CS193P - Lecture 5

iPhone Application Development

Views
Drawing
Animation

Announcements

- Assignment 1 grades are out. Contact Paul or Dave if you didn't get yours
- Contact Paul or Dave if you need a loaner iPod Touch
- Assignments 2A and 2B due Wednesday, 1/20

Questions from Monday?

- Model, View, Controller
- Interface Builder & Nibs
- Delegate
  - Allows one object to act on behalf of another object
- Target-Action

Today's Topics

- Views
- Drawing
- Text & Images
- Animation

Views

View Fundamentals

- Rectangular area on screen
- Draws content
- Handles events
- Subclass of UIResponder (event handling class)
- Views arranged hierarchically
  - every view has one superview
  - every view has zero or more subviews

View Hierarchy - UIWindow

- Views live inside of a window
- UIWindow is actually just a view
  - adds some additional functionality specific to top level view
- One UIWindow for an iPhone app
  - Contains the entire view hierarchy
  - Set up by default in Xcode template project

View Hierarchy - Manipulation

- Add/remove views in IB or using UIView methods

```
- (void)addSubview:(UIView *)view;
- (void)removeFromSuperview;
```

- Manipulate the view hierarchy manually:

```
- (void)insertSubview:(UIView *)view atIndex:(int)index;
- (void)insertSubview:(UIView *)view belowSubview:(UIView *)view;
- (void)insertSubview:(UIView *)view aboveSubview:(UIView *)view;
- (void)exchangeSubviewAtIndex:(int)index
        withSubviewAtIndex:(int)otherIndex;
```

View Hierarchy - Ownership

- Superviews retain their subviews

- Not uncommon for views to only be retained by superview
  - Be careful when removing!
  - Retain subview before removing if you want to reuse it

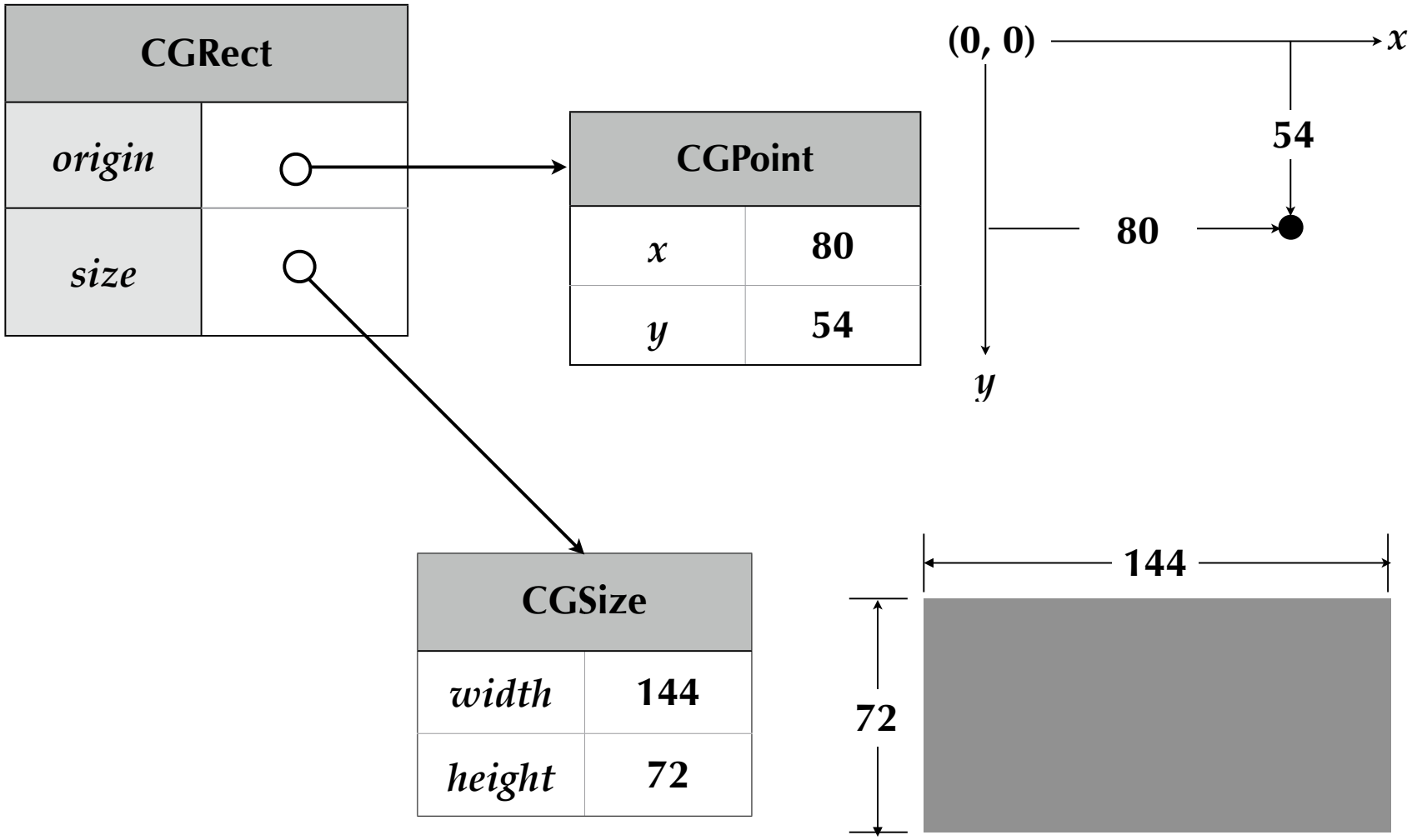
- Views can be temporarily hidden

```
theView.hidden = YES;
```

View-related Structures

- CGPoint
  - location in space: { x , y }
- CGSize
  - dimensions: { width , height }
- CGRect
  - location and dimension: { origin , size }

Rects, Points and Sizes



View-related Structure

Creation Function	Example

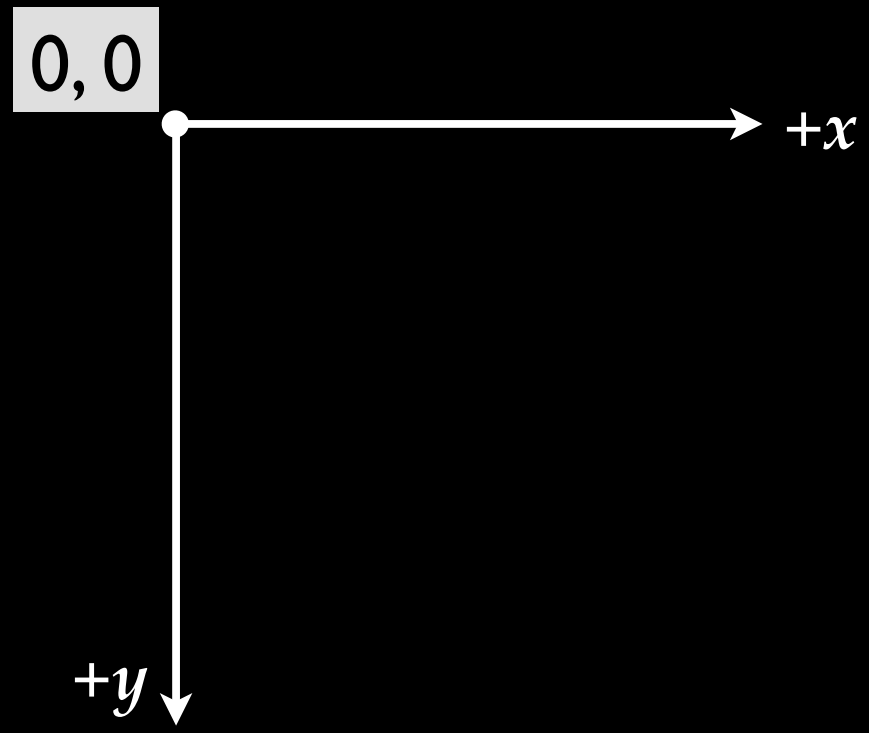
`CGPointMake (x, y)`	`CGPoint point = CGPointMake (100.0, 200.0);`
	`point.x = 300.0;`
	`point.y = 30.0;`

`CGSizeMake (width, height)`	`CGSize size = CGSizeMake (42.0, 11.0);`
	`size.width = 100.0;`
	`size.height = 72.0;`

`CGRectMake (x, y,`
`width, height)`
	`CGRect rect = CGRectMake (100.0, 200.0,`
	`42.0, 11.0);`
	
