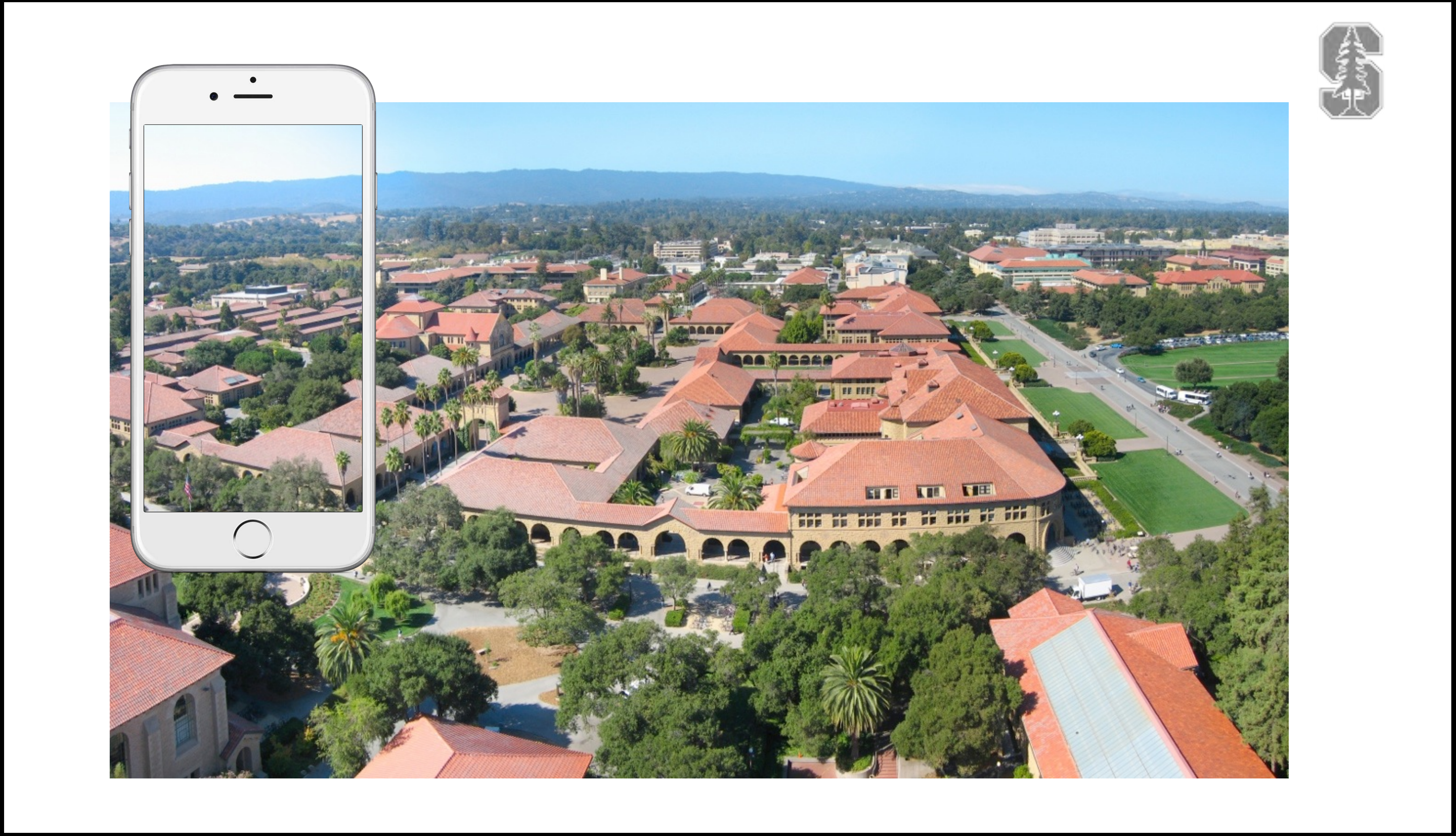
```
scrollView.contentSize = CGSize(width: 3000, height: 2000)
aerial.frame = CGRect(x: 150, y: 200, width: 2500, height: 1600)
scrollView.addSubview(aerial)
```



