
Blurable

###Apply a Gaussian Blur to any UIView with Swift Protocol Extensions

Adds `blur()` and `unBlur()` methods to `UIView` components which applies a Core Image Gaussian blur filter to the contents. #####Companion project to this blog post: <http://flexmonkey.blogspot.co.uk/2015/0/gaussian-blur-to-uiviews-with.html>



Here's a fun little experiment showing the power of Swift's Protocol Extensions to apply a `CIGaussianBlur` Core Image filter to any `UIView` with no developer overhead. Blurable components can be simple labels or buttons or more complex composite components such as `UISegmentedControls` and they can reside as subviews of other `UIView`s including `UIStackViews`. The code could be extended to apply any Core Image filter such as a half tone screen or colour adjustment.

`Blurable` is a simple protocol that borrows some of the methods and variables from a `UIView`:

```
1    var layer: CALayer { get }
2    var subviews: [UIView] { get }
3    var frame: CGRect { get }
4    var superview: UIView? { get }
5
6    func addSubview(view: UIView)
7
8    func bringSubviewToFront(view: UIView)
```

...and adds a few of its own:

```
1    func blur(blurRadius blurRadius: CGFloat)
2    func unBlur()
3
4    var isBlurred: Bool { get }
```

Obviously, just being a protocol, it doesn't do much on its own. However, by adding an extension, I can introduce default functionality. Furthermore, by extending `UIView` to implement `Blurable`, every component from a label to a segmented control to a horizontal slider can be blurred:

```
1  extension UIView: Blurable
2  {
3
4  }
```

Installation

Manually

1. Download and drop `FMBearable.swift` in your project.
2. Congratulations!

##The Mechanics of Blurable

Getting a blurred representation of a `UIView` is pretty simple: I need to begin an image context, use the view's `renderInContext` method to render into the context and then get a `UIImage` from the context:

```
1  UIGraphicsBeginImageContextWithOptions(CGSize(width: frame.width,
2      height: frame.height), false, 1)
3  layer.renderInContext(UIGraphicsGetCurrentContext()!)
4
5  let image = UIGraphicsGetImageFromCurrentImageContext()
6
7  UIGraphicsEndImageContext();
```

Once I have the image populated, it's a fairly standard workflow to apply a Gaussian blur to it:

```
1  guard let blur = CIFilter(name: "CIGaussianBlur") else
2  {
3      return
4  }
5
6  blur.setValue(CIImage(image: image), forKey: kCIInputImageKey)
7  blur.setValue(blurRadius, forKey: kCIInputRadiusKey)
8
9  let ciContext = CIContext(options: nil)
10
11  let result = blur.valueForKey(kCIOutputImageKey) as! CIImage!
12
13  let boundingRect = CGRect(x: 0,
14      y: 0,
15      width: frame.width,
16      height: frame.height)
17
```

```
18     let cgImage = ciContext.createCGImage(result, fromRect:
19         boundingRect)
20     let filteredImage = UIImage(CGImage: cgImage)
```

A blurred image will be larger than its input image, so I need to be explicit about the size I require in `createCGImage`.

The next step is to swap out the blurred component from its superview for a `UIImageView` containing the blurred image. The technique for doing this differs depending on whether the superview is a `UIStackView` and the blurred component is an arranged subview or not. I've already created a constant named `this` that is a non-optional, strongly typed reference to `self` as a `UIView`, so I can go ahead and check its superview and, if it's a `UIStackView` insert the blurred view as an arranged subview:

```
1     if let superview = superview as? UIStackView,
2         index = (superview as UIStackView).arrangedSubviews.indexOf(
3             this)
4     {
5         removeFromSuperview()
6         superview.insertArrangedSubview(blurOverlay, atIndex: index)
```

However, if the blurred component isn't an arranged subview, we can use a nice animation to cross fade between the original and the blurred view:

```
1     else
2     {
3         blurOverlay.frame.origin = frame.origin
4
5         UIView.transitionFromView(this,
6             toView: blurOverlay,
7             duration: 0.2,
8             options: UIViewAnimationOptions.CurveEaseIn,
9             completion: nil)
10    }
```

Finally, we need to create a reference between the original blurred component and its blur overlay. Since protocol extensions don't allow for stored properties, I use `objc_setAssociatedObject` to effectively add a `blurOverlay` property to the component:

```
1     objc_setAssociatedObject(this,
2         &BlurableKey.blurable,
3         blurOverlay,
4         objc_AssociationPolicy.OBJC_ASSOCIATION_RETAIN)
```

When it comes to unblurring in `unBlur()`, it's essentially the same process but in reverse. First I

create the same **this** constant and ensure the component has an associated blur overlay:

```
1    guard let this = self as? UIView,
2        blurOverlay = objc_getAssociatedObject(self as? UIView, &
3            BlurableKey.blurable) as? BlurOverlay else
4    {
5        return
6    }
```

Then do the same checks to see if **blurOverlay**'s superview is a **UIStackView** and either insert **self** as an arranged subview if it is or do the same **transitionFromView** animation as above, but backwards, if it isn't:

```
1    if let superview = blurOverlay.superview as? UIStackView,
2        index = (blurOverlay.superview as! UIStackView).
3            arrangedSubviews.indexOf(blurOverlay)
4    {
5        blurOverlay.removeFromSuperview()
6        superview.insertArrangedSubview(this, atIndex: index)
7    }
8    else
9    {
10        this.frame.origin = blurOverlay.frame.origin
11        UIView.transitionFromView(blurOverlay,
12            toView: this,
13            duration: 0.2,
14            options: UIViewAnimationOptions.CurveEaseIn,
15            completion: nil)
16    }
```

The last step of **unBlur()** is to remove the association between the original blurred component and its blur overlay:

```
1    objc_setAssociatedObject(this,
2        &BlurableKey.blurable,
3        nil,
4        objc_AssociationPolicy.OBJC_ASSOCIATION_RETAIN)
```

Finally, to see if a **UIView** is currently blurred, I created **isBlurred()** which just needs to check if it has an associated blur overlay:

```
1    var isBlurred: Bool
2    {
3        return objc_getAssociatedObject(self as? UIView, &BlurableKey.
4            blurable) is BlurOverlay
5    }
```

##Blurring a UIView

To blur and de-blur, just invoke `blur()` and `unBlur()` on an `UIView`:

```
1 segmentedControl.unBlur()
2 segmentedControl.blur(blurRadius: 2)
```

##Source Code

As always, the source code for this project is available at my GitHub repository [here](#). Enjoy!