	`rect.origin.x = 0.0;`
	`rect.size.width = 50.0;`

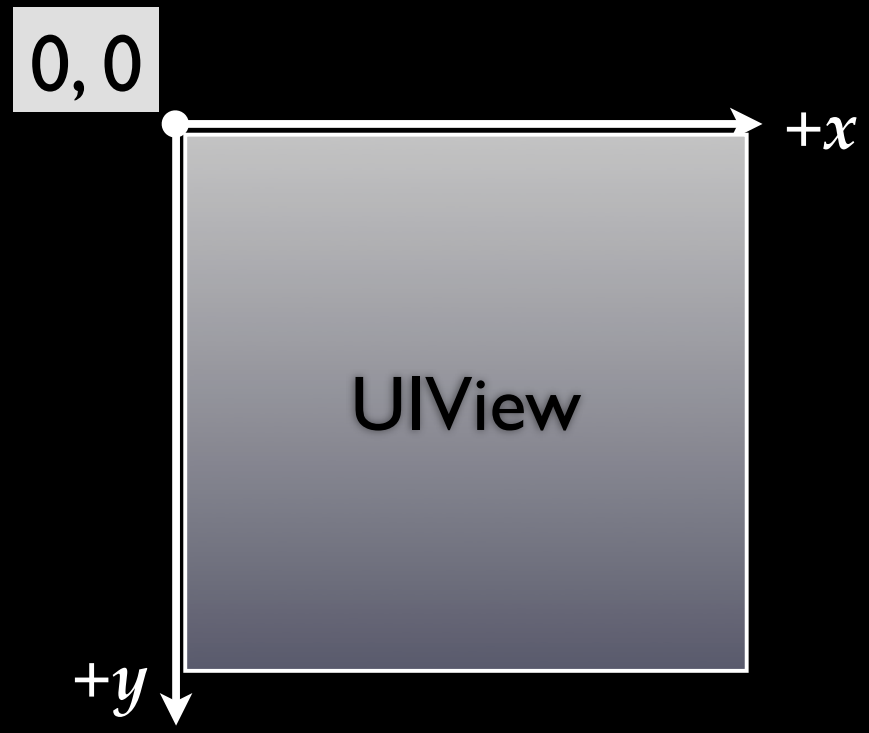
UIView Coordinate System

- Origin in upper left corner
- y axis grows downwards



UIView Coordinate System

- Origin in upper left corner
- y axis grows downwards



Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

Location and Size

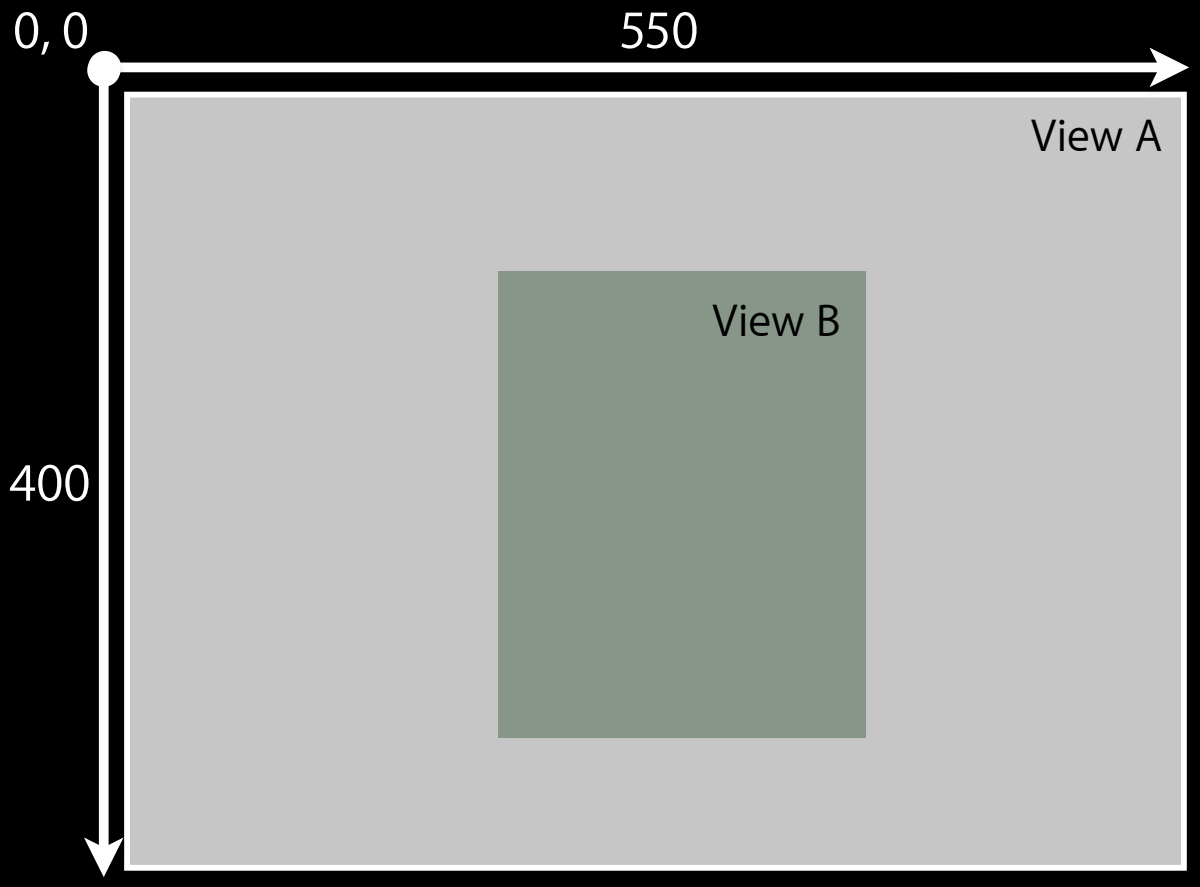
- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

