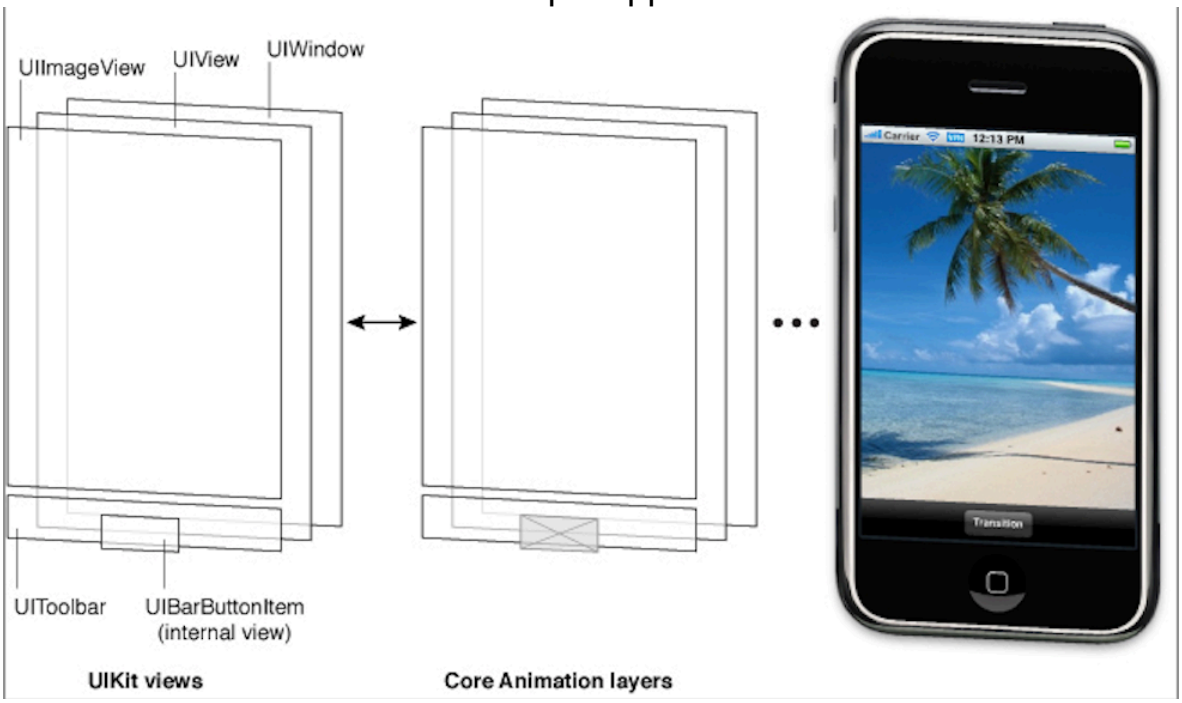
View and Window Architecture

View Architecture Fundamentals
A view object(instances of the UIView class) defines a rectangular region on the screen and handles the drawing and touch events in that region. A view can also act as a parent of other views and coordinate the placement and sizing of those views.
Architecture of the views in a sample application



> View Hierarchies and Subview Management
1. When one view contains another, a parent-child relationship is created between the two views. The child view in the relationship is known as **subview** and the parent view is **superview**.

2. The content of subview obscures all or part of the superview.

3. The superview-subview impacts several view behaviors. Changing the size of parent view has an effect that can cause size and position of the subview to change too.

4. The arrangement of views in a view hierarchy determine how your application responds to events. An events is sent to the view objects to handle, if not, pass it to the superview, if still not, pass it to its superview,

> The View Drawing Cycle
1. A view first appears on the screen, the system asks it to draw the

content.
=> The system captures a snapshot of this content and use that snapshot as representation.
=> If the view's content is never changed, drawing code is not call, and reuse the snapshot;
=> If the view's content is changed, you notify the system. And the view repeats the process of drawing-capture.

