Designing Apps with Scroll Views _ Part I: Zooming and Scrolling on Photos

Based on Apple WWDC 2010 Designing Apps with Scroll Views

Seyed Samad Gholamzadeh Jan 16, 2018

Introduction

This is the first part of a set of tutorials related to using scroll views in our app based on **Apple WWDC 2010 Designing Apps with Scroll Views** which you can watch it from here (**HD**|**SD**).

Here we want to make an app similar to theiOS photos app with the help of `UIScrollView` useful features. You can study the next two parts of this tutorial from the links below:

Start From Scratch!

Open Xcode, create a new project, choose Single View App under the iOS tab bar, type **PhotoScroller** as Product Name and create aproject.
In the Project navigator under the **ViewController.swift**, Create a new file of Cocoa Touch Class, name it **ImageScrollView** and set it a subclass of `UIScrollView`. Delete all commented lines between class curly braces and finally, you should see something like this:

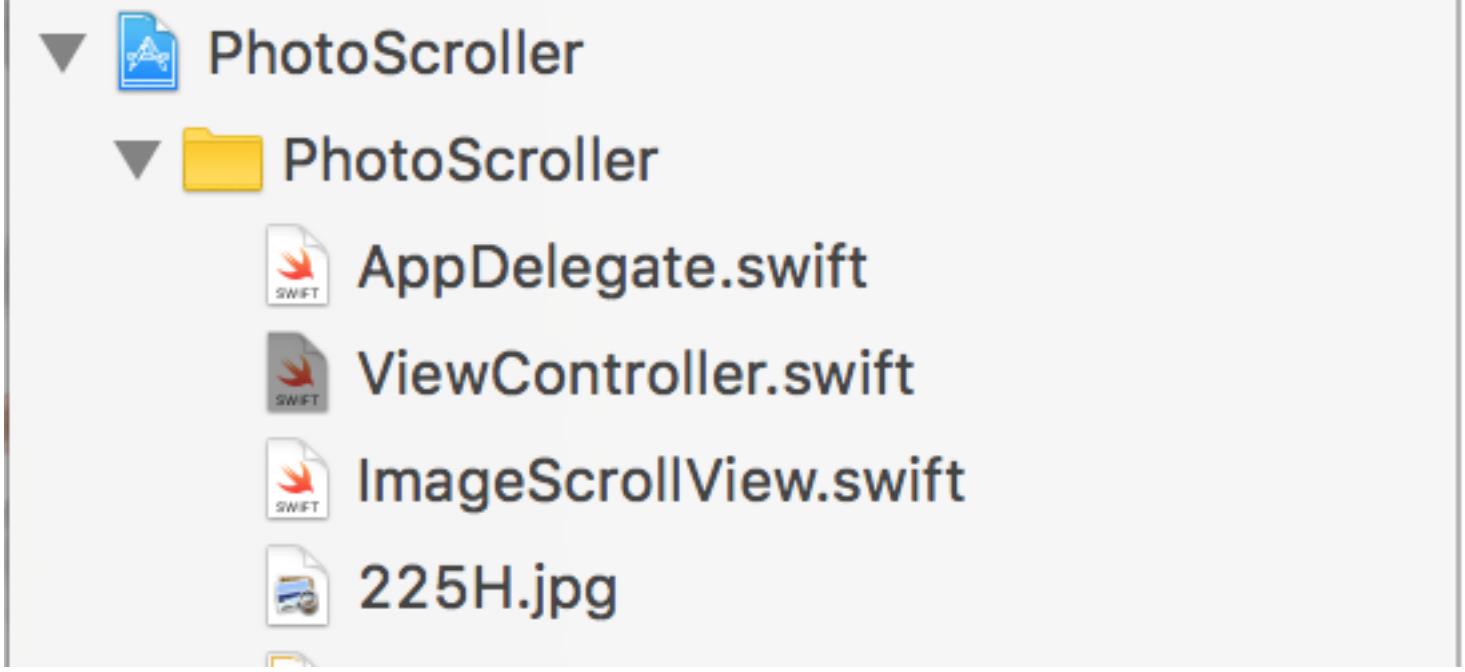
This is a simple `UIScrollView` subclass. Let's change it and make it kind of scroll views that shows our images. Add this lines of code into the class curly braces:

We added a UIImageView instance that named **zoomView** and it's going to show our image in scroll view.
Next, we added a method that:
1. Clears `zoomView`, then reinitialized it with the new image given to method as an input argument.
2. Adds zoomView to Scroll view as its subview.

Now its time to see what shows us scroll view!

> ***Note:*** *We do not set any frame for zoomView. In such situations, UIImageView automatically set its frame based on image size.*

Before we test our app, we need some high-quality images to scroll on it! Download this photo and put it blew the **ImageScrollView.swift** in Xcode as you can see in the picture (Do not add the photo into Assets library):



Do not add photo to Assets library, and add it directly in to Xcode like other files.

Now go to **ViewController.swift.** make an instance of `ImageScrollView` at the top of `ViewController` class like this:

```
var imageScrollView: ImageScrollView!
```

And add this bunch of codes in to `viewDidLoad()` method after `super.viewDidLoad()`:

The codes are simple:

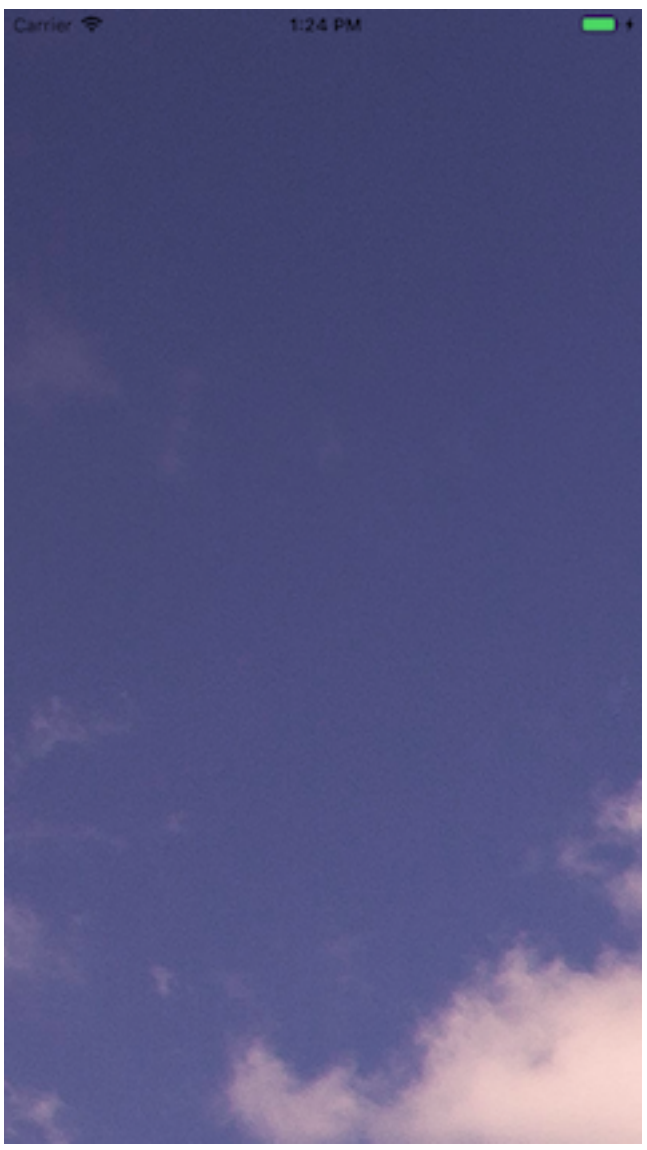
1. We initialized `imageScrollView` and added it to ViewControllers `view`. Note that we set the imageScrollView `frame` as viewControllers view `bounds`. It's important that always you set scroller view frames even if you used constraints to fix it in the parent view. That's because scroll view must know in which size it shows its content.
2. We created an image object. Notice we do not use of `UIImage(named:)` initializer. That's because `UIImage(named:)` caches image in memory even If you only use it one time and never use it again, instead `UIImage(contentsOfFile:)` caches image in memory until you need it and as soon as you leave image, it purges image from memory too, and if you need to image be reloaded later, the image object loads that data again from the specified path. Since we use a high-quality photo here, then it is better to use `UIImage(contentsOfFile:)` . It's not a good idea to hold a high-

quality photo in memory all the time and that's why I said do not add the photo to assets library.

> ***Note:*** *For more information about UIImage initialization check out [Apple Developer Documentation](#).*

3. We asked `imageScrollView` to display our photo in the scroll view.

Build and run the app. You should see something like this on your phone screen:



A huge blue sky! It was probably not what you expected to see.
Try to scroll photo or pinch it. Did nothing happen?
Don't worry, we fix it soon.

Let's see what's the problem. The problem is that the scroll view does not know what we want! All information we give it till now is the photo we wanted to show and the frame we wanted to see the scroll view in that dimension.
See? In fact, It has done its job very well! So if we want more, we should

give it more information. Let's do it!

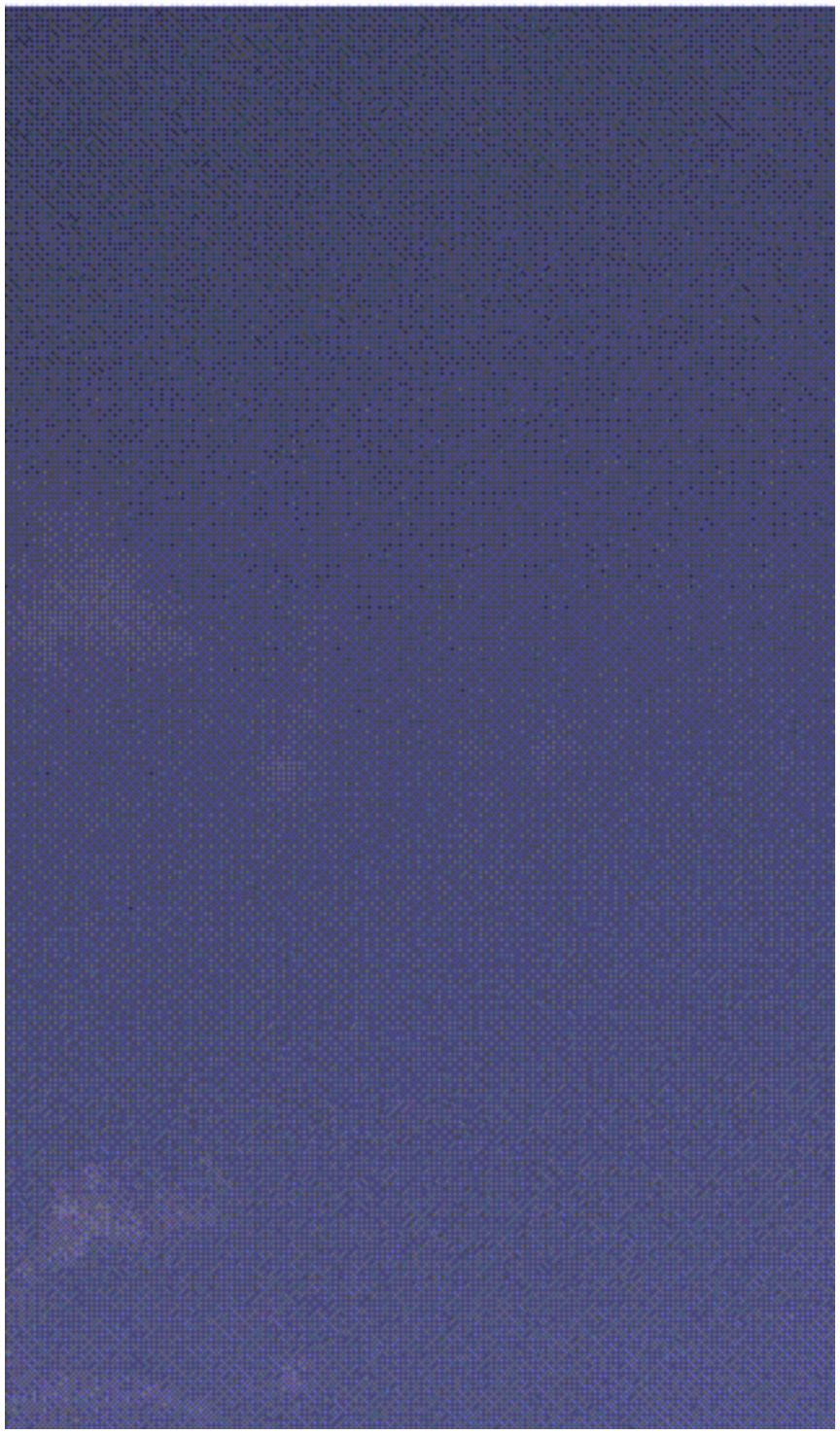
Make it scrolls

One of the information, we have to give it to the scroll view is that How big is the size of our photo. For this purpose, go to **ImageScrollView.swift** and add this method implementation after `display(_ image: UIImage) {…}`:

In this simple method, We give the size of the image to scroll view as its content size.

add `self.configureFor(image.size)` at the end of `display(_ image: UIImage)` method after `self.addSubview(zoomView)` line.