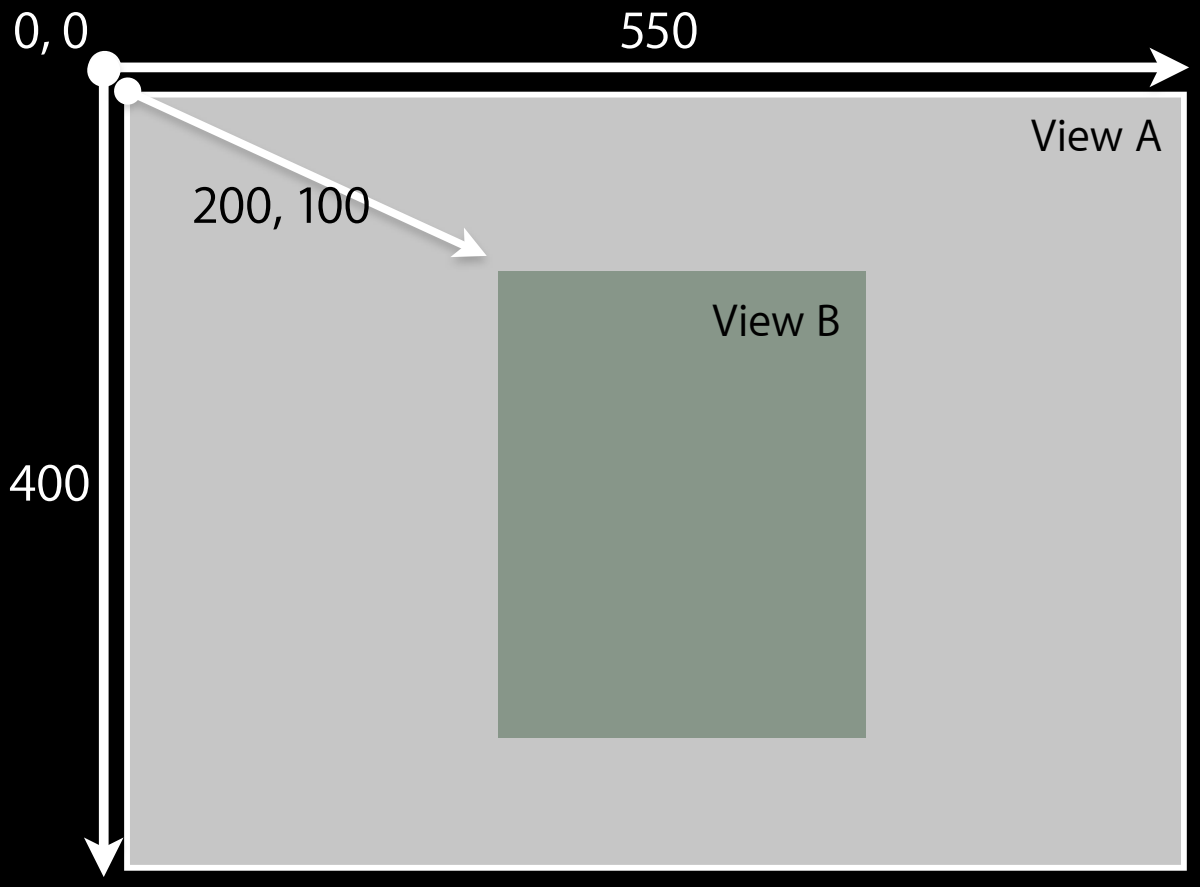


View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

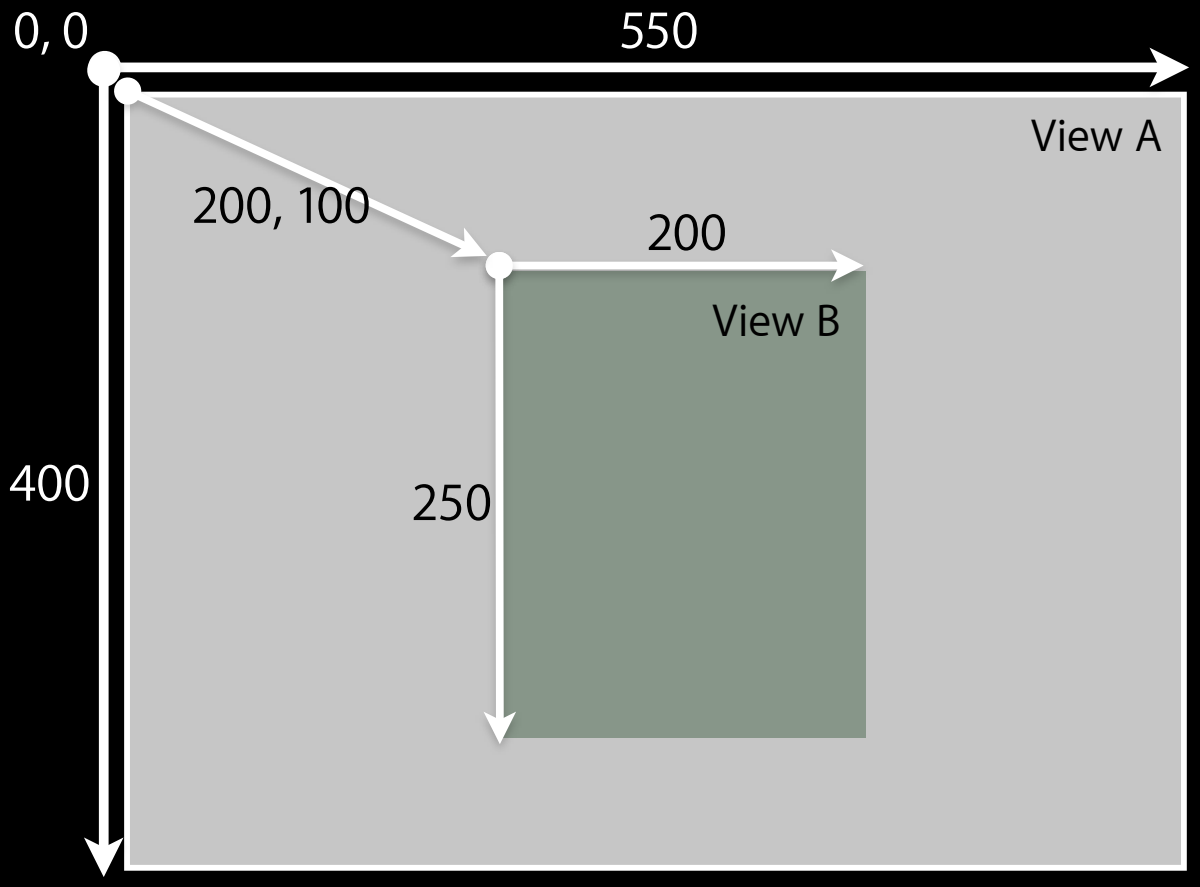


View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

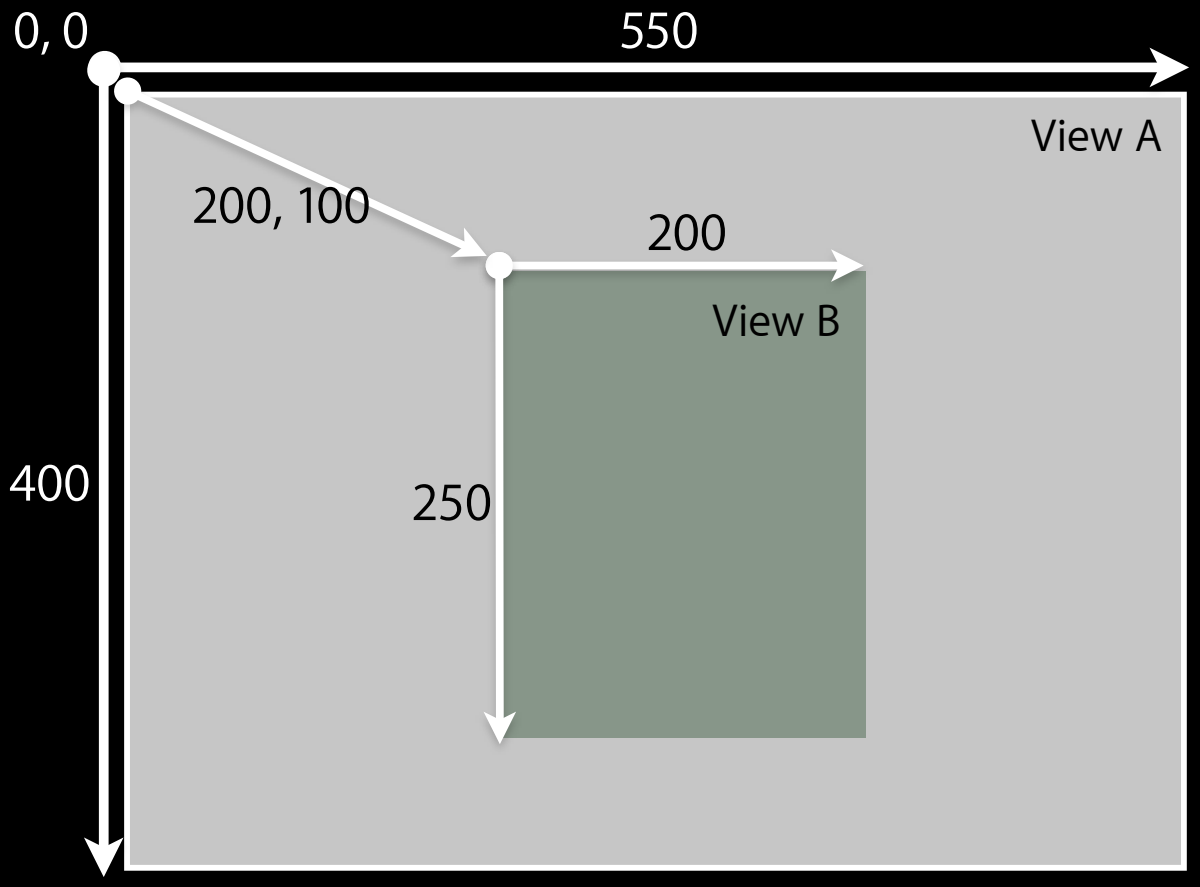


View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system



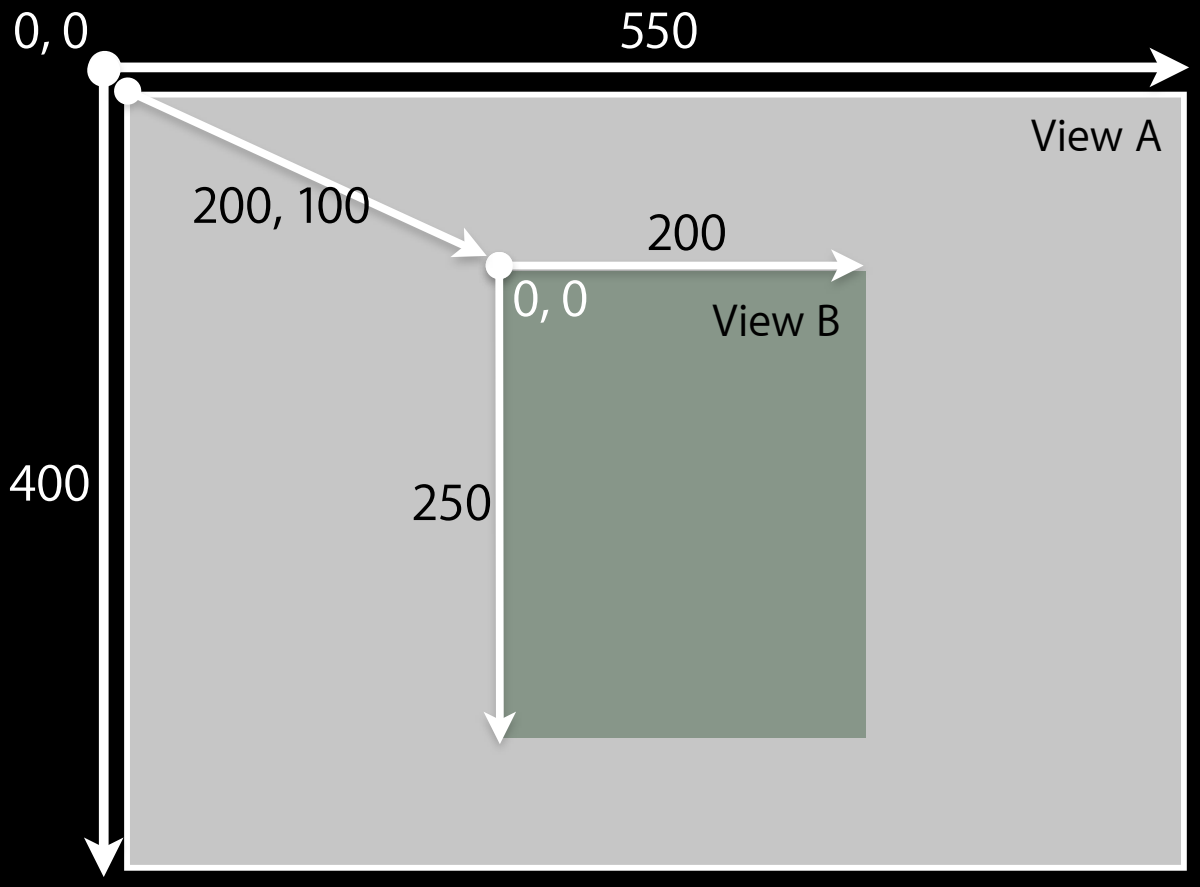
View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

View B frame:
origin: 200, 100
size: 200 x 250

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system



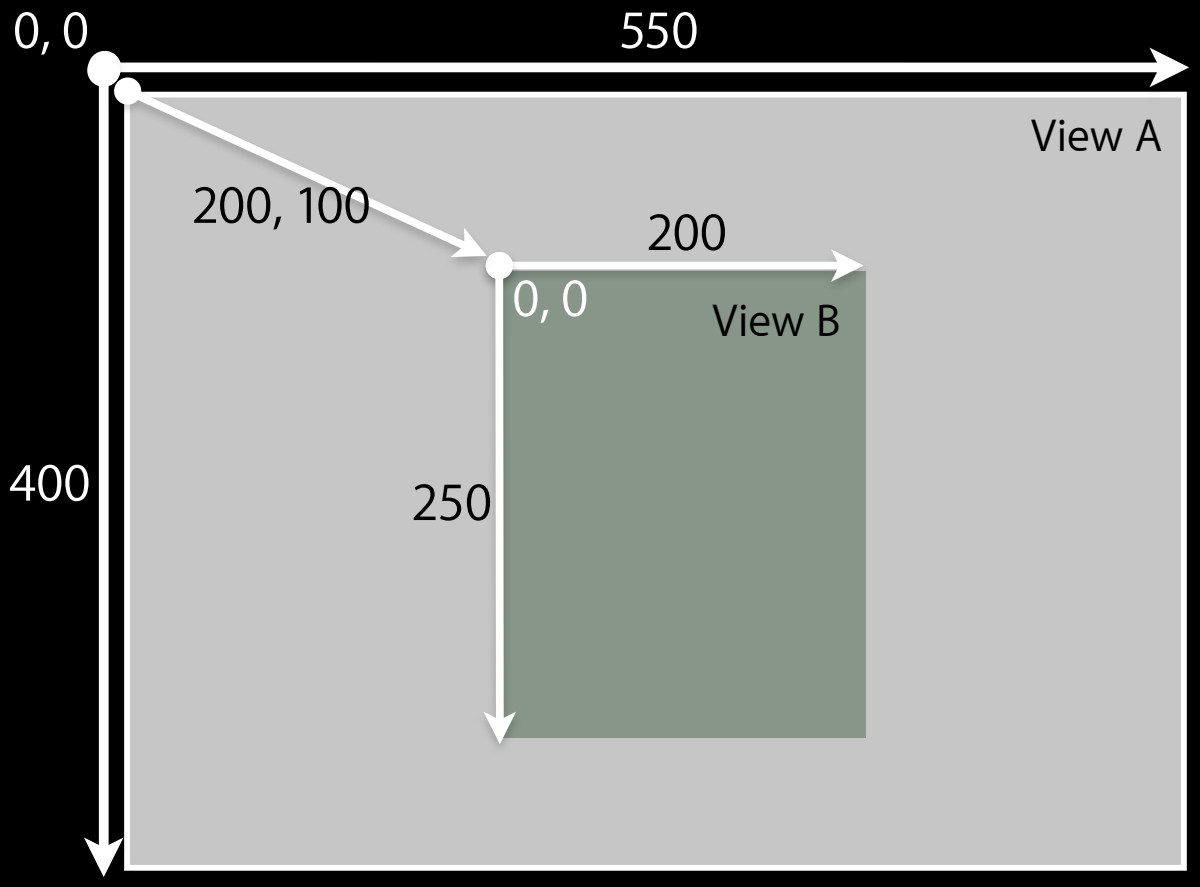
View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

