CA For Fun and Profit


360|iDev
August 22-25, 2021
Denver, CO
Mihaela Mihaljevic Jakic
Presents:
Core Animation for Fun and Profit

01. What is Core Animation?

Core Animation is not (just) an “animation engine”.
Actually, at its heart, it is not that at all.

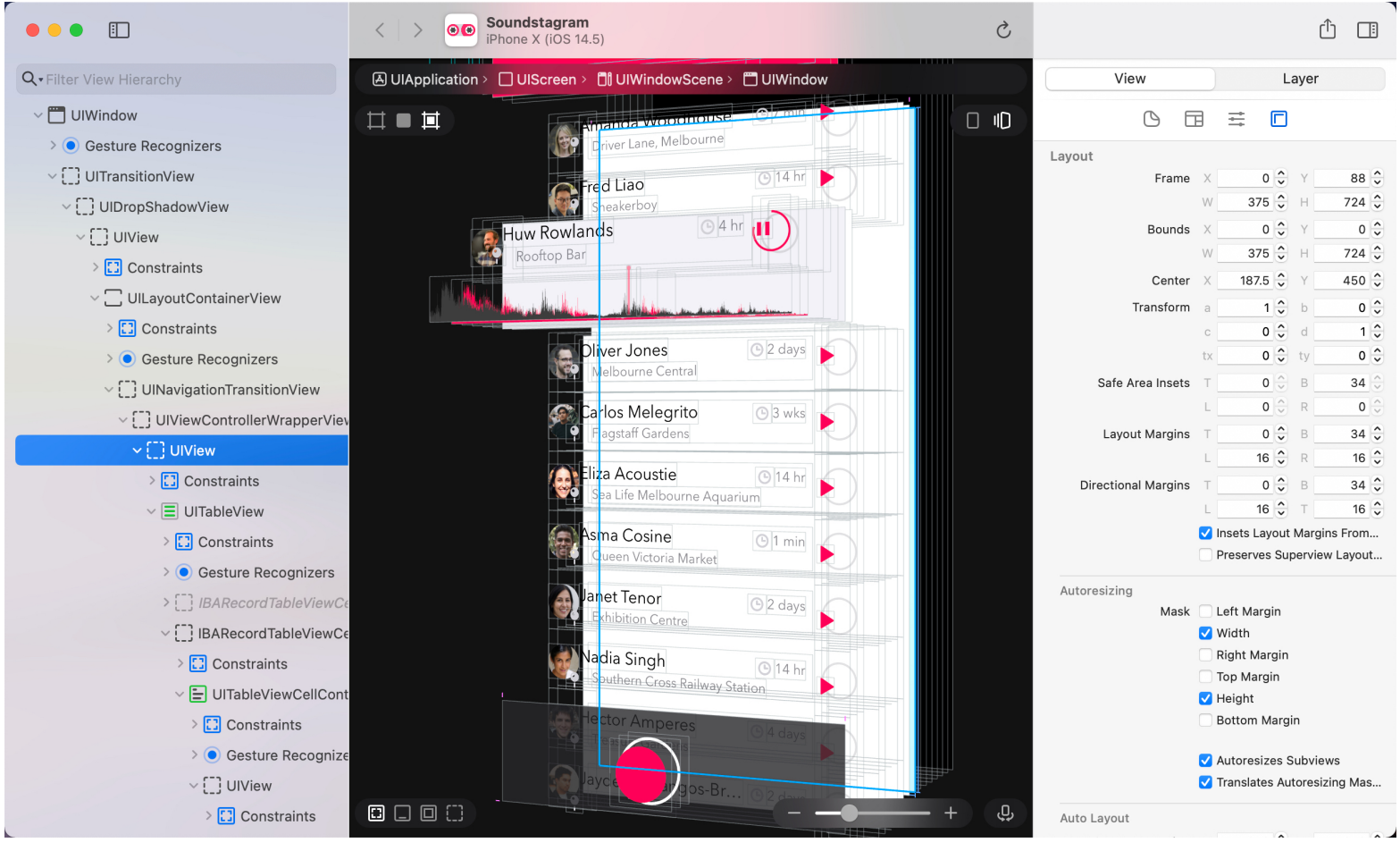
Core Animation is really a 2D compositing framework.
(2.5D actually).

What is “layer compositing”?

- Layer compositing is the foundation of 2D graphics.

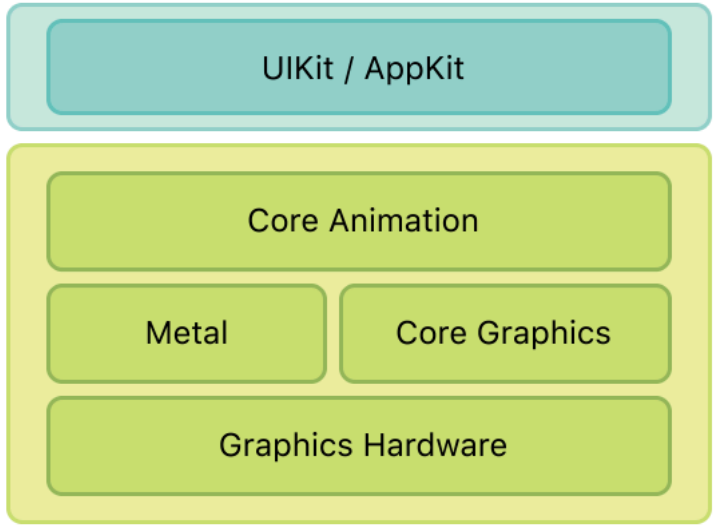
User Interface on Apple platforms

- On most of the operating systems, the User Interface elements are created by issuing drawing commands.
- On Apple platforms, the User Interface is a tree of layers.