Build and run the app. Try to scroll photo, It should scroll now.
Does it scroll? Good!



Now you can scroll on the photo.

Well, let's just do some configuration on the scroll view. Add these codes at the top of `ImageScrollView` class just after `var zoomView: UIImageView!`:

In the above codes, we set some changes on scroll view when it initializes;
The first two lines hide horizontal and vertical scroll indicators.
The third line cause to scroll view decelerate faster its speed to stop (test the app with an without this line, to see differences).

Pinch it

What about pinching and zooming?

For this case, the scroll view needs to know two things:

The first is the view which we like to apply zoom on it and the second is the range of zoom scale

For the first one, `ImageScrollView` must adopt `UIScrollViewDelegate` protocol and implement its `viewForZoomng(in:)` method.
Add `UIScrollViewDelegate` to `ImageScrollView` like blew:

```
class ImageScrollView: UIScrollView, UIScrollViewDelegate {
```

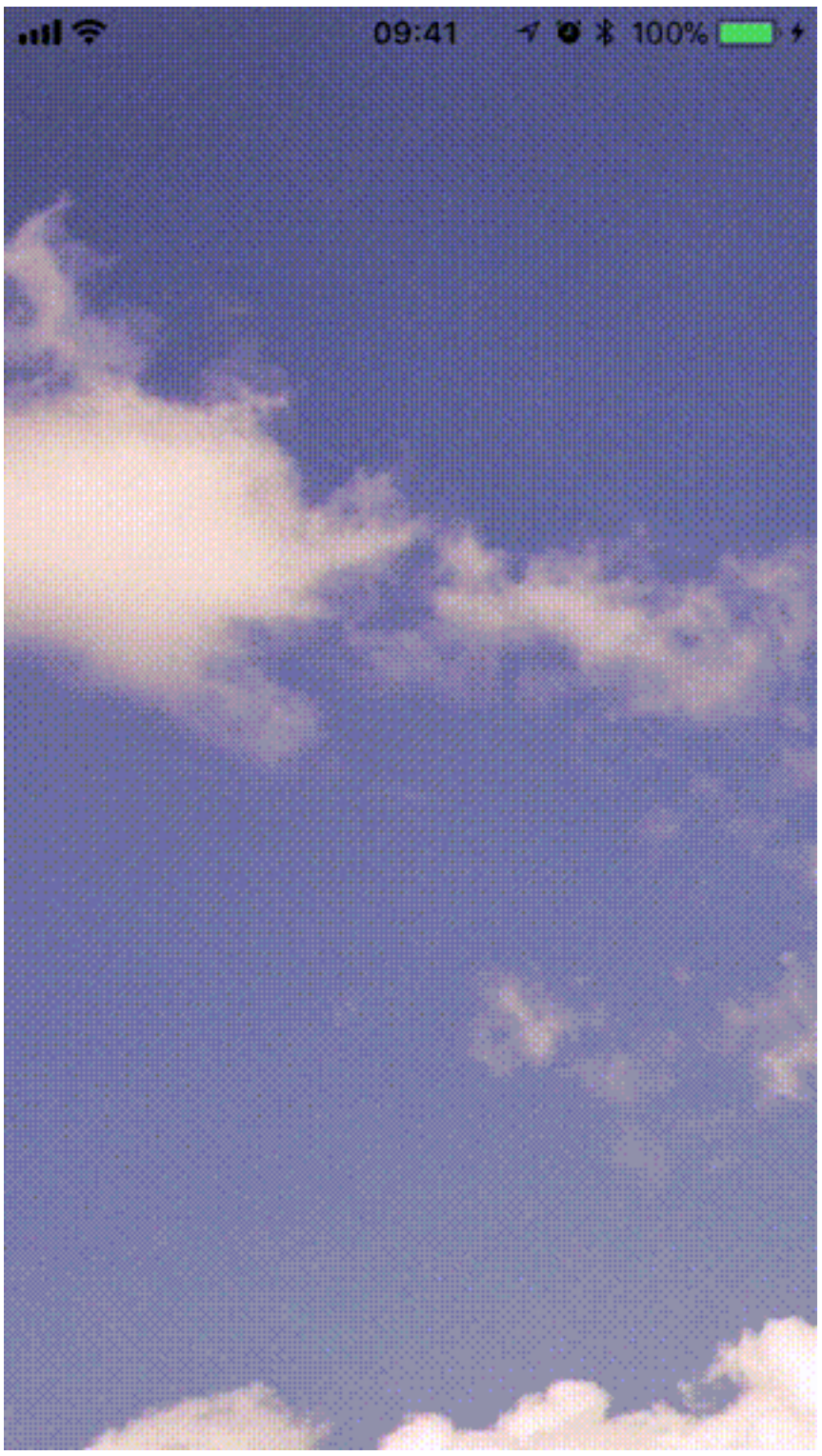
And add this method after `configureFor(_ imageSize: CGSize) {…}` method:

This method asks the delegate (which here is `ImageScrollView`) for the view to scale when zooming is about to occur in the scroll view.
One more thing to do has remained, and that is we should set the scroll view as the delegate of itself. For this purpose, just add `self.delegate = self` into the `init(frame: CGRect)` initializer after `self.decelerationRate = UIScrollViewDecelerationRateFast` line.
With this line, scrollView's delegate will be set when it initializes.