View B frame:
origin: 200, 100
size: 200 x 250

Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system



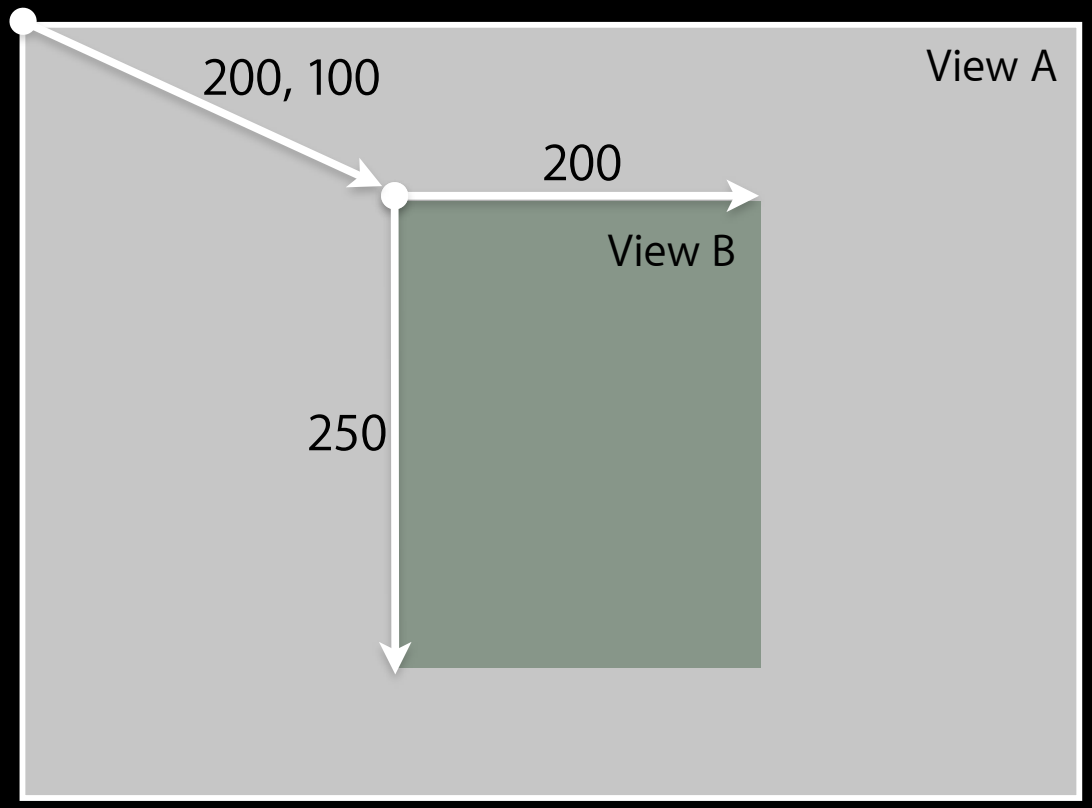
View A frame:
origin: 0, 0
size: 550 x 400

View A bounds:
origin: 0, 0
size: 550 x 400

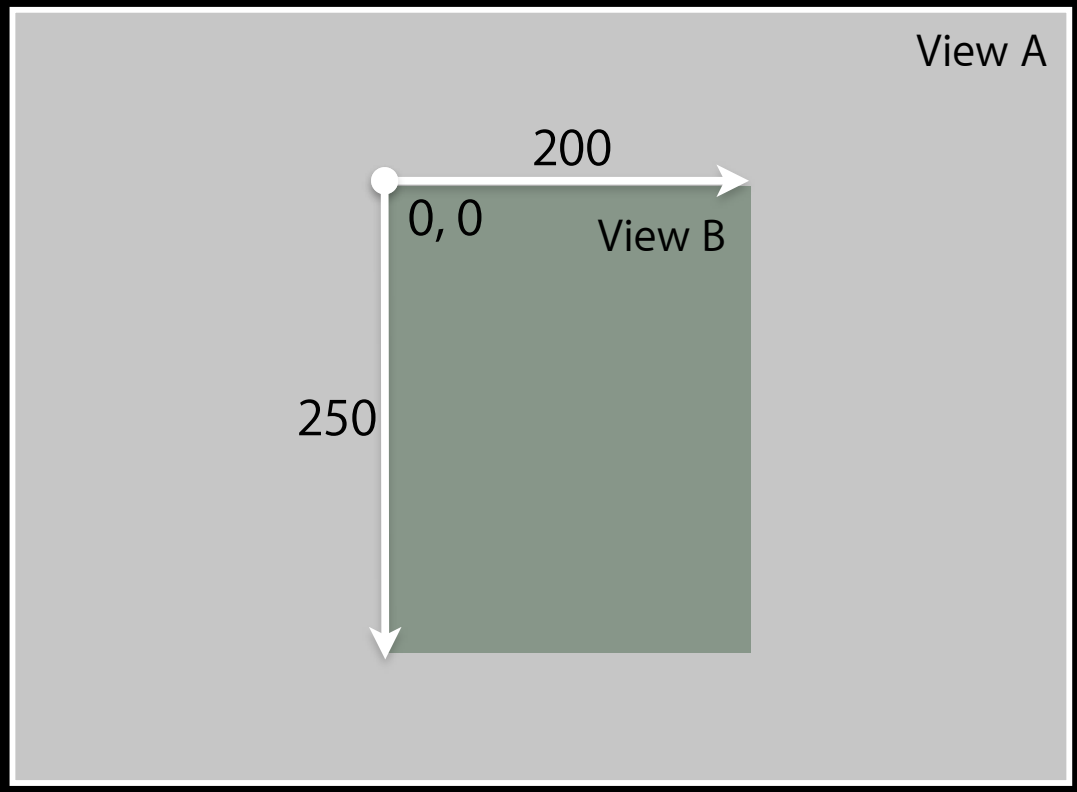
View B frame:
origin: 200, 100
size: 200 x 250

