2017 Jan, MOSA

UI/NSScrollView CoreGraphics / Metal

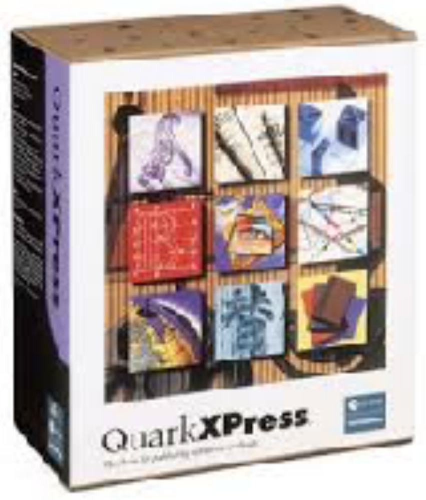
Kaz Yoshikawa

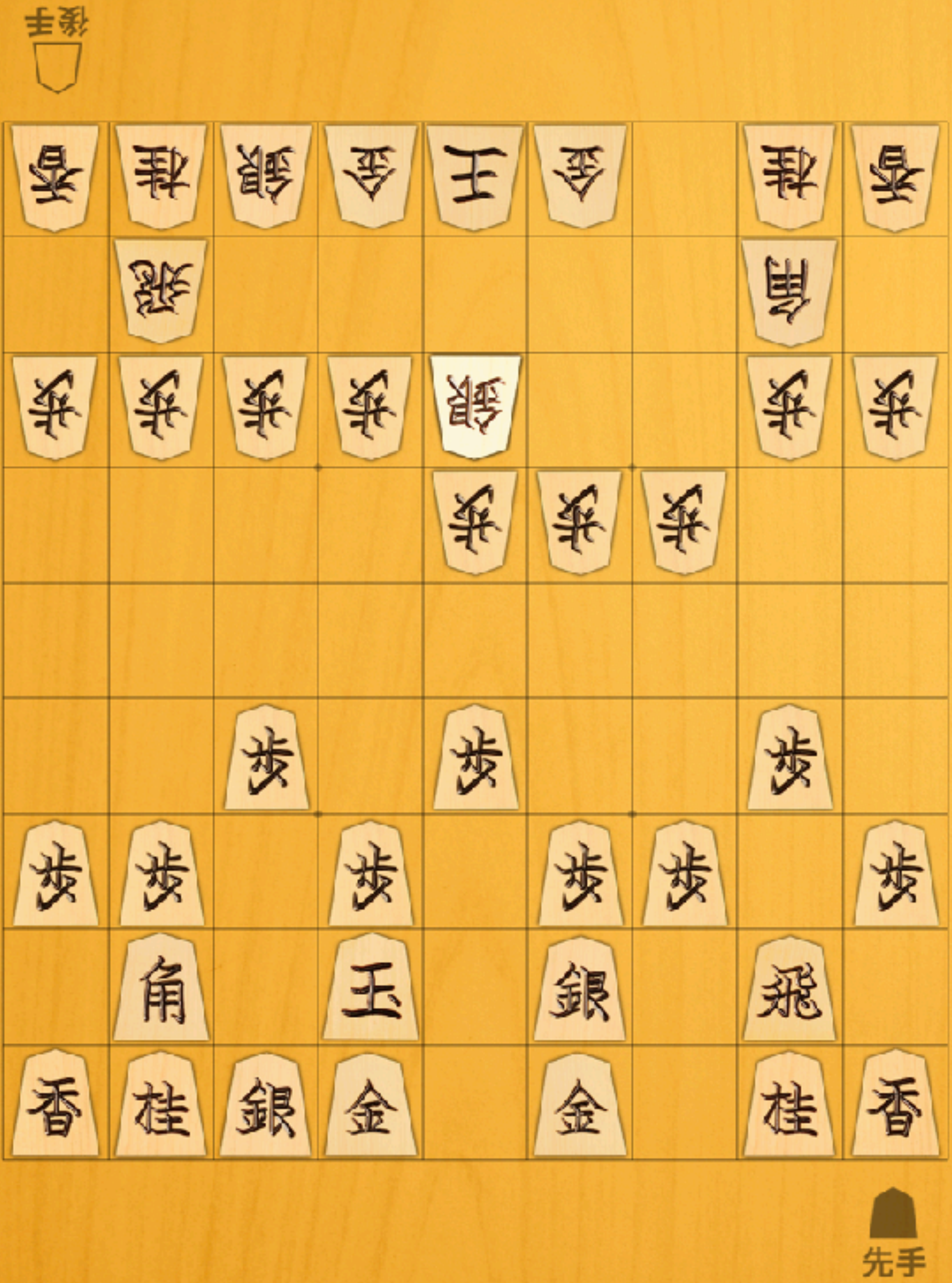
About me

Kaz Yoshikawa

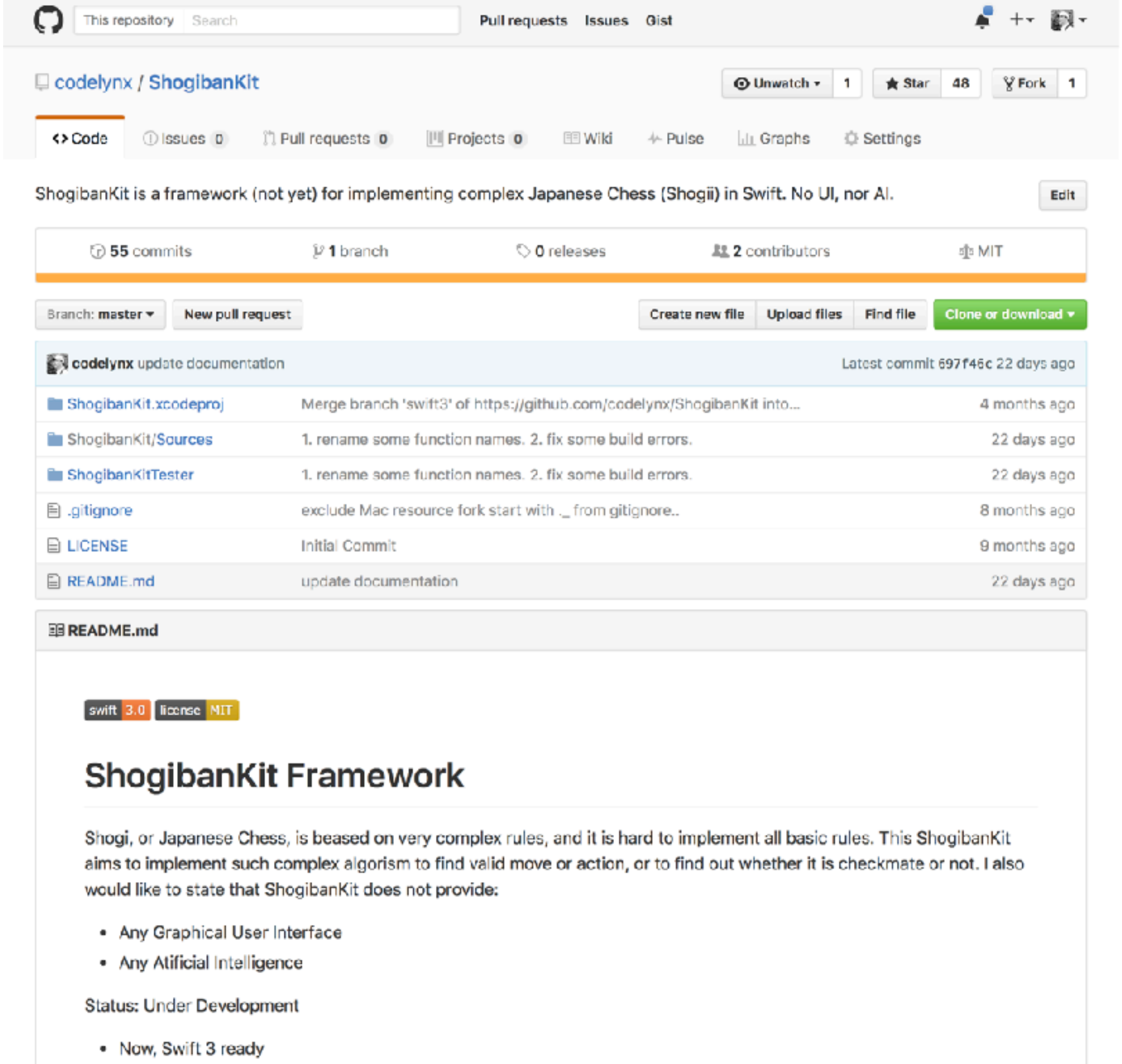
- Electricwoods LLC 代表 / Digital Lynx Systems Inc. 副代表
  - e-mail: kyoshikawa@electricwoods.com
  - twitter: @codelynx1
- Working History
  - Adobe Systems (Tokyo)
  - Lionbridge (Tokyo)
  - Quark (Tokyo / Denver)
  - Hummingbird Communications (Mt. View, USA)
  - Fact International (Vancouver, Canada)
  - Perle Systems (Toronto, Canada), etc.

Involved Products




後手
先手

ShogibanKit



ShogibanKit is a framework (not yet) for implementing complex Japanese Chess (Shogii) in Swift. No UI, nor AI.

codelynx update documentation — Latest commit 697f46c 22 days ago

ShogibanKit.xcodeproj	Merge branch 'swift3' of https://github.com/codelynx/ShogibanKit into...	4 months ago
ShogibanKit/Sources	1. rename some function names. 2. fix some build errors.	22 days ago
ShogibanKitTester	1. rename some function names. 2. fix some build errors.	22 days ago
.gitignore	exclude Mac resource fork start with ._ from gitignore..	8 months ago
LICENSE	Initial Commit	9 months ago
README.md	update documentation	22 days ago

ShogibanKit Framework

Shogi, or Japanese Chess, is beased on very complex rules, and it is hard to implement all basic rules. This ShogibanKit aims to implement such complex algorism to find valid move or action, or to find out whether it is checkmate or not. I also would like to state that ShogibanKit does not provide:

- Any Graphical User Interface
- Any Atificial Intelligence

Status: Under Development

- Now, Swift 3 ready

Do you use UI/NS ScrollView?

Are they zoomable?

Have you had any issues about zooming?

Do they work on both iOS and macOS?