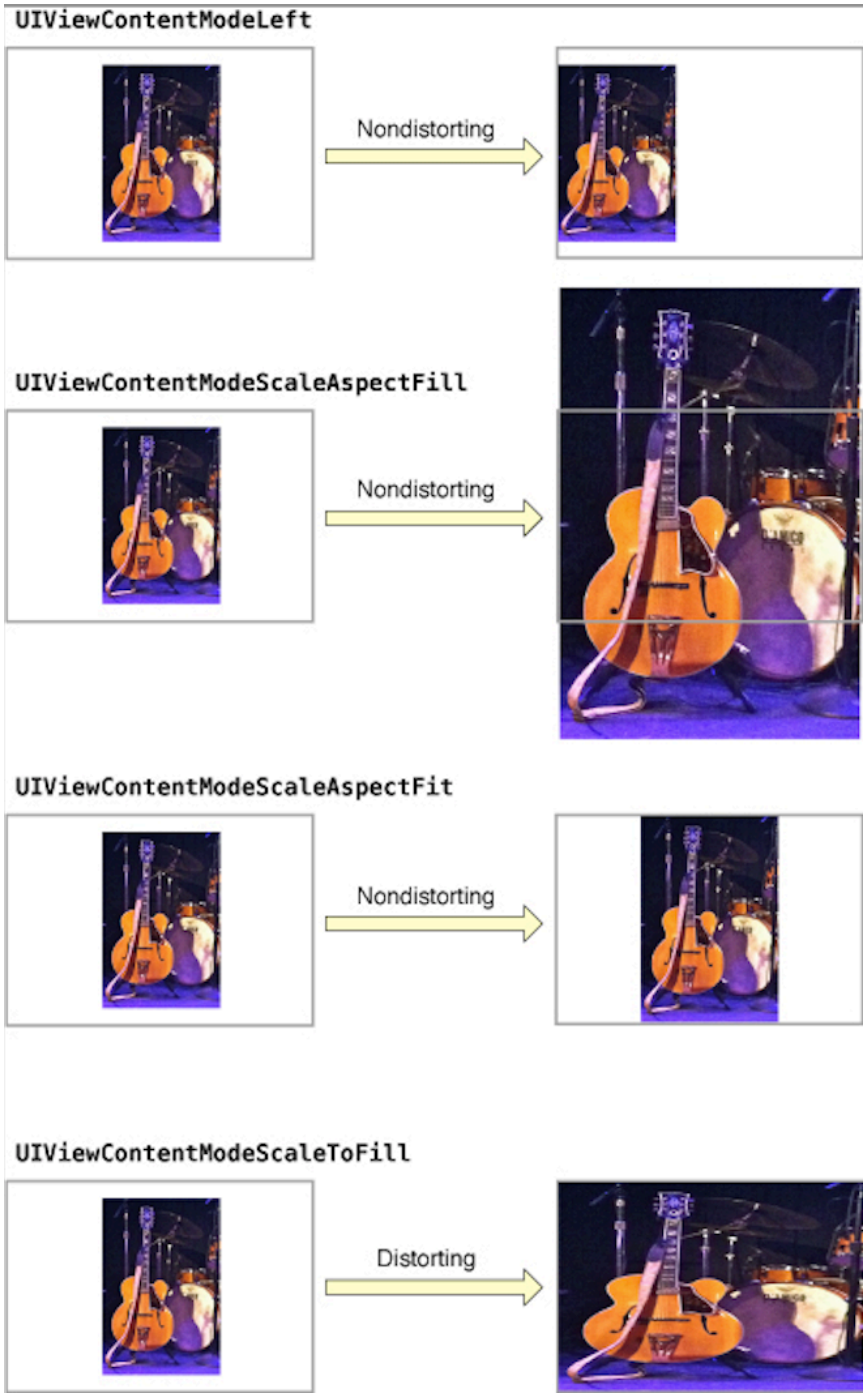
2. When the view's content is changed, you do not redraw the content directly, the system will wait until the end of the run loop before initiating any drawing operations.

