April 2012

Hit testing animating layers

tl;dr hit test the presentation layer instead.

At the core of Core Animation[1] is the fact that it changes the animated property right away (or not at all[2]) but smoothly animates the change visually over time. If you inspect the value of the animated property during the animation it will already have been set to its final value (or never been changed at all[2]). Knowing this, what do you do when you need to know the values that is being used to render the layer during the animation? How do you hit test a moving layer?

Spoiler: The answer is simply to *look at the value of the presentation layer* instead.

Make a small experiment

You can learn a lot about Core Animation by writing a small piece of code where you hit test a moving layer and its presentations layer. While we are at it, we will have a poke at key-frame animations as well.

Write some code

First we create a layer and add it to our views layer. Don't forget to keep a reference to the layer, we'll need it when we hit test it later.

```
CALayer * movingLayer = [CALayer layer];
[movingLayer setBounds: CGRectMake(0, 0, layerSize, layerSize)];
[movingLayer setBackgroundColor:[UIColor orangeColor].CGColor];
[movingLayer setPosition:CGPointMake(layerCenterInset, layerCenterInset)
];
// Additional styling of the layer ...
[[[self view] layer] addSublayer:movingLayer];
[self setMovingLayer:movingLayer];
```

Next we set up the key-frame animation for the position and add it to our layer.

```
CAKeyframeAnimation * moveLayerAnimation = [CAKeyframeAnimation animatio
nWithKeyPath:@"position"];
[moveLayerAnimation setValues:[NSArray arrayWithObjects: /* some NSValue
-wrapped CGPoints */, nil]];

        [moveLayerAnimation setDuration:10.0];
[moveLayerAnimation setRepeatCount:HUGE_VALF];
[moveLayerAnimation setTimingFunction: [CAMediaTimingFunction functionWi
thName:kCAMediaTimingFunctionLinear]];
[[self movingLayer] addAnimation:moveLayerAnimation forKey:@"move"];
```

To be able to tap the layer we need to add a tap gesture recognizer and connect it to an action method that is hit testing both the layer and its presentation layer, like this.