View B bounds:
origin: 0, 0
size: 200 x 250

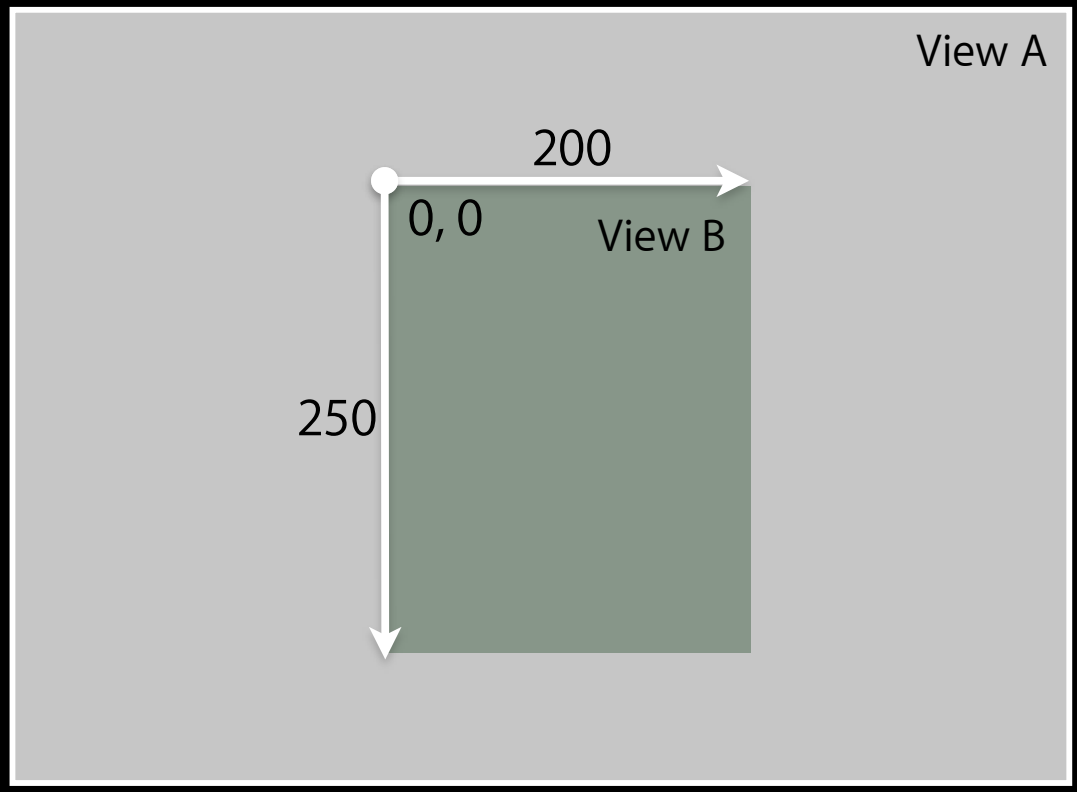
Frame is Computed



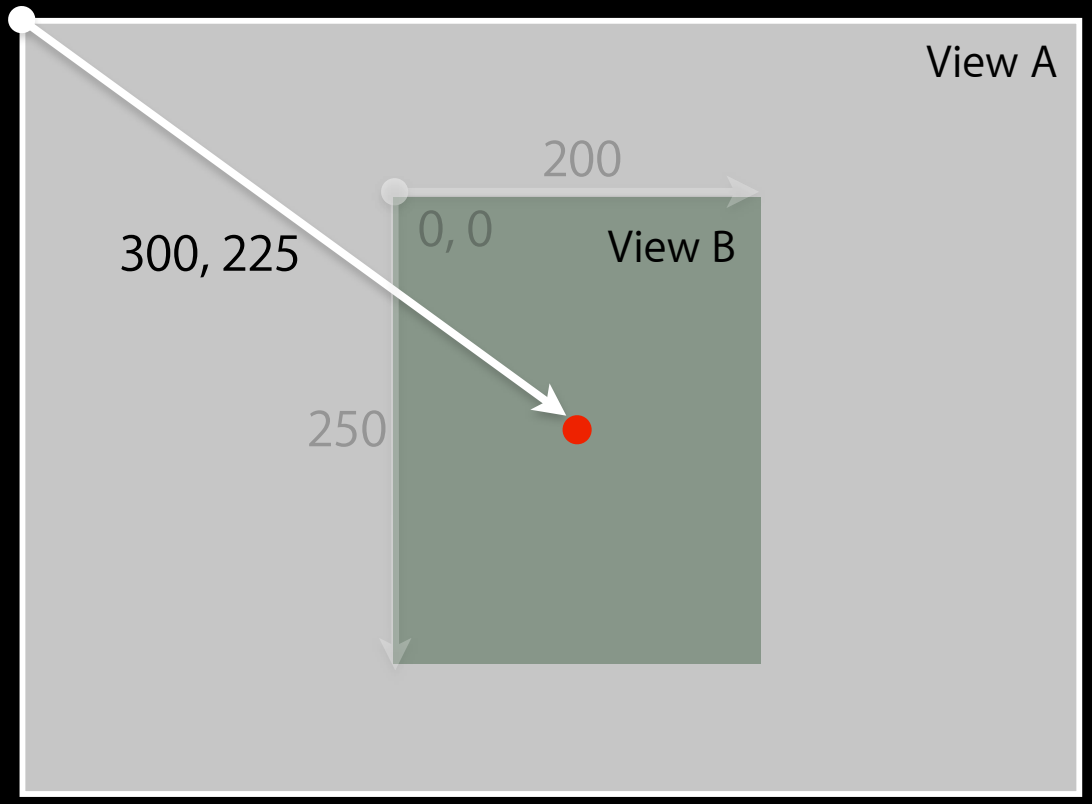
Bounds



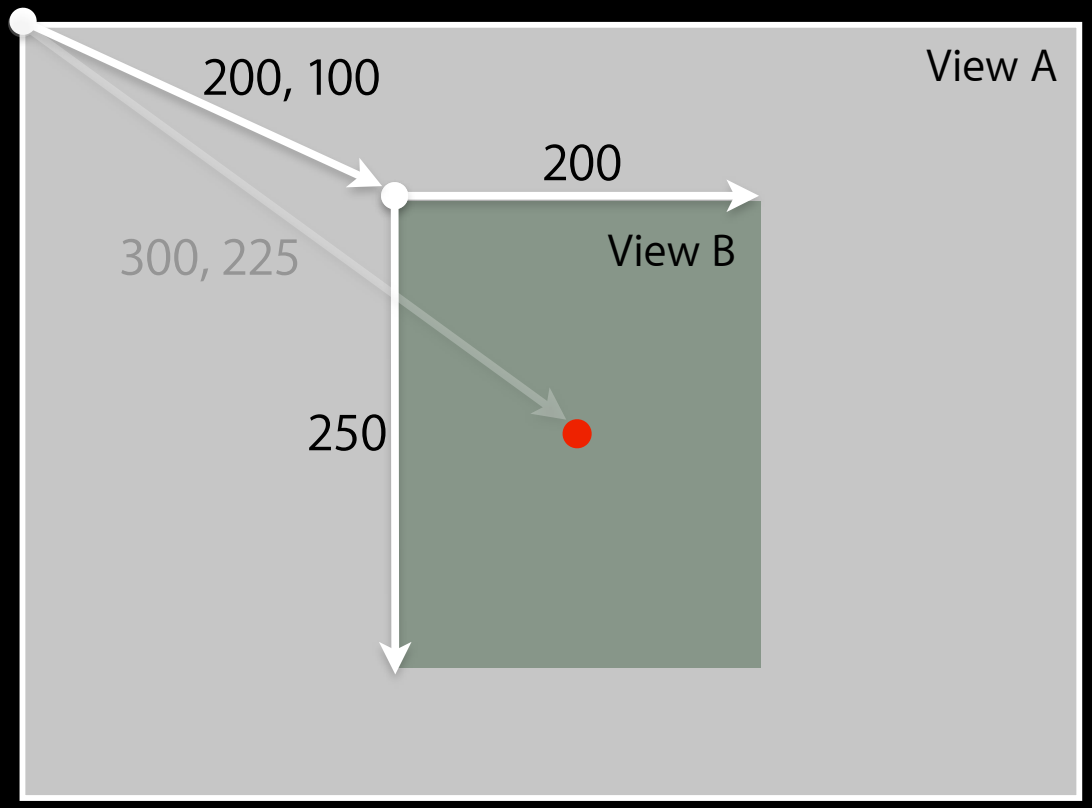
Center



Center

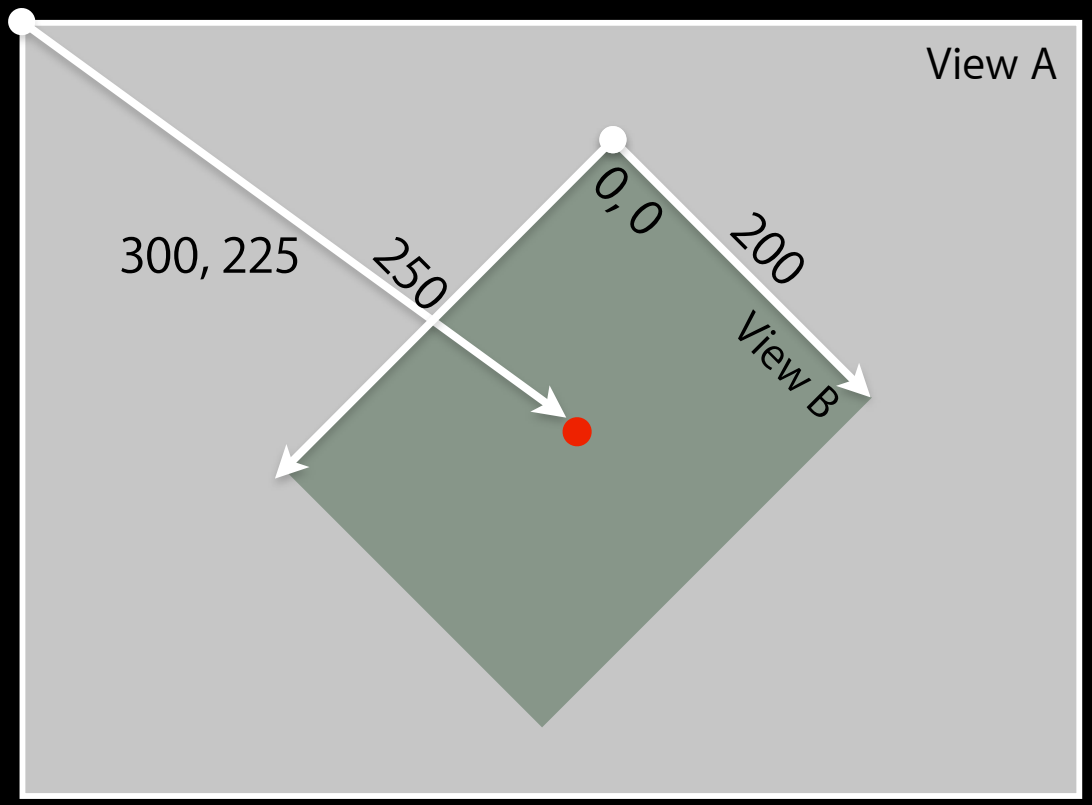


Frame



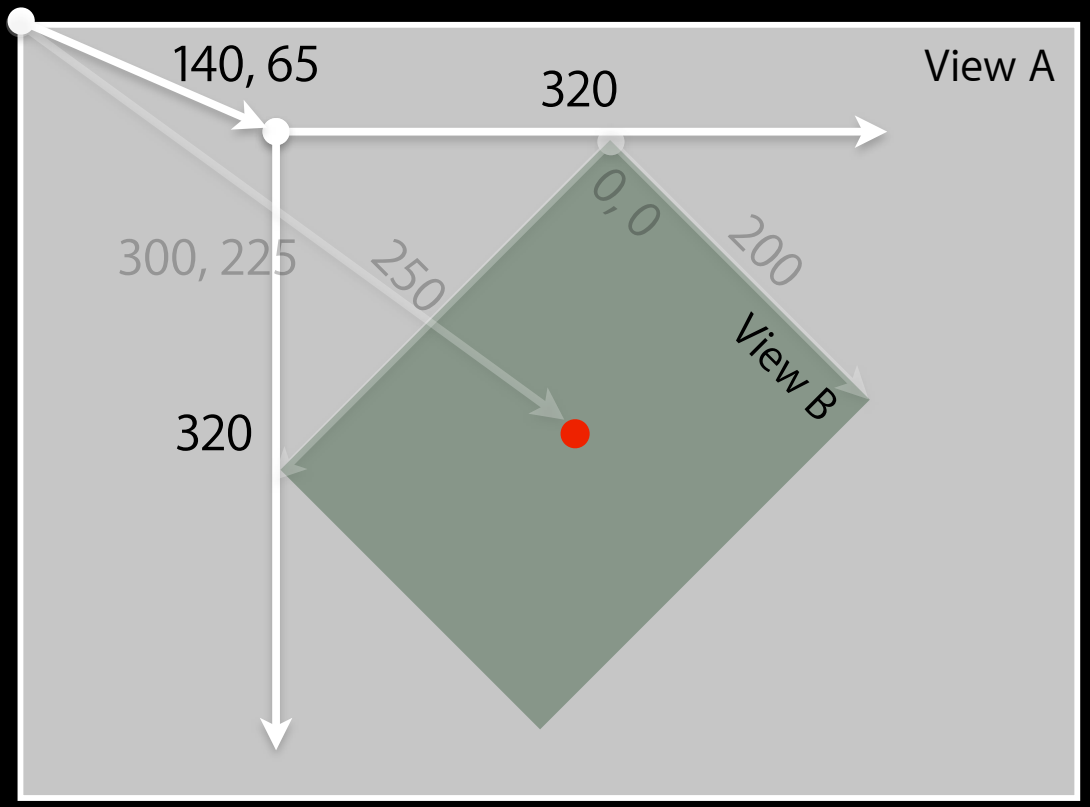
Transform

- 45° Rotation



Frame

- The smallest rectangle in the superview's coordinate system that fully encompasses the view itself



Frame and Bounds

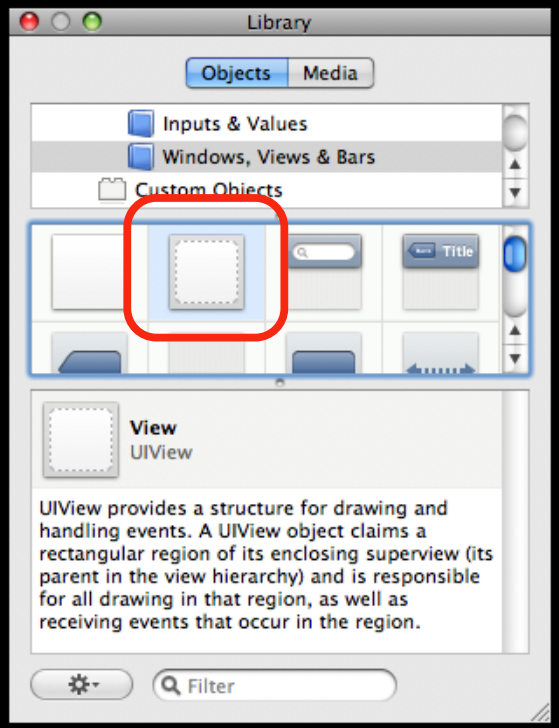
- Which to use?
  - Usually depends on the context
- If you are *using* a view, typically you use frame
- If you are *implementing* a view, typically you use bounds
- Matter of perspective
  - From outside it's usually the frame
  - From inside it's usually the bounds