3. When the time comes to render your view's content, the actual drawing process is a little different depending on the view and its configuration..
For system views, implement private drawing process, they often provide you interfaces that you can use to configure it;
For custom views, override the drawRect: method of your view and use the method to draw content.

> Content Modes
1. Each view has a content mode that controls how the view recycles its content in response to changes in the view's geometry and whether recycles its content at all.

2. The content mode of a view is applied whenever you do the following:
=> Change the width of height of the view's frame or bounds rectangles.
=> Assign a transform that includes a scaling factor to the view's transform property.

3. Content modes comparisons



> Stretchable Views

1. You can designate a portion of view as stretchable so that when the size of the view changes only the content in the stretchable portion is affected.

2. You specify the stretchable area of a view using the contentStretch property.

3. Stretchable areas are only used when the content mode would cause

the view's content to be scaled.

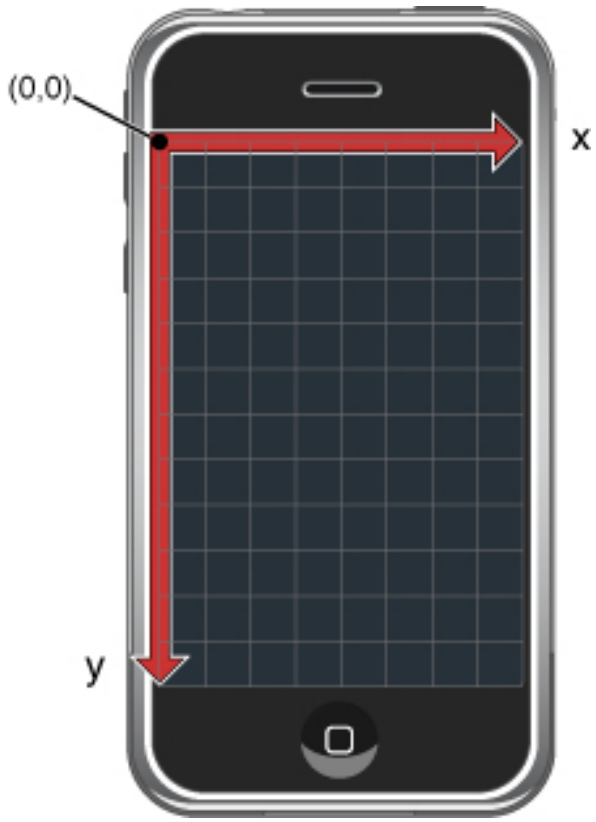
> Built-In Animation Support
1. To perform an animation, all you have to do is:
=> Tell UIKit you want to perform an animation
=> Change the value of the property.

2. Properties that you can animate on a UIView object:
=> frame—Use this to animate position and size changes for the view.
=> bounds—Use this to animate changes to the size of the view.
=> center—Use this to animate the position of the view.
=> transform—Use this to rotate or scale the view.
=> alpha—Use this to change the transparency of the view.
=> backgroundColor—Use this to change the background color of the view.
=> contentStretch—Use this to change how the view's contents stretch.

3. You use a view controller to manage the animation associated with major changes between part of your user interface.

4. In addition using UIKit classes to create animation, you can also use Core Animation layers.

View Geometry and Coordinate Systems
1. The default coordinate system in UIKit has its origin in the top-left and has axes that extend down and to the right from the origin point.
windows and views can define their own local coordinate systems.
Coordinate system orientation in UIKit