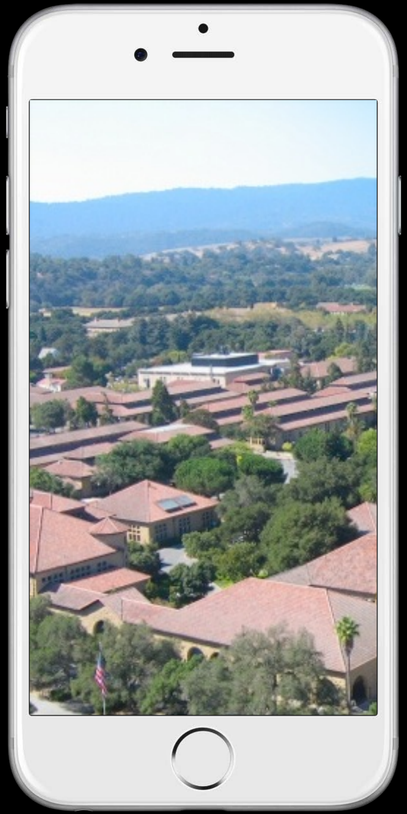


Adding subviews to a UIScrollView …

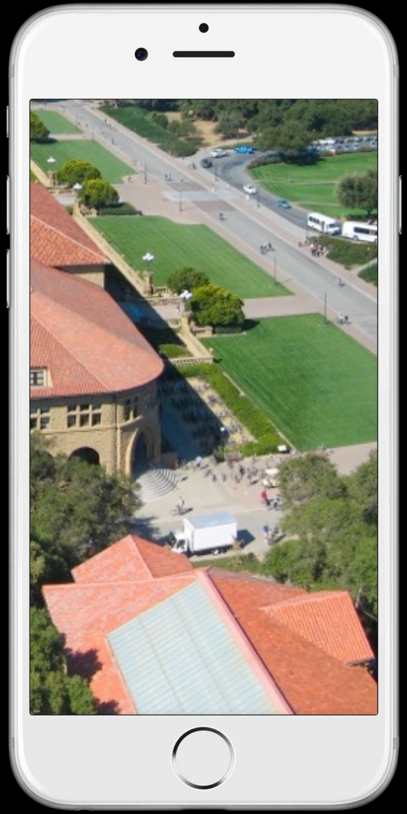
```
scrollView.contentSize = CGSize(width: 3000, height: 2000)
aerial.frame = CGRect(x: 150, y: 200, width: 2500, height: 1600)
scrollView.addSubview(aerial)
```