But, how to set a range of zoom scale?
For this purpose, scroll view has two properties which are `minimumZoomScale` and `maximumZoomScale`. `minimumZoomScale` determines how many you could zoom out the content of scroll view and `maximumZoomScale` determines how many you could zoom in the content of scroll view, relative to the current scale of its content. The default value of both of these properties is `1.0`.
Add blew codes in to `configureFor(_ imageSize: CGSize)` method after `self.contentSize = imageSize`:

Now build and run the project. You should be able to pinch and zoom now.

> ***Note:*** *If you testing app on the simulator, for pinch you must hold* ***option*** *(⌥) key on the keyboard.*



If you played enough with the pinching and zoom in/out photo let's answer to a question:

What's the appropriate value for these properties?

For `minimumZoomScale` , it's usually better to be fit on the screen, when the photo gets zoom out.
But `maximumZoomScale` is a little complicated. When we set `maximumZoomScale` to `1.0`, it allows the photo to be scaled to its actual size, but this is not a good idea for photos like the one we used in this tutorial. As you saw When this photo is in its actual size, We could only

see a very small part of the photo, so in such cases, It's better to have a `maximumZoomScale` value, less than `1.0`. But this is not always true. There are photos with smaller size out there and they need their own `maximumZoomScale` and we should consider them as well.
Let's dive in codes, and see the calculations needed for each of these values.
Add this method after `configureFor(_ imageSize: CGSize) {…}` method:

For `minimumZoomScale` we calculated two value named `xScale` and `yScale`, which are respectively the scroll view bound's width to image width ratio and scroll view bound's height to image height ratio. The minimum of these values causes the photo to be fit in the screen, so `minimumZoomScale = minScale`.
For `maximumZoomScale` , we set several conditions with respect to `minimumZoomScale` to be sure we have the right amount of `maximumZoomScale` for photos of different sizes.

Replace `self.setMaxMinZoomScaleForCurrentBounds()` with the last two lines in `configureFor(_ imageSize: CGSize)` method. Your `configureFor(_ imageSize: CGSize)` now must be like this:

```
func configureFor(_ imageSize: CGSize) {
    self.contentSize = imageSize
    self.setMaxMinZoomScaleForCurrentBounds()
}
```

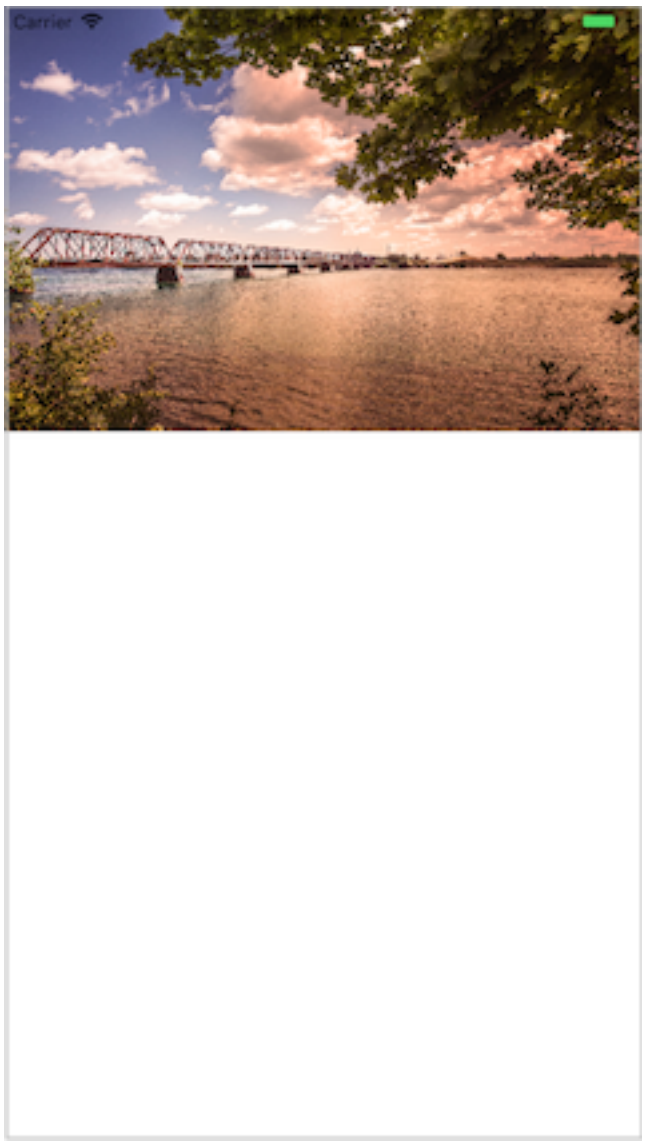
Build and run the app. Now if you zoom out the photo with pinching, It should be fit inside the screen.

Fit Photo To Screen at First Launch

How we have a fit screen photo when we first launch the app?
Scroll view has another property named `zoomScale`.
This property is responsible for the scale of the photo at any moment and its default value is `1.0`. That's why you see the photo at the actual size

when you run the app. So if you want to see it fit you should set `zoomScale` to `minimumZoomScale`.

Add `self.zoomScale = self.minimumZoomScale` at the end of `configureFor(_ imageSize: CGSize)` after `self.setMaxMinZoomScaleForCurrentBounds()`.



Put the Photo in the Middle

There is one drawback here which is that the photo is up at the top which isn't really what we want. We want as the image gets zoomed out to be smaller than the screen is centered in the screen rather than hugging the upper left corner.
How do that?

To do this we overriding the `layoutSubviews()` method. The advantage of the `layoutSubviews()` is that it's called at every frame of both zooming and scrolling. So if we want to keep a view centered, this is a perfect place to do it.