Soundstagram
iPhone X (iOS 14.5)
Filter View Hierarchy
UIWindow
Gesture Recognizers
UITransitionView
UIDropShadowView
UIView
Constraints
UILayoutContainerView
Constraints
Gesture Recognizers
UINavigationTransitionView
UIViewControllerWrapperVie
UIView
Constraints
UITableView
Constraints
Gesture Recognizers
IBARecordTableViewCe
IBARecordTableViewCe
Constraints
UITableViewCellCont
Constraints
Gesture Recognize
UIView
Constraints
UIApplication
UIScreen
UIWindowScene
UIWindow
Driver Lane, Melbourne
Fred Liao
Sneakerboy
14 hr
Huw Rowlands
Rooftop Bar
4 hr
Oliver Jones
Melbourne Central
2 days
Carlos Melegrito
Flagstaff Gardens
3 wks
Eliza Acoustie
Sea Life Melbourne Aquarium
14 hr
Asma Cosine
Queen Victoria Market
1 min
Janet Tenor
Exhibition Centre
2 days
Nadia Singh
Southern Cross Railway Station
14 hr
Hector Amperes
4 days
View
Layer
Layout
Frame X 0 Y 88
W 375 H 724
Bounds X 0 Y 0
W 375 H 724
Center X 187.5 Y 450
Transform a 1 b 0
c 0 d 1
tx 0 ty 0
Safe Area Insets T 0 B 34
L 0 R 0
Layout Margins T 0 B 34
L 16 R 16
Directional Margins T 0 B 34
L 16 T 16
Insets Layout Margins From...
Preserves Superview Layout...
Autoresizing
Mask
Left Margin
Width
Right Margin
Top Margin
Height
Bottom Margin
Autoresizes Subviews
Translates Autoresizing Mas...
Auto Layout

Frameworks

- Core Animation sits below UIKit / AppKit
- and above the hardware layer



- Core Animation uses Core Graphics to draw the content of its layers
- It uses Metal to output the final result to the GPU (it used OpenGL in the past)

02. What is a Layer?

- As a data structure, it is a non-visual (Objective-C) object (non-abstract, you may instantiate it)
- As a drawing destination, it is a 2D surface in a 3D world.

What defines a layer?

- Geometry
- Properties
- Content

Layer geometry

- Bounds (width and height)
- Position (in the super-layer coordinate space)
- Anchor point (the point of transformation, where are you holding that layer with your two fingers)
- Transform

Layer properties

Animatable

Non-animatable

03. Layer - View Relationship

How do views and layers relate to each other?

- There is no class hierarchy!
- Views do not inherit from CALayer because that would be very limited.

- There are two parallel class hierarchies: views own a CALayer.

Geometry

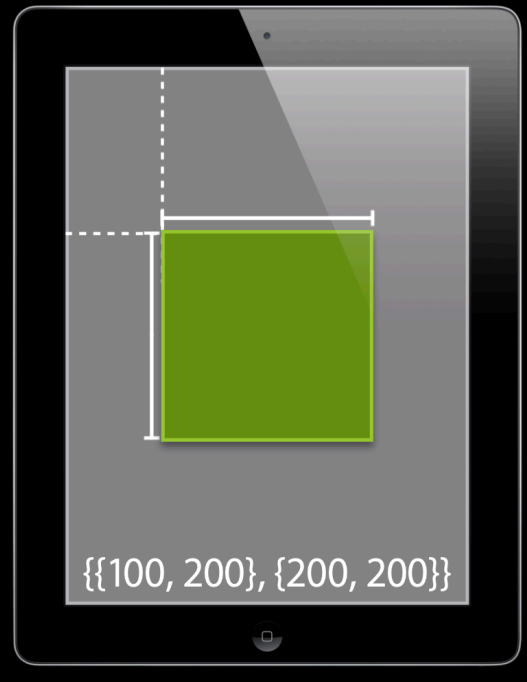
- Position and size of the view is always defined in its superview (super-layer).