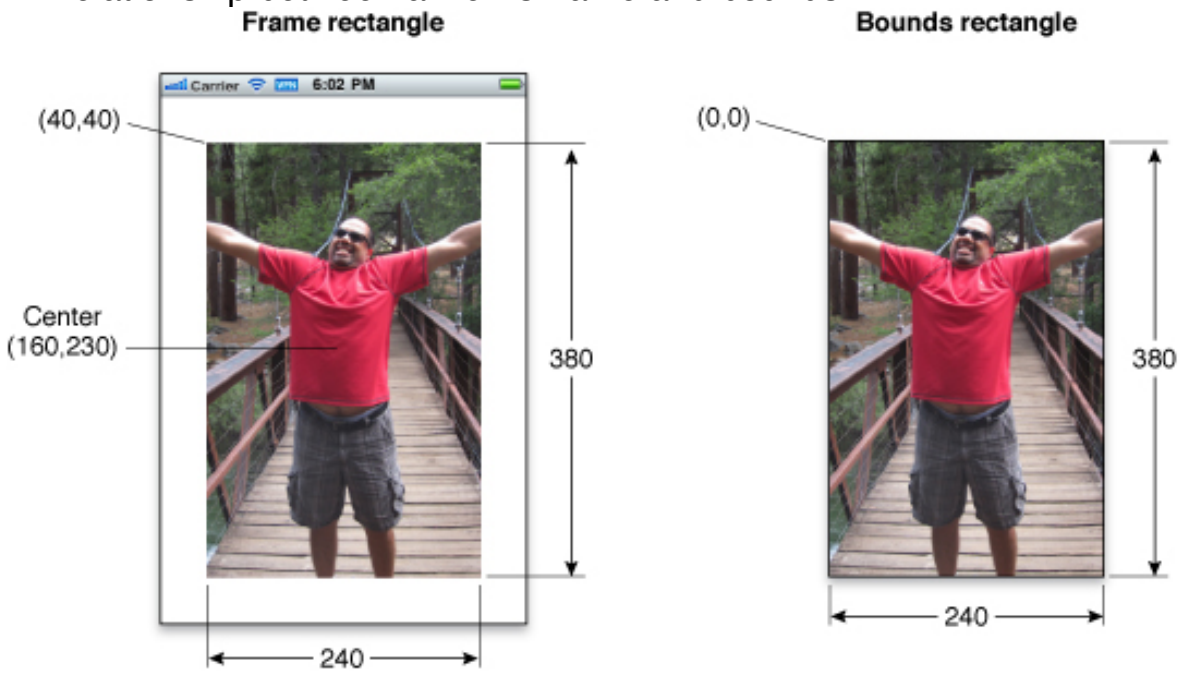
2. You need to be aware of the which coordinate system is in effect.
For drawing, you specify coordinates relative to the view's own coordinate system;
For geometry changes, you specify coordinates relative to superview's coordinate system.

> The Relationship of the Frame, Bounds, and Center Properties
1. A view object tracks its size and position using its frame, bounds and center properties:
=> The frame property contains the frame rectangle, which specifies the size and location of the view in its superview's coordinate system.
=> The bounds property contains the bounds rectangle, which specifies the size of the view (and its content origin) in the view's own local coordinate system.
=> The center property contains the known center point of the view in the superview's coordinate system.

2. You use frame and center properties for manipulating the geometry of the current view.

3. You use bounds for drawing.

4. Relationship between a view's frame and bounds.



5. Changes to one property affect the others in the following ways:
=> When you set the frame property, the size value in the bounds property changes to match the new size of the frame rectangle. The value in the center property similarly changes to match the new center point of the frame rectangle.
=> When you set the center property, the origin value in the frame changes accordingly.
=> When you set the size of the bounds property, the size value in the frame property changes to match the new size of the bounds rectangle.

6. By default, a view's frame is not clipped to its superview's frame.

> Coordinate System Transformations
1. Coordinate system transformations offer a way to alter your view or its contents. An affine transform is a matrix that specify how points in one coordinate system map to points in a different coordinate system. How you apply the affine transform depends on context:
=> To modify your entire view, modify the affine transform in the transform property of your view.
=> To modify specific pieces of content in your view's drawRect: method, modify the affine transform associated with the active graphics context.