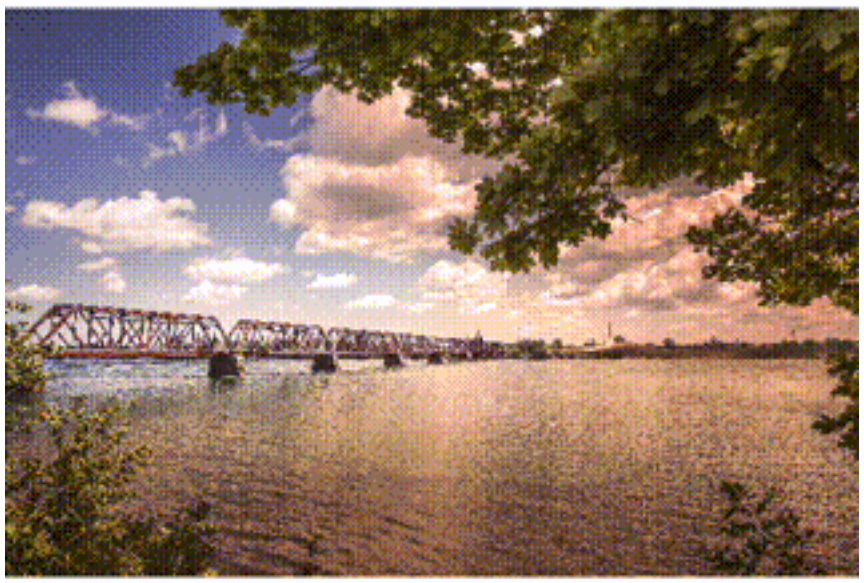
Add these method after `setMaxMinZoomScaleForCurrentBounds() {…}` method:

In this method, we calculate and check if each Side of the photo is smaller than the screen, change the position of photo and put at the middle of the screen.
Now override add this method after `required init?(coder aDecoder: NSCoder) {…}` initializer:

This method overrides `layoutSubviews()` and calls our `centerImage()` method, each time `layoutSubviews()` being called.

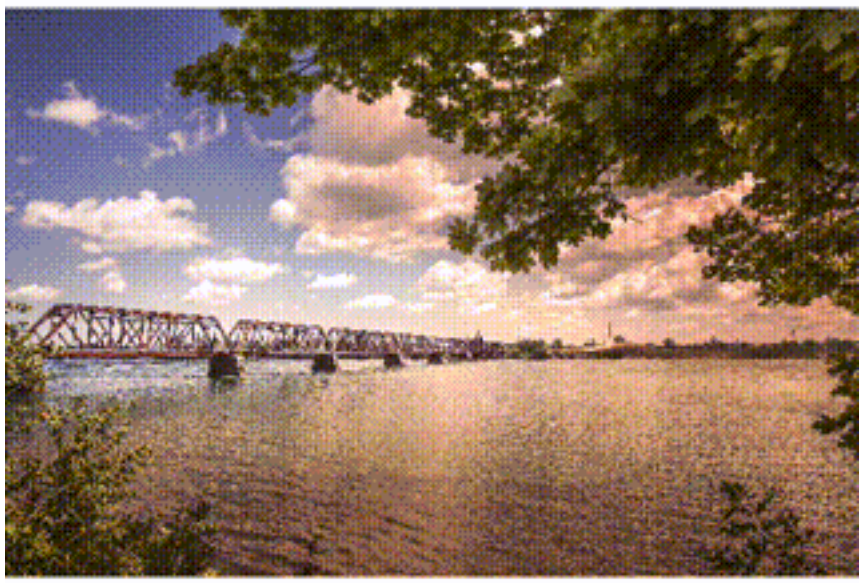
Build and run and you will see the photo is now in the middle.

If you noticed, when you zooming out the photo and then you leave it, The photo goes to the left first and then comes to the middle. It’s not good.



To solve this problem, we need to implement another of `UIScrollViewDelegate` methods and call `centerImage()` method into it too. Add this method after `viewForZooming(in scrollView: UIScrollView) -> UIView? {…}` method:

Problem solved!



> ***Note:*** *Have you noticed that bouncing mode when you zoom out the photo more than its minimumZoomScale and then you leave it? This is because of a property of* `UIScrollView` *named* `bounces`*, which is a Boolean value that controls whether the scroll view bounces past the edge of content and back again. If the value of this property is* `true`*, the scroll view bounces when it encounters a boundary of the content. Bouncing visually indicates that scrolling has reached an edge of the content. If the value is* `false`*, scrolling stops immediately at the content boundary without bouncing. The default value of this property is* `true`*. Try to add* `self.bounces = false` *in to* `override init(frame: CGRect) {`*, and see what happens.*

Turn to Landscape!

If you turn your phone to landscape, you will see photo falling to down left of screen.

The photo is falling down letft in landscape, because of wrong static layout

That's because we've not set up `imageScrollView` layout yet. Go to **ViewController.swift** file and add This method after `viewDidLoad() {…}` method:

In this method, we fixed all edges of imageScrollView to the **viewControllers** `view`. So if you turn phone to landscape `imageScrollView` will be turn with **viewControllers** `view` too.

> ***Note:*** *If you're not familiar with using* ***layout constraints*** *programmatically, it's very similar to using layout constraints in story board. Pay attention to the above codes. Be sure you will learn very soon.*
> *just be careful each time you set layout programmatically be sure to set* `translatesAutoresizingMaskIntoConstraints` *of the view to* `false`*.*
> *Because at the default this property is on* `true` *and it means system automatically creates a set of constraints that duplicate the behavior specified by the view's autoresizing mask (Some static, frame-based layout within Auto Layout). Therefore, you cannot add additional constraints to modify this size or position without introducing conflicts. If you want to use Auto Layout to dynamically calculate the*

> *size and position of your view, you must set this property to `false`.*

Add `self.layoutImageScrollView()` into the `viewDidLoad()` method after `self.view.addSubview(self.imageScrollView)` line:

```
override func viewDidLoad() {
    super.viewDidLoad()

    //1. Initialize imageScrollView and adding it to viewControllers view
    self.imageScrollView = ImageScrollView(frame: self.view.bounds)
    self.view.addSubview(self.imageScrollView)
    self.layoutImageScrollView()
```

Build and run the app. The photo does no longer fall!

Photo in landscape, with correct layout. But Scroll view must reset its zoom scale range in landscape

Now there is another problem. As you see the photo is not fit to screen anymore. That's because we did not tell to the `imageScrollView` we turned phone to landscape so reset your scale range please!
But this is not all problems. Try to zoom in photo in portrait and then turn iPhone to landscape, If you look carefully, you'll notice zooming location on the photo is changed. Our plan for this problem is we save the position and zoom scale of photo before changes of rotation be applied on screen and then ask to the `imageScrollView` to restore to our saved scale and position of zooming!

Zoomed photo in portrait

Zoomed photo in landscape. As you see the zoom location is changed.