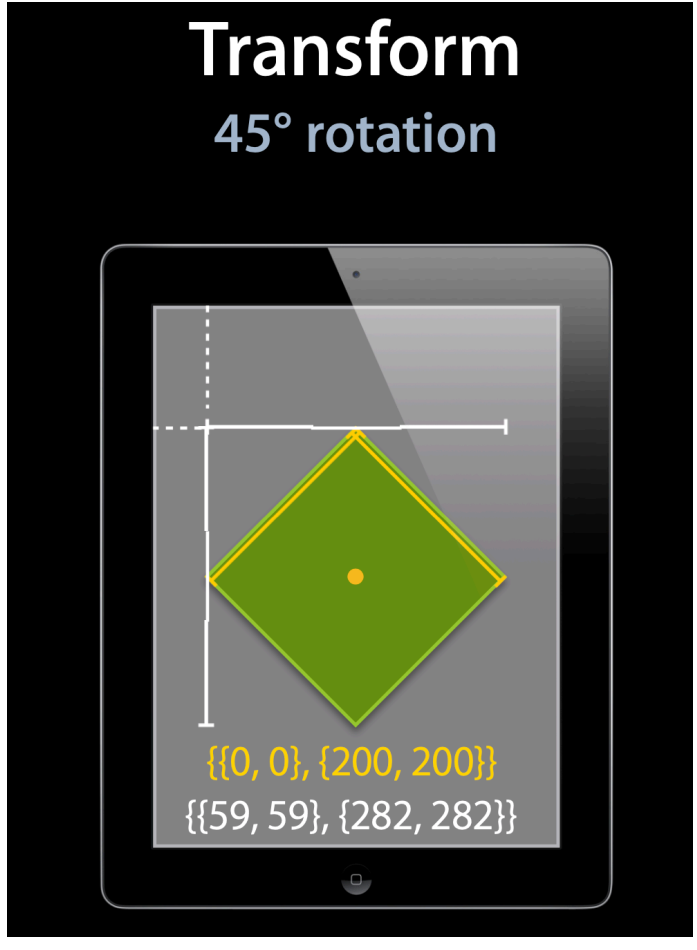
Frame

- The smallest rectangle that encloses that view, in its superview

Frame

Superview's Coordinate Space

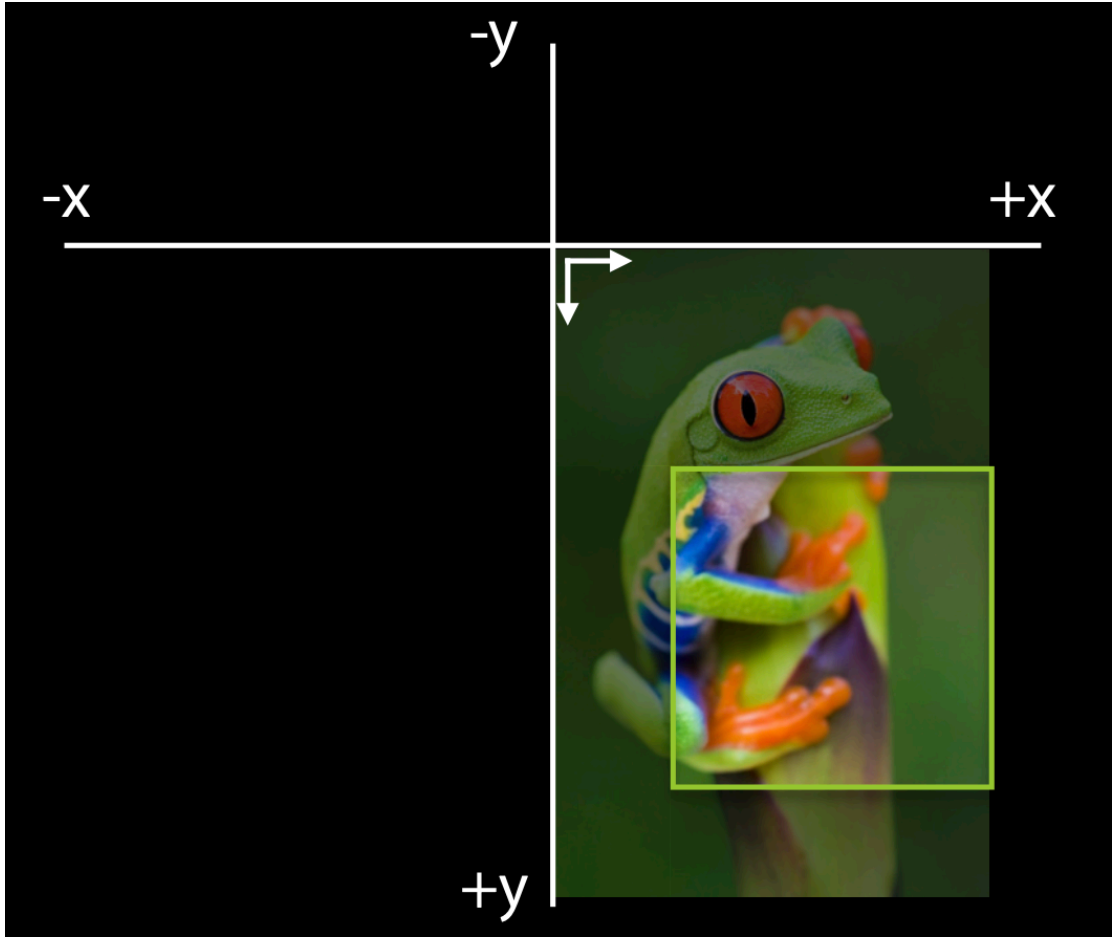



Transform
45° rotation
{{0, 0}, {200, 200}}
{{59, 59}, {282, 282}}

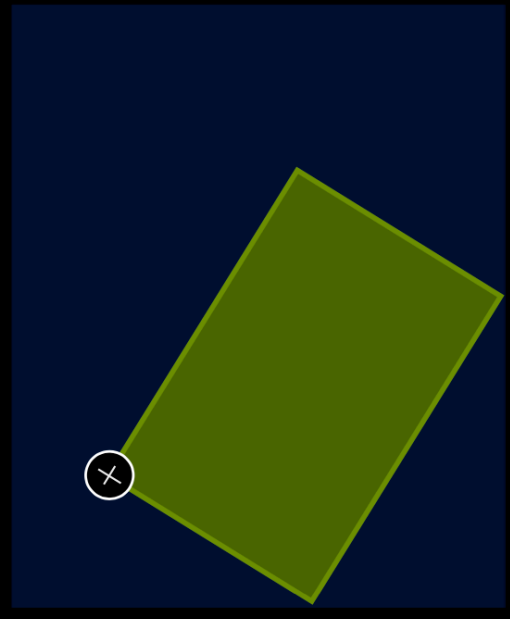
- Transform property of a View is a Core Graphics 2D transform.

Components of the frame

- “bounds.size”: view / layer sets its bounds (coordinate space is View)
- “bounds.origin” gives us a window to a view