Core Graphics

+

UI/NSScrollView

Zoomable Contents

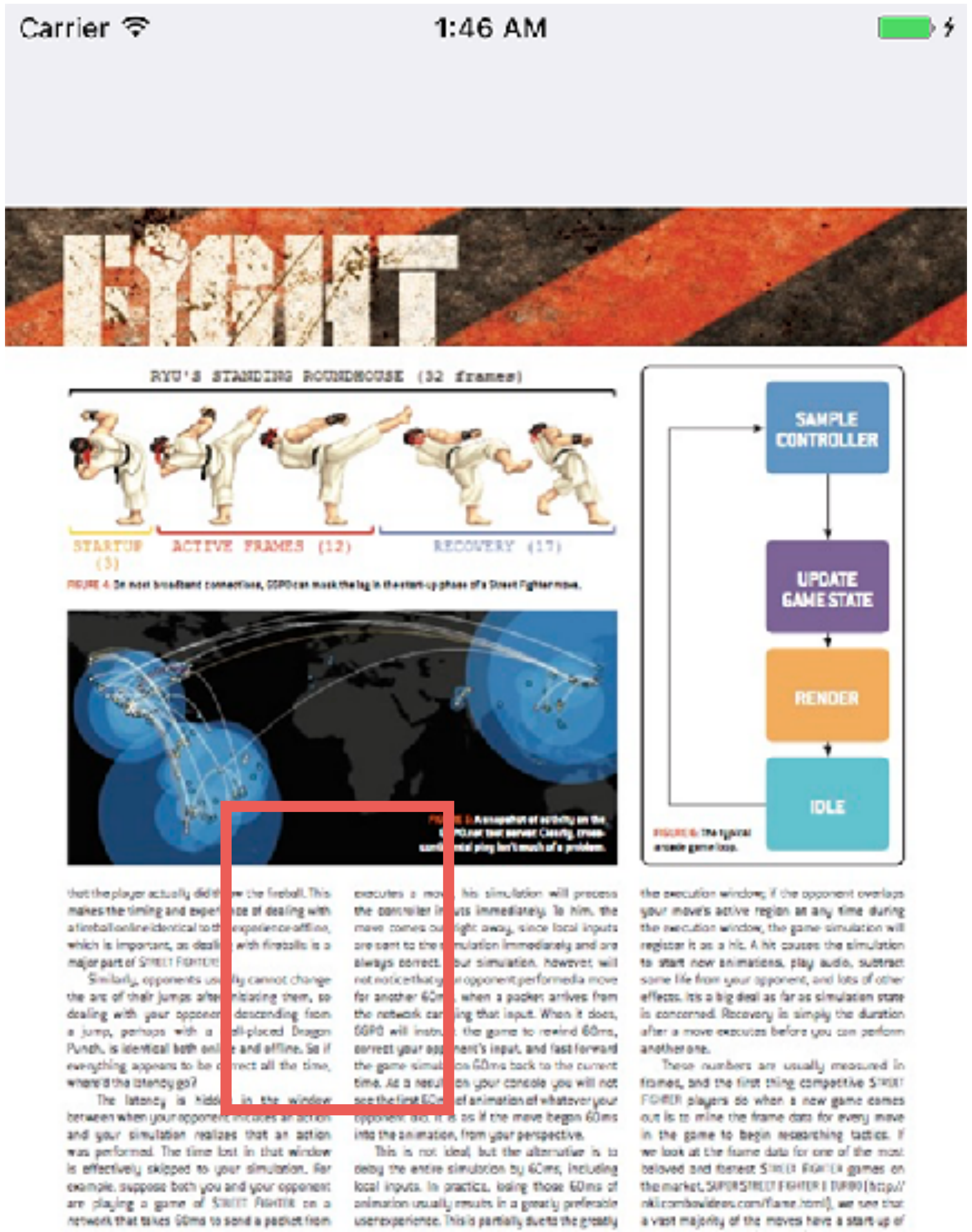


FIGURE 4: On most broadband connections, GGPO can mask the lag in the start-up phase of a Street Fighter move.

FIGURE 5: A snapshot of activity on the GGPO.net test server. Clearly, cross-continental play isn't much of a problem.

FIGURE 6: The typical arcade game loop.

that the player actually did throw the fireball. This makes the timing and experience of dealing with a fireball online identical to the experience offline, which is important, as dealing with fireballs is a major part of STREET FIGHTER.

Similarly, opponents usually cannot change the arc of their jumps after initiating them, so dealing with your opponent descending from a jump, perhaps with a well-placed Dragon Punch, is identical both online and offline. So if everything appears to be correct all the time, where'd the latency go?

The latency is hidden in the window between when your opponent initiates an action and your simulation realizes that an action was performed. The time lost in that window is effectively skipped in your simulation. For example, suppose both you and your opponent are playing a game of STREET FIGHTER on a network that takes 60ms to send a packet from

executes a move, his simulation will process the controller inputs immediately. To him, the move comes out right away, since local inputs are sent to the simulation immediately and are always correct. Your simulation, however, will not notice that your opponent performed a move for another 60ms, when a packet arrives from the network carrying that input. When it does, GGPO will instruct the game to rewind 60ms, correct your opponent's input, and fast forward the game simulation 60ms back to the current time. As a result, on your console you will not see the first 60ms of animation of whatever your opponent did. It is as if the move began 60ms into the animation, from your perspective.

This is not ideal, but the alternative is to delay the entire simulation by 60ms, including local inputs. In practice, losing those 60ms of animation usually results in a greatly preferable user experience. This is partially due to the greatly

the execution window; if the opponent overlaps your move's active region at any time during the execution window, the game simulation will register it as a hit. A hit causes the simulation to start new animations, play audio, subtract some life from your opponent, and lots of other effects. It's a big deal as far as simulation state is concerned. Recovery is simply the duration after a move executes before you can perform another one.

These numbers are usually measured in frames, and the first thing competitive STREET FIGHTER players do when a new game comes out is to mine the frame data for every move in the game to begin researching tactics. If we look at the frame data for one of the most beloved and fiercest STREET FIGHTER games on the market, SUPER STREET FIGHTER II TURBO (http://wiki.comboviddeos.com/frame.html), we see that a vast majority of the moves have a start up of

Blurring when zoom!?

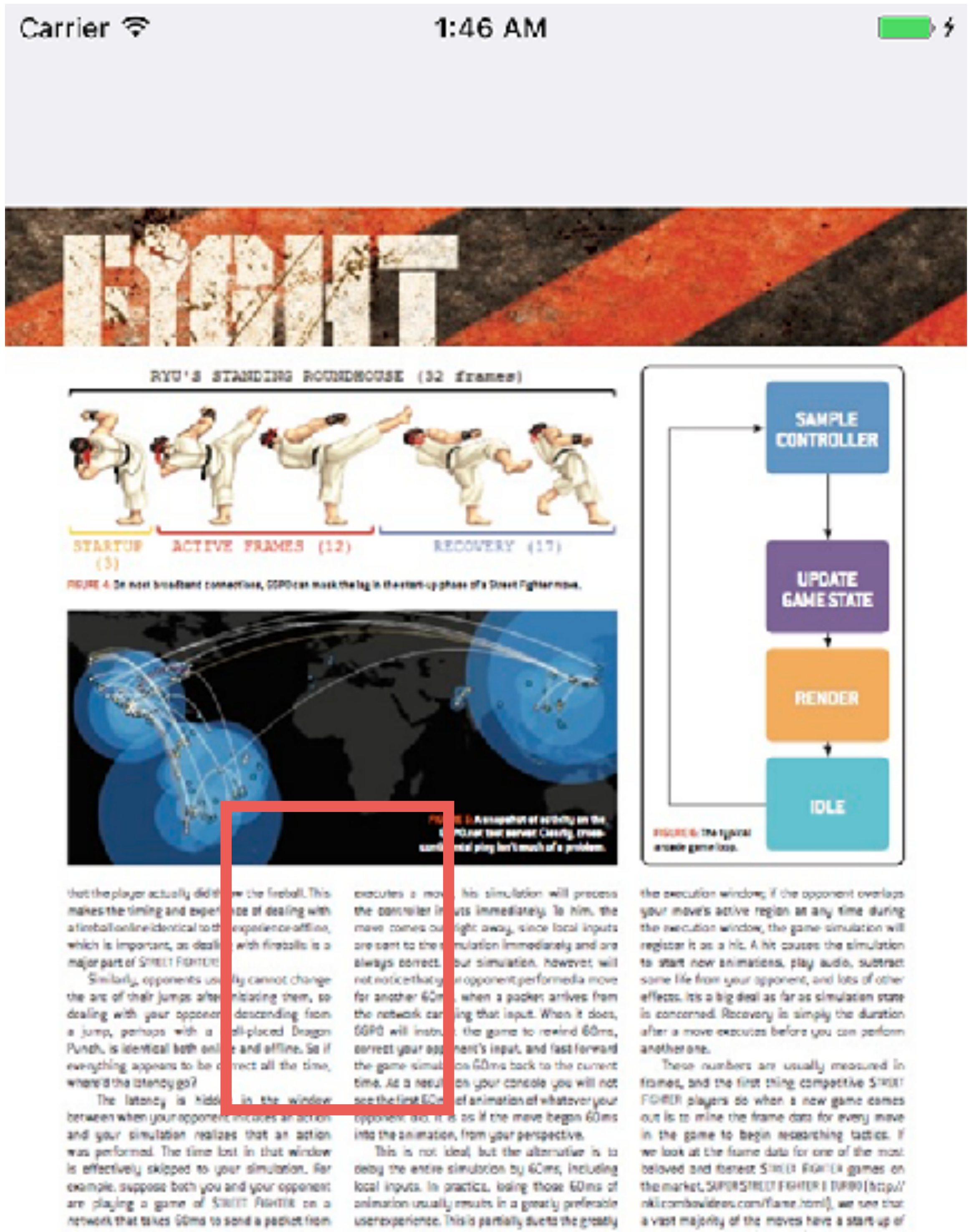


FIGURE 4: On most broadband connections, GGPO can mask the lag in the start-up phase of a Street Fighter move.

FIGURE 5: A snapshot of activity on the GGPO.net test server. Clearly, cross-continental ping isn't much of a problem.

FIGURE 6: The typical arcade game loop.

that the player actually did throw the fireball. This makes the timing and experience of dealing with a fireball online identical to the experience offline, which is important, as dealing with fireballs is a major part of STREET FIGHTER!

Similarly, opponents usually cannot change the arc of their jumps after initiating them, so dealing with your opponent descending from a jump, perhaps with a well-placed Dragon Punch, is identical both online and offline. So if everything appears to be correct all the time, where'd the latency go?