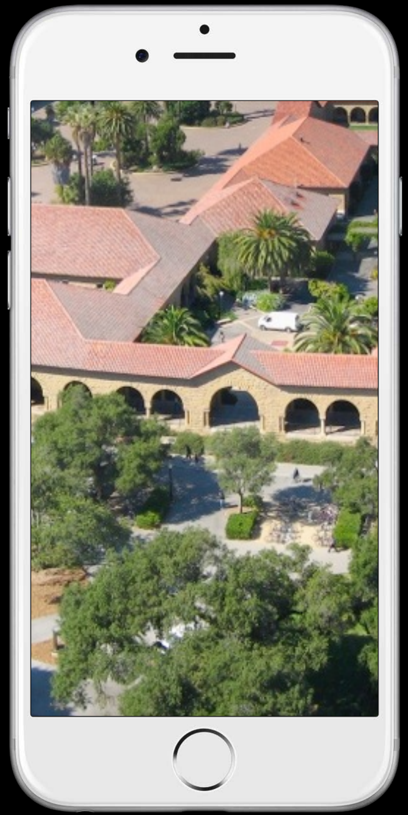
Scrolling in a UIScrollView …



Scrolling in a `UIScrollView` …



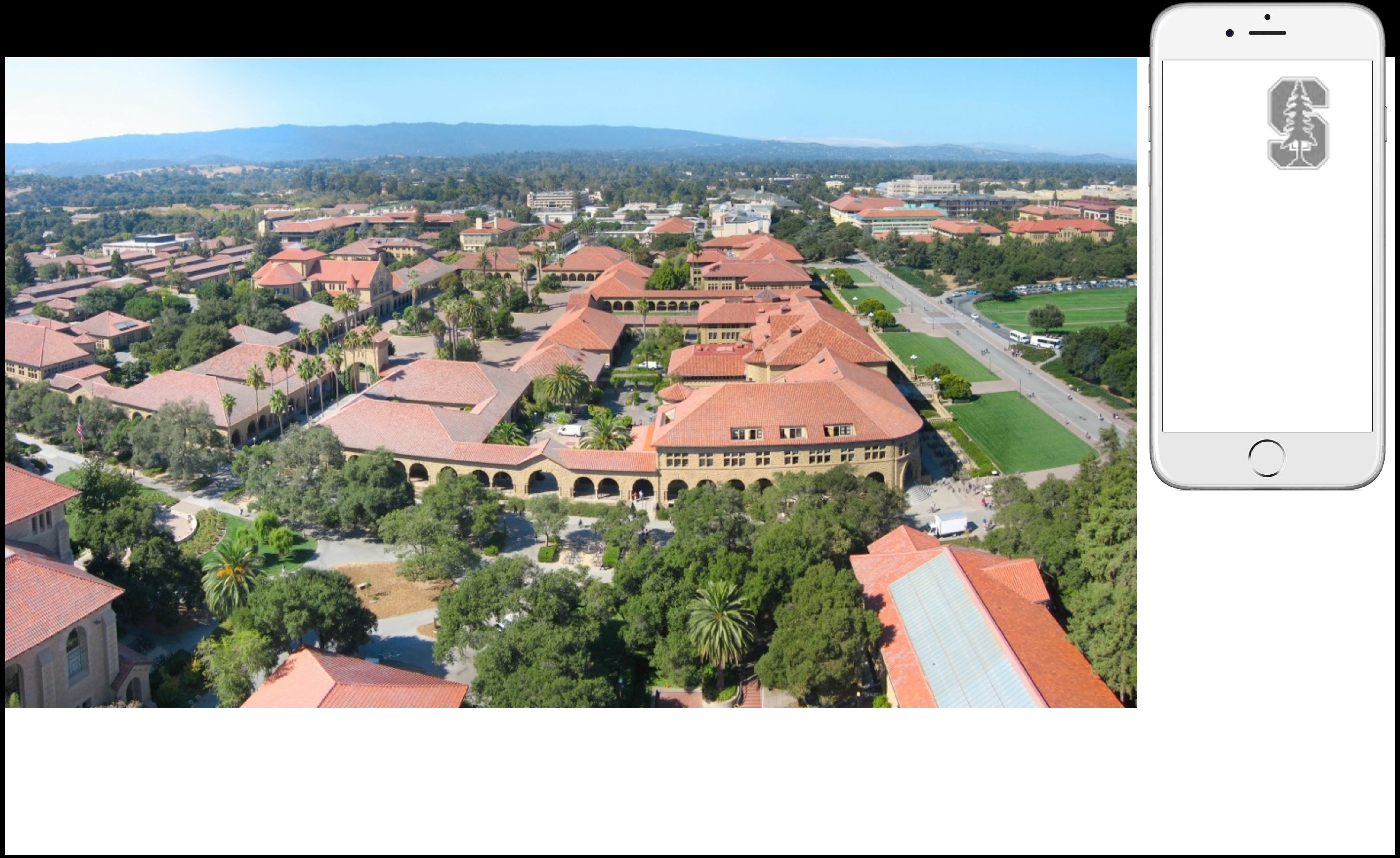
Scrolling in a `UIScrollView` …



Scrolling in a `UIScrollView` …

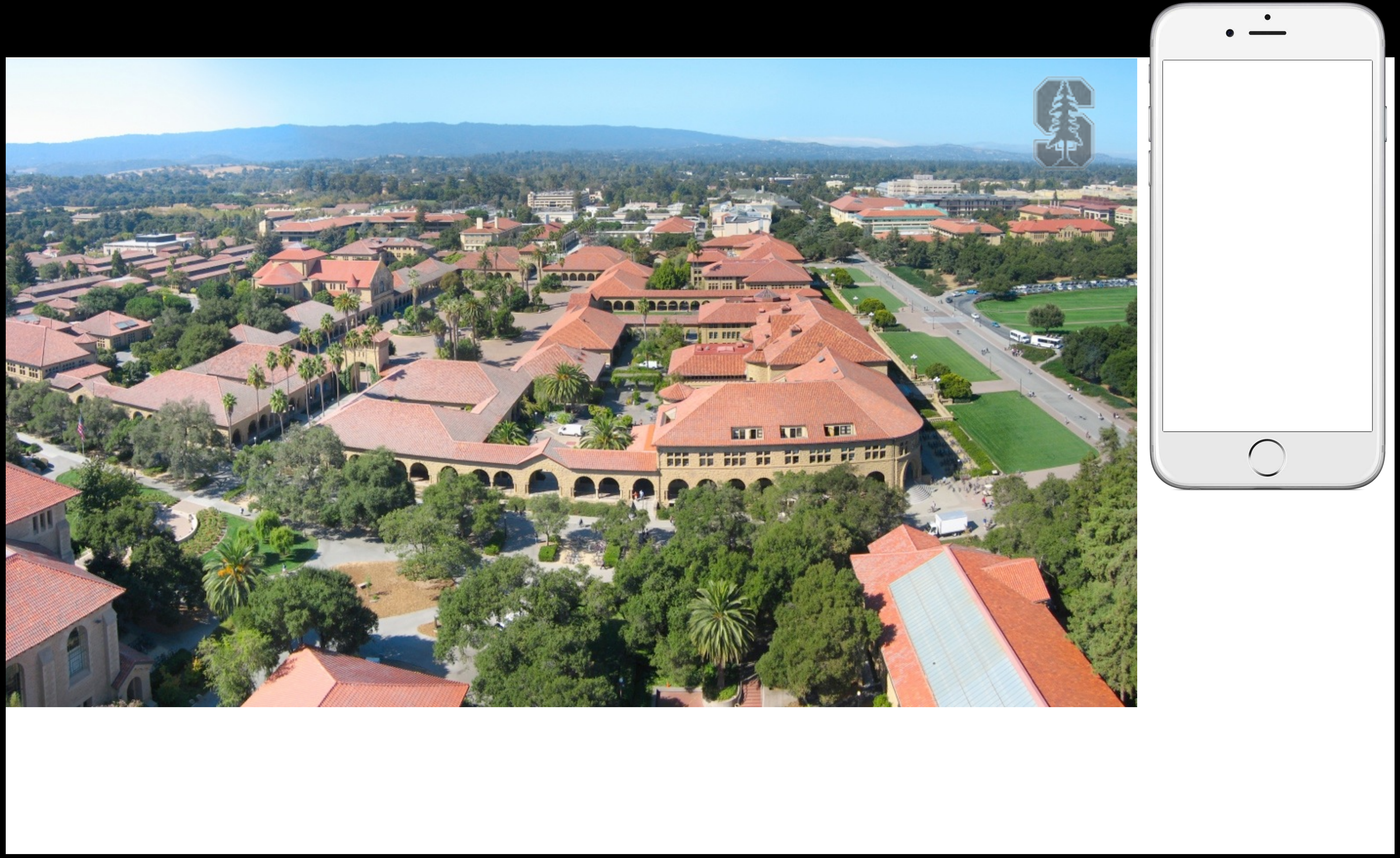
Positioning subviews in a UIScrollView …

```
aerial.frame = CGRect(x: 0, y: 0, width: 2500, height: 1600)
```