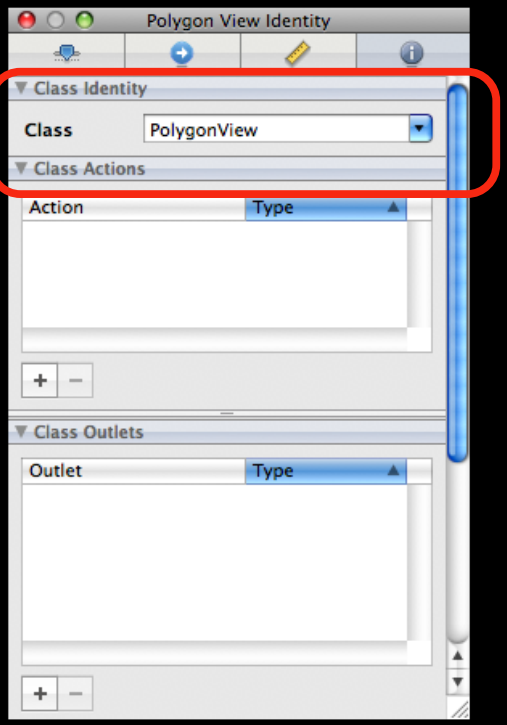
- Examples:
  - Creating a view, positioning a view in superview - use frame
  - Handling events, drawing a view - use bounds

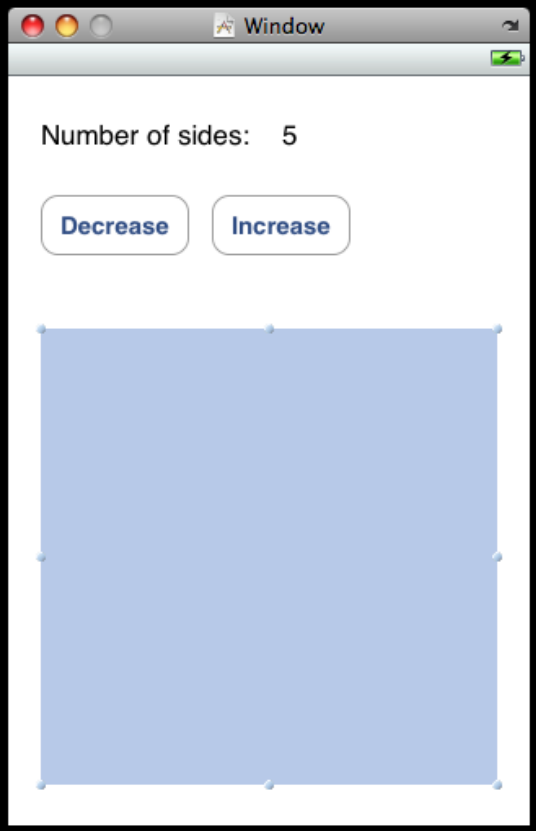
Creating Views

Where do views come from?

- Commonly Interface Builder
- Drag out any of the existing view objects (buttons, labels, etc)
- Or drag generic UIView and set custom class







Manual Creation

- Views are initialized using -initWithFrame:

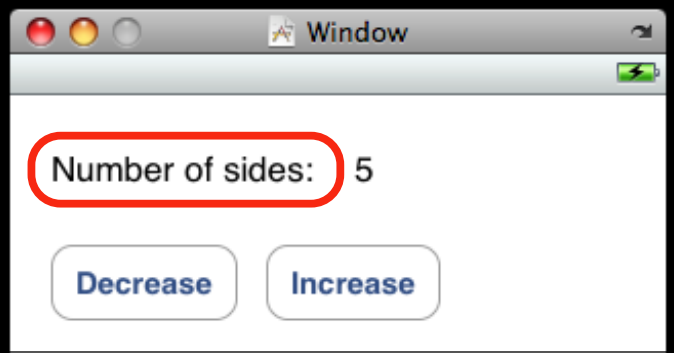
```
CGRect frame = CGRectMake(0, 0, 200, 150);

UIView *myView = [[UIView alloc] initWithFrame:frame];
```

- Example:

```
CGRect frame = CGRectMake(20, 45, 140, 21);
UILabel *label = [[UILabel alloc] initWithFrame:frame];

[window addSubview:label];
[label setText:@"Number of sides:"];
[label release];  // label now retained by window
```



Defining Custom Views

- Subclass UIView

- For custom drawing, you override:

```
- (void)drawRect:(CGRect)rect;
```

- For event handling, you override:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

Drawing Views

- (void)drawRect:(CGRect)rect

- -[UIView drawRect:] does nothing by default
  - If not overridden, then backgroundColor is used to fill
- Override - drawRect: to draw a custom view
  - rect argument is area to draw
- When is it OK to call drawRect:?

Be Lazy

- drawRect: is invoked automatically
  - Don't call it directly!

- Being lazy is good for performance

- When a view needs to be redrawn, use:

```
- (void)setNeedsDisplay;
```

- For example, in your controller:

```
- (void)setNumberOfSides:(int)sides {
    numberOfSides = sides;
    [polygonView setNeedsDisplay];
}
```

CoreGraphics and Quartz 2D

- UIKit offers very basic drawing functionality

```
UIRectFill(CGRect rect);
UIRectFrame(CGRect rect);
```

- CoreGraphics: Drawing APIs
- CG is a C-based API, not Objective-C
- CG and Quartz 2D drawing engine define simple but powerful graphics primitives
  - Graphics context
  - Transformations
  - Paths
  - Colors
  - Fonts
  - Painting operations

Graphics Contexts

- All drawing is done into an opaque graphics context
- Draws to screen, bitmap buffer, printer, PDF, etc.
- Graphics context setup automatically before invoking drawRect:
  - Defines current path, line width, transform, etc.
  - Access the graphics context within drawRect: by calling

```
(CGContextRef)UIGraphicsGetCurrentContext(void);
```

  - Use CG calls to change settings
- Context only valid for current call to drawRect:
  - Do not cache a CGContext!

CG Wrappers

- Some CG functionality wrapped by UIKit
- UIColor
  - Convenience for common colors
  - Easily set the fill and/or stroke colors when drawing

```
UIColor *redColor = [UIColor redColor];
[redColor set];
// drawing will be done in red
```

- UIFont
  - Access system font
  - Get font by name

```
UIFont *font = [UIFont systemFontOfSize:14.0];
[myLabel setFont:font];
```

Simple drawRect: example

- Draw a solid color and shape

```
- (void)drawRect:(CGRect)rect {
    CGRect bounds = [self bounds];

    [[UIColor grayColor] set];
    UIRectFill (bounds);

    CGRect square = CGRectMake (10, 10, 50, 100);
    [[UIColor redColor] set];
    UIRectFill (square);

    [[UIColor blackColor] set];
    UIRectFrame (square);
}
```

Simple drawRect: example

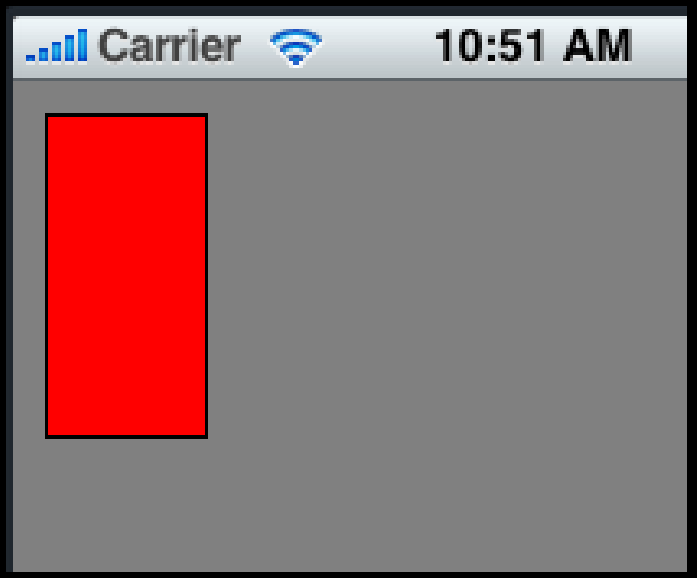
- Draw a solid color and shape

```
- (void)drawRect:(CGRect)rect {
    CGRect bounds = [self bounds];

    [[UIColor grayColor] set];
    UIRectFill (bounds);

    CGRect square = CGRectMake (10, 10, 50, 100);
    [[UIColor redColor] set];
    UIRectFill (square);

    [[UIColor blackColor] set];
    UIRectFrame (square);
}
```



Drawing More Complex Shapes

- Common steps for drawRect: are
  - Get current graphics context
  - Define a path
  - Set a color
  - Stroke or fill path
  - Repeat, if necessary

Paths

- CoreGraphics paths define shapes
- Made up of lines, arcs, curves and rectangles
- Creation and drawing of paths are two distinct operations
  - Define path first, then draw it

CGPath

- Two parallel sets of functions for using paths
  - CGContext "convenience" throwaway functions
  - CGPath functions for creating reusable paths

CGContext	CGPath
CGContextMoveToPoint	CGPathMoveToPoint
CGContextAddLineToPoint	CGPathAddLineToPoint
CGContextAddArcToPoint	CGPathAddArcToPoint
CGContextClosePath	CGPathCloseSubPath
And so on and so on . . .	