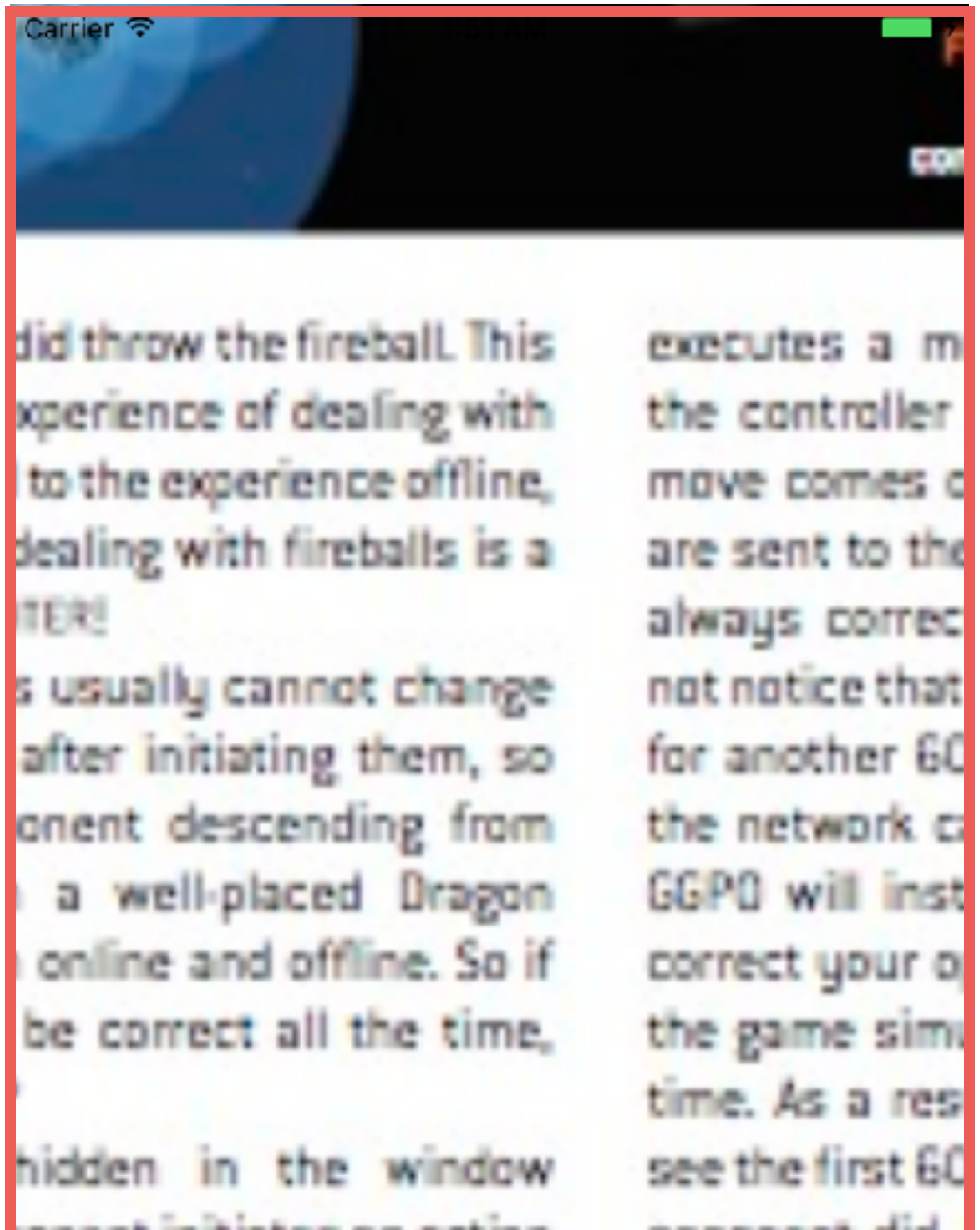
The latency is hidden in the window between when your opponent initiates an action and your simulation realizes that an action was performed. The time lost in that window is effectively skipped to your simulation. For example, suppose both you and your opponent are playing a game of STREET FIGHTER on a network that takes 60ms to send a packet from

executes a move, his simulation will process the controller inputs immediately. To him, the move comes out right away, since local inputs are sent to the simulation immediately and are always correct. Your simulation, however, will not notice that your opponent performed a move for another 60ms, when a packet arrives from the network carrying that input. When it does, GGPO will instruct the game to rewind 60ms, correct your opponent's input, and fast forward the game simulation 60ms back to the current time. As a result, on your console you will not see the first 60ms of animation of whatever your opponent did. It is as if the move began 60ms into the animation, from your perspective.

This is not ideal, but the alternative is to delay the entire simulation by 60ms, including local inputs. In practice, losing those 60ms of animation usually results in a greatly preferable user experience. This is partially due to the greatly

the execution window; if the opponent overlaps your move's active region at any time during the execution window, the game simulation will register it as a hit. A hit causes the simulation to start new animations, play audio, subtract some life from your opponent, and lots of other effects. It's a big deal as far as simulation state is concerned. Recovery is simply the duration after a move executes before you can perform another one.

These numbers are usually measured in frames, and the first thing competitive STREET FIGHTER players do when a new game comes out is to mine the frame data for every move in the game to begin researching tactics. If we look at the frame data for one of the most beloved and fiercest STREET FIGHTER games on the market, SUPER STREET FIGHTER II TURBO (http://wiki.comboideas.com/frame.html), we see that a vast majority of the moves have a start up of

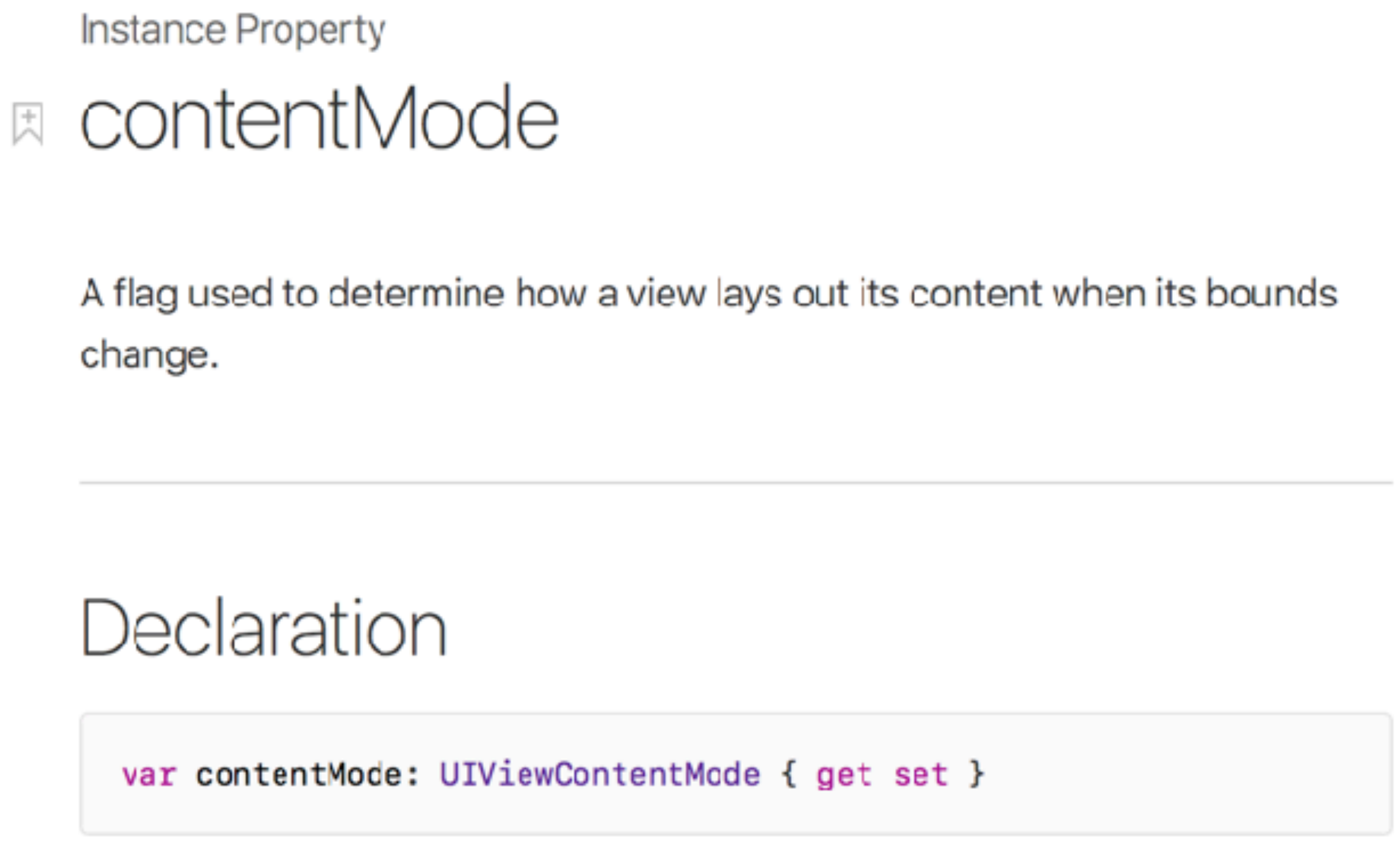


did throw the fireball. This
xperience of dealing with
to the experience offline,
dealing with fireballs is a
TER!
s usually cannot change
after initiating them, so
onent descending from
a well-placed Dragon
online and offline. So if
be correct all the time,

hidden in the window

executes a m
the controller
move comes o
are sent to the
always correc
not notice that
for another 60
the network c
GGPO will inst
correct your o
the game simu
time. As a res
see the first 60