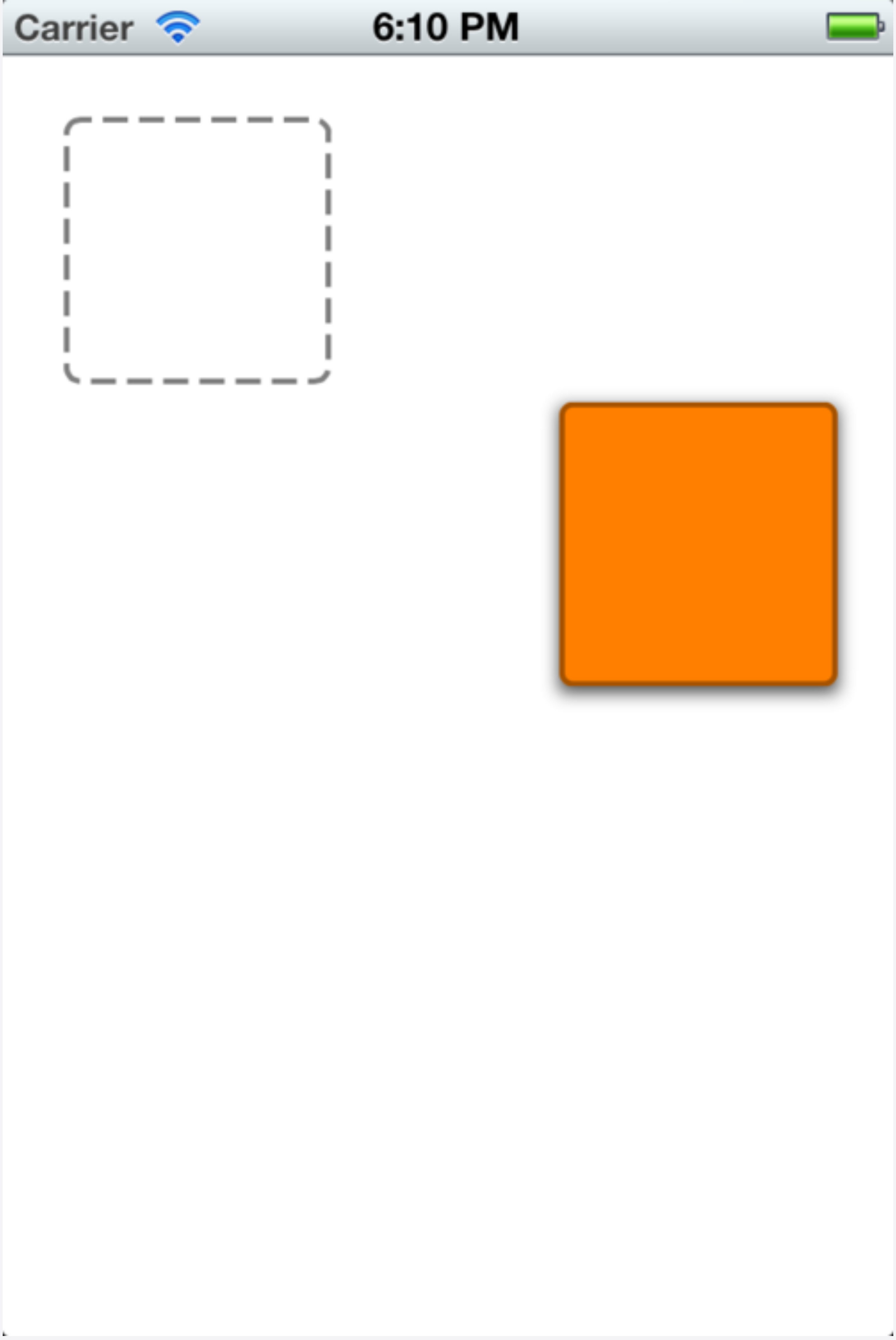
```
- (IBAction)pressedLayer:(UIGestureRecognizer *)gestureRecognizer {
    CGPoint touchPoint = [gestureRecognizer locationInView:[self view]];

    if ([[[self movingLayer] presentationLayer] hitTest:touchPoint]) {
        [self blinkLayerWithColor:[UIColor yellowColor]];
    } else if ([[self movingLayer] hitTest:touchPoint]) {
        [self blinkLayerWithColor:[UIColor redColor]];
    }
}
```

Blinking the layer is done by setting the background color in a swift animation and reversing to the current value:

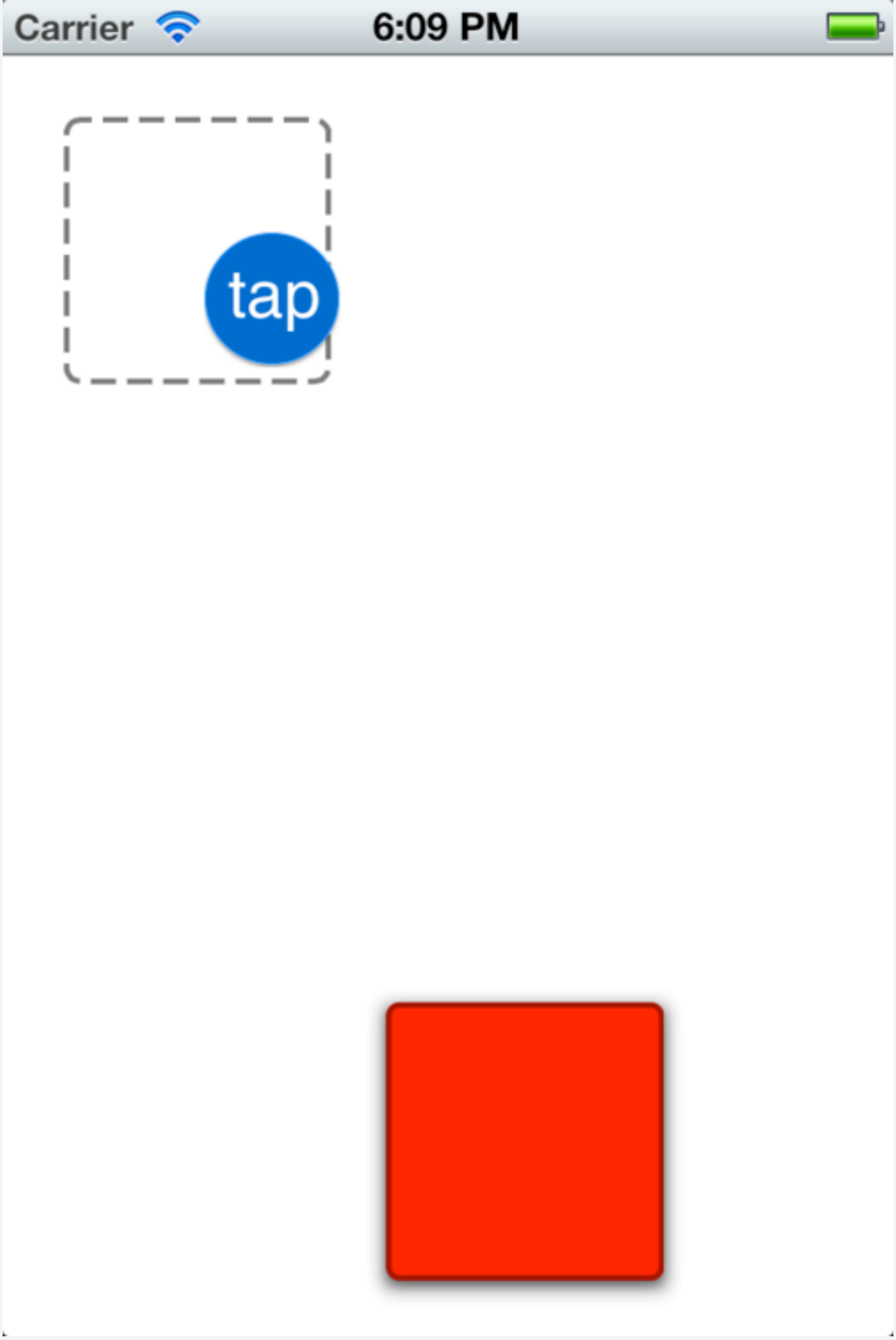
```
- (void)blinkLayerWithColor:(UIColor *)color {
    CABasicAnimation * blinkAnimation = [CABasicAnimation animationWithK
eyPath:@"backgroundColor"];
    [blinkAnimation setDuration:0.1];
    [blinkAnimation setAutoreverses:YES];
    [blinkAnimation setFromValue:(id)[[self movingLayer] backgroundColor
]];
    [blinkAnimation setToValue:(id)color.CGColor];

    [[self movingLayer] addAnimation:blinkAnimation forKey:@"blink"];
}
```