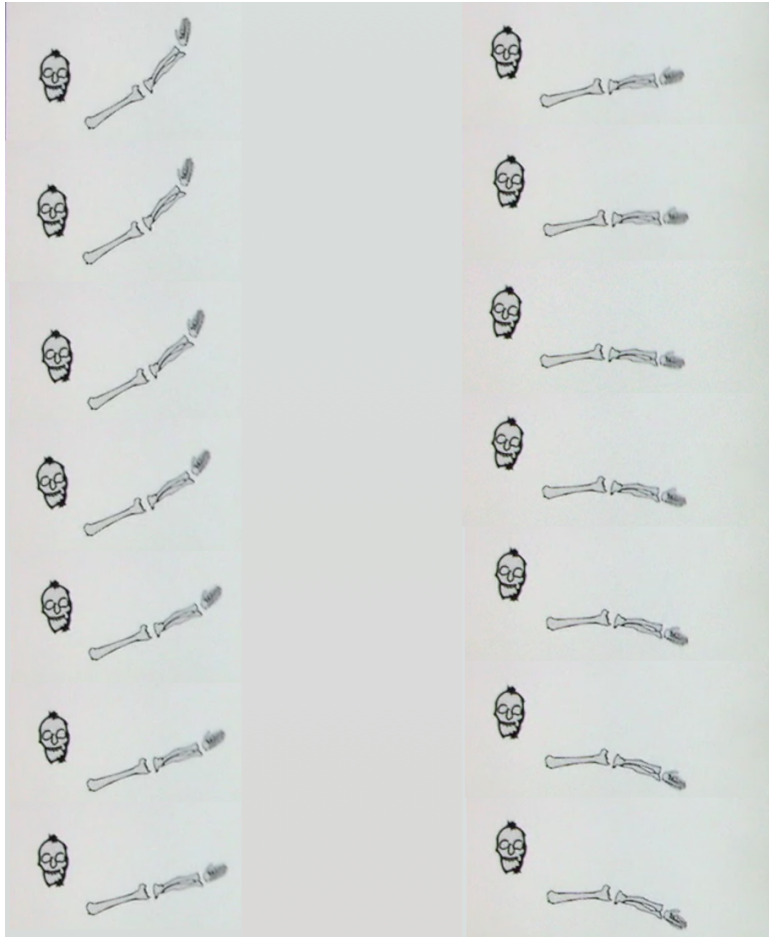

-y
-x
+x
+y

- “center”:
  - It is an easier representation of “anchor points” and “position” properties of CALayer.
  - “Position” says where the “anchor point” is in the coordinate system of the super-layer
  - “Anchor point” is a property of a layer that determines how the `transforms` are applied.



```
// B: Rotate about lower left
layer.anchorPoint = CGPointMake(0.0,1.0);
layer.transform = rotationTransform;
```

Here’s the demo of animation with the `anchor point`



04. Layer Compositing

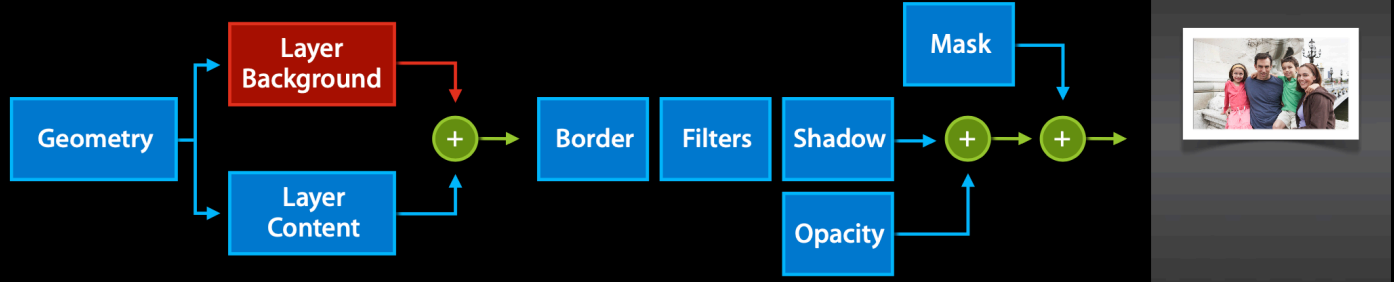
Core Animation is really a `2D compositing engine`

Before each layer (in a layer-tree) get rendered, many steps have to be executed:

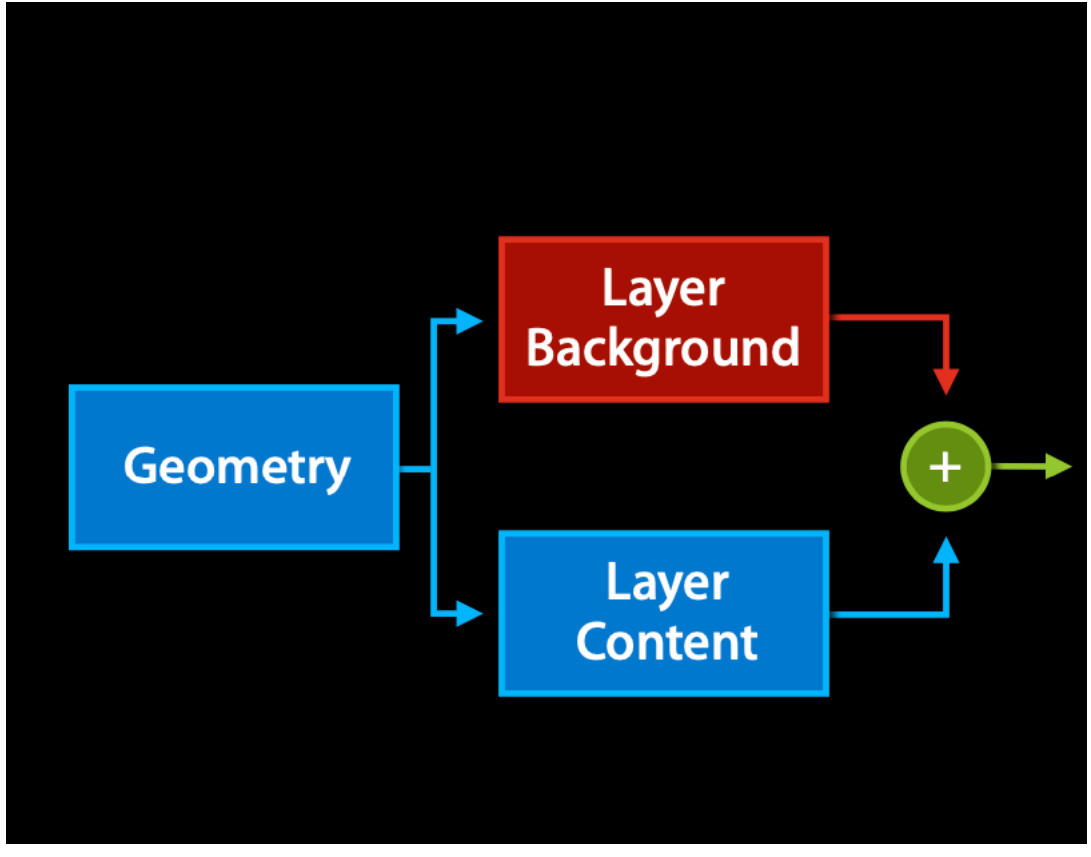
- Layers are sorted (which one goes on top of each other)
- Layers background is merged with its content
- After that opacity, borders, shadows, filters, masks are applied

These steps occur for the production of each frame.