So let's do it. Go to **ImageScrollView.swift** file and add these methods at the end of `ImageScrollView` class after `scrollViewDidZoom(_ scrollView: UIScrollView) {…}` method:

Don't be afraid! Most of the codes are clear and are commented. But I try to explain some Some obscure parts:

1. In `pointToCenterAfterRotation()` method we calculate the center

position of the screen, this would be the center position of `ImageScrollView` to. Meaning of `convert(boundsCenter, to: zoomView)` is we ask `ImageScrollerView` class to tell us what is its center point on screen in the zoomView coordinate?

2. `CGFloat.ulpOfOne` In `scaleToRestoreAfterRotation()`, is like epsilon in the math. It means : *The positive difference between 1.0 and the next greater representable number.*
   We want to be sure we aren't out of zoomScale range and if we are, we just return `0.0` as `zoomScale` and system automatically restores `zoomScale` to the `minimumZoomScale`.
3. **contentOffset** in `maximumContentOffset()` and `minimumContentOffset()` is the amount of scrollView content that is out of screen. We calculate the maximum and minimum of contentOffset to be sure the center point that we saved before rotation, is not out of range.

I think there is no dark side in `restoreCenterPoint(to oldCenter: CGPoint, oldScale: CGFloat)` method. So let's do the rest of job to fix our scroll view problem.

Go to **ViewController.swift** file and add these methods after `viewDidLoad() {…}` method and before `layoutImageScrollView() {…}` method:

In the above codes, `viewWillTransition(to size: CGSize, with coordinator: UIViewControllerTransitionCoordinator)` is a method that **UIKit** calls this method before changing the size of a presented view controller's view and this method notifies the container that the size of its view is about to change.
The `size` parameter giving us the new `size` for the container's view.
So we get the new `size` and calculating new frame for `imageScrollView` saving `imageScrollView` last center point and last zoom Scale, give to `imageScrollView` new frame ask it to reset its zoom scale range and restore its center point and zoom scale in new frame and that's all!

Build and run the app.

Enjoy of turning your phone to landscape!

Now scroll view resets the range of its zoom scale in landscape

Zoomed photo in portrait

Zoomed photo in landscape. Now the zoom location is fixed

One more thing!

The scroll view now works perfect. But do you want to add a feature that makes your user more happy?
That is tapping, that enables user to zoom with twice tapping on screen. Let's add it. Go to **ImageScrollView.swift** file and add these lines of code at top of the `ImageScrollView` subclass, after `var zoomView: UIImageView!`:

This is a lazy instance of `UITapGestureRecognizer`. The reason of that we make it lazy is it's not being initialized until user taps on screen and as user taped it being initialized and cached in memory and after that, it's available in memory without need to initialize again. Xcode now gives you an error about that it does not know any action method with `handleZoomingTap(_:)` name. We fix that now, add these lines of code at the end of `ImageScrollView` subclass, after `restoreCenterPoint(oldCenter: CGPoint, oldScale: CGFloat)` method:

When user taps on photo, `zoomingTap` gesture recognizer calls `handleZoomingTap(_ sender: UITapGestureRecognizer)` method.
In this method we get the location of users tap on `zoomView`. `zoomingTap` itself gives us this location because we will attach it to `zoomView`.

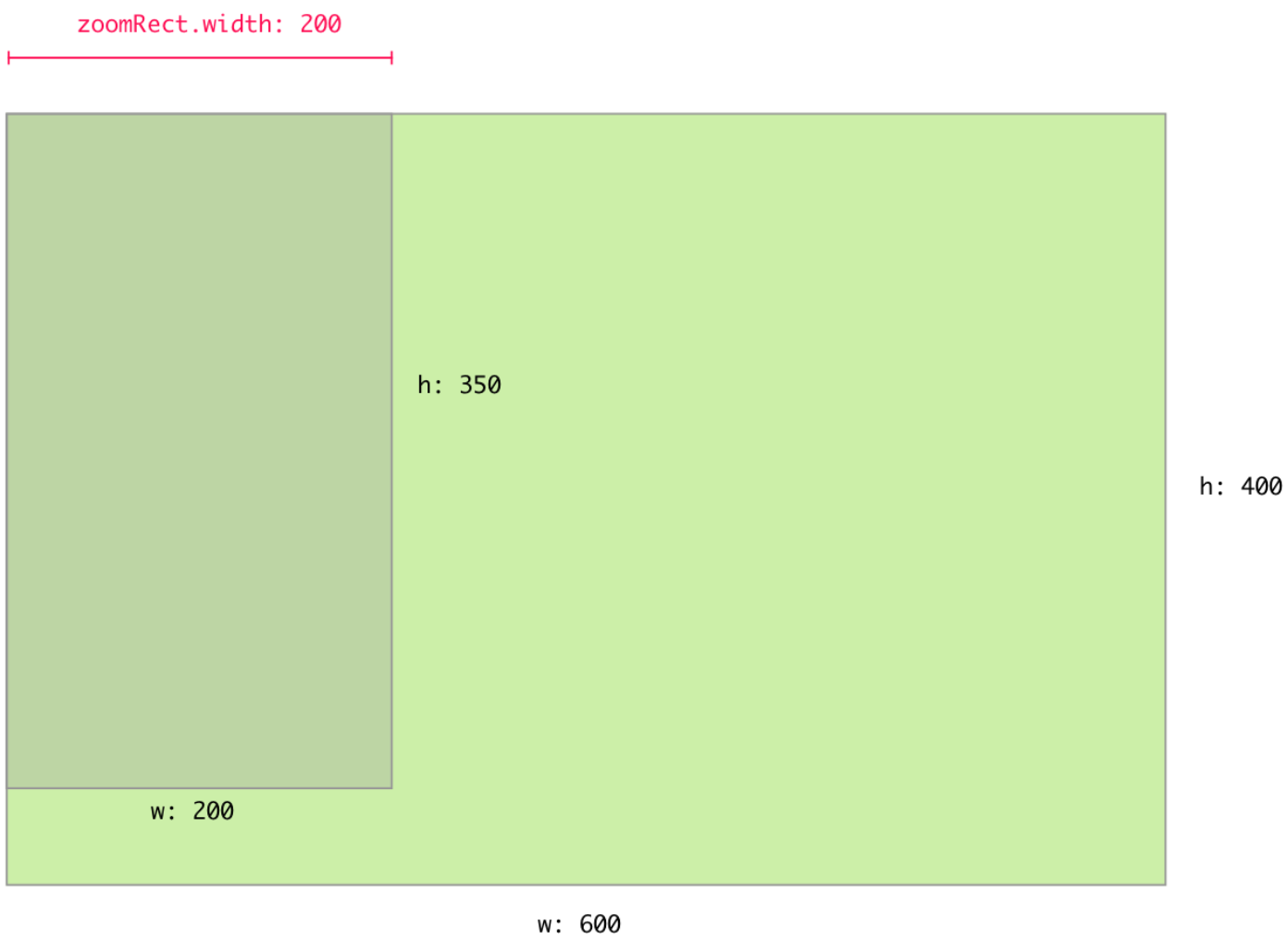
`UIScrollView` has a method that zooms to a specific area of the content that we specified. But this method gets a rectangle with coordinate of content view (which here is `zoomView`) as zoom area. So we need convert the point that zoomingTap gives us to this area. We do this in `zoomRect(for scale: CGFloat, withCenter center: CGPoint)` method. But how?