Make observations

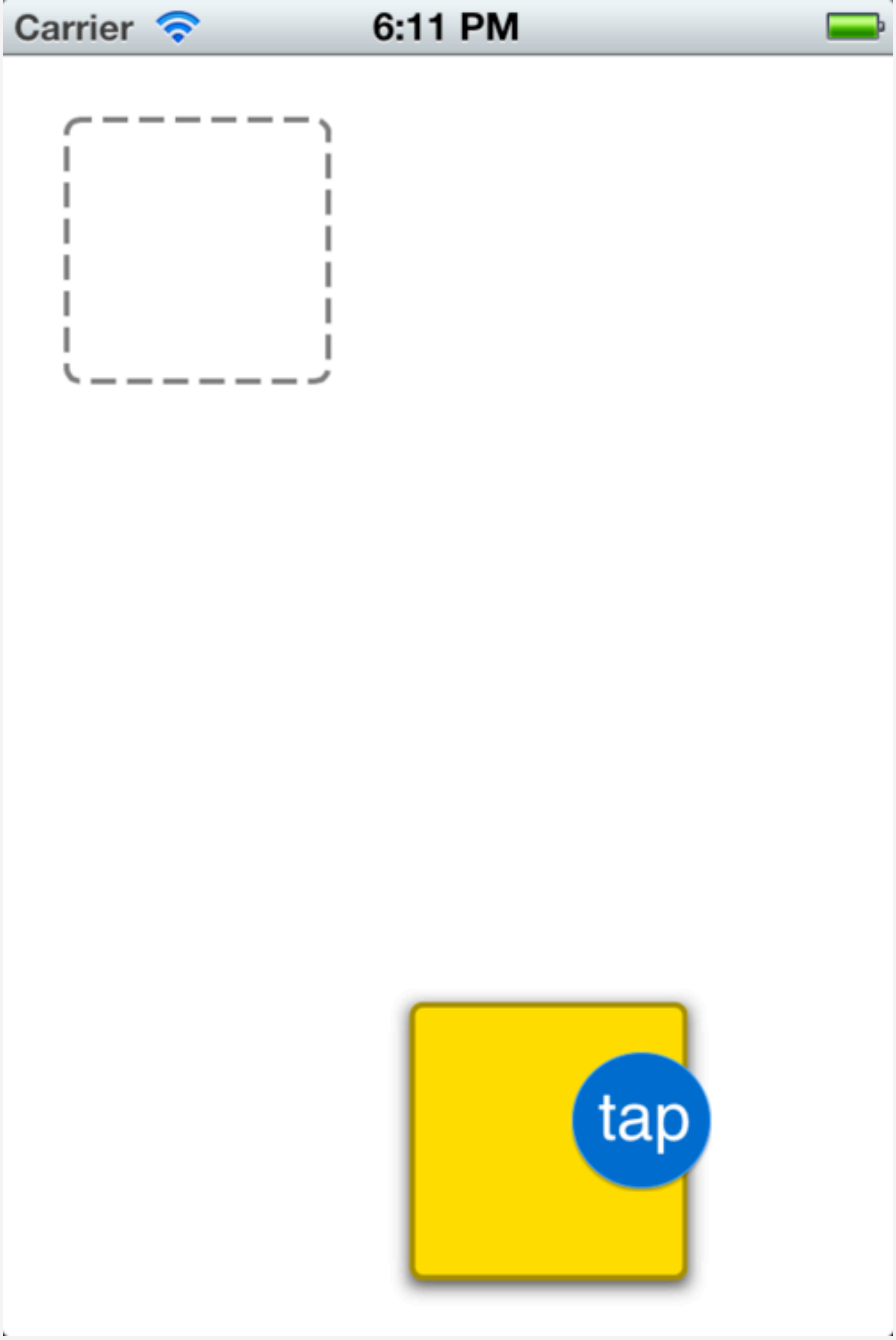
When you run the application you will see a moving orange layer.



If you tap the where the layer originated that will cause it to blink red, even when the layer is far away from where you are tapping. This is what happens if you inspect “real” values during an animation.



Also note that the layer actually blinks in a yellow tone when you are tapping it. We hit the moving layer. Success!



Have some extra fun

Go back to the key-frame animation code and change the calculation mode by adding this line of code.

```
[moveLayerAnimation setCalculationMode:kCAAnimationCubic];
```

Now, run the application again and see the cool smooth curve that your layer is animating along. No more of this boring straight-line movement. Awesome! Also note that even though the layer is moving along this curve, you can still tap on it. Amazing!

Bug or feature?

The documentation for the presentation layer clearly states that it provides "a close approximation to the version of the layer that is currently being displayed", *not* the exact value. This approximation seems extremely good as long as you don't mess with the animation timings. If you set the calculation mode on a key-frame animation to any of the paced modes (`kCAAnimationPaced` or `kCAAnimationCubicPaced`) then it seems to be off, by *a lot*. In fact, it seems to me like the values you get back from the presentation mode is being calculated linearly (the default calculation mode).

The following can quite nicely be illustrated by slowly animating the position of two identical layers, one using the default calculation mode and one using a paced calculation mode. Trying to hit the linearly moving layer also hits the paced layer but trying to hit the paced layer hits nothing. I of course did the responsible thing and filed a radar. I'll post an update when I know its in fact the intended behavior (part of the approximation) or a real bug.

1. No pun intended. ↩

2. Explicit animations (like `CABasicAnimation` or `CAKeyFrameAninmation`) doesn't change the value of the animated property. Only implicit animations (setting the value of an animatable layer property changes the value). ↩ ↩2

@davidronnqvist