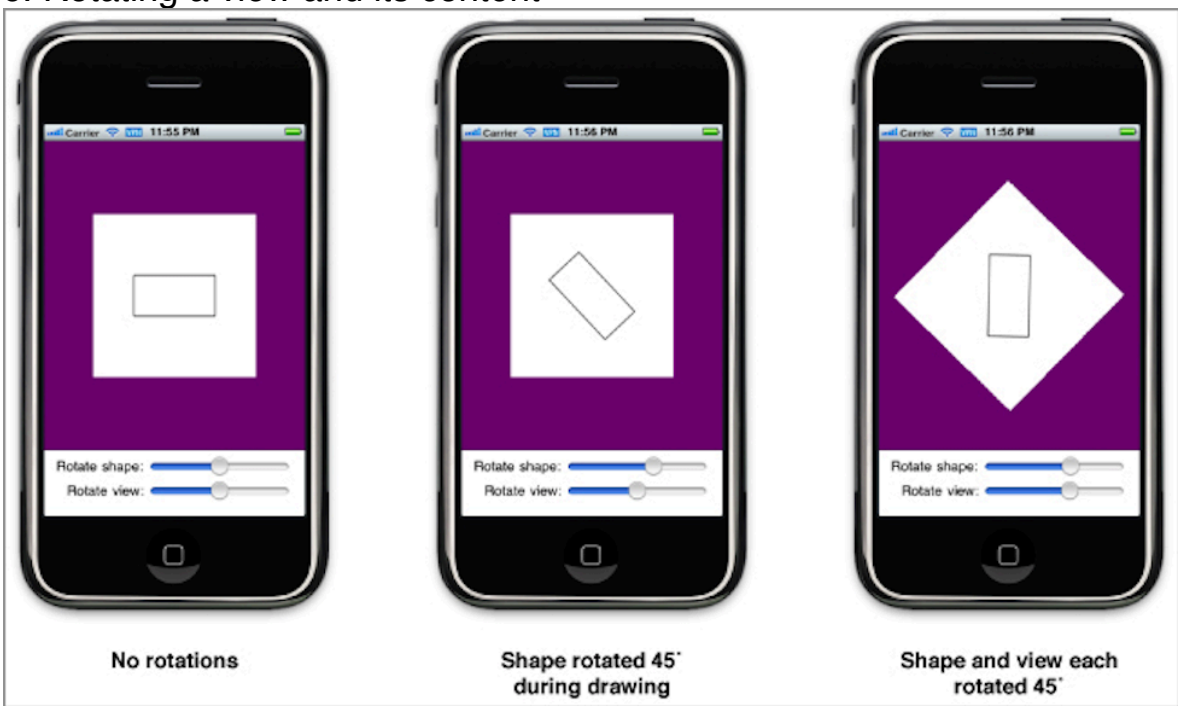
2. You can modify the transform property of a view when you want to implement an animations. And you would not use this property to make permanent changes to your view.

3. In your view's drawRect: method, you use affine transformations to position and orient the items you plan to draw.

4. The current transformation matrix (CTM) is the affine transform in use at any given time. When manipulating the geometry of your entire view, the CTM is the affine transform stored in your view's transform property. Inside your drawRect: method, the CTM is the affine transform associated with the active graphics context.

5. When you modify a view's transform property, that changes affects the view and its subview. However, these changes only affect only the final rendering of the views on the screen.

6. Rotating a view and its content



> Points Versus Pixels

1. The main thing to understand about **points** is that they provide a fixed frame of reference for drawing.

2. The point-based measuring system used for each type of device defines what is known as **user coordinate space**. This is the standard coordinate space you use for nearly all of your code.