Set contentMode to redraw…



Instance Property

contentMode

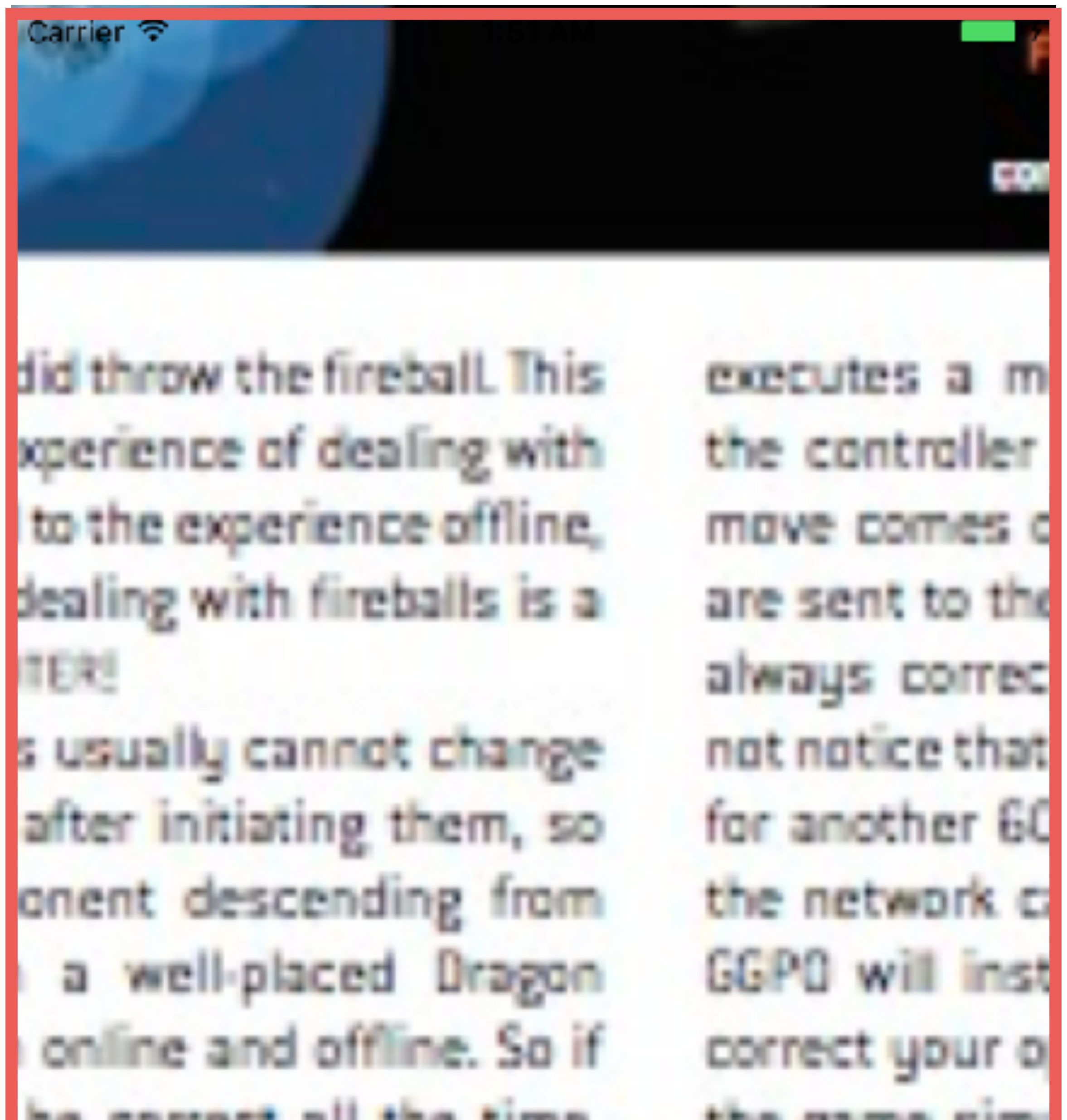
A flag used to determine how a view lays out its content when its bounds change.

Declaration

```
var contentMode: UIViewContentMode { get set }
```

Nope… Still Blurry

- contentMode to redraw won't help

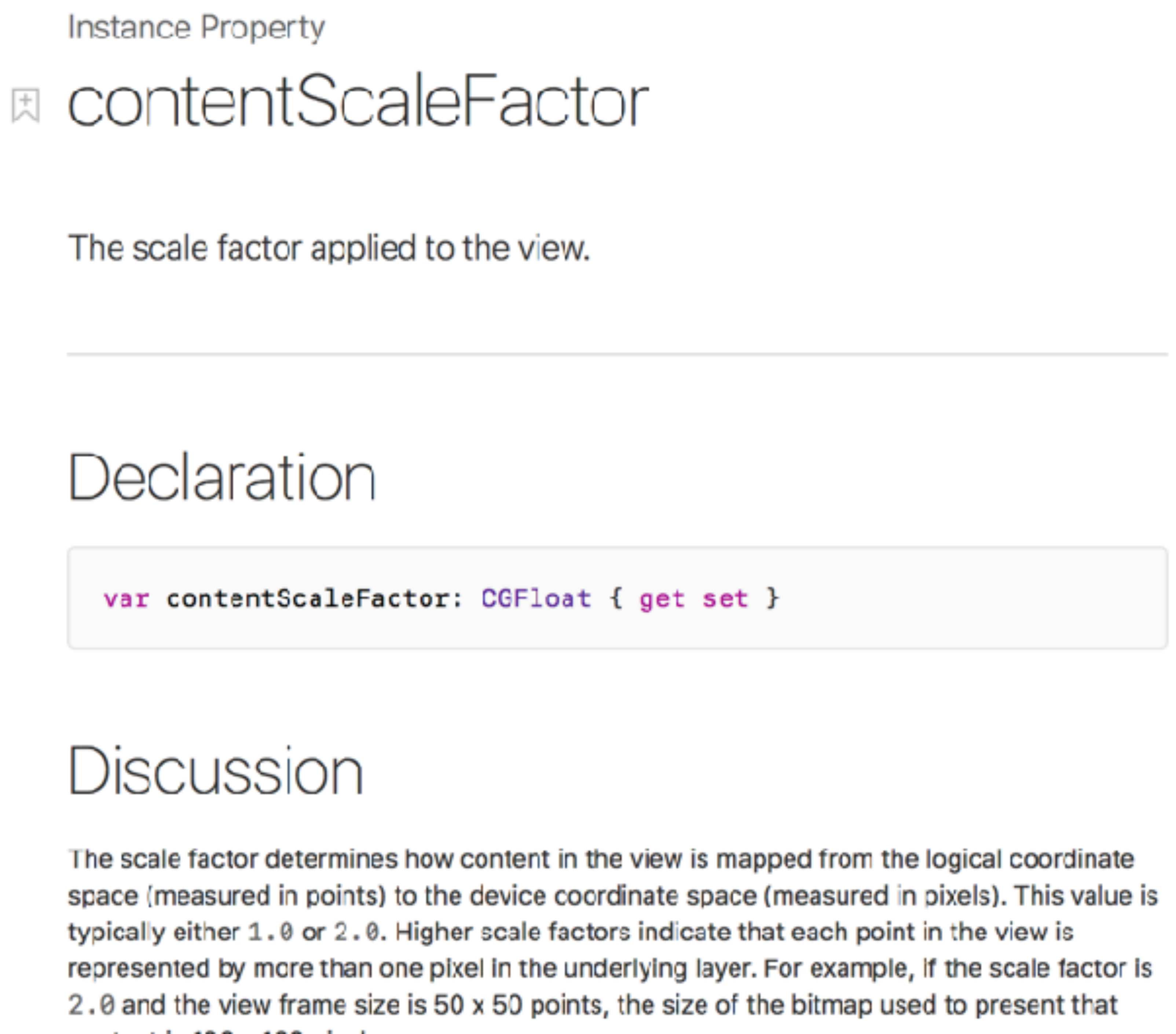


Carrier

did throw the fireball. This
xperience of dealing with
to the experience offline,
dealing with fireballs is a
TER!
s usually cannot change
after initiating them, so
onent descending from
a well-placed Dragon
online and offline. So if

executes a m
the controller
move comes
are sent to the
always correc
not notice that
for another 60
the network c
GGPO will inst
correct your o

update contentScaleFactor …



Instance Property

contentScaleFactor

The scale factor applied to the view.

Declaration

```
var contentScaleFactor: CGFloat { get set }
```

Discussion

The scale factor determines how content in the view is mapped from the logical coordinate space (measured in points) to the device coordinate space (measured in pixels). This value is typically either `1.0` or `2.0`. Higher scale factors indicate that each point in the view is represented by more than one pixel in the underlying layer. For example, if the scale factor is `2.0` and the view frame size is 50 x 50 points, the size of the bitmap used to present that

contentScaleFactor ⚠️

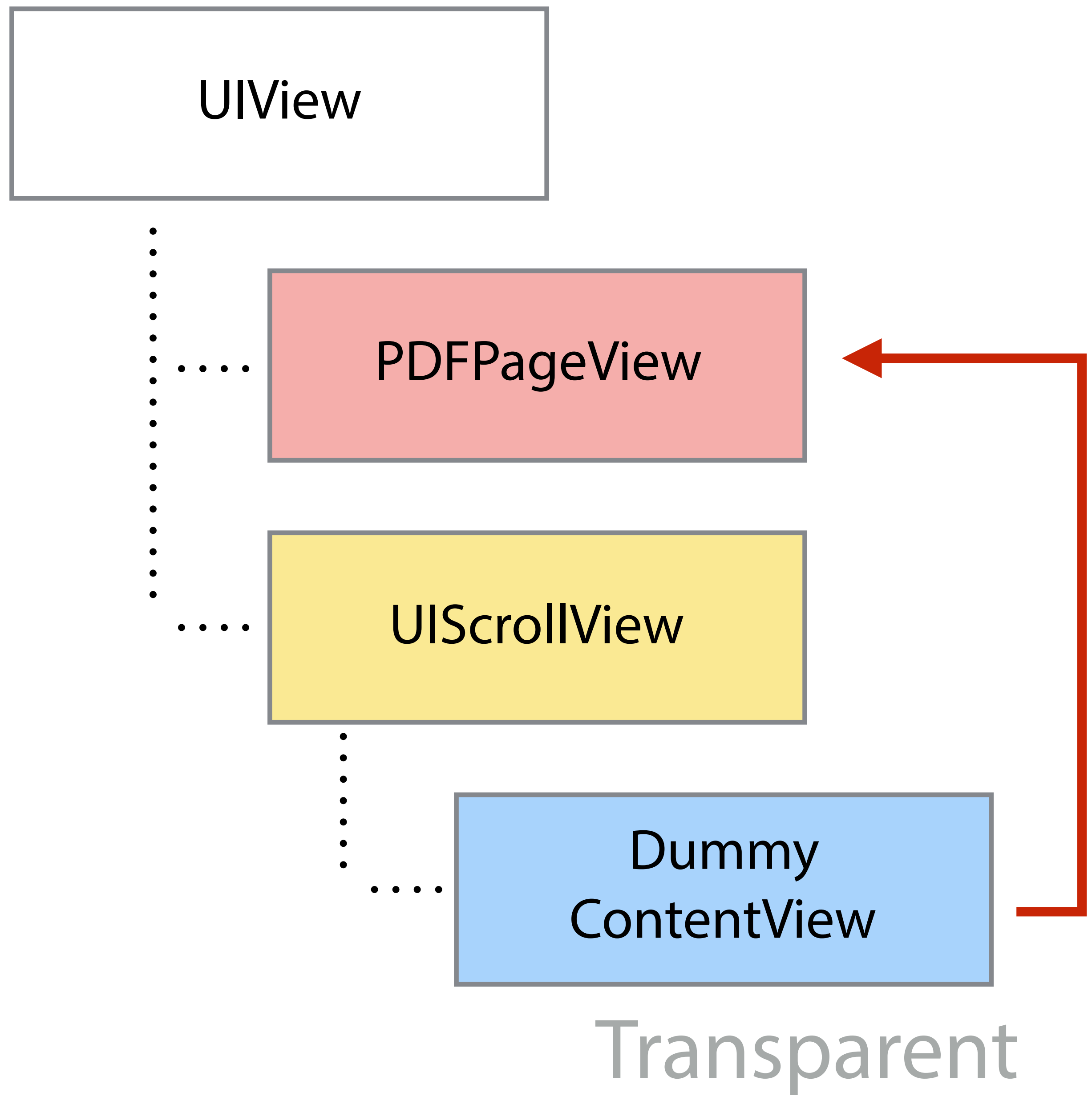
```
func scrollViewDidZoom(_ scrollView: UIScrollView) {

    self.pdfPageView.contentScaleFactor =
            self.scrollView.zoomScale * UIScreen.main.scale

}
```