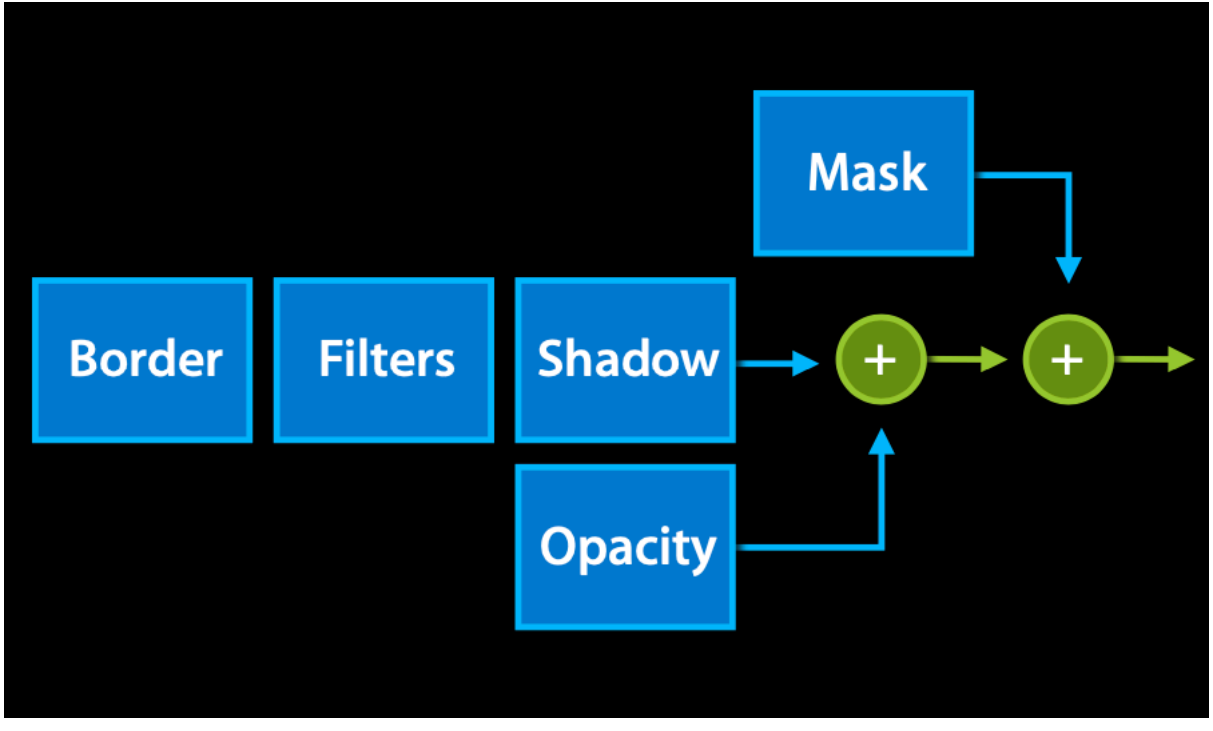
Compositing Model



Layers background is merged with its content


Layer
Background
Geometry
+
Layer
Content

Borders, shadows, filters, masks are applied



And we get the resulting image.

05. What is Animation?

Oxford English Dictionary says `Animation is the state of being alive`.

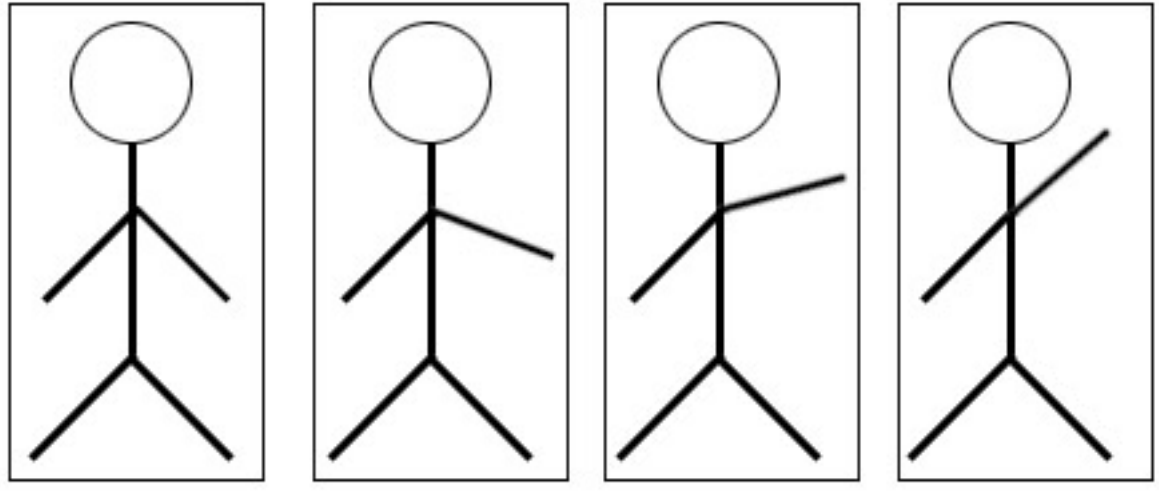
Animation serves a purpose in User Interface design in that that it:

- adds realism to user experience
- adds feedback for the user
- conveys a state change