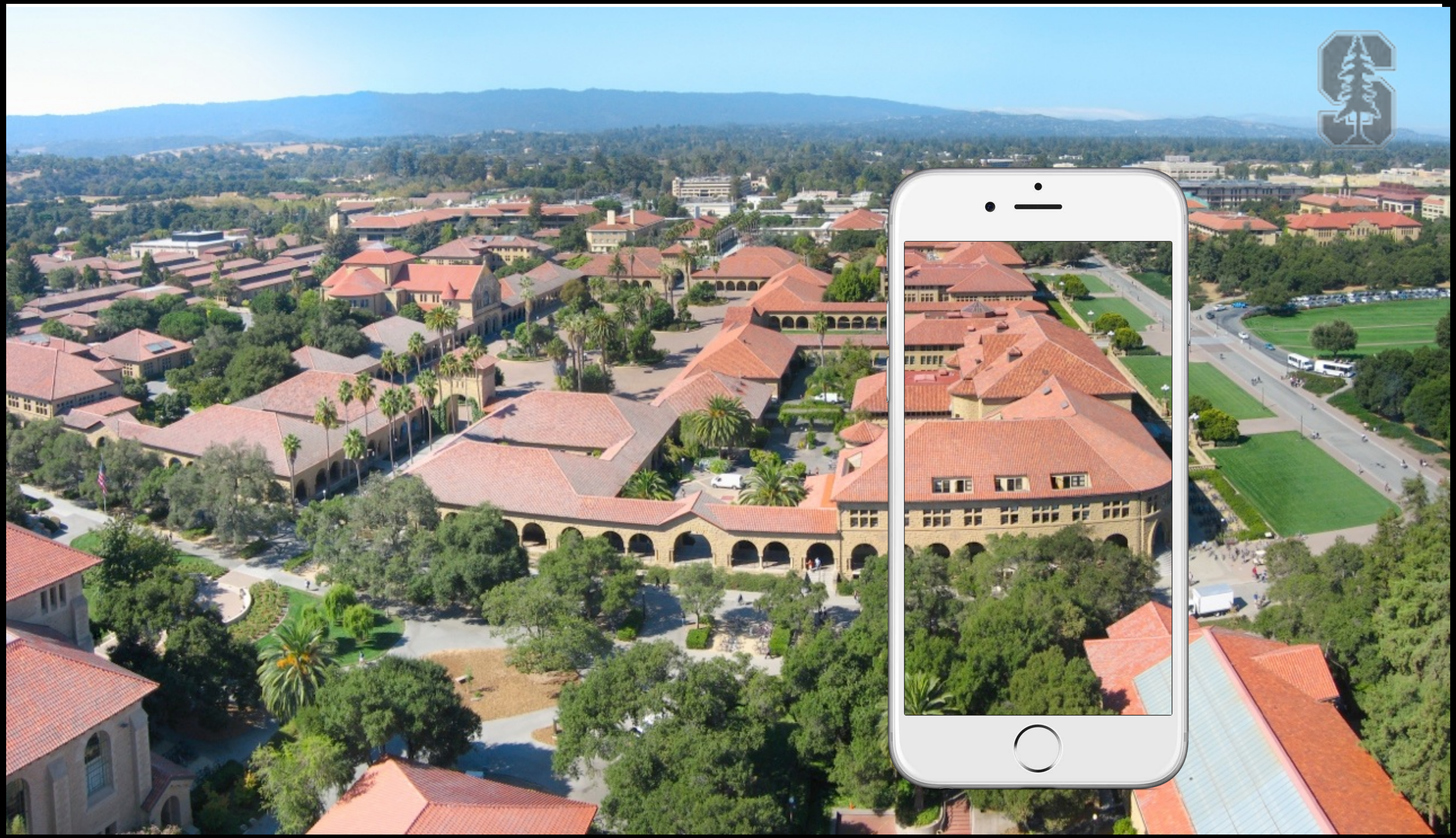
Positioning subviews in a UIScrollView …

```
aerial.frame = CGRect(x: 0, y: 0, width: 2500, height: 1600)
logo.frame = CGRect(x: 2300, y: 50, width: 120, height: 180)
```

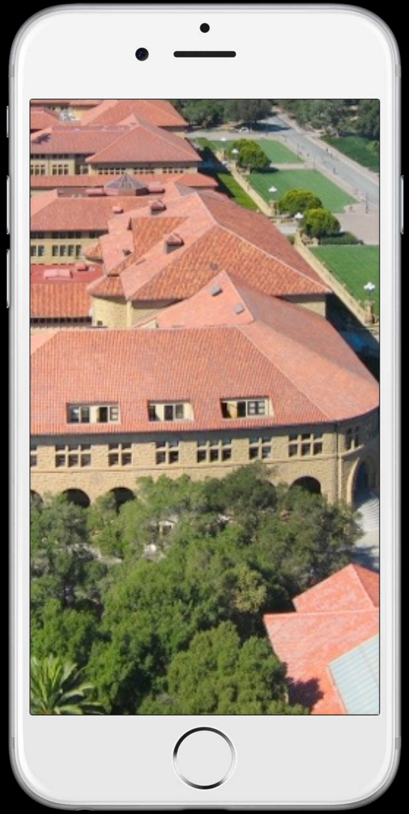


Positioning subviews in a `UIScrollView` …

```
aerial.frame = CGRect(x: 0, y: 0, width: 2500, height: 1600)
logo.frame = CGRect(x: 2300, y: 50, width: 120, height: 180)
scrollView.contentSize = CGSize(width: 2500, height: 1600)
```

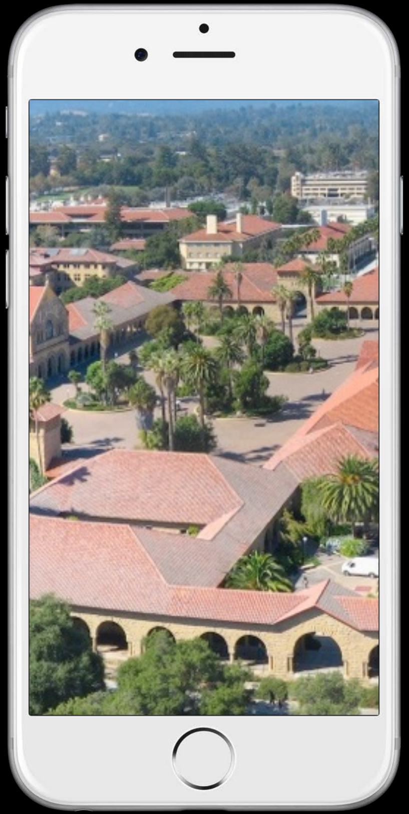


That’s it!