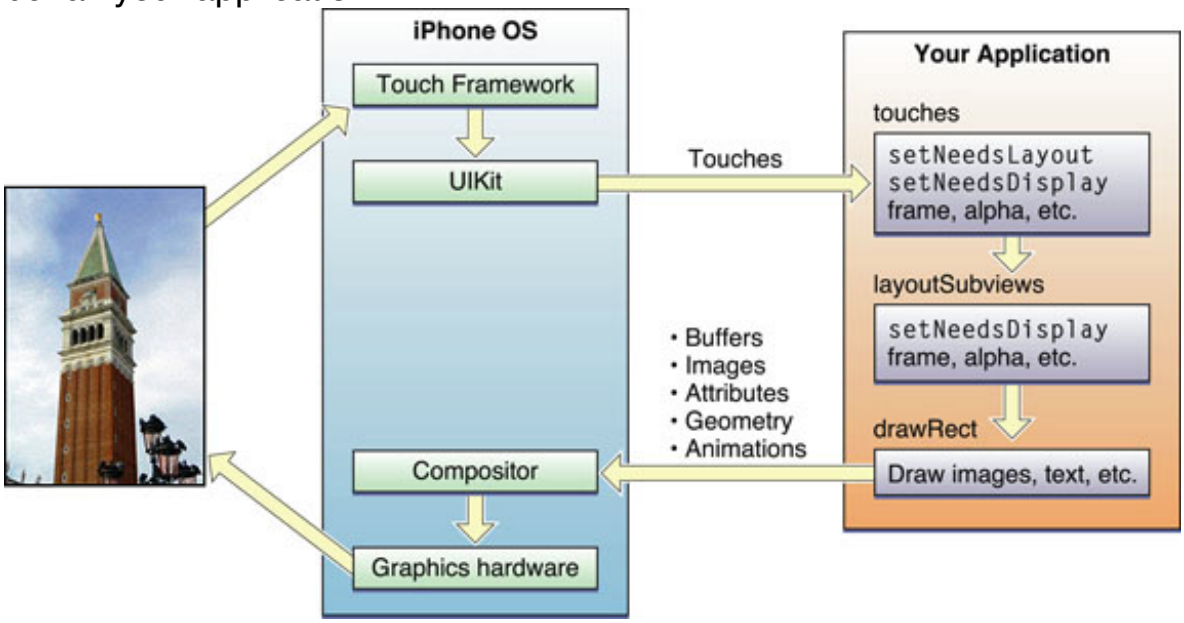
3. At the device level, all the coordinates in your view must be converted to pixels at some point. The mapping of points in user coordinate space

to pixels in **device coordinate space** is handled by the system.

4. When you work with images or pixel-based technologies such as OpenGL ES, iOS provides help in managing those pixels.

The Runtime Interaction Model for Views

1. Any time a user interact with your user interface, or any time your code changes something, a sequence takes places inside of UIKit to handle that interaction. At specified point during that sequence, UIKit calls out to your view classes and give them a chance to respond on behalf your application.



Steps in figure above:

1. The user touches the screen.
2. The hardware reports the touch event to the UIKit framework.
3. The UIKit framework packages the touch into a UIEvent object and dispatches it to the appropriate view.
4. The event-handling code of your view responds to the event.
5. If the geometry of a view changed for any reason, UIKit updates its subviews according to the following rules:
    a. If you have configured autoresizing rules for your views, UIKit adjusts each view according to those rules.
    b. If the view implements the layoutSubviews method, UIKit calls it.
6. If any part of any view was marked as needing to be redrawn, UIKit asks the view to redraw itself.
7. Any updated views are composited with the rest of the application's visible content and sent to the graphics hardware for display.
8. The graphics hardware transfers the rendered content to the screen.

Tips for Using Views Effectively

> Views Do Not Always Have a Corresponding View Controller

1. There is rarely one-to-one relationship between individual views and view controllers. The job of view controller is to manage a view hierarchy, which often consists of more than one view used to implement some self-contained feature.

2. It's important to consider the role that view controllers will play.

> Minimize Custom Drawing

Any time your content can be assembled with a combination of existing views, your best bet is to combine those view objects into a custom view hierarchy.

> Take Advantage of Content Modes

> Declare Views as Opaque Whenever Possible

> Adjust Your View's Drawing Behavior When Scrolling

> Do Not Customize Controls by Embedding Subviews