- improves the visual production value

05.01. Core Animation is a `Tweening` engine

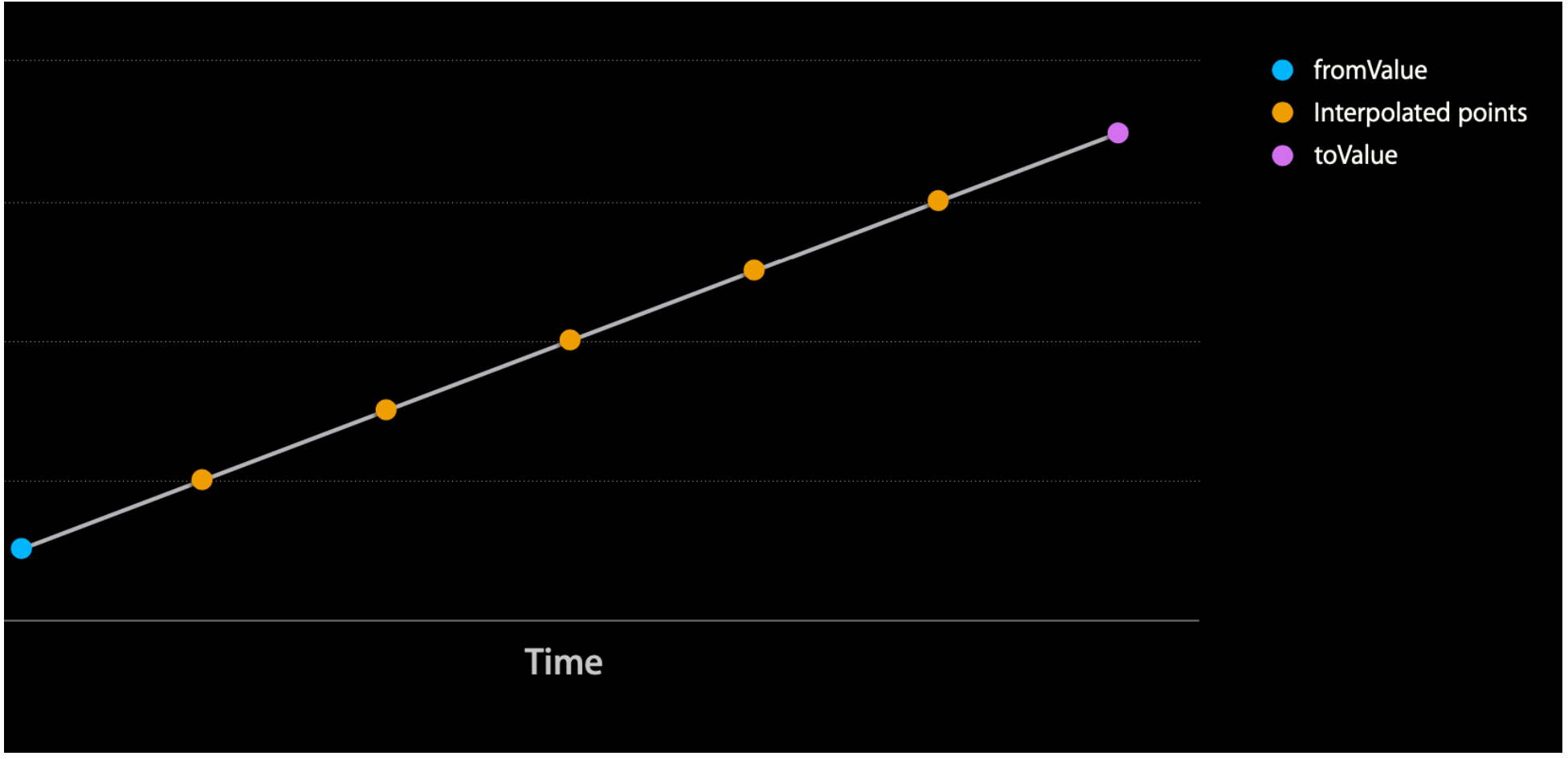
If we look at out layer-tree as our models, we get that animation is `changing the values of our model properties over time`



Animation is

- the interpolation of points
- starting at an initial point,

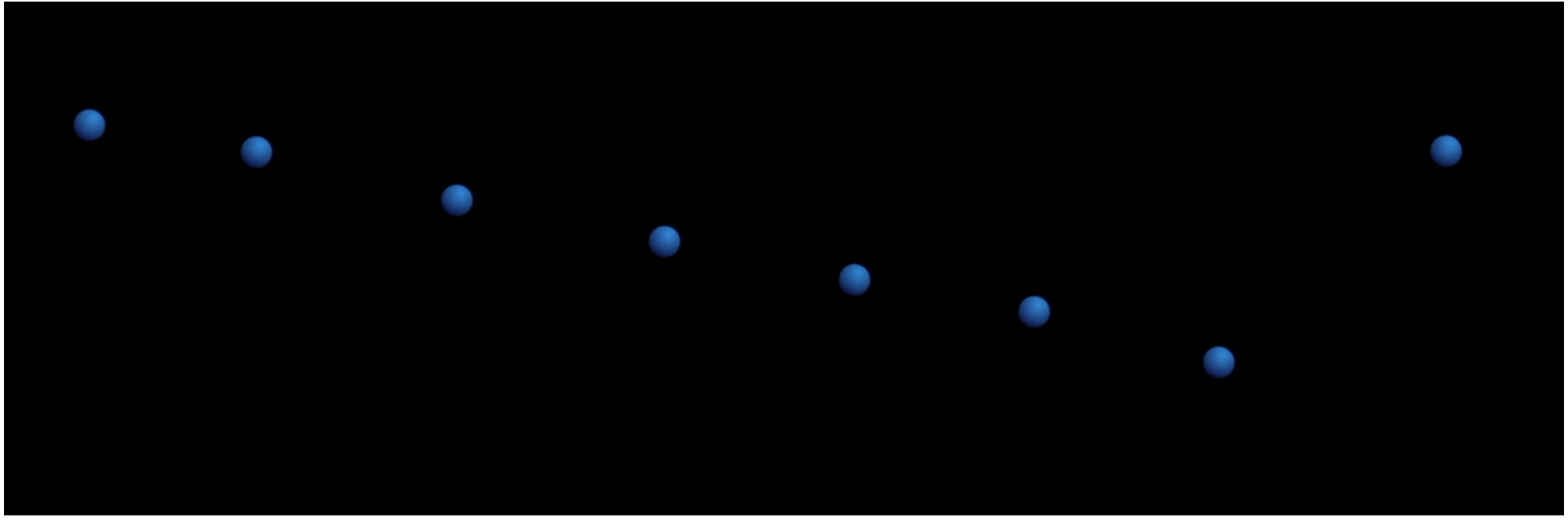
- going through some interpolated points,
- and ending at a destination.