- OK – Device size like small contents
- NG – Magazine / Newspaper like large contents
    - eg. 2048x1536 12MB — (4x zoom) — 8192 x 6144 48MB

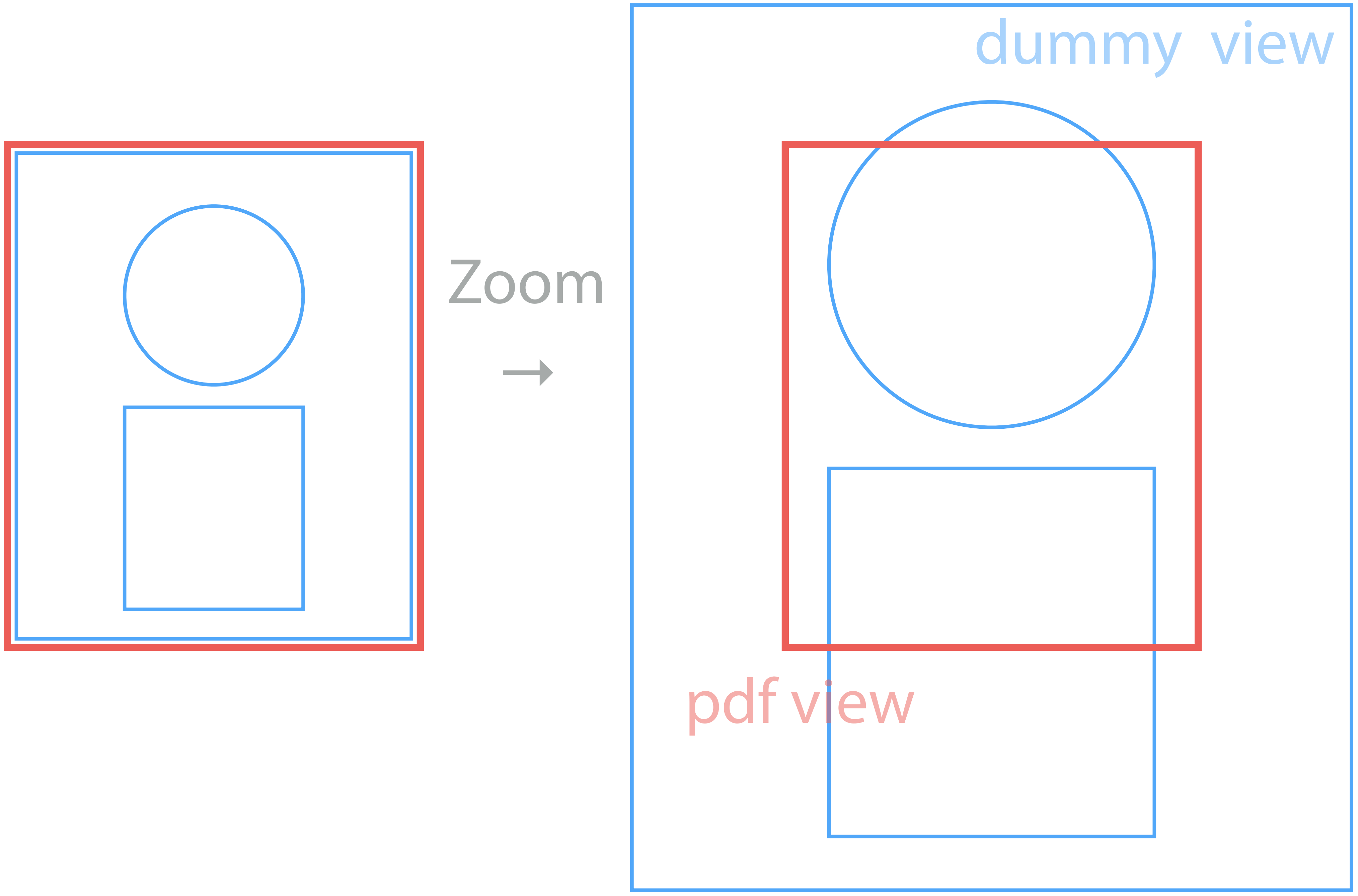
Tips

Dummy Content View

Dummy Content View



PDF view draws based on dummy view coordinate



Convert rect from content dummy view

```
  override func draw(_ layer: CALayer, in ctx: CGContext) {
    UIGraphicsPushContext(ctx)
    ctx.saveGState()

    let box = page.getBoxRect(.cropBox)
→   let rect = self.contentView.convert(self.contentView.bounds, to: self)
    ctx.translateBy(x: rect.minX, y: rect.minY)
    ctx.translateBy(x: 0, y: rect.height)
    ctx.scaleBy(x: 1, y: -1)

    ctx.scaleBy(x: rect.width / box.width, y: rect.height / box.height)

    ctx.drawPDFPage(page)

    ctx.restoreGState()
    UIGraphicsPopContext()
  }
```

UIScrollView delegate

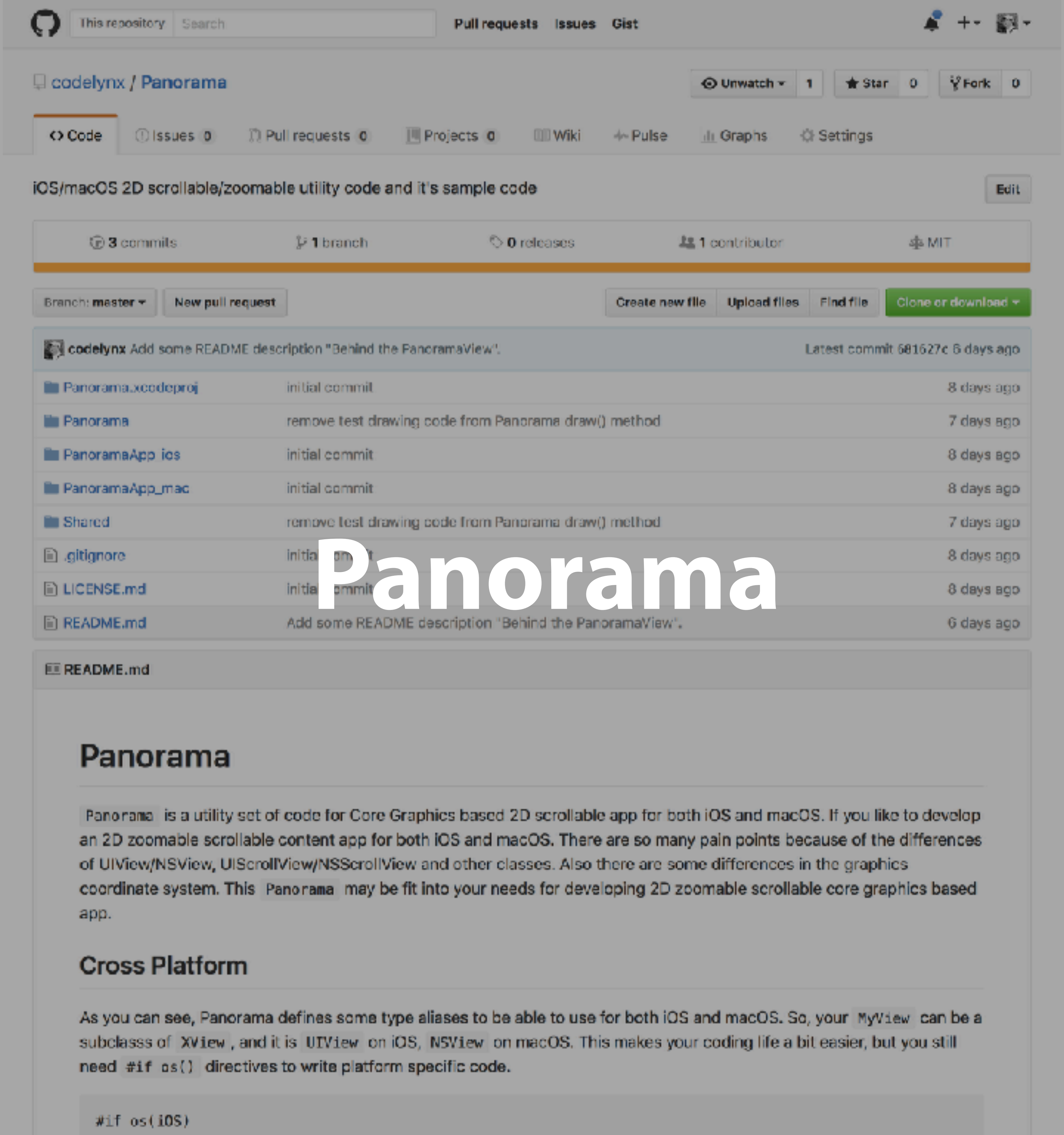
```
func viewForZooming(in scrollView: UIScrollView) -> UIView? {
  return contentView
}

func scrollViewDidZoom(_ scrollView: UIScrollView) {
  self.pdfPageView.setNeedsDisplay()
}

func scrollViewDidScroll(_ scrollView: UIScrollView) {
  self.pdfPageView.setNeedsDisplay()
}
```

Draw it

```
class PDFPageView: UIView {
    // ...
   override func setNeedsDisplay() {
        super.setNeedsDisplay()
        self.layer.setNeedsDisplay()
    }

    override func layoutSubviews() {
        super.layoutSubviews()
        self.layer.drawsAsynchronously = true
    }
    // ...
}
```

I got the idea
Is there easy way?