Simple Path Example

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    [[UIColor grayColor] set];
    UIRectFill ([self bounds]);

    CGContextBeginPath (context);
    CGContextMoveToPoint (context, 75, 10);
    CGContextAddLineToPoint (context, 10, 150);
    CGContextAddLineToPoint (context, 160, 150);
    CGContextClosePath (context);

    [[UIColor redColor] setFill];
    [[UIColor blackColor] setStroke];
    CGContextDrawPath (context, kCGPathFillStroke);
}
```

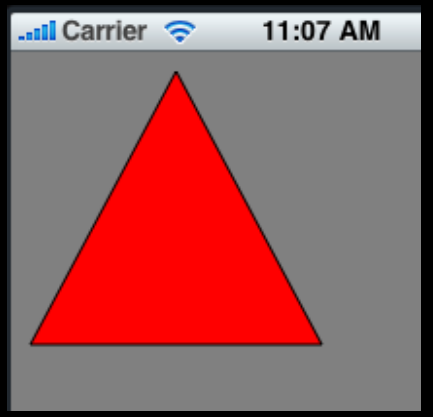
Simple Path Example

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    [[UIColor grayColor] set];
    UIRectFill ([self bounds]);

    CGContextBeginPath (context);
    CGContextMoveToPoint (context, 75, 10);
    CGContextAddLineToPoint (context, 10, 150);
    CGContextAddLineToPoint (context, 160, 150);
    CGContextClosePath (context);

    [[UIColor redColor] setFill];
    [[UIColor blackColor] setStroke];
    CGContextDrawPath (context, kCGPathFillStroke);
}
```



More Drawing Information

- UIView Class Reference
- CGContext Reference
- “Quartz 2D Programming Guide”
- Lots of samples in the iPhone Dev Center

Images & Text

UIImage

- UIKit class representing an image

- Creating UIImages:
  - Fetching image in application bundle
    - Use +[UIImage imageNamed:(NSString *)name]
    - Include file extension in file name, e.g. @"myImage.jpg"
  - Read from file on disk
    - Use -[UIImage initWithContentsOfFile:(NSString *)path]
  - From data in memory
    - Use -[UIImage initWithData:(NSData *)data]

Creating Images from a Context

- Need to dynamically generate a bitmap image
- Same as drawing a view
- General steps
  - Create a special CGGraphicsContext with a size
  - Draw
  - Capture the context as a bitmap
  - Clean up

Bitmap Image Example

```
- (UIImage *)polygonImageOfSize:(CGSize)size {
    UIImage *result = nil;

    UIGraphicsBeginImageContext (size);

    // call your drawing code...

    result = UIGraphicsGetImageFromCurrentContext();

    UIGraphicsEndImageContext();

    return result;
}
```

Getting Image Data

- Given UIImage, want PNG or JPG representation

```
NSData *UIImagePNGRepresentation (UIImage * image);
NSData *UIImageJPGRepresentation (UIImage * image);
```

- UIImage also has a CGImage property which will give you a CGImageRef to use with CG calls

Drawing Text & Images

- You can draw UIImages in -drawRect:

```
- [UIImage drawAtPoint:(CGPoint)point]
- [UIImage drawInRect:(CGRect)rect]
- [UIImage drawAsPatternInRect:(CGRect)rect]
```

- You can draw NSString in -drawRect:

```
- [NSString drawAtPoint:(CGPoint)point withFont:(UIFont *)font]
```

Drawing Text & Images

- You can draw UIImages in -drawRect:

```
- [UIImage drawAtPoint:(CGPoint)point]
- [UIImage drawInRect:(CGRect)rect]
- [UIImage drawAsPatternInRect:(CGRect)rect]
```

- You can draw NSString in -drawRect:

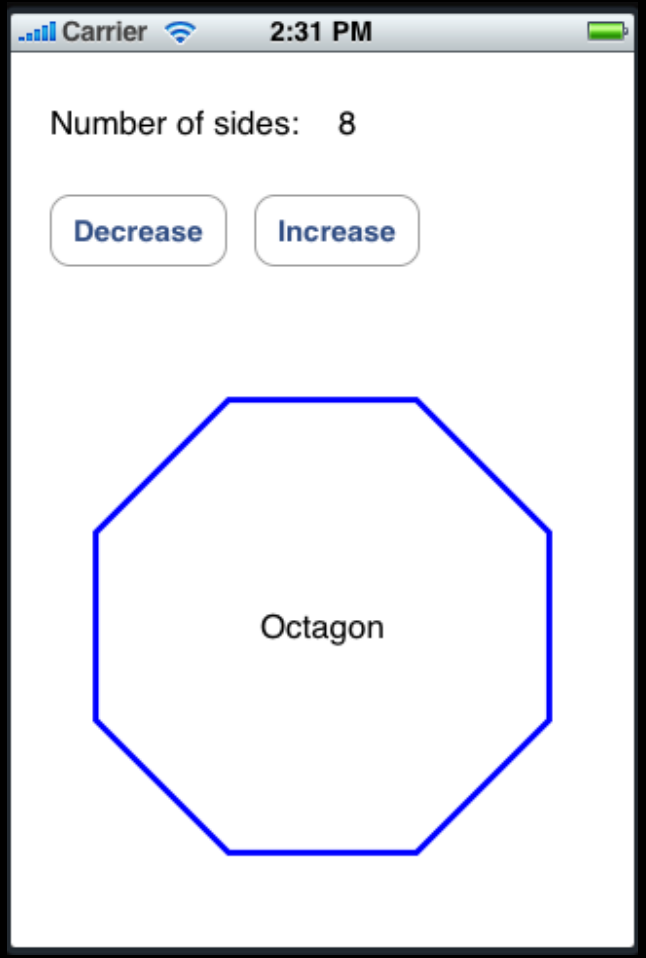
```
- [NSString drawAtPoint:(CGPoint)point withFont:(UIFont *)font]
```

But there is a better way!

Text, Images, and UIKit views

Constructing Views

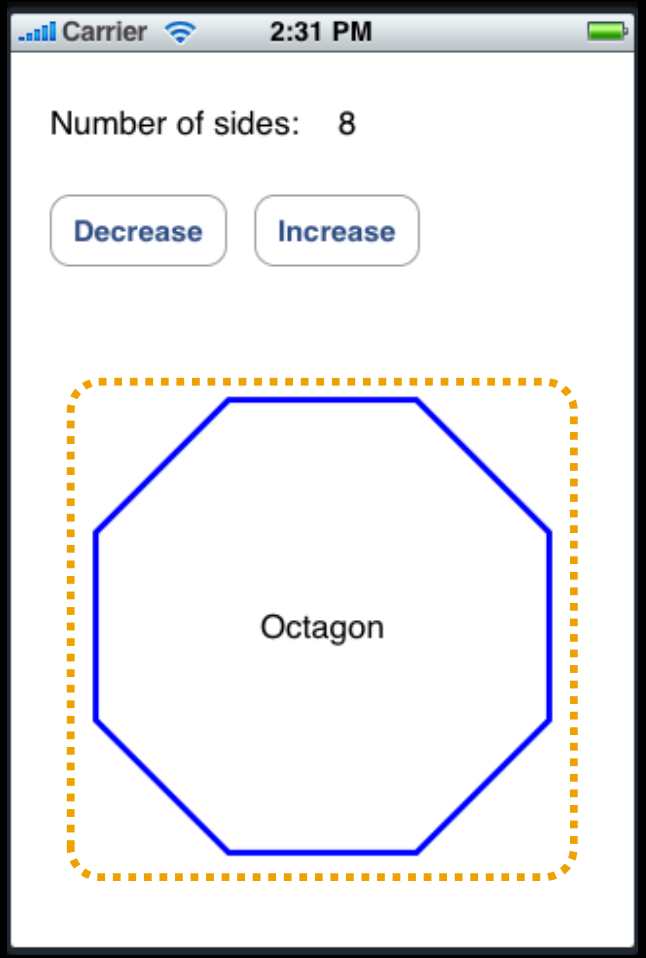
- How do I implement this?



Constructing Views

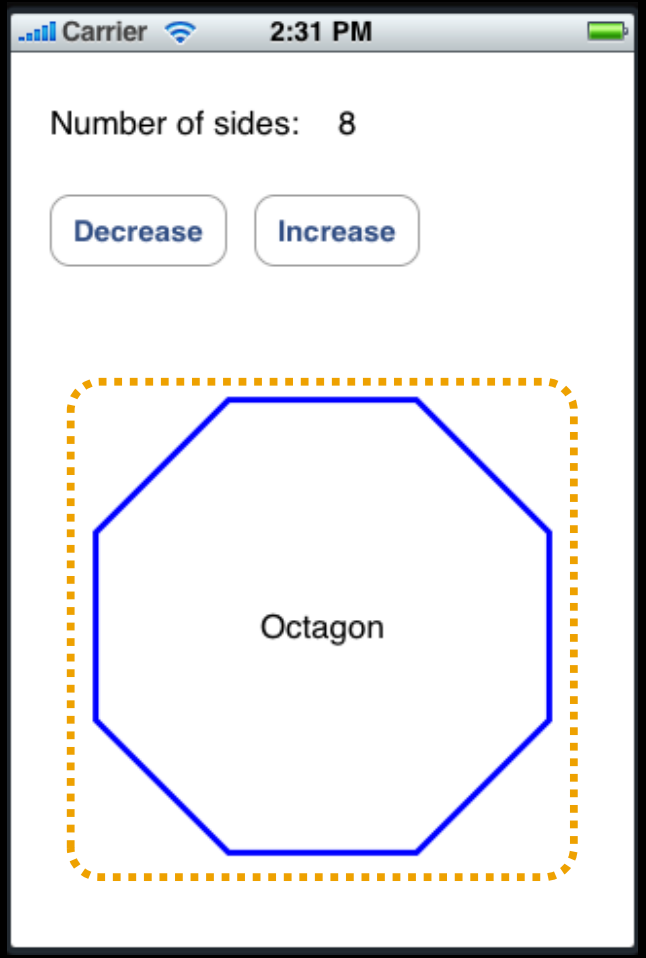
- How do I implement this?