fromValue
Interpolated points
toValue
Time

Timing:

- Is it very jerky sort of infinite speed? And then stop?
- Or does it start going faster, reaches maximum speed, slow down?
- That is determined by a timing function

Here’s the image of all the frames generated for a ball animation



05. 02. What Properties can be Animatable?

Only those properties that

- can be represented numerically
- so that the Core Animation engine can interpolate between `from value` and `to value`

06. Implicit and Explicit Animations

Implicit Animations

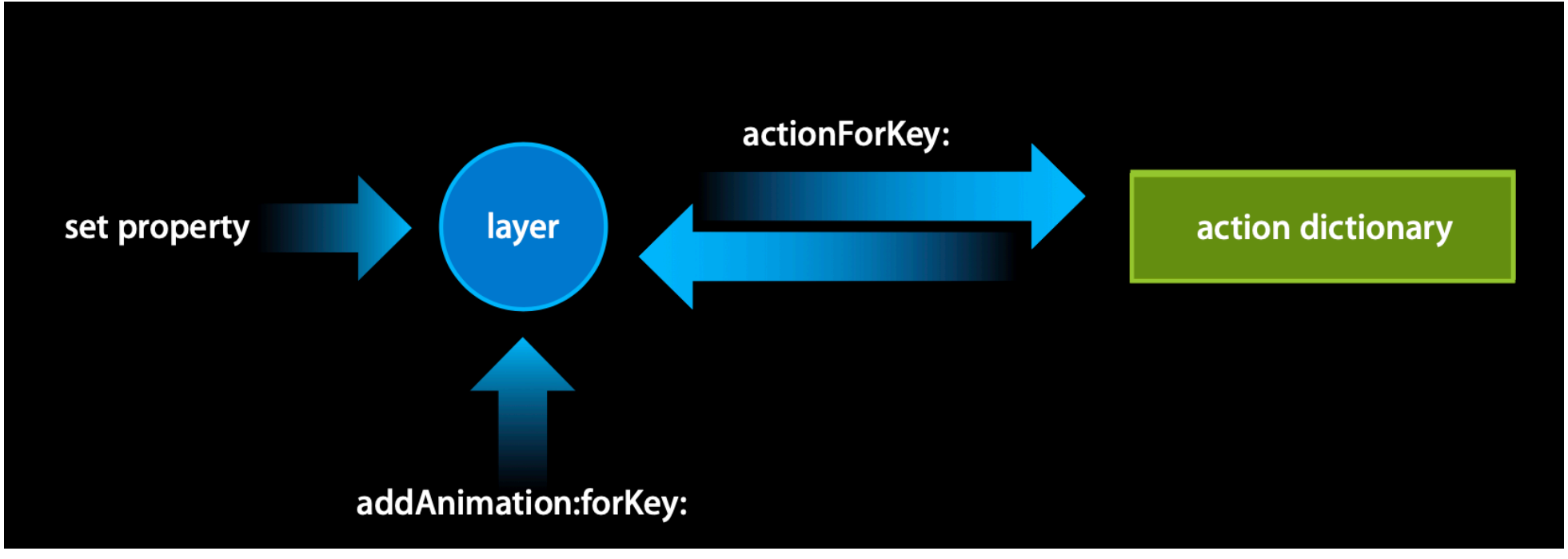
When you change an `animatabale` property of a `layer`, it will animate in the background.

That is `implicit` animation.

It happens automatically.

Here, I'm just going to change the opacity and position of the image.

06.01. Implicit Animation `Dance`



The implicit animation dance: (we need to find that `action`*)*

In short:

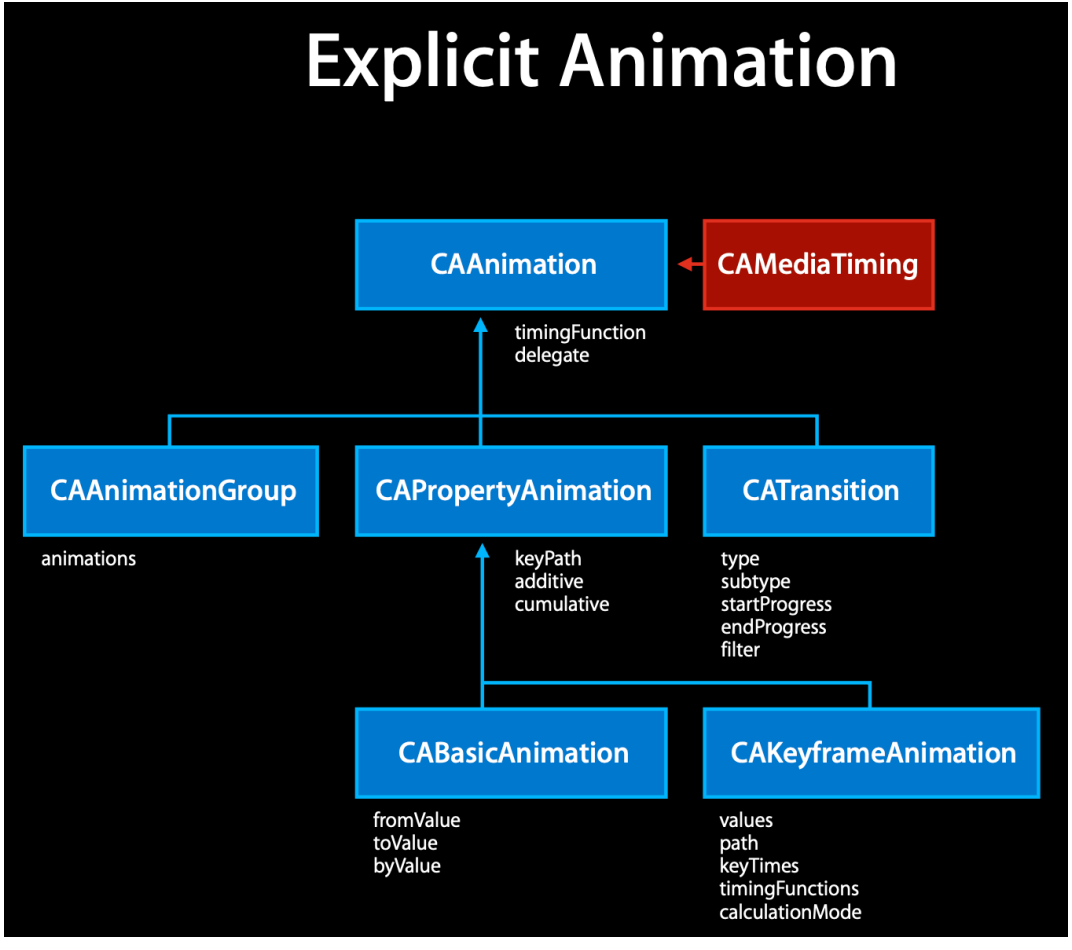
1. setter
2. `actionForKey`
3. `action dictionary`
4. `styles dictionary` (for CoreImage filters on macOS)
5. `defaultActionForKey` class method on `CALayer`

06.02. Explicit Animations

Implicit animations are great for property changes, but what if you want to animate something with no state changing?

In that case, you want to `explicitlly` program the changes to the animation model.

Explicit Animation



07. Custom Animatable Properties

Here's how it's done:

- Define a new property to be animated
- Override `needsDisplayForKey` to mark that changing the property should re-draw the layer
- Use property's value in the `drawInContext` method
- For implicit animations, override `actionForkey` method

```
class MyView : UIView { // exists only to host MyLayer
    override class var layerClass : AnyClass {
        return MyLayer.self
    }
```

```
}

class MyLayer : CALayer {

    // Define a new property to be animated
    // Make it `NSManaged` (previously `dynamic`) so that `CALayer` implements the accessors
    @NSManaged var thickness : CGFloat

    // Override `needsDisplayForKey` to mark that changing the property should re-draw the
layer
    override class func needsDisplay(forKey key: String) -> Bool {
        if key == #keyPath(thickness) {
            return true
        }
        return super.needsDisplay(forKey:key)
    }

    // Use property's value in the `drawInContext` method
    override func draw(in con: CGContext) {
```

```
        let r = self.bounds.insetBy(dx:10, dy:10)
        con.setFillColor(UIColor.red.cgColor)
        con.fill(r)
        con.setLineWidth(self.thickness)
        con.stroke(r)
    }

    // For implicit animations, override `actionForkey` method
    // crate your own `CAAction`
    override func action(forKey key: String) -> CAAction? {
        if key == #keyPath(thickness) {
            let ba = CABasicAnimation(keyPath: key)
            ba.fromValue = self.presentation()!.value(forKey:key)
            return ba
        }
        return super.action(forKey:key)
    }
}
```

```
let lay = v.layer as! MyLayer
lay.thickness = 30
```

08. Difference Between Animations and Transitions

Implicit and explicit animations are both dealing with properties of the layers.

More precisely, they are dealing with the properties that can be numerically interpolated.

The default transition is `cross-fade`.

Where available, you can use `CoreImage` to write your transitions.

While doing transitions, you might want to disable implicit animations and return `nil` for `actionForKey` method on the layer.

```
// For disabling implicit animations, override `actionForkey` method and return `nil` fro your
keyPath
override func action(forKey key: String) -> CAAction? {
    if key == #keyPath(myKeyPath) {
        return nil
    }
    return super.action(forKey:key)
}
```

09. Core Animation Performance

For us, application developers, GPU is really just the device that converts triangles to pixels

- Each triangle is filled with a color or an image
- Each can `blend` over background
- Destination can also be an image
- Core Animation takes each layer and maps them into triangles.
- Each rectangle (a layer) is just two triangles.
- Blended triangles have to draw triangles into a piece of memory and then use that as a source image to apply to another triangle.
- That interrupts the GPU pipeline, which is costly.

09.01. Write Bandwidth

Write fewer pixels

See this image with label


SURF

Each pixel inside the text needs to be written twice.

Checklist:

- Set your views to be `opaque` if possible
- Use iOS simulator's `Color Blended Layers` analyser
- Only use transparency if you really need it
- Make sure that opaque images have no alpha channel (on older iOS versions)
- Cut views with opaque regions into sublayers
- If your views are still too complex, consider using `Core Graphics` for drawing content

09.02. Read Bandwidth

Read fewer pixels.

Every pixel needs to be read.

- Use images that, as much as possible, match screen resolution.
- Use `Color Misalagnied images` instrument
- Resize big images by creating them with `Core Graphics`.

09.03. Rendering Passes

Avoid off-screen rendering

Your application should have one rendering pass per frame.

Expensive operations that cause off-screen rendering pass :

- Masking (rounded corners, masks)
- Shadows

Use the `Color Offscreen` Instruments option