Panorama



iOS/macOS 2D scrollable/zoomable utility code and it's sample code

codelynx Add some README description "Behind the PanoramaView". Latest commit 681627c 6 days ago

Panorama.xcodeproj	initial commit	8 days ago
Panorama	remove test drawing code from Panorama draw() method	7 days ago
PanoramaApp_ios	initial commit	8 days ago
PanoramaApp_mac	initial commit	8 days ago
Shared	remove test drawing code from Panorama draw() method	7 days ago
.gitignore	initial commit	8 days ago
LICENSE.md	initial commit	8 days ago
README.md	Add some README description "Behind the PanoramaView".	6 days ago

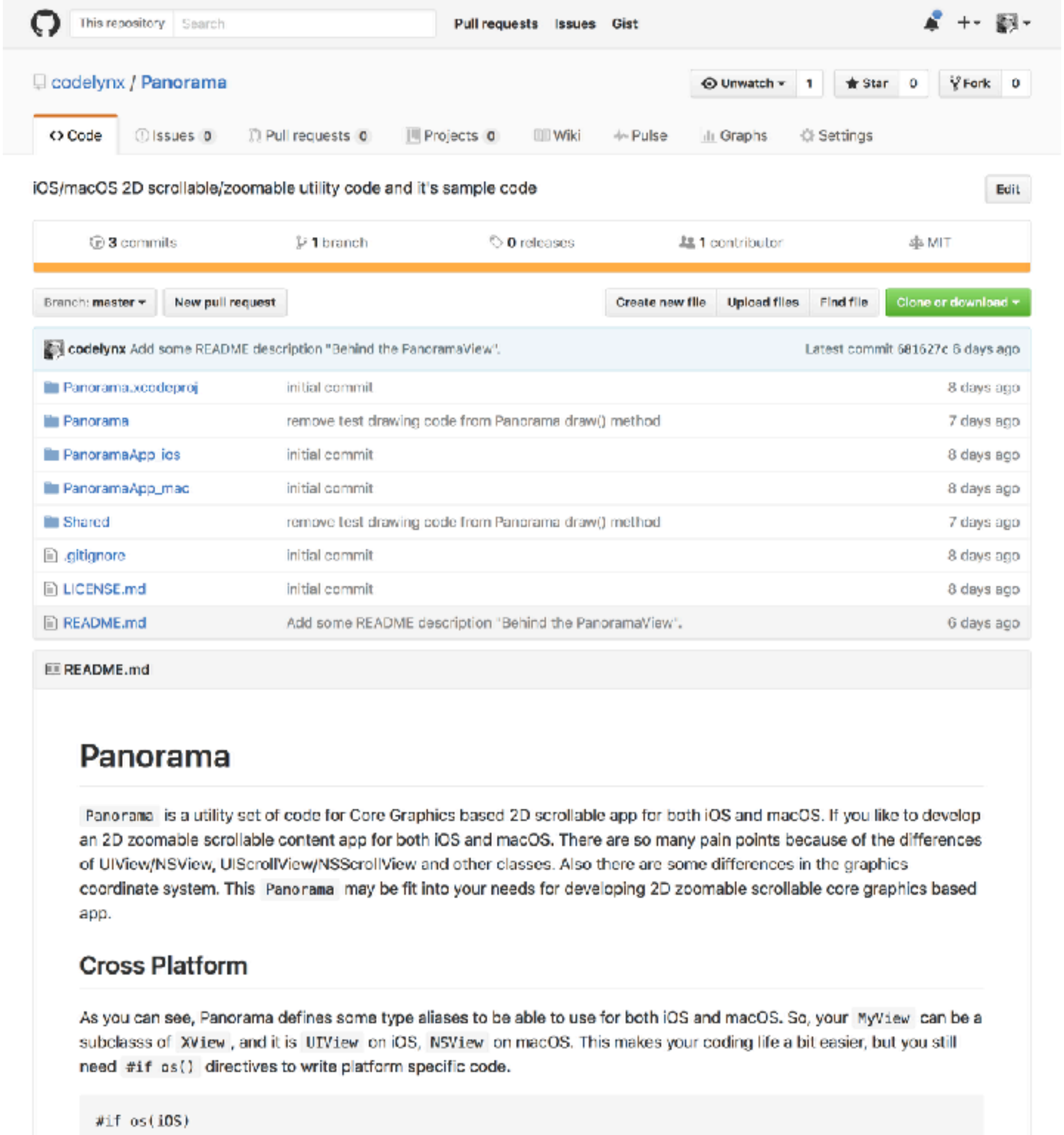
Panorama

`Panorama` is a utility set of code for Core Graphics based 2D scrollable app for both iOS and macOS. If you like to develop an 2D zoomable scrollable content app for both iOS and macOS. There are so many pain points because of the differences of UIView/NSView, UIScrollView/NSScrollView and other classes. Also there are some differences in the graphics coordinate system. This `Panorama` may be fit into your needs for developing 2D zoomable scrollable core graphics based app.

Cross Platform

As you can see, Panorama defines some type aliases to be able to use for both iOS and macOS. So, your `MyView` can be a subclass of `XView`, and it is `UIView` on iOS, `NSView` on macOS. This makes your coding life a bit easier, but you still need `#if os()` directives to write platform specific code.

```
#if os(iOS)
```



iOS/macOS 2D scrollable/zoomable utility code and it's sample code

codelynx Add some README description "Behind the PanoramaView". Latest commit 681627c 6 days ago

Panorama.xcodeproj	initial commit	8 days ago
Panorama	remove test drawing code from Panorama draw() method	7 days ago
PanoramaApp_ios	initial commit	8 days ago
PanoramaApp_mac	initial commit	8 days ago
Shared	remove test drawing code from Panorama draw() method	7 days ago
.gitignore	initial commit	8 days ago
LICENSE.md	initial commit	8 days ago
README.md	Add some README description "Behind the PanoramaView".	6 days ago

README.md

Panorama

`Panorama` is a utility set of code for Core Graphics based 2D scrollable app for both iOS and macOS. If you like to develop an 2D zoomable scrollable content app for both iOS and macOS. There are so many pain points because of the differences of UIView/NSView, UIScrollView/NSScrollView and other classes. Also there are some differences in the graphics coordinate system. This `Panorama` may be fit into your needs for developing 2D zoomable scrollable core graphics based app.

Cross Platform

As you can see, Panorama defines some type aliases to be able to use for both iOS and macOS. So, your `MyView` can be a subclass of `XView`, and it is `UIView` on iOS, `NSView` on macOS. This makes your coding life a bit easier, but you still need `#if os()` directives to write platform specific code.

```
#if os(iOS)
```

Panorama Features

- Cross Platform iOS / macOS
- Crips Sharp 8x+ Zoom scale
- Core Graphics
- May not suitable for game and/or animation

Open GL / Metal

+

UI/NSScrollView

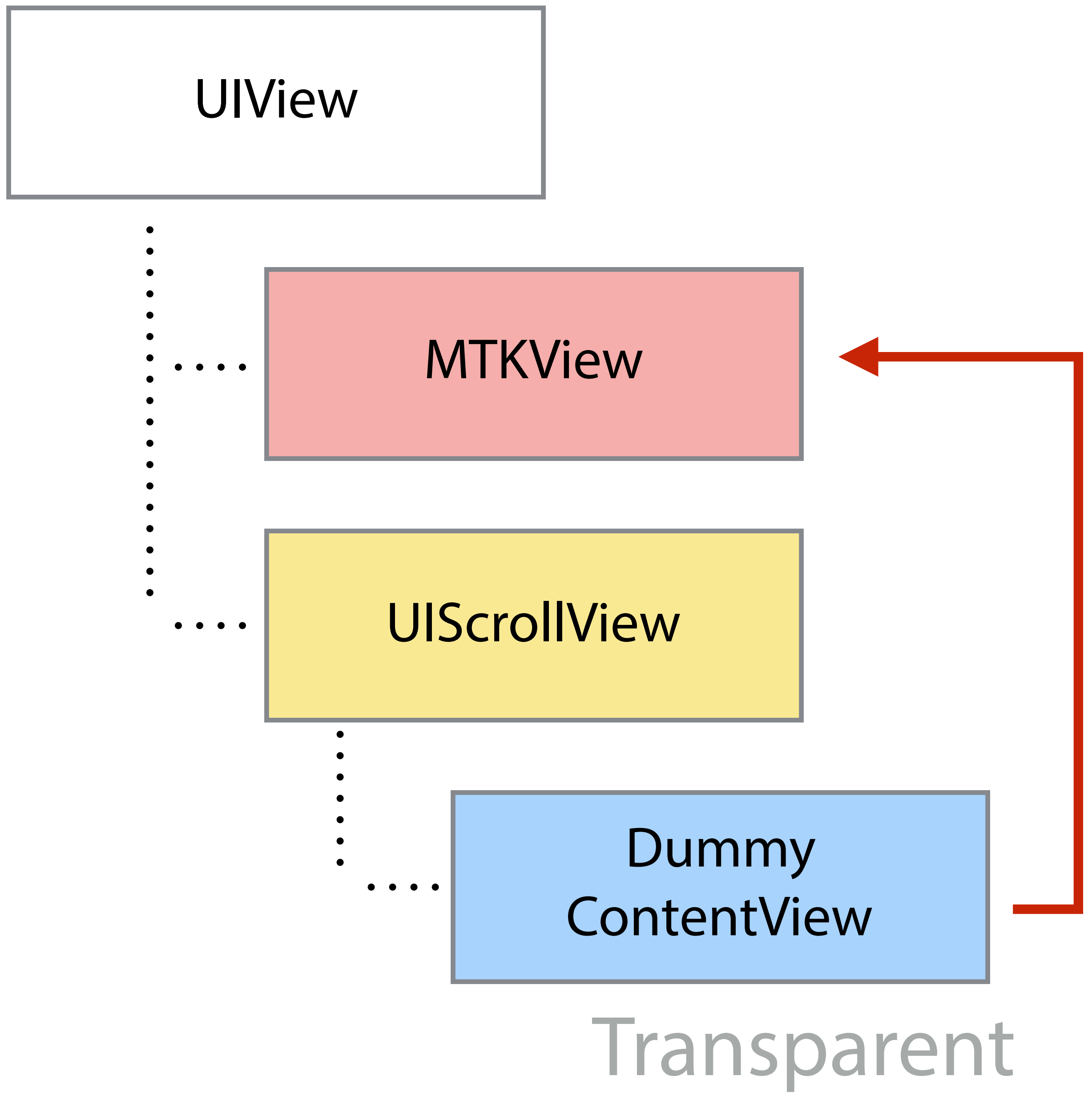
Not suitable to be a subview of UI/NSScrollView

- GLKView / EAGLLayer backed UI/NSView
- MTKView / CAMetalLayer backed UI/NSView

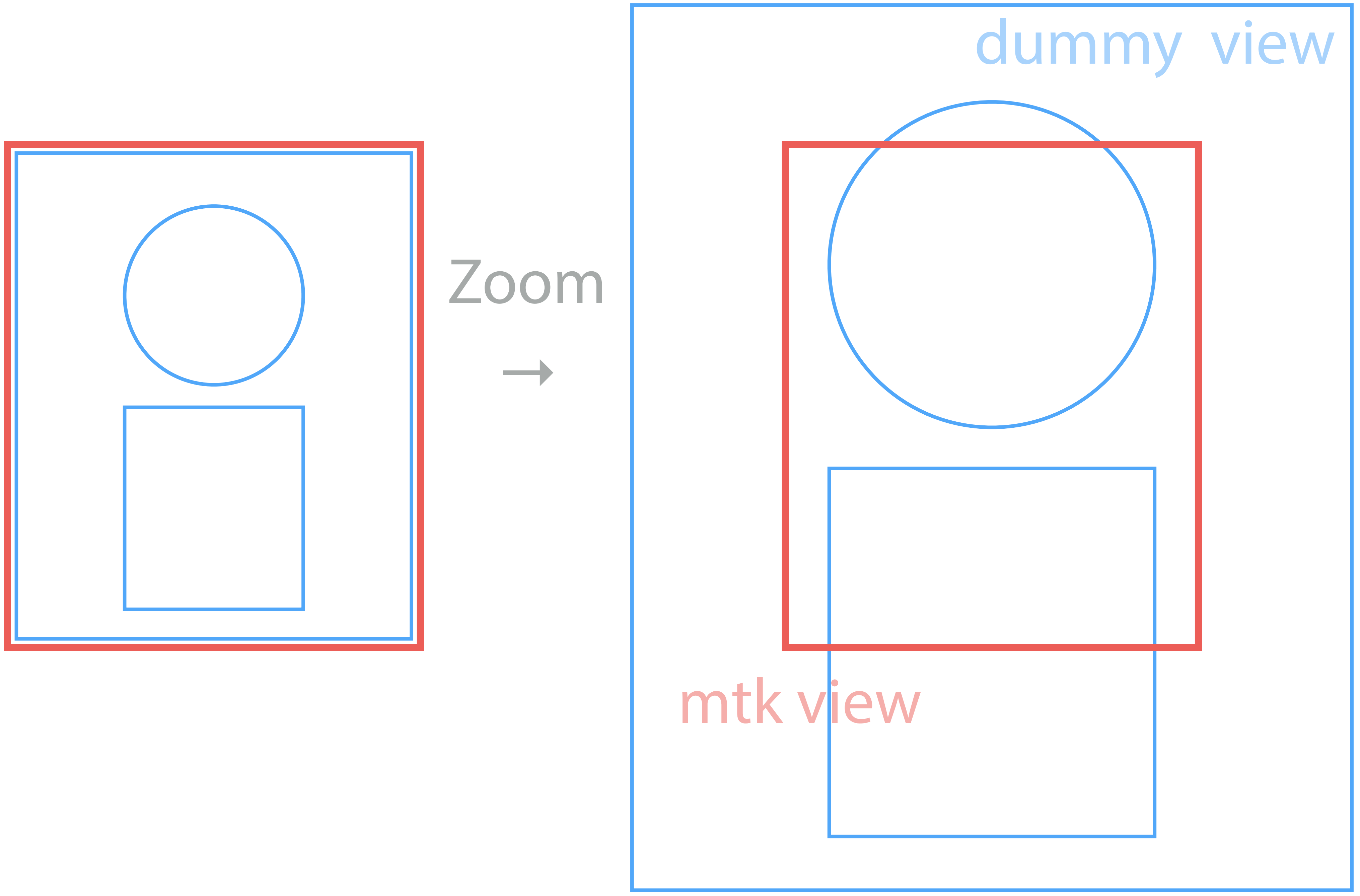
Tips

Dummy Content View

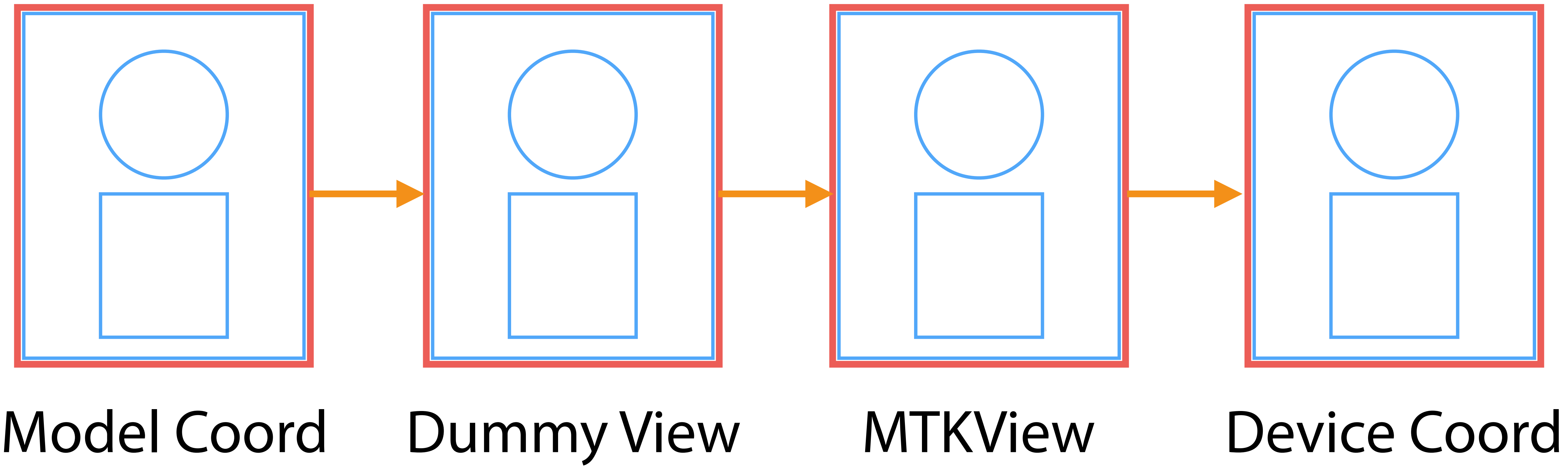
Dummy Content View



Metal view draws based on dummy view coordinate

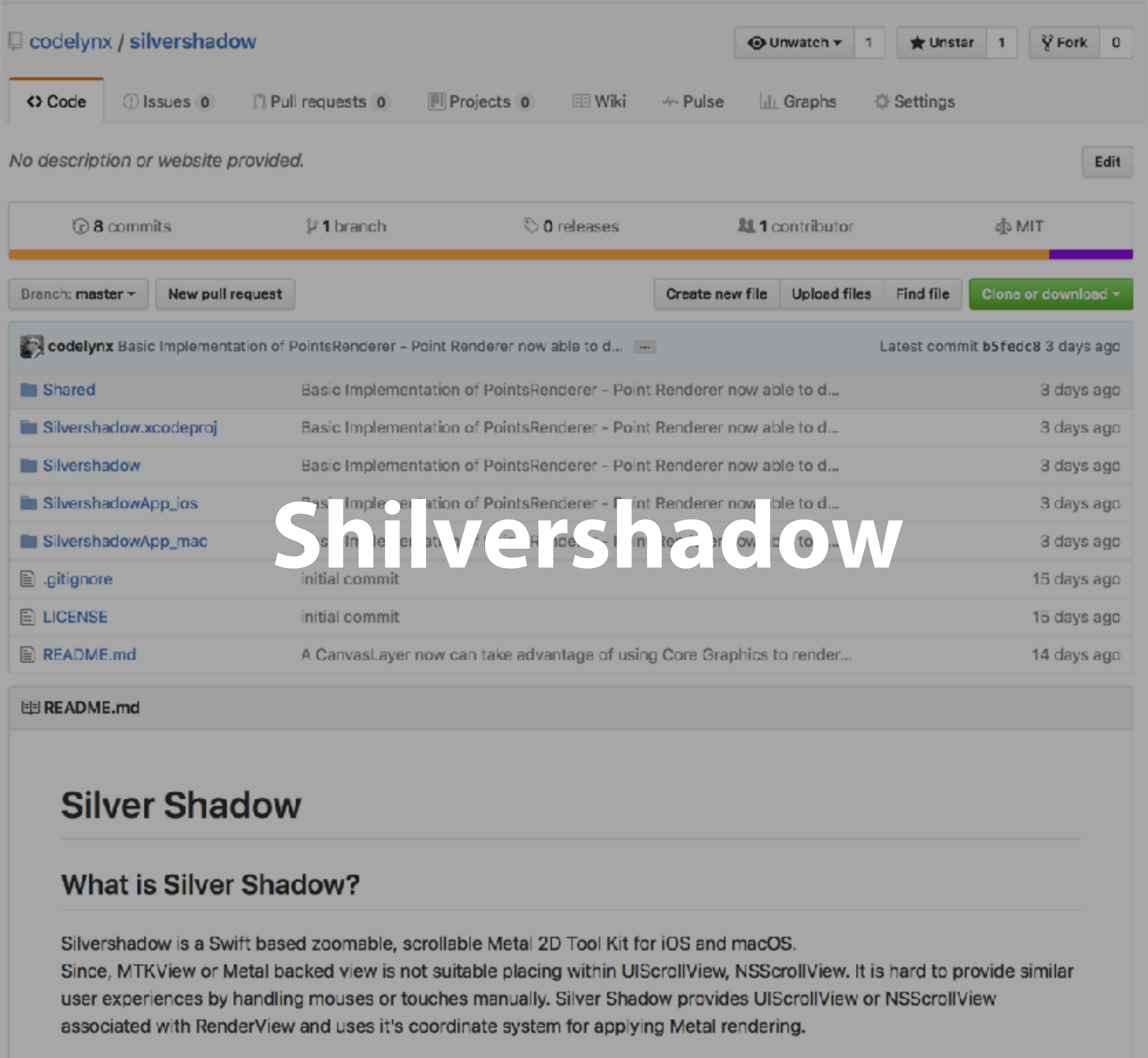


Transform Coordinate



I got the idea
Is there easy way?

Shilvershadow



No description or website provided.

8 commits | 1 branch | 0 releases | 1 contributor | MIT

codelynx Basic Implementation of PointsRenderer - Point Renderer now able to d... Latest commit b5fedc8 3 days ago

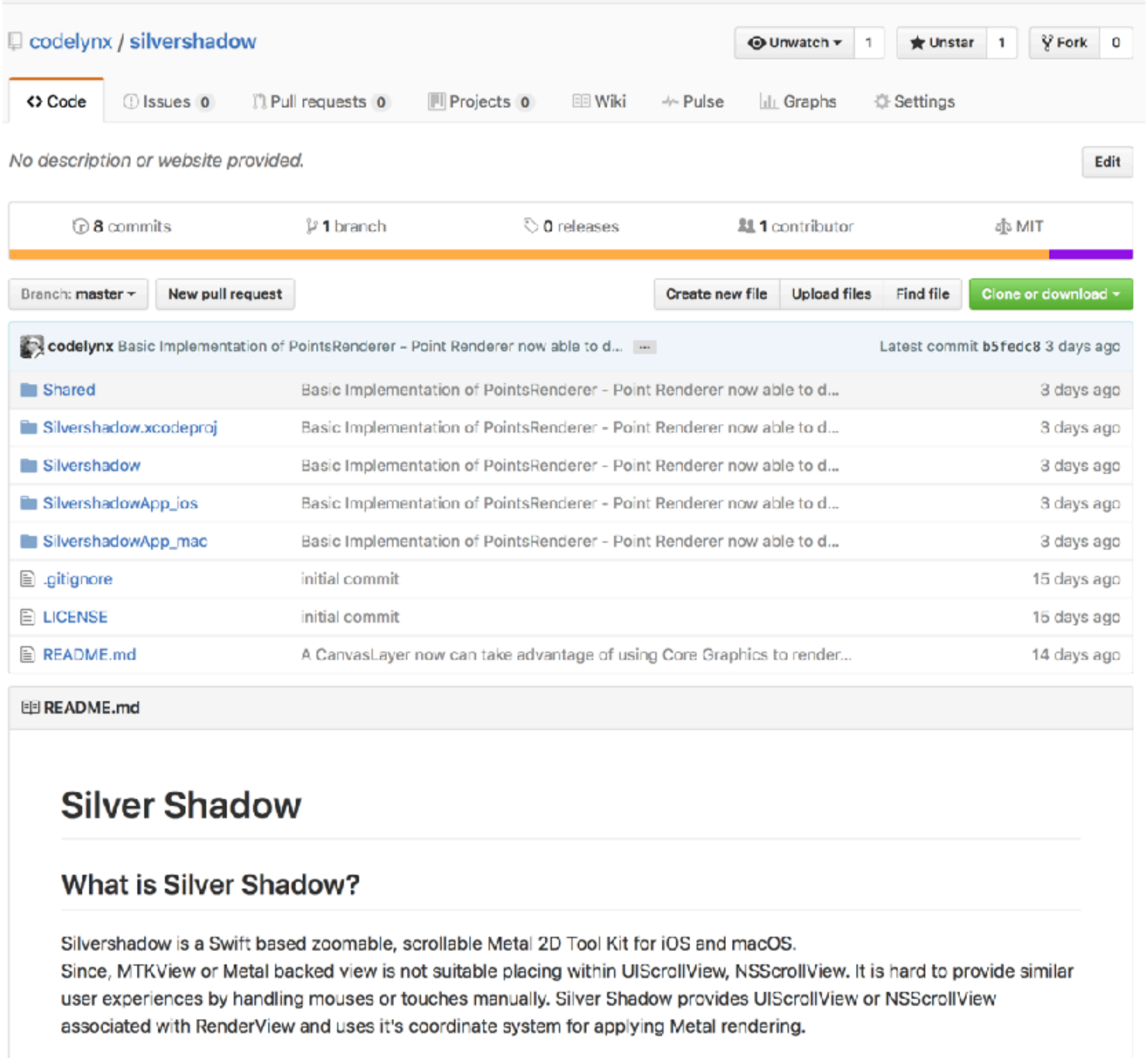
Shared	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
Silvershadow.xcodeproj	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
Silvershadow	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
SilvershadowApp_ios	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
SilvershadowApp_mac	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
.gitignore	initial commit	15 days ago
LICENSE	initial commit	15 days ago
README.md	A CanvasLayer now can take advantage of using Core Graphics to render...	14 days ago

README.md

Silver Shadow

What is Silver Shadow?

Silvershadow is a Swift based zoomable, scrollable Metal 2D Tool Kit for iOS and macOS.
Since, MTKView or Metal backed view is not suitable placing within UIScrollView, NSScrollView. It is hard to provide similar user experiences by handling mouses or touches manually. Silver Shadow provides UIScrollView or NSScrollView associated with RenderView and uses it's coordinate system for applying Metal rendering.



No description or website provided.

8 commits | 1 branch | 0 releases | 1 contributor | MIT

codelynx Basic Implementation of PointsRenderer - Point Renderer now able to d... … Latest commit b5fedc8 3 days ago

Shared	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
Silvershadow.xcodeproj	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
Silvershadow	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
SilvershadowApp_ios	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
SilvershadowApp_mac	Basic Implementation of PointsRenderer - Point Renderer now able to d...	3 days ago
.gitignore	initial commit	15 days ago
LICENSE	initial commit	15 days ago
README.md	A CanvasLayer now can take advantage of using Core Graphics to render...	14 days ago

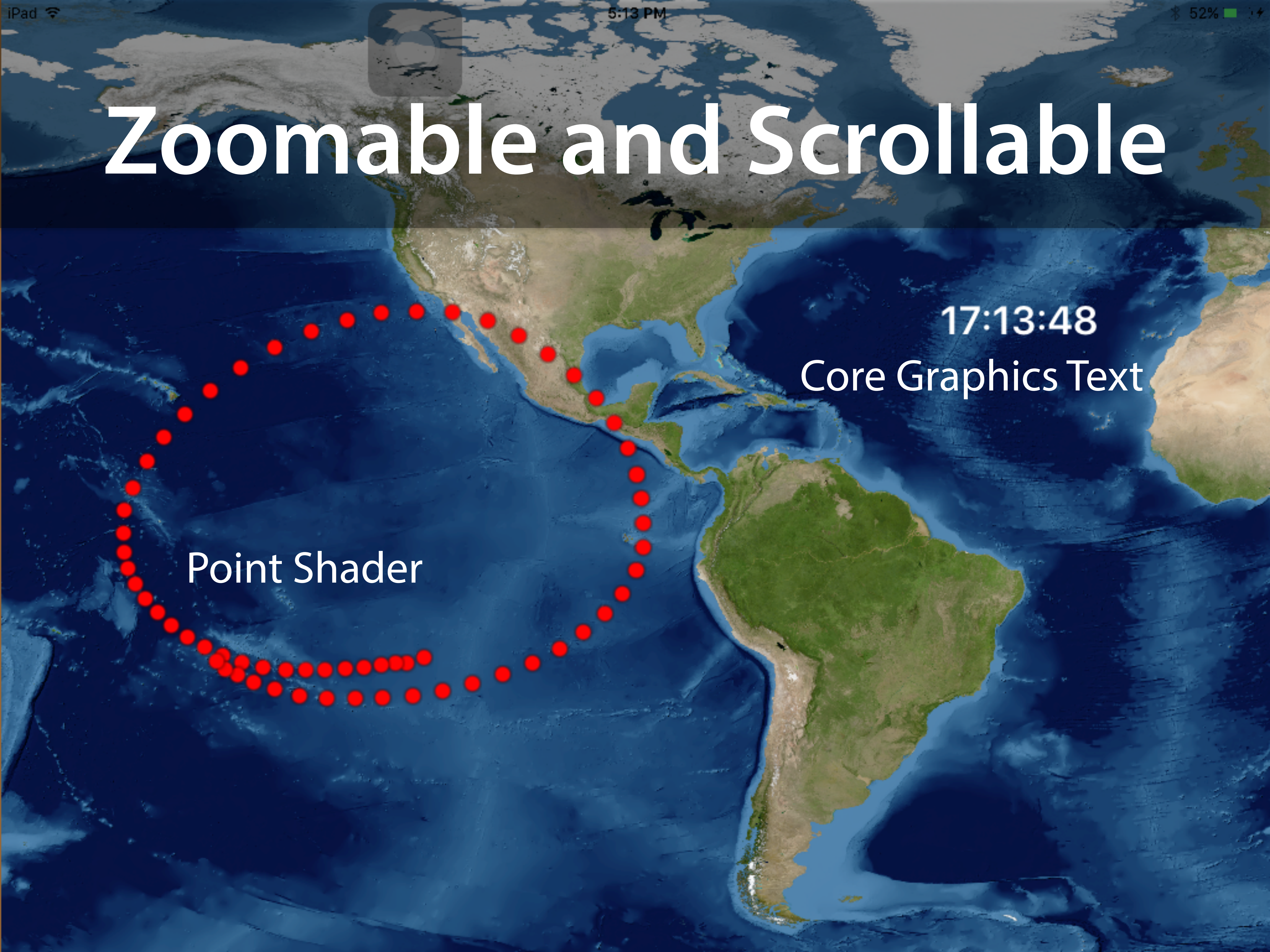
README.md

Silver Shadow

What is Silver Shadow?

Silvershadow is a Swift based zoomable, scrollable Metal 2D Tool Kit for iOS and macOS.
Since, MTKView or Metal backed view is not suitable placing within UIScrollView, NSScrollView. It is hard to provide similar user experiences by handling mouses or touches manually. Silver Shadow provides UIScrollView or NSScrollView associated with RenderView and uses it's coordinate system for applying Metal rendering.

Zoomable and Scrollable



Features

- No limitation for writing shaders – unlike SKShaders
- No complex storyboard configuration – Just place RenderView
- Subclass Scene or Canvas for your own displayable contents
- Write your own shaders
- Subclass Renderer to use your shaders
- Ability to render Bezier Path with shaders
- Possible Hybrid Displaying with Core Graphics

Feedback and Star Please

Thank you

kyoshikawa@electricwoods.com