- Goal
  - PolygonView that displays shape as well as name



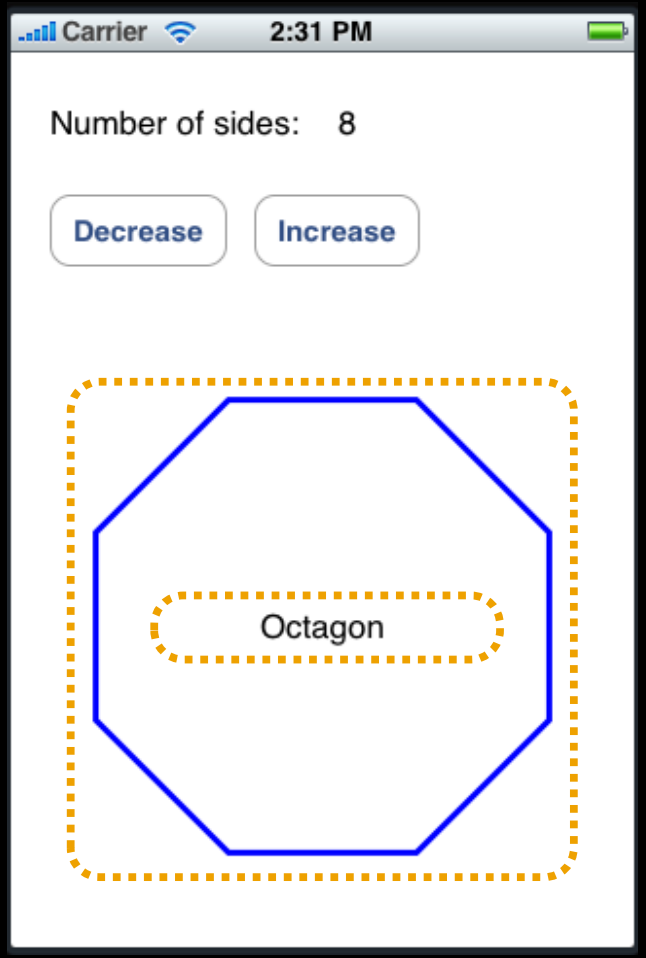
Constructing Views

- How do I implement this?
- Goal
  - PolygonView that displays shape as well as name
- Initial thought
  - Have PolygonView draw the text
  - Inefficient when animating



Constructing Views

- How do I implement this?
- Goal
  - PolygonView that displays shape as well as name
- Initial thought
  - Have PolygonView draw the text
  - Inefficient when animating
- Instead use UILabel!
  - Tastes great
  - Less filling



UILabel

- UIView subclass that knows how to draw text

- Properties include:
  - font
  - textColor
  - shadow (offset & color)
  - textAlignment

UIImageView

- UIView that draws UIImages

- Properties include:
  - image
  - animatedImages
  - animatedDuration
  - animatedRepeatCount

- contentMode property to align and scale image wrt bounds

UIControl

- UIView with Target-Action event handling

- Properties include:
  - enabled
  - selected
  - highlighted

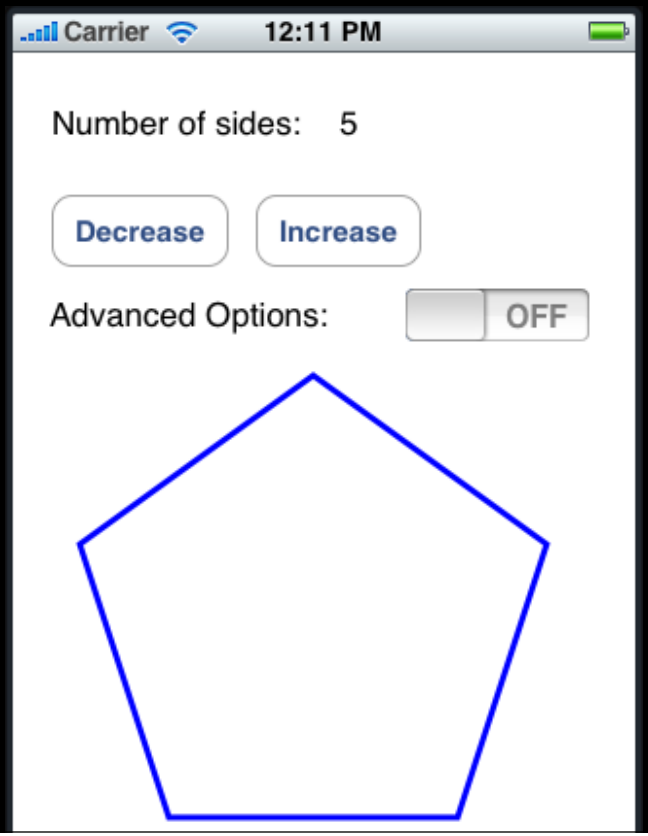
- UIButton: font, title, titleColor, image, backgroundImage
- UITextField: font, text, placeholder, textColor

- See UIKit headers for plenty more

View Properties & Animation

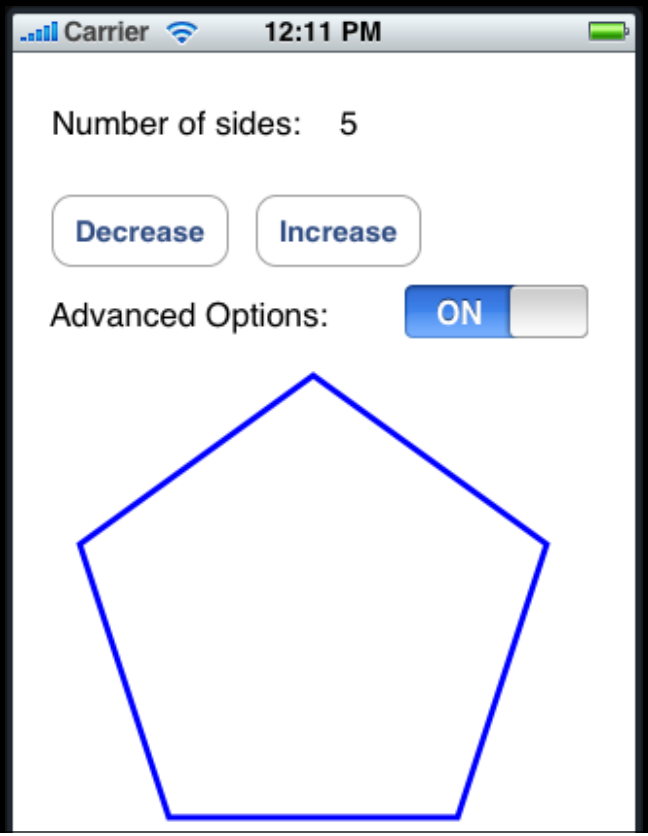
Animating Views

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...



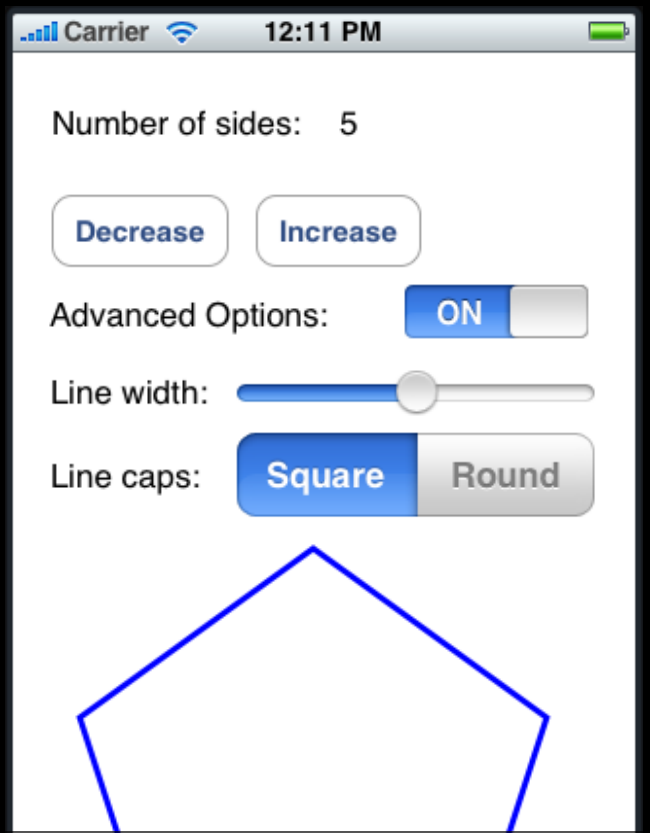
Animating Views

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...



Animating Views

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...



UIView Animations

- UIView supports a number of animatable properties
  - frame, bounds, center, alpha, transform
- Create "blocks" around changes to animatable properties
- Animations run asynchronously and automatically

Other Animation Options

- Additional animation options
  - delay before starting
  - start at specific time
  - curve (ease in/out, ease in, ease out, linear)
  - repeat count
  - autoreverses (e.g. ping pong back and forth)

View Animation Example

```
- (void)showAdvancedOptions {
    // assume polygonView and optionsView
    [UIView beginAnimations:@"advancedAnimations" context:nil];
    [UIView setAnimationDuration:0.3];


    // make optionsView visible (alpha is currently 0.0)
    optionsView.alpha = 1.0;

    // move the polygonView down
    CGRect polygonFrame = polygonView.frame;
    polygonFrame.origin.y += 200;
    polygonView.frame = polygonFrame;


    [UIView commitAnimations];

}
```

Knowing When Animations Finish

- UIView animations allow for a delegate

```
[UIView setAnimationDelegate:myController];
```

- myController will have callbacks invoked before and after

```
- (void)animationWillStart:(NSString *)animationID
                   context:(void *)context;

- (void)animationDidStop:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context;
```

- Can provide custom selectors if desired, for example

```
[UIView setAnimationWillStartSelector:
                    @selector(animationWillStart)];
[UIView setAnimationDidStopSelector:
                    @selector(animationDidStop)];
```

How Does It Work?

- Is drawRect: invoked repeatedly?
- Do I have to run some kind of timer in order to drive the animation?
- Is it magic?

Core Animation

- Hardware accelerated rendering engine
- UIViews are backed by "layers"
- -drawRect: results are cached
  - Cached results used to render view
  - -drawRect: called only when contents change
  - Layers drawn from a separate render tree managed by separate process
- Property animations done automatically by manipulating layers

View Transforms

- Every view has a transform property
  - used to apply scaling, rotation and translation to a view
- Default "Identity transform"
- CGAffineTransform structure used to represent transform
- Use CG functions to create, modify transforms

CGAffineTransform Functions (just a small example set)
CGAffineTransformScale (transform, xScale, yScale)
CGAffineTransformRotate (transform, angle)
CGAffineTransformTranslate (transform, xDelta, yDelta)

More Animation Information

- *iPhone OS Programming Guide*
  - "Modifying Views at Runtime" section
- *Core Animation Programming Guide*

Assignment 3 Hints

Saving State Across App Launches

- NSUserDefaults to read and write prefs & state

- Singleton object:

```
+ (NSUserDefaults *)standardUserDefaults;
```

- Methods for storing & fetching common types:

```
- (int)integerForKey:(NSString *)key;
- (void)setInteger:(int)value forKey:(NSString *)key;

- (id)objectForKey:(NSString *)key;
- (void)setObject:(id)value forKey:(NSString *)key;
```

- Find an appropriate time to store and restore your state

Questions?