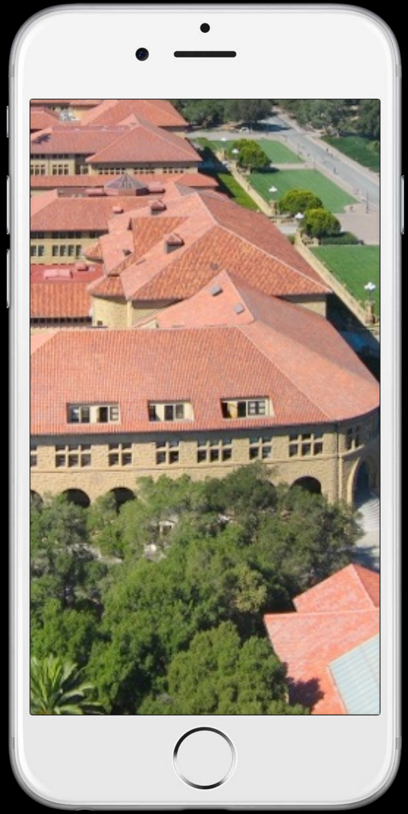
That's it!

That's it!



That's it!

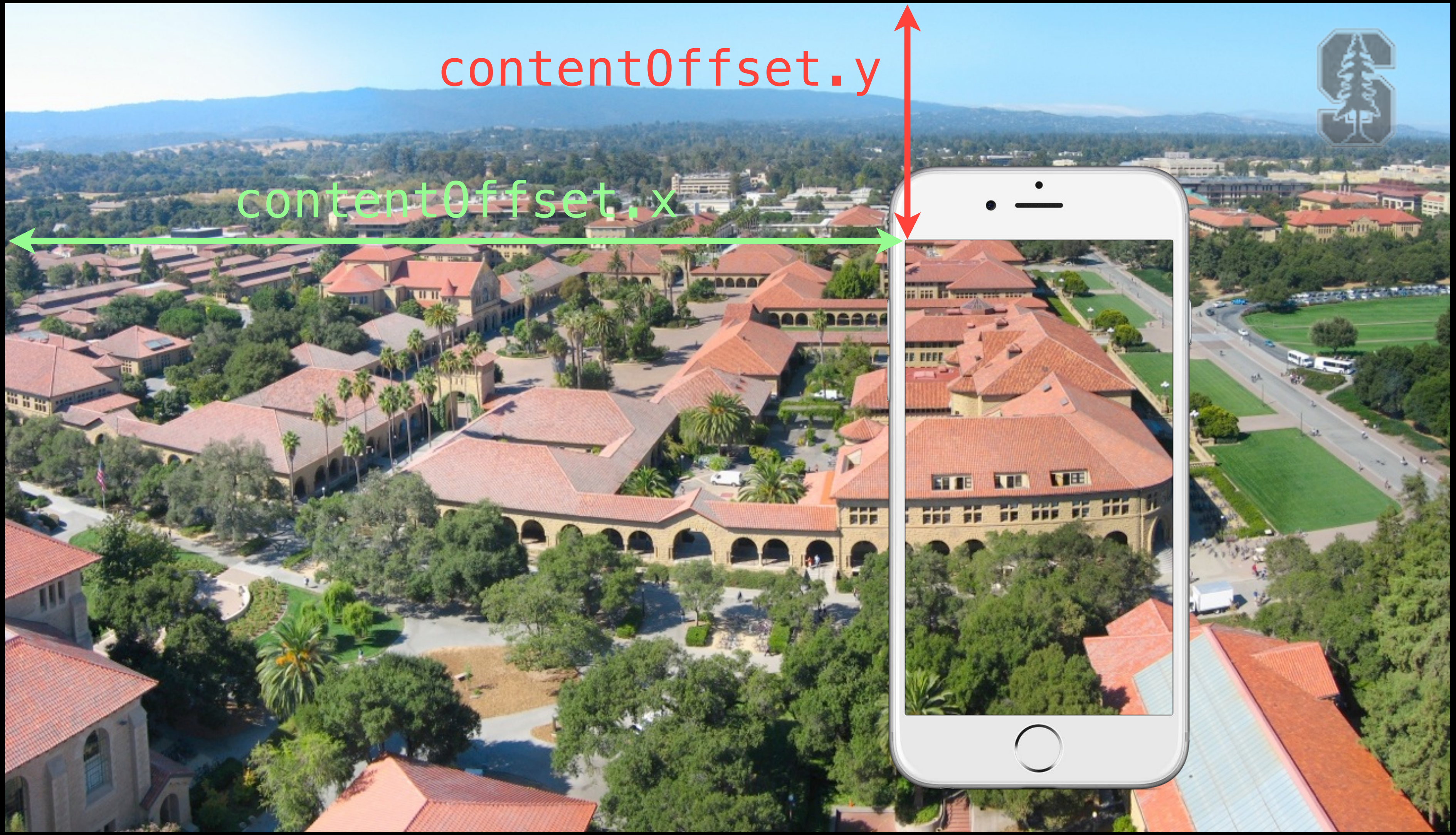
That's it!



Where in the content is the scroll view currently positioned?

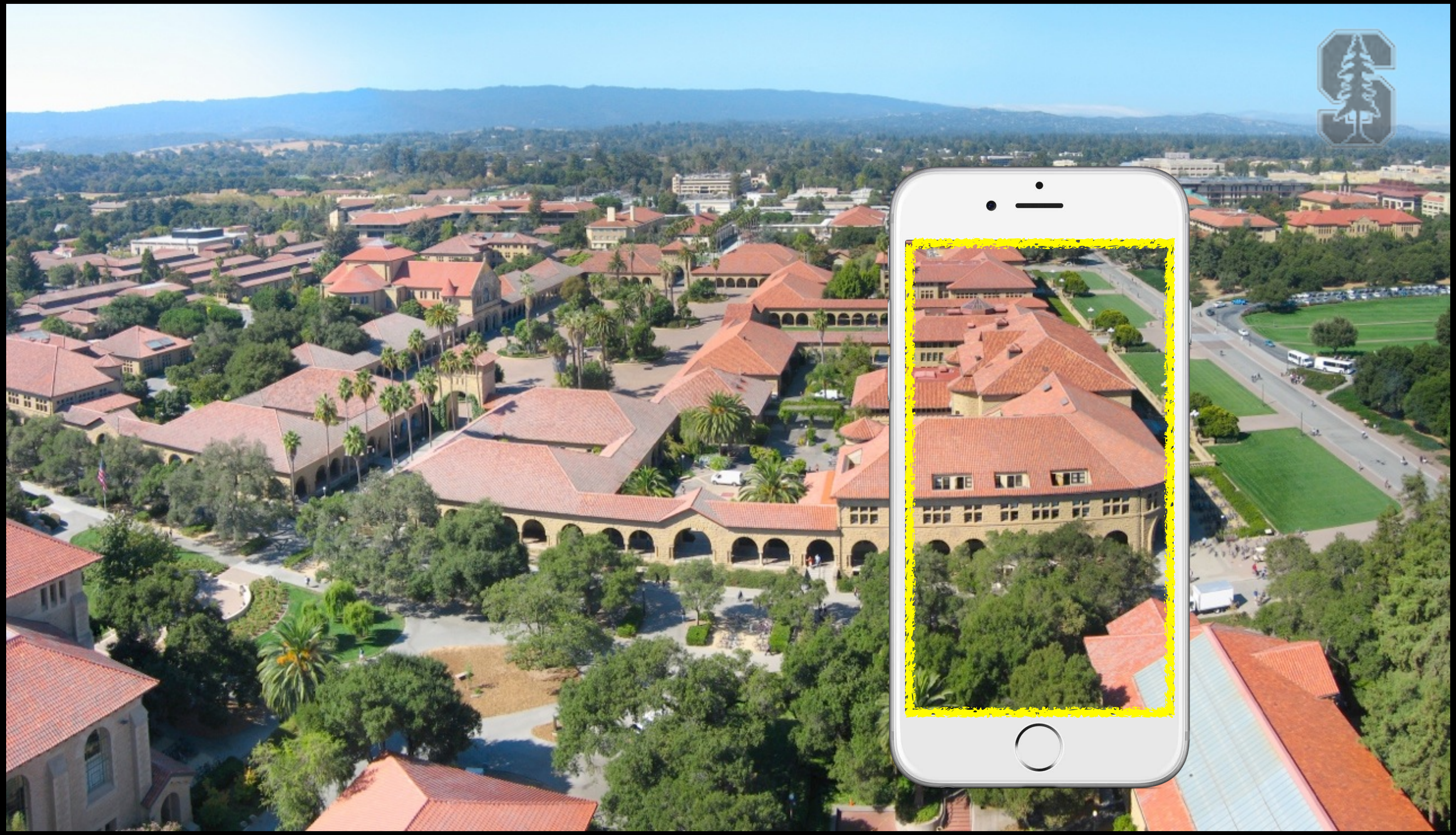
```
let upperLeftOfVisible: CGPoint = scrollView.contentOffset
```

In the content area's coordinate system.



What area in a subview is currently visible?

```
let visibleRect: CGRect = aerial.convert(scrollView.bounds, from: scrollView)
```



Why the convertRect? Because the scrollView's bounds are in the scrollView's coordinate system.
And there might be zooming going on inside the scrollView too …

UIScrollView

- How do you create one?

  Just like any other UIView. Drag out in a storyboard or use UIScrollView(frame:).
  Or select a UIView in your storyboard and choose "Embed In -> Scroll View" from Editor menu.

- To add your "too big" UIView in code using addSubview …

```
if let image = UIImage(named: "bigimage.jpg") {
    let iv = UIImageView(image: image)  // iv.frame.size will = image.size
    scrollView.addSubview(iv)
}
```

  Add more subviews if you want.
  All of the subviews' frames will be in the UIScrollView's content area's coordinate system
  (that is, (0,0) in the upper left & width and height of contentSize.width & .height).

- Now don't forget to set the contentSize

  Common bug is to do the above lines of code (or embed in Xcode) and forget to say:

  `scrollView.contentSize = imageView.frame.size` (for example)

UIScrollView

- Scrolling programmatically

  `func scrollRectToVisible(CGRect, animated: Bool)`

- Other things you can control in a scroll view

  Whether scrolling is enabled.

  Locking scroll direction to user's first "move".

  The style of the scroll indicators (call `flashScrollIndicators` when your scroll view appears).

  Whether the actual content is "inset" from the content area (`contentInset` property).

UIScrollView

- Zoming

  All UIView's have a property (`transform`) which is an affine transform (translate, scale, rotate).
  Scroll view simply modifies this transform when you zoom.
  Zooming is also going to affect the scroll view's `contentSize` and `contentOffset`.

- Will not work without minimum/maximum zoom scale being set

```
scrollView.minimumZoomScale = 0.5   // 0.5 means half its normal size
scrollView.maximumZoomScale = 2.0   // 2.0 means twice its normal size
```

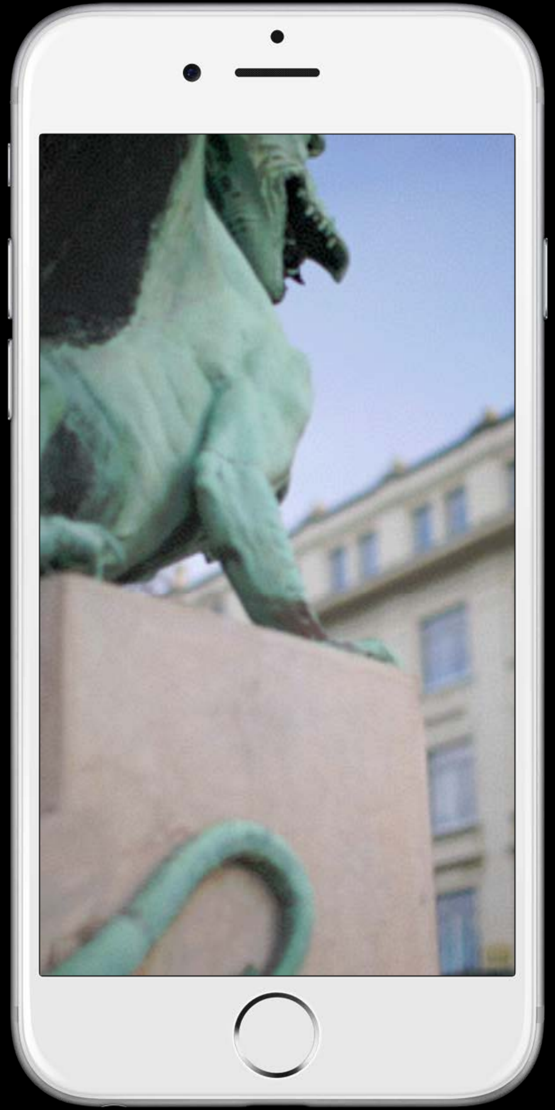
- Will not work without <u>delegate</u> method to specify view to zoom

```
func viewForZooming(in scrollView: UIScrollView) -> UIView
```

  If your scroll view only has one subview, you return it here. More than one? Up to you.

- Zooming programatically

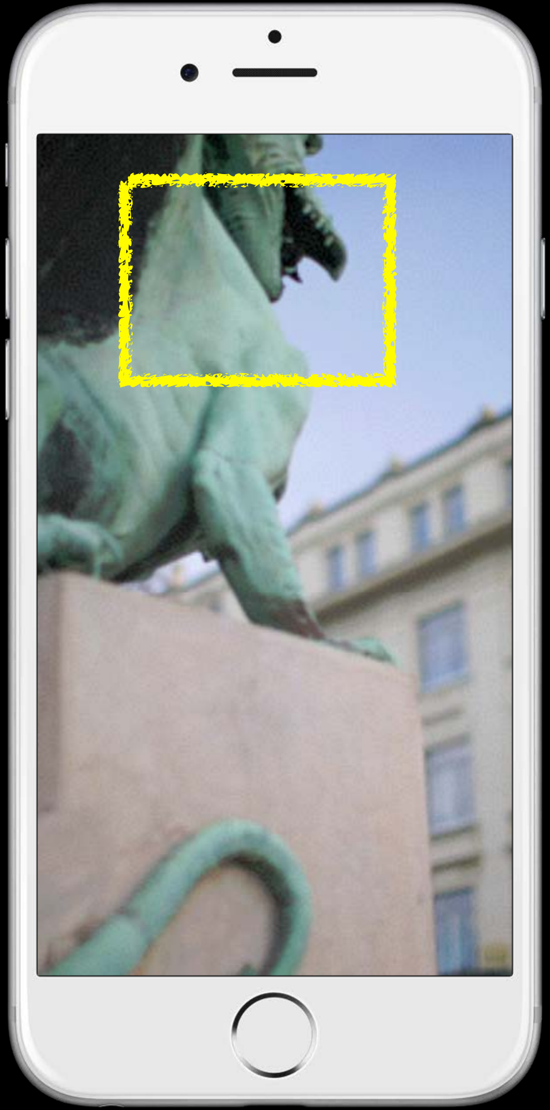
```
var zoomScale: CGFloat
func setZoomScale(CGFloat, animated: Bool)
func zoom(to rect: CGRect, animated: Bool)
```



```
scrollView.zoomScale = 1.2
```

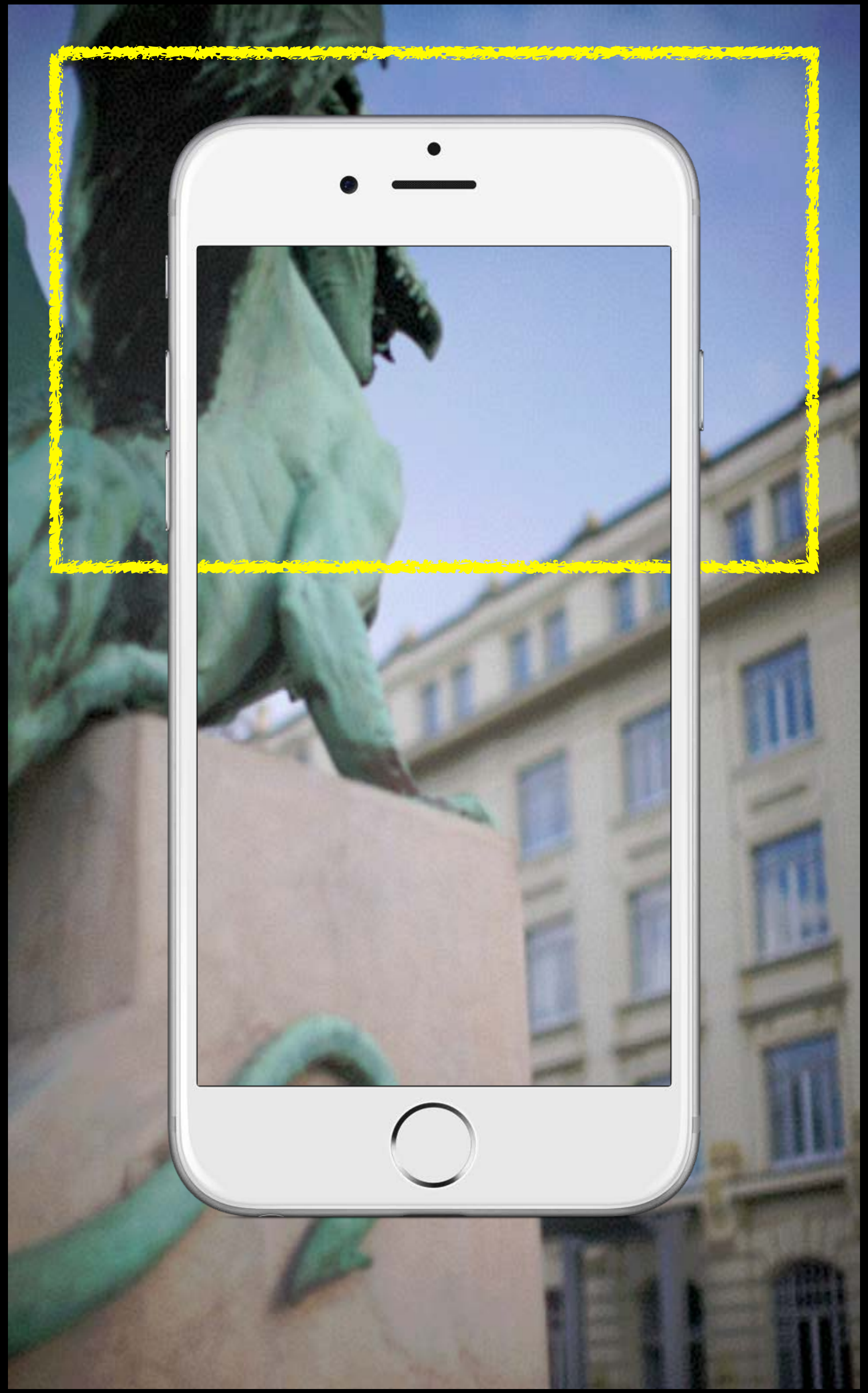
```
scrollView.zoomScale = 1.0
```

```
scrollView.zoomScale = 1.2
```

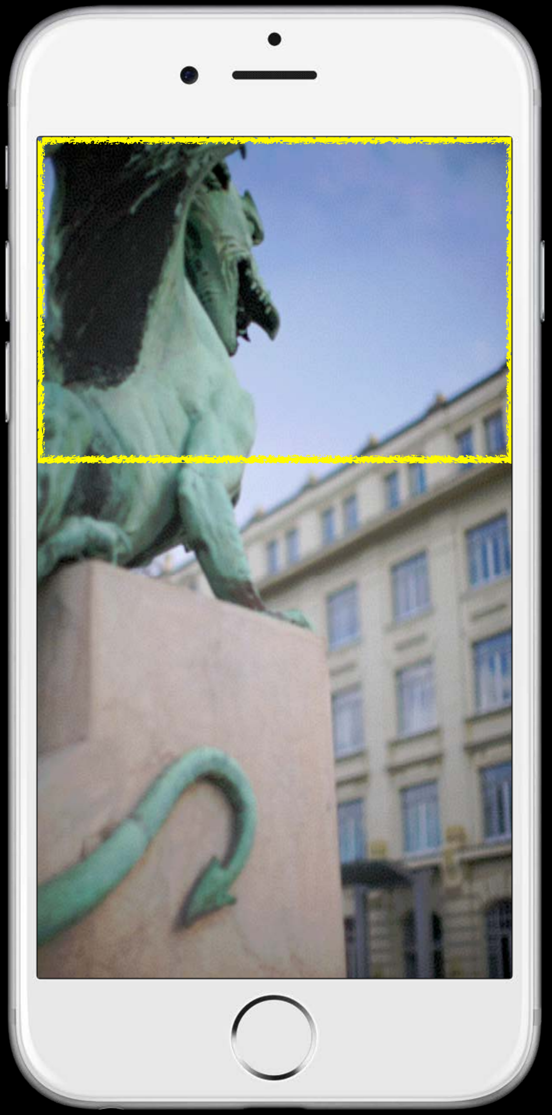


`zoom(to rect: CGRect, animated: Bool)`

```
zoom(to rect: CGRect, animated: Bool)
```



`zoom(to rect: CGRect, animated: Bool)`



`zoom(to rect: CGRect, animated: Bool)`

UIScrollView

- Lots and lots of `delegate` methods!

  The scroll view will keep you up to date with what's going on.

- Example: `delegate` method will notify you when zooming ends

```
func scrollViewDidEndZooming(UIScrollView,
                             with view: UIView,  // from delegate method above
                             atScale: CGFloat)
```

  If you redraw your view at the new scale, be sure to reset the transform back to identity.

Demo Code

Download the Cassini demo code from today's lecture.

Download the VCL logging UIViewController from today's lecture.