We set `maximumZoomScale` as final zoom scale when user taps on photo. For a rect area we need for value: `x`, `y`, `width` and `height`.
The `width` and `height` are the part of photo that fits in screen at final zoom scale. We obtain these values by subtracting screen width/height on final zoom Scale (*Don't use these codes in Xcode*):

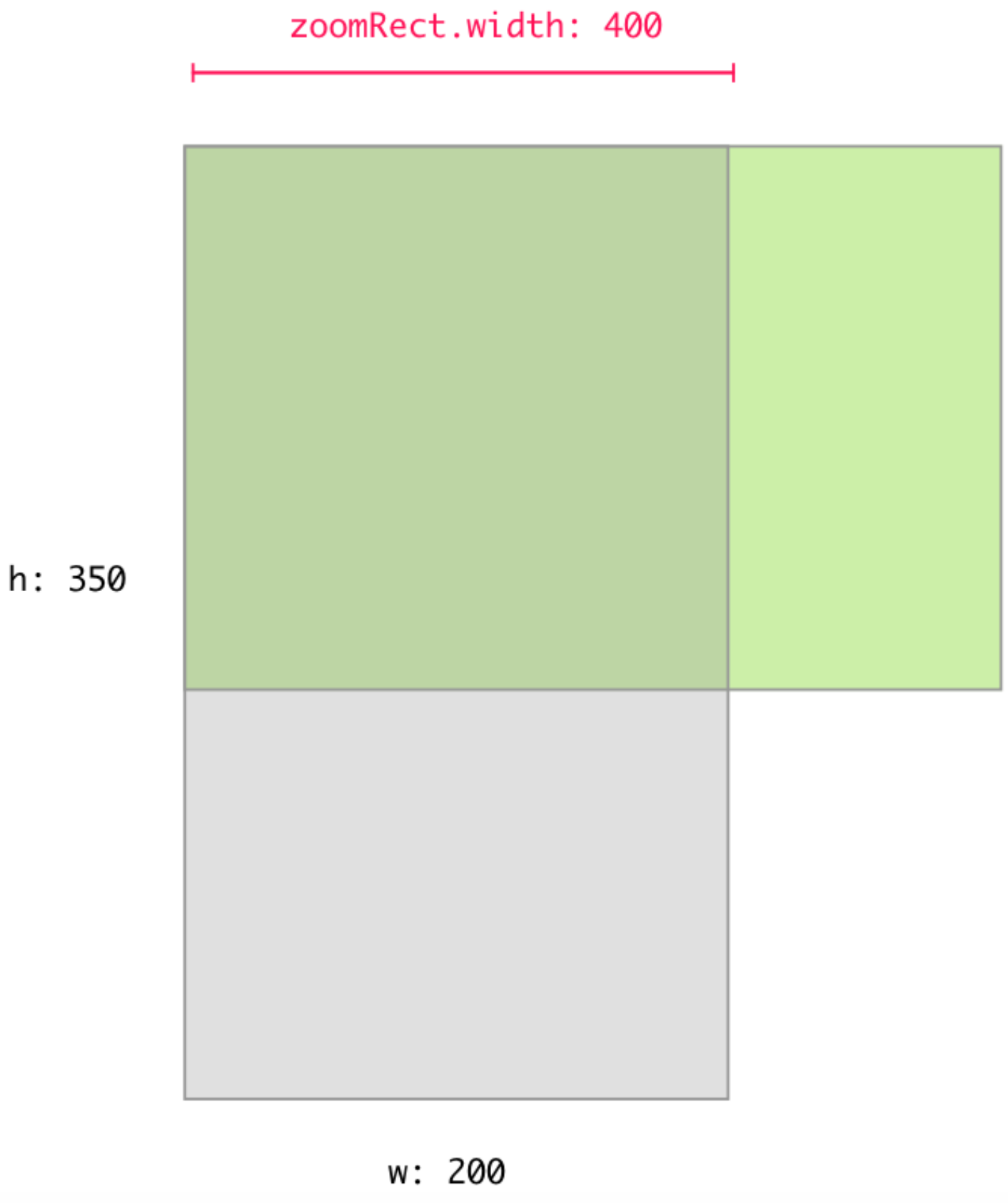
```
zoomRect.width = phoneScreen.width / finalzoomScale

zoomRect.height = phoneScreen.height / finalZoomScale
```

See blew pictures for a better understanding:

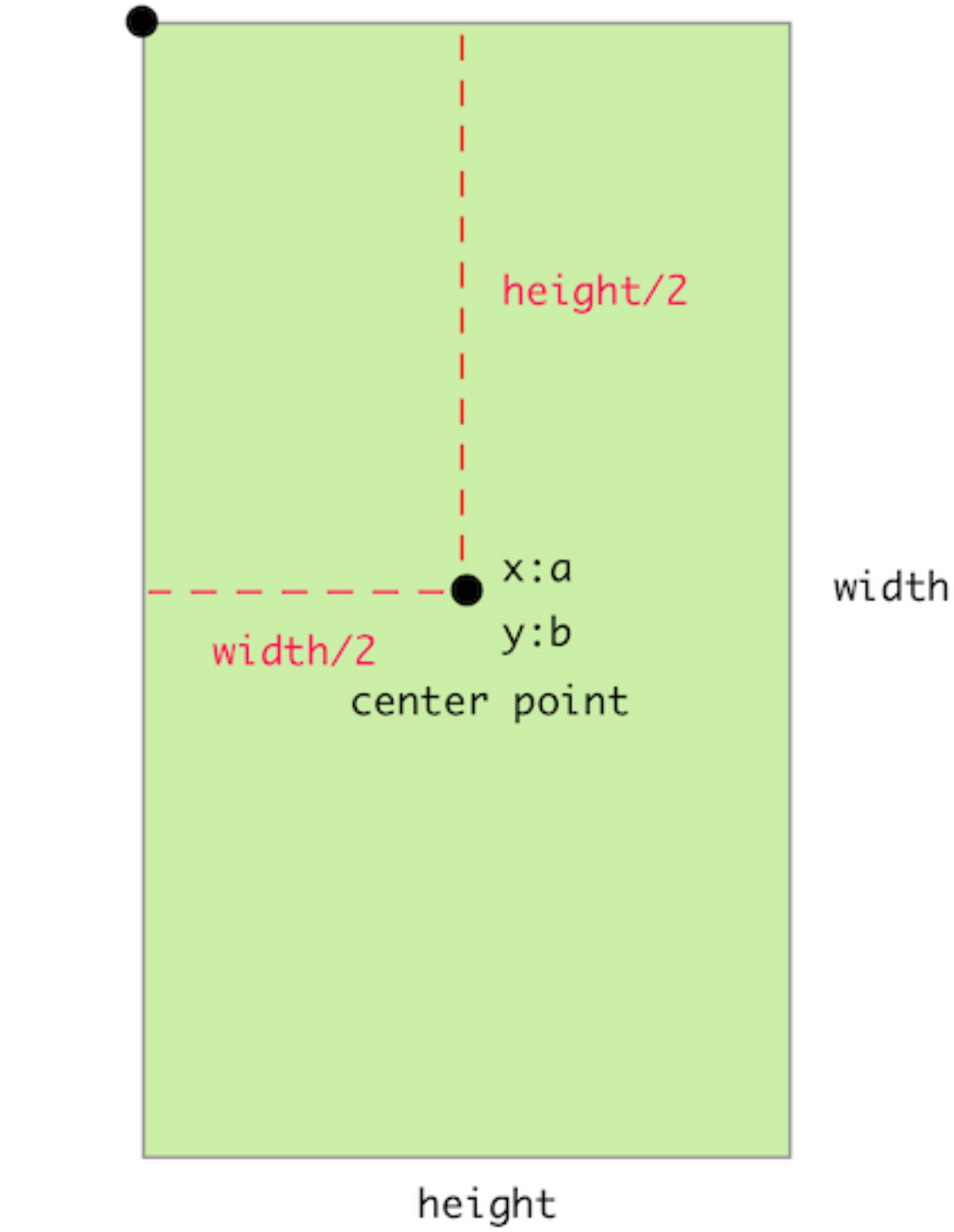


width of zoomRect at **1.0** zoom scale. As you can see it is **equal** to the width of screen.



width of zoomRect at **0.5** zoom scale. As you can see it is **twice** of the width of screen.

To know that how we calculate value of `x` and `y` see blew picture:



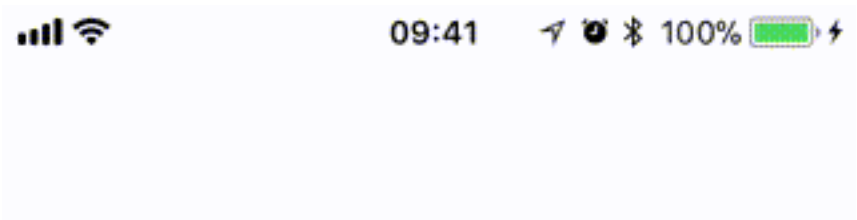
Finally add these two lines into the `configureFor(_ imageSize: CGSize)` method after `self.zoomScale = self.minimumZoomScale` :

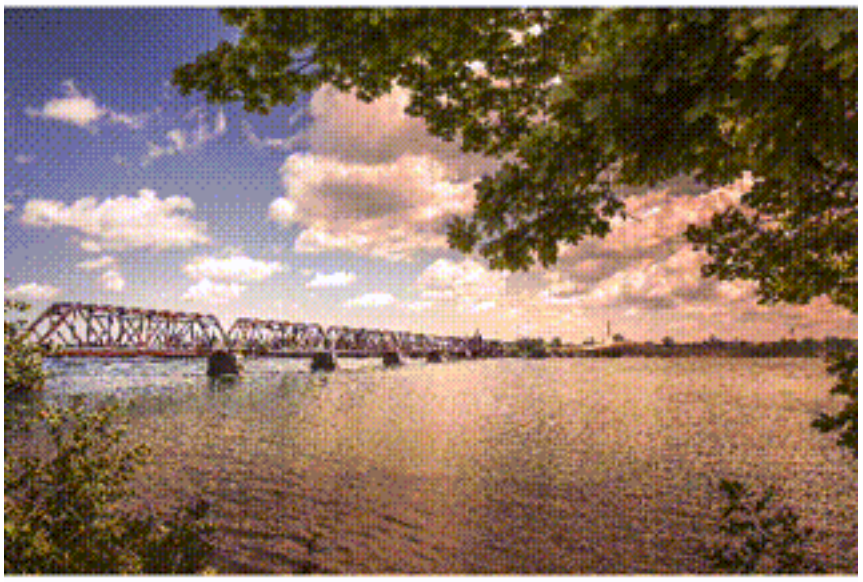
`configureFor(_ imageSize: CGSize)` method should be like this:

```
func configureFor(_ imageSize: CGSize) {
    self.contentSize = imageSize
    self.setMaxMinZoomScaleForCurrentBounds()
    self.zoomScale = self.minimumZoomScale

    //Enable zoom tap
    self.zoomView.addGestureRecognizer(self.zoomingTap)
    self.zoomView.isUserInteractionEnabled = true
}
```

First line attaches `zoomingTap` gesture recognizer to `zoomView` and second lines enables `zoomView` to accept our taps.
Build and run the app and enjoy of tapping!





Users can now zoom in/out by tapping

You can download the completed sample code of this tutorial from here.

Please clap this tutorial and share it to your friends if you enjoyed it.

You can study the next two parts of this tutorial from the links below: