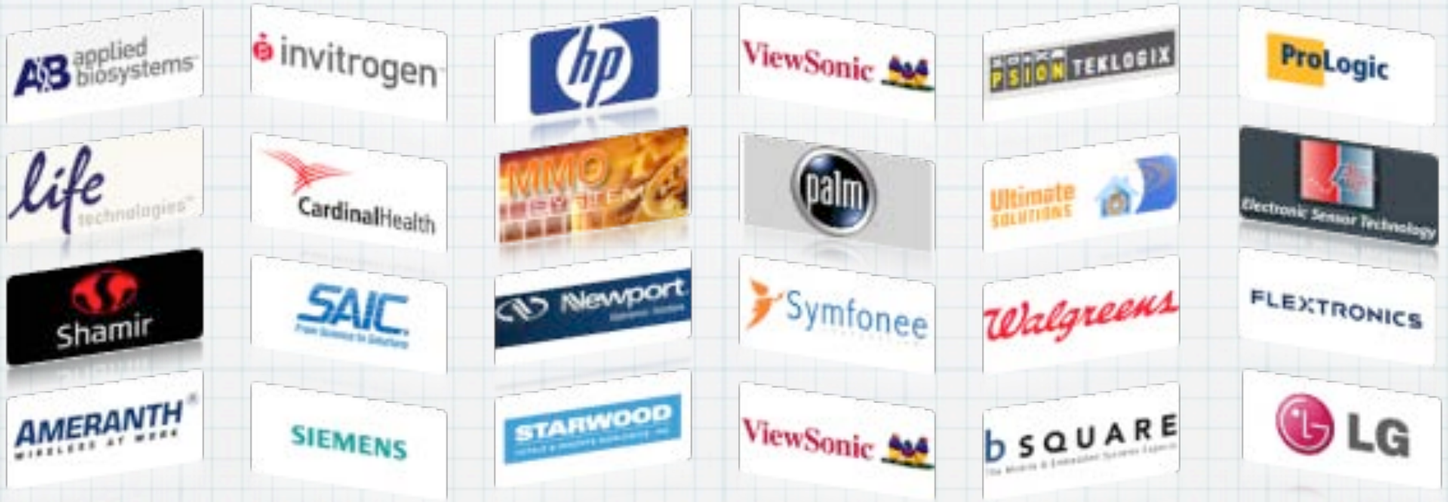
Mark Pospesel
Senior Lead Developer
Odyssey Computing, Inc.

Who am I?

* Senior Lead Developer with Odyssey Computing, Inc.
* 14 years industry experience
* In-house development & project consulting
* Programming for mobile since 1999!
* C++ → C# → Obj-C
* Windows → iOS

odyssey COMPUTING

Some of our clients



odyssey COMPUTING

Course Materials

available for download

* Slides: www.odysseyinc.com/downloads/EnterTheMatrix.pdf
* Code: github.com/mpospese/EnterTheMatrix

Outline

* Intro to matrices
* API's
* Quartz 2D
* CGAffineTransform
* CATransform3D

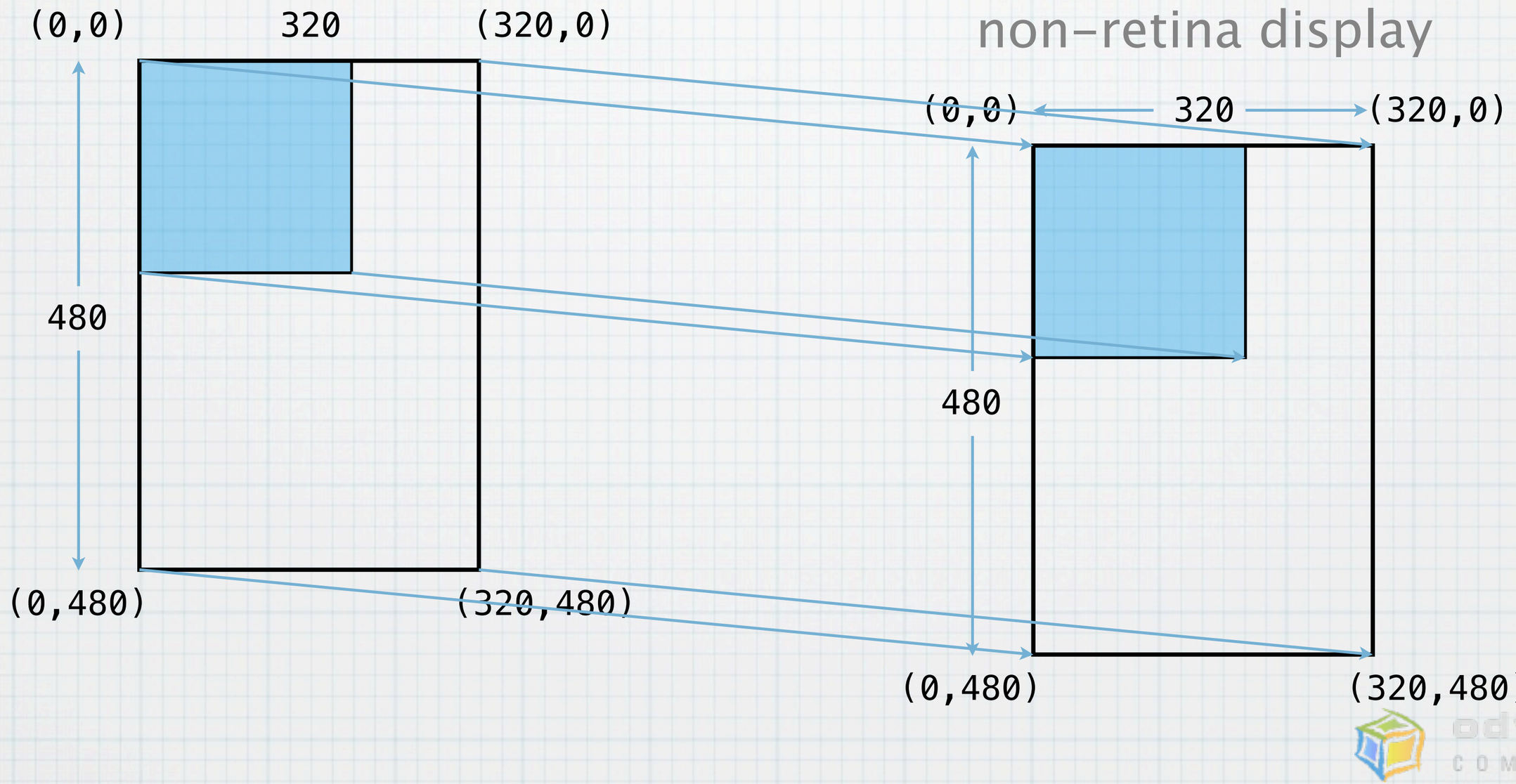
Outline

* Intro to matrices
* API's
* Animations
  * Basic
  * Keyframe
  * Fold
  * Flip

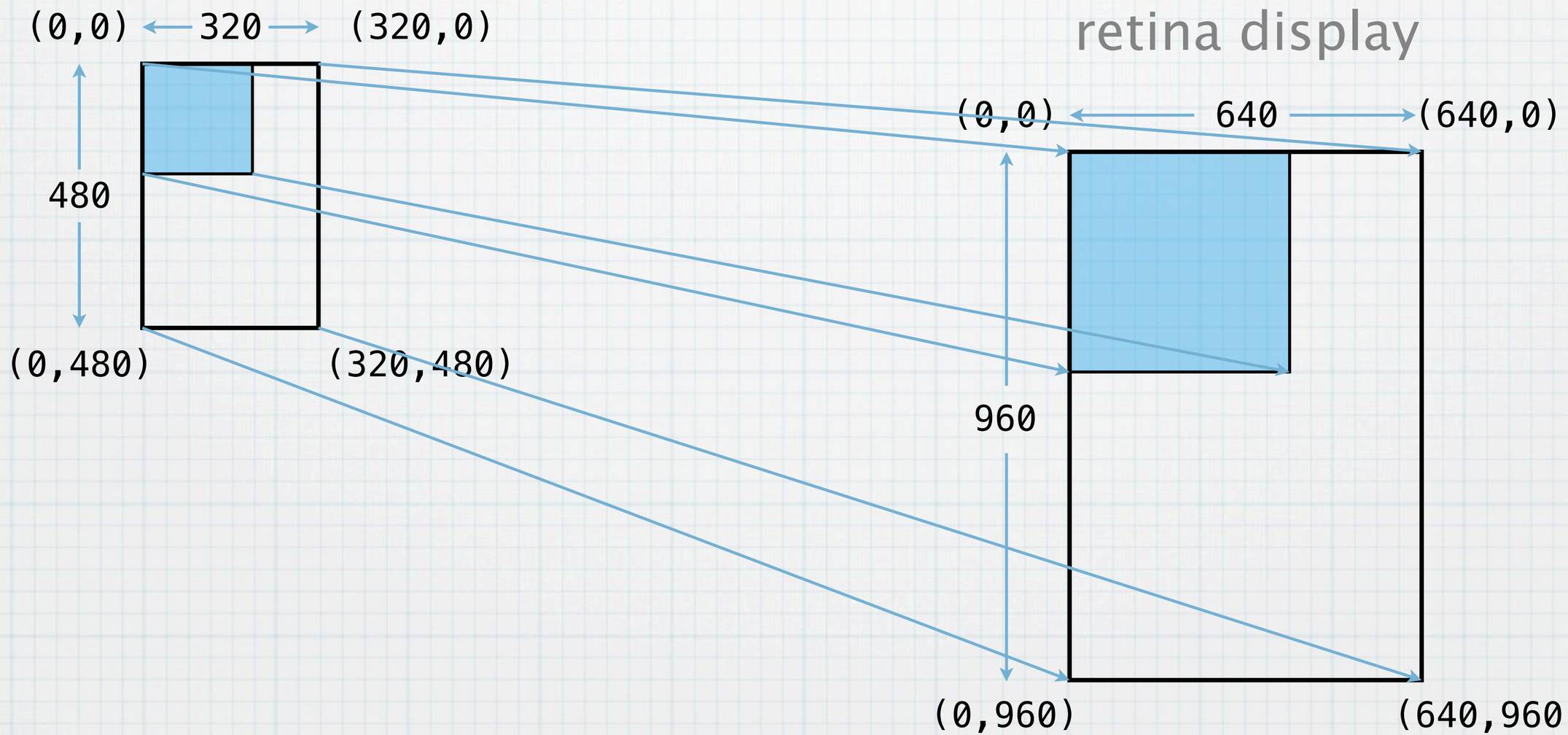
User space vs. Device space

* User space = your view (points)
* Device space = hardware device native resolution (pixels)
* On print or display, Quartz maps user space coordinates to device space coordinates

User space vs. Device space



User space vs. Device space



What is the matrix?

* Control. The Matrix is a computer-generated dream world built to keep us under control in order to change a human being into this. [holds up a Duracell battery]

What is a matrix?

* A way to transform user space
  * Translate – move left/right, up/down
  * Scale – make bigger/smaller
  * Rotate – spin

What are they good for?

* Pretty much all operations in OpenGL
* Creating custom views
* Shortcuts for efficient Quartz drawing
* Animations – UIView.transform is an animatable property

Matrix math basics

* Identity matrix – similar to 1

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Matrix math basics

* Identity matrix – similar to 1

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Matrix math basics

* To perform multiple operations matrices are multiplied to each other (concatenation)
* All the operations you perform on a single object are concatenated into a single matrix

$$\begin{bmatrix} M_1 \end{bmatrix} \times \begin{bmatrix} M_2 \end{bmatrix} \times \begin{bmatrix} M_3 \end{bmatrix} \times \begin{bmatrix} M_4 \end{bmatrix} = \begin{bmatrix} M_n \end{bmatrix}$$

Matrix math basics

* Order of operation matters! (not commutative)

$$M_1 \times M_2 \neq M_2 \times M_1$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \times \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} = \begin{bmatrix} \text{something} \end{bmatrix}$$

Matrix math basics

* Order of operation matters! (not commutative)

$$M_1 \times M_2 \neq M_2 \times M_1$$

$$\begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} \times \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} \text{some other thing} \end{bmatrix}$$

Matrix math basics

* Inverted matrix – similar to 1/x

$$M \times M_{inv} = M_{ident}$$

$$\begin{bmatrix} M \end{bmatrix} \times \begin{bmatrix} M_{inv} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix math basics

* Inverted matrix – similar to 1/x

$$M \times M_{inv} = M_{ident}$$

$$\begin{bmatrix} M_{inv} \end{bmatrix} \times \begin{bmatrix} M \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix math basics

* Inverted matrix – similar to 1/x

$$M \times M_{inv} = M_{ident}$$

* Useful for reverting transformations on points, rects, etc.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} M \end{bmatrix} = \begin{bmatrix} x^1 \\ y^1 \\ 1 \end{bmatrix} \times \begin{bmatrix} M_{inv} \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Matrix math basics

* Transforming points: 2D

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} ax + by + t_x \\ cx + dy + t_y \\ 1 \end{bmatrix}$$

Matrix math basics

* Transforming points: Identity

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1x + 0y + 0z \\ 0x + 1y + 0z \\ 1 \end{bmatrix}$$

Matrix math basics

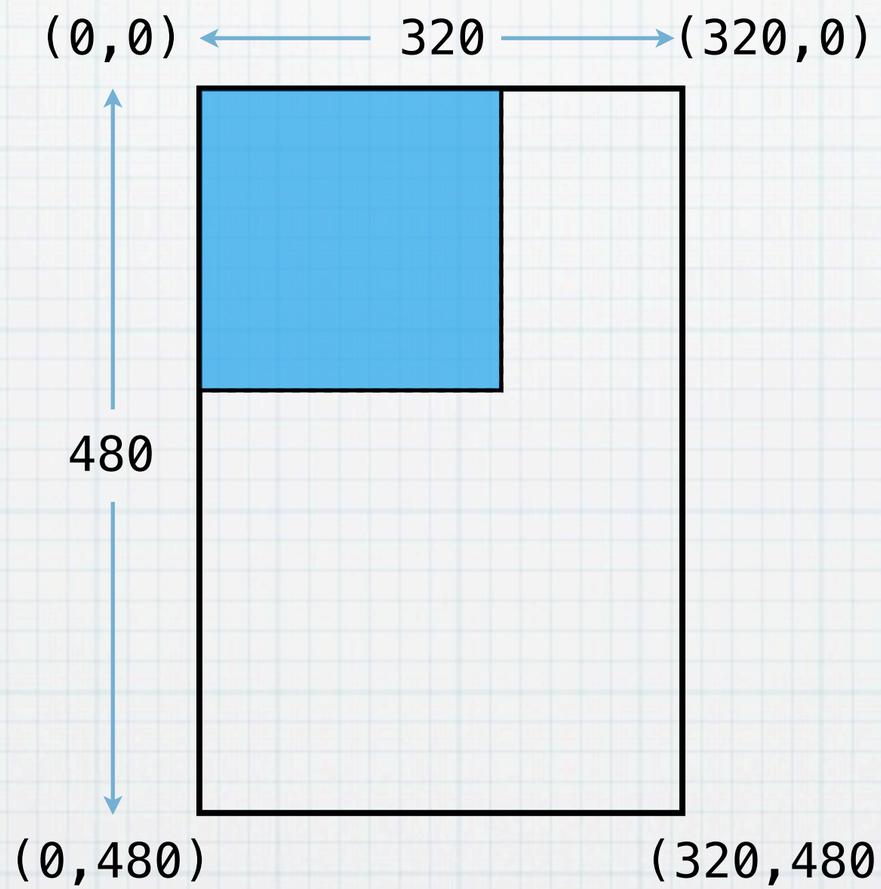
* Transforming points: Identity

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

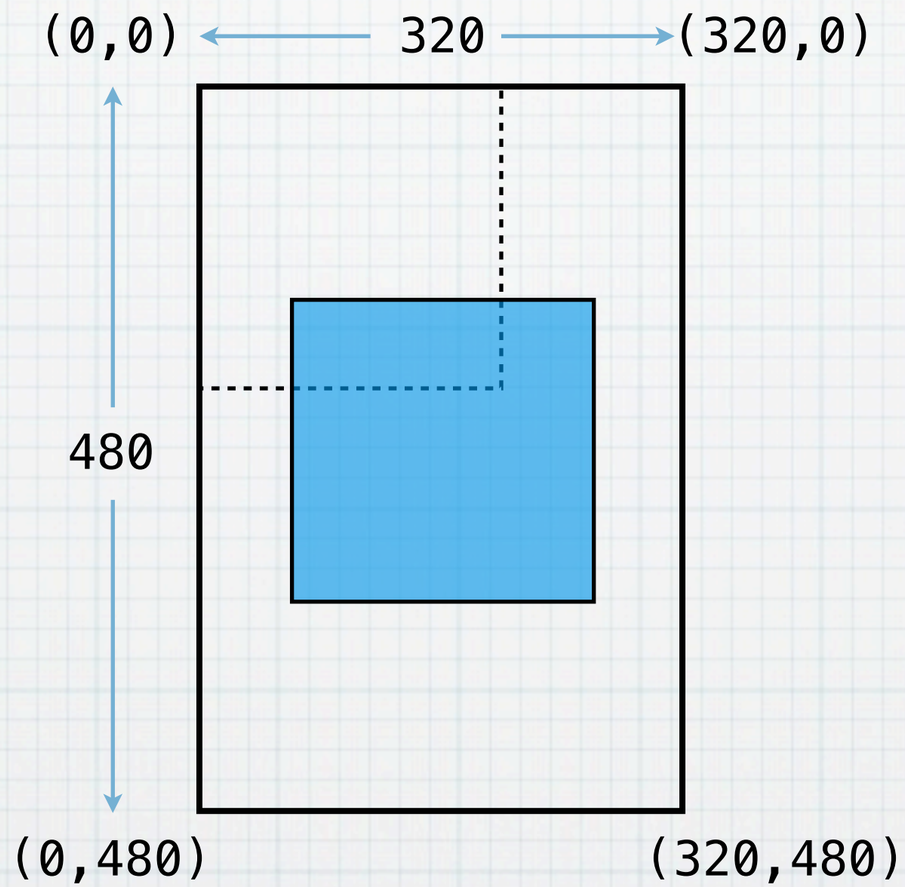
Basic operations

* Translation
* Scale
* Rotation

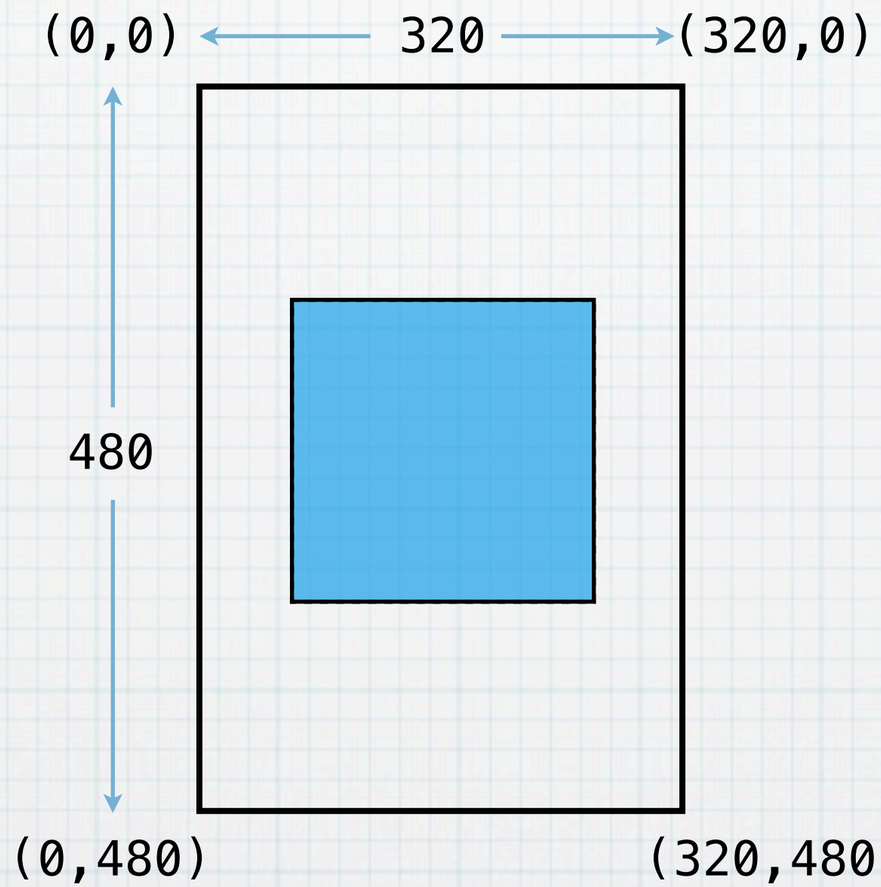
Translation



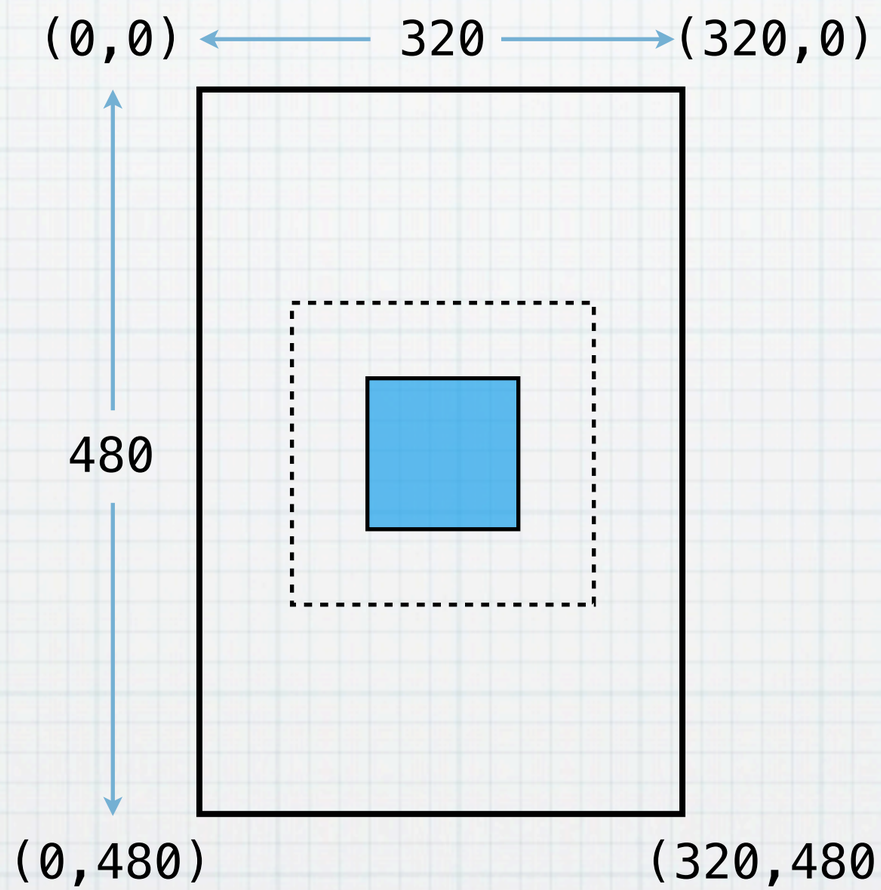
Translation



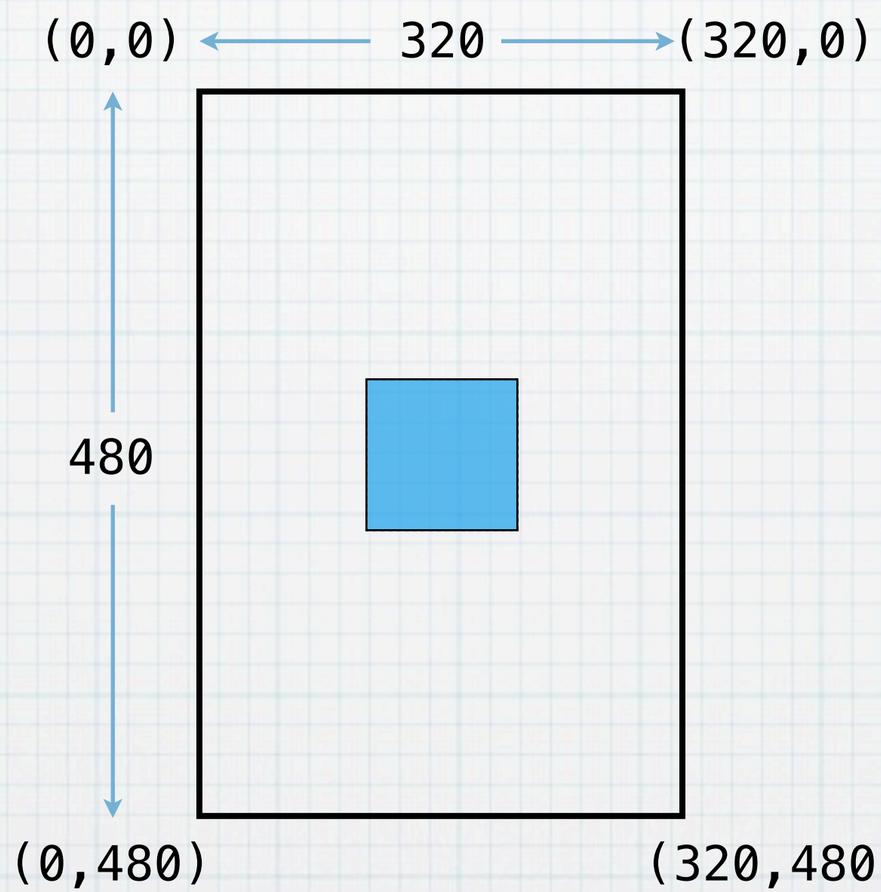
Scale



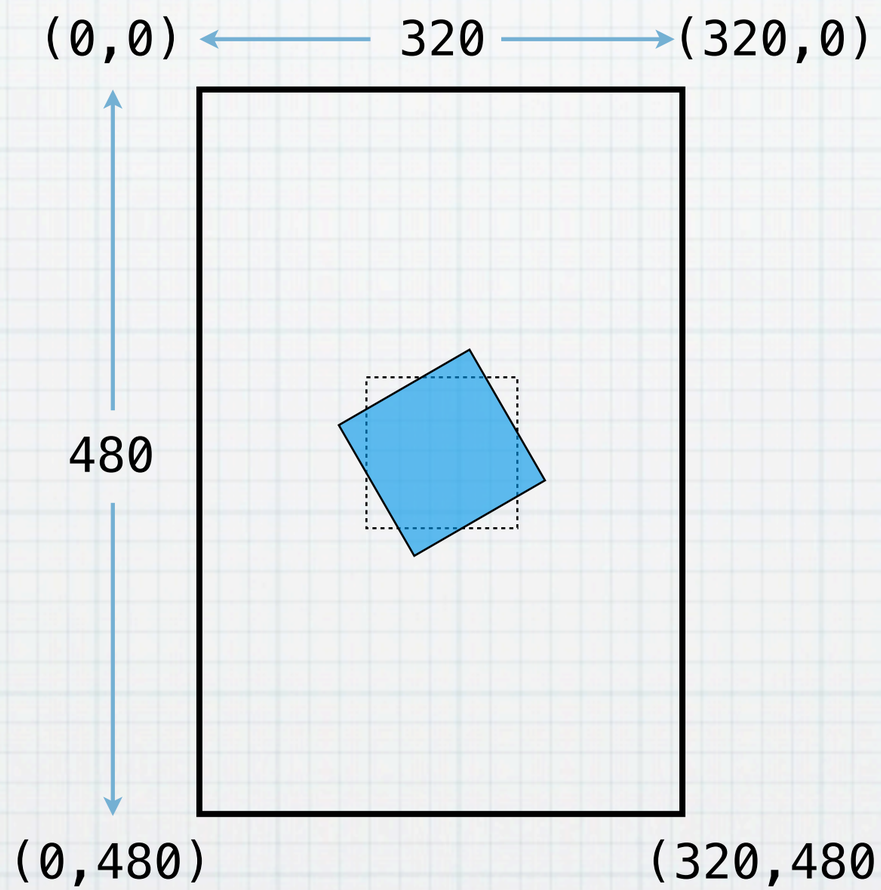
Scale



Rotation



Rotation



Quartz 2D

Quartz 2D

* CGContextConcatCTM
* CGContextGetCTM
* CGContextTranslateCTM
* CGContextScaleCTM
* CGContextRotateCTM

CTM = Current Transformation Matrix

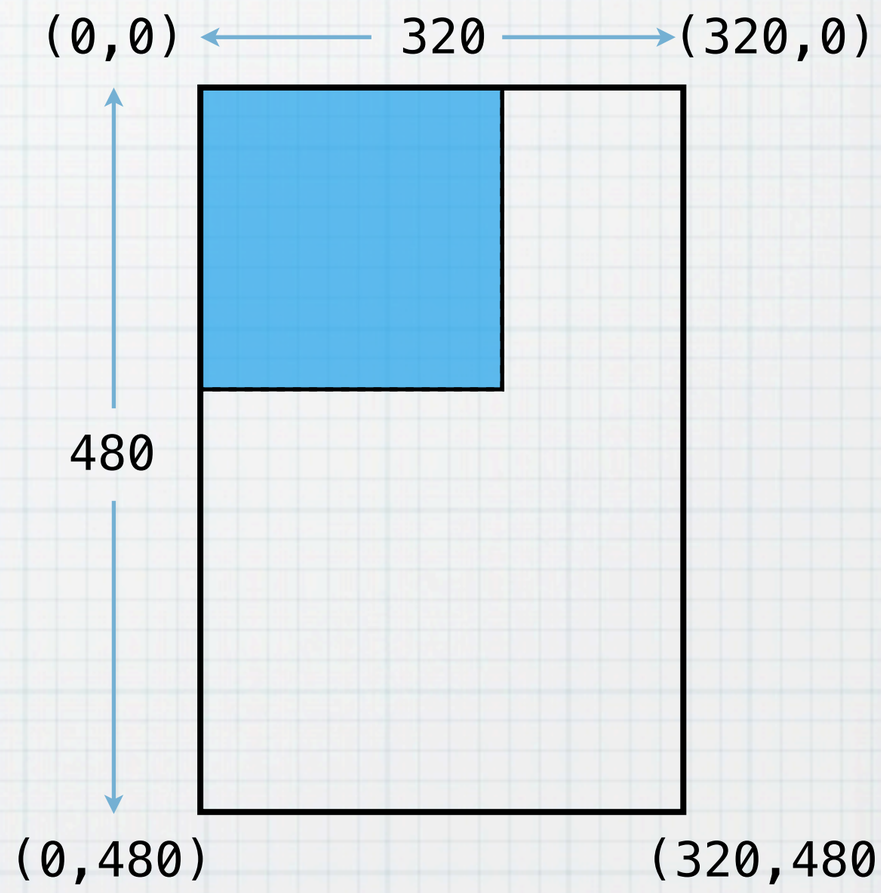
Quartz 2D

- Concat — applies transform to CTM
- Get — gets the CTM
- Translate
- Scale
- Rotate

CGContextTranslateCTM

Drawing - drawRect:

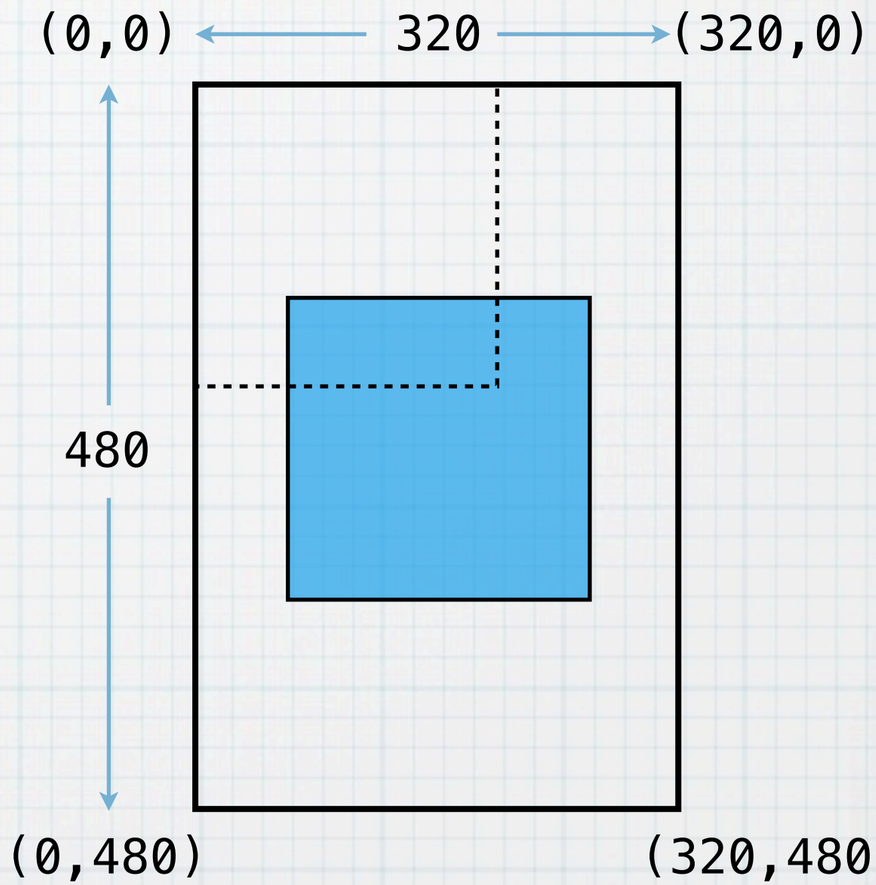
```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
   CGContextRef context = UIGraphicsGetCurrentContext();
   [[UIColor blackColor] setStroke];
   [[UIColor blueColor] setFill];
   UIBezierPath *path = [UIBezierPath bezierPathWithRect:
      (  CGRect){{0,0}, {200,200}}];
   [path fill];
   [path stroke];
}
```



CGContextTranslateCTM

Drawing - drawRect:

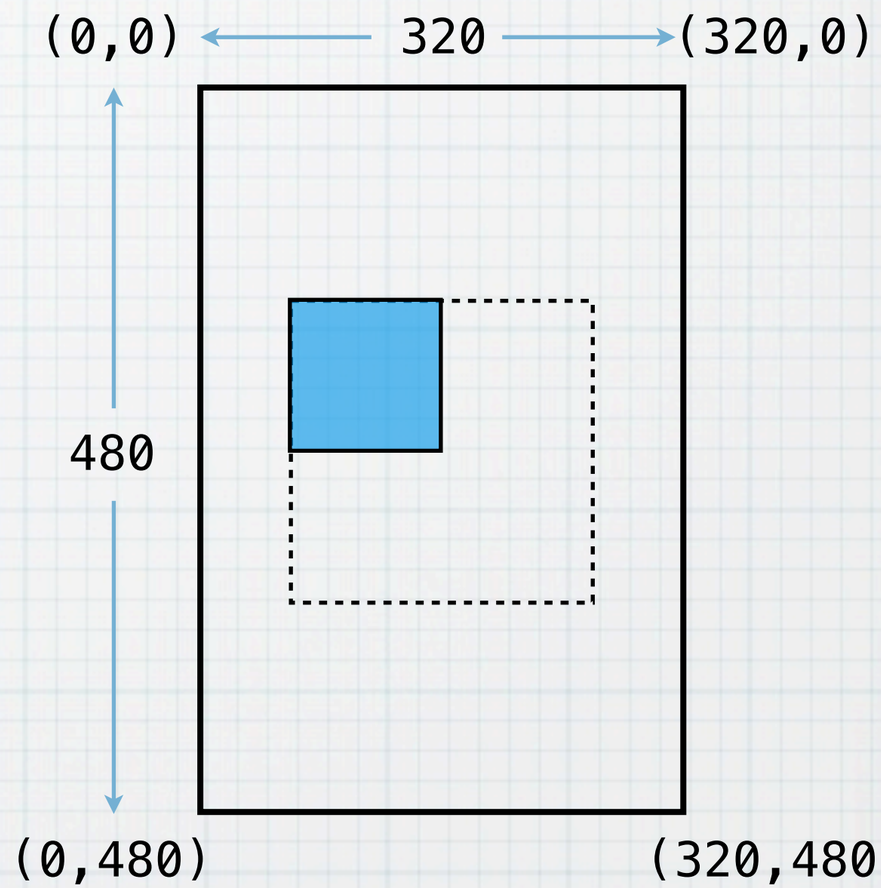
```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
   CGContextRef context = UIGraphicsGetCurrentContext();
   CGContextTranslateCTM(context, 60, 140);  // Translate
   [[UIColor blackColor] setStroke];
   [[UIColor blueColor] setFill];
   UIBezierPath *path = [UIBezierPath bezierPathWithRect:
      (  CGRect){{0,0}, {200,200}}];
   [path fill];
   [path stroke];
}
```



CGContextScaleCTM

Drawing - drawRect:

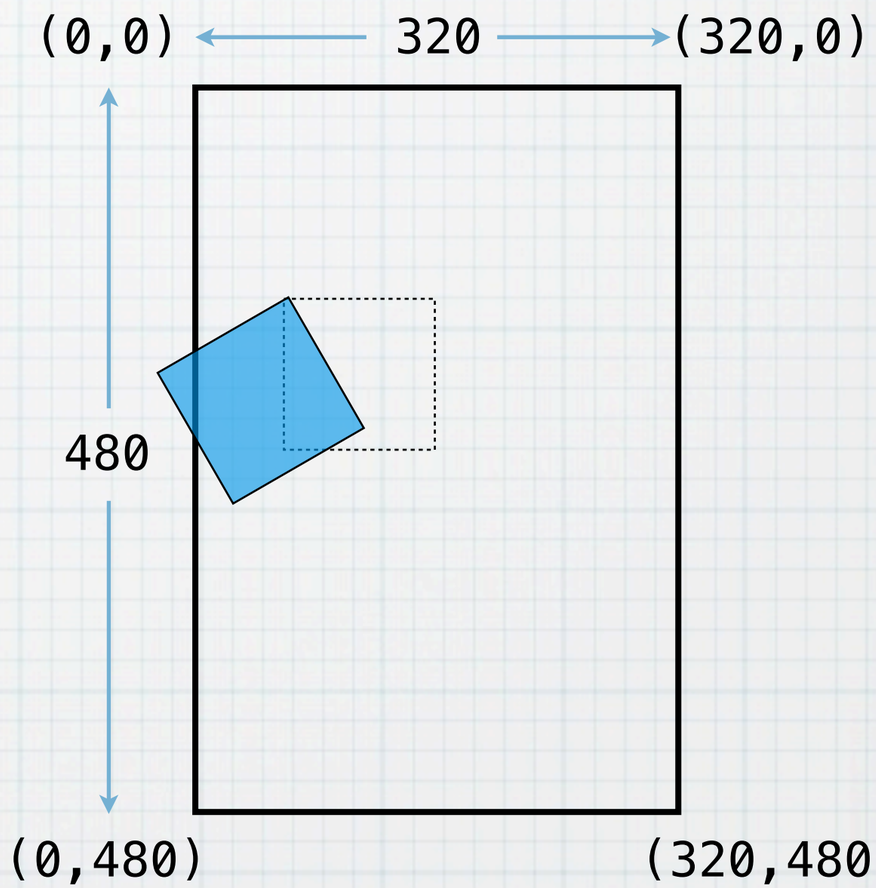
```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
   CGContextRef context = UIGraphicsGetCurrentContext();
   CGContextTranslateCTM(context, 60, 140);  // Translate
   CGContextScaleCTM(context, 0.5, 0.5);     // Scale
   [[UIColor blackColor] setStroke];
   [[UIColor blueColor] setFill];
   UIBezierPath *path = [UIBezierPath bezierPathWithRect:
      (  CGRect){{0,0}, {200,200}}];
   [path fill];
   [path stroke];
}
```



CGContextRotateCTM

Drawing - drawRect:

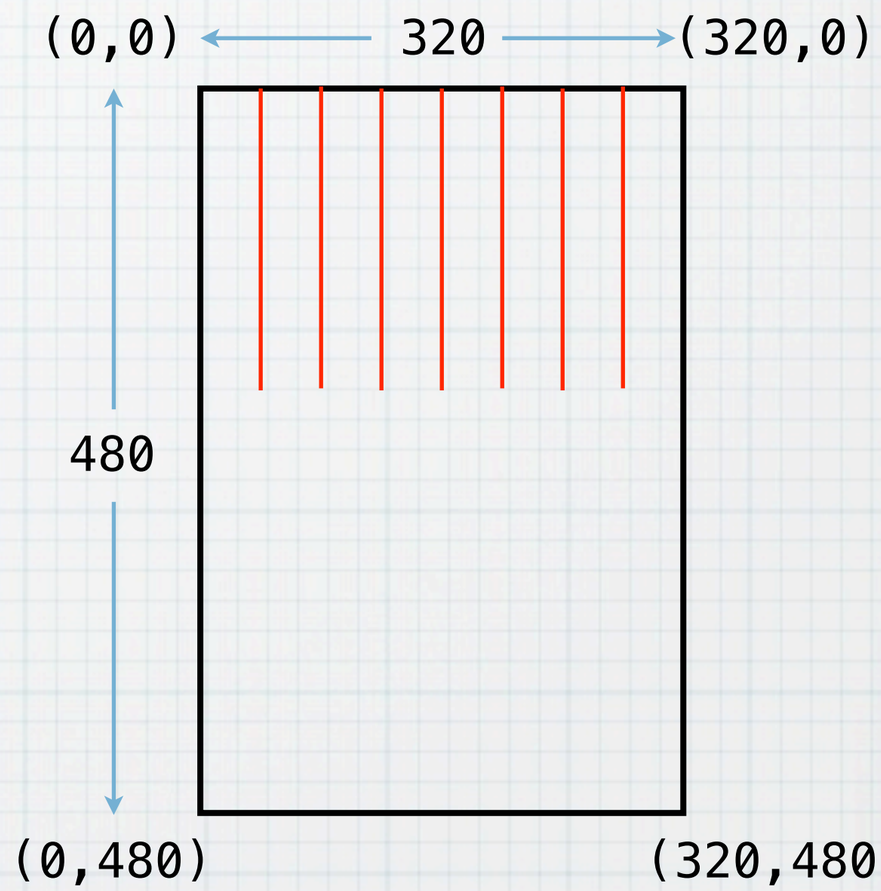
```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
   CGContextRef context = UIGraphicsGetCurrentContext();
   CGContextTranslateCTM(context, 60, 140);  // Translate
   CGContextScaleCTM(context, 0.5, 0.5);     // Scale
   CGContextRotateCTM(context, radians(60)); // Rotate
   [[UIColor blackColor] setStroke];
   [[UIColor blueColor] setFill];
   UIBezierPath *path = [UIBezierPath bezierPathWithRect:
      (  CGRect){{0,0}, {200,200}}];
   [path fill];
   [path stroke];
}
```



Quartz 2D drawing

A way to draw some lines

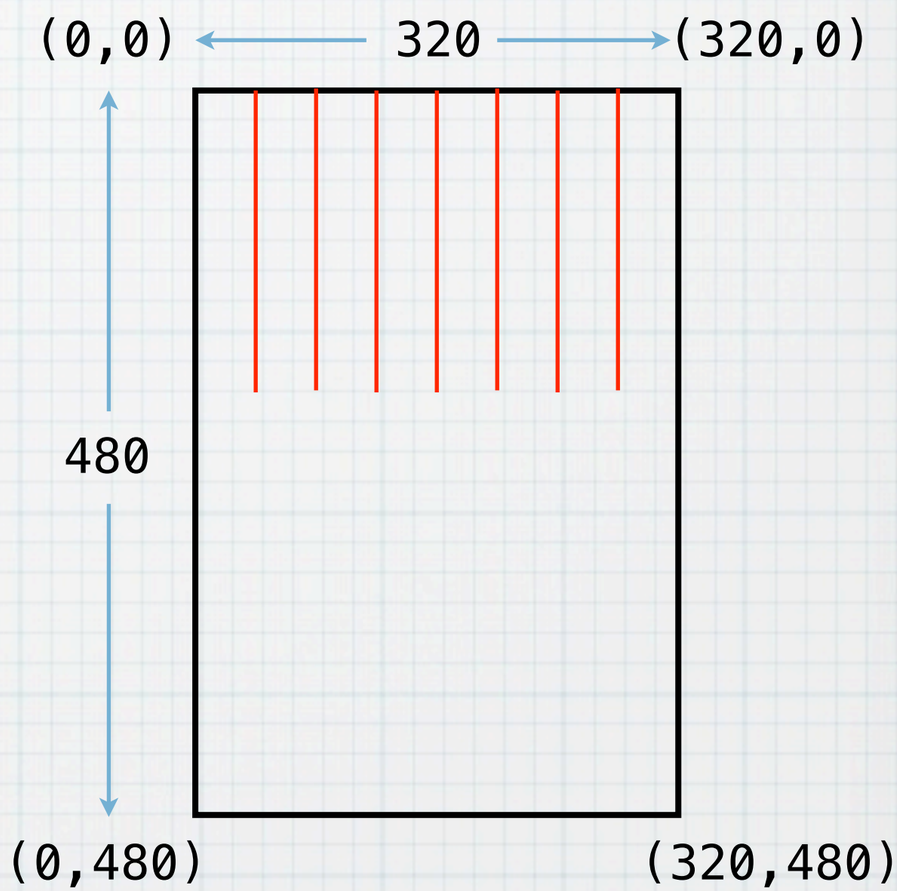
```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 2);
    [[UIColor redColor] setStroke];

    for (int i = 1; i <= 7; i++)
    {
       CGContextMoveToPoint(context, 40 * i, 0);
       CGContextAddLineToPoint(context, 40 * i, 200);
    }

    CGContextStrokePath(context);
}
```



Quartz 2D drawing

A better way to draw some lines

```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
    CGContextRef context = UIGraphicsGetCurrentContext();
    [[UIColor redColor] setStroke];

    UIBezierPath *path = [UIBezierPath bezierPath];
    [path moveToPoint:(CGPoint){0, 0}];
    [path addLineToPoint:(CGPoint){0, 200}];
    [path setLineWidth:2];

    CGContextSaveGState(context);

    for (int i = 1; i <= 7; i++)
    {
       CGContextTranslateCTM(context, 40, 0);
       [path stroke];
    }

    CGContextRestoreGState(context);
}
```



DEMO

CGContext CTM methods

CGAffineTransform

CGAffineTransform

* 2D
* UIView transform property (animatable)
* CALayer affineTransform & setAffineTransform convenience methods (indirectly animatable)

CGAffineTransform

```
struct CGAffineTransform {
  CGFloat a, b, c, d;
  CGFloat tx, ty;
};
```

CGAffineTransform
Identity

* CGAffineTransformIdentity

```
[view setTransform:CGAffineTransformIdentity];
```

CGAffineTransform
Creating Transforms

* CGAffineTransformMake
* CGAffineTransformMakeTranslation
* CGAffineTransformMakeScale
* CGAffineTransformMakeRotation

CGAffineTransform

Modifying Transforms

* CGAffineTransformTranslate
* CGAffineTransformScale
* CGAffineTransformRotate
* CGAffineTransformInvert
* CGAffineTransformConcat

CGAffineTransform

Creating vs. Modifying

Create

CGAffineTransformMake

CGAffineTransformMakeTranslation

CGAffineTransformMakeScale

CGAffineTransformMakeRotation

Modify

CGAffineTransformConcat

CGAffineTransformTranslate

CGAffineTransformScale

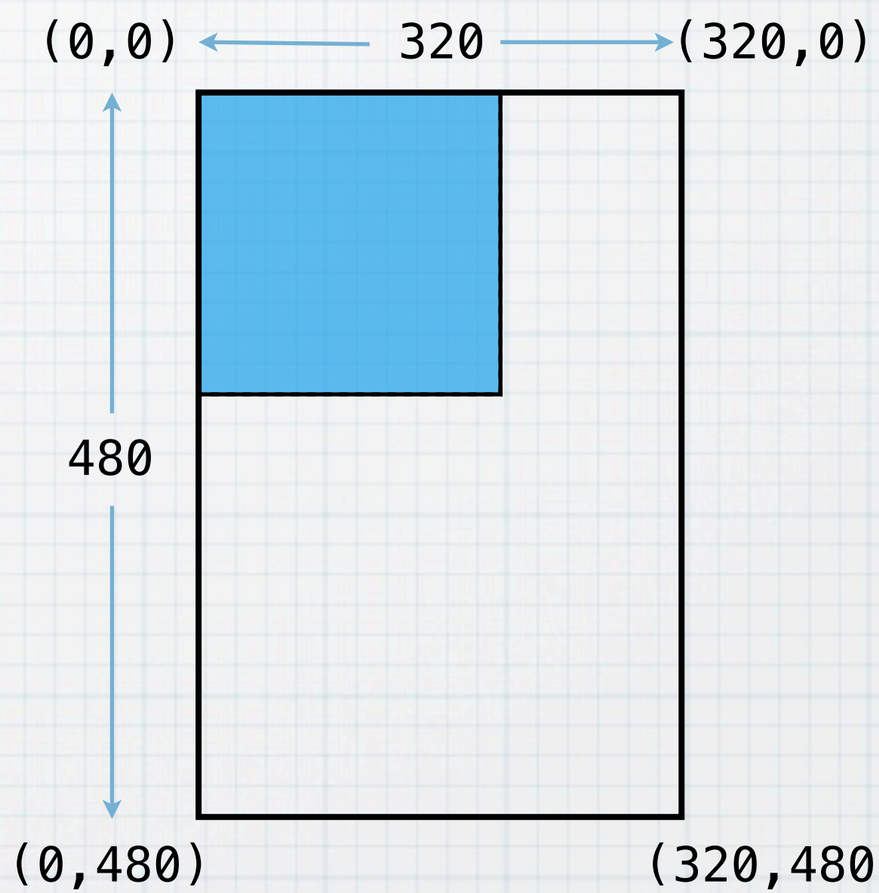
CGAffineTransformRotate

CGAffineTransformInvert

Translation

```
UIView *square = [[UIView alloc] initWithFrame:
    (CGRect){{0,0}, {200,200}}];
[square setBackgroundColor:[UIColor blueColor]];
[self.view addSubview:square];

// Translate
CGAffineTransform transform =
    CGAffineTransformMakeTranslation(60, 140);

[square setTransform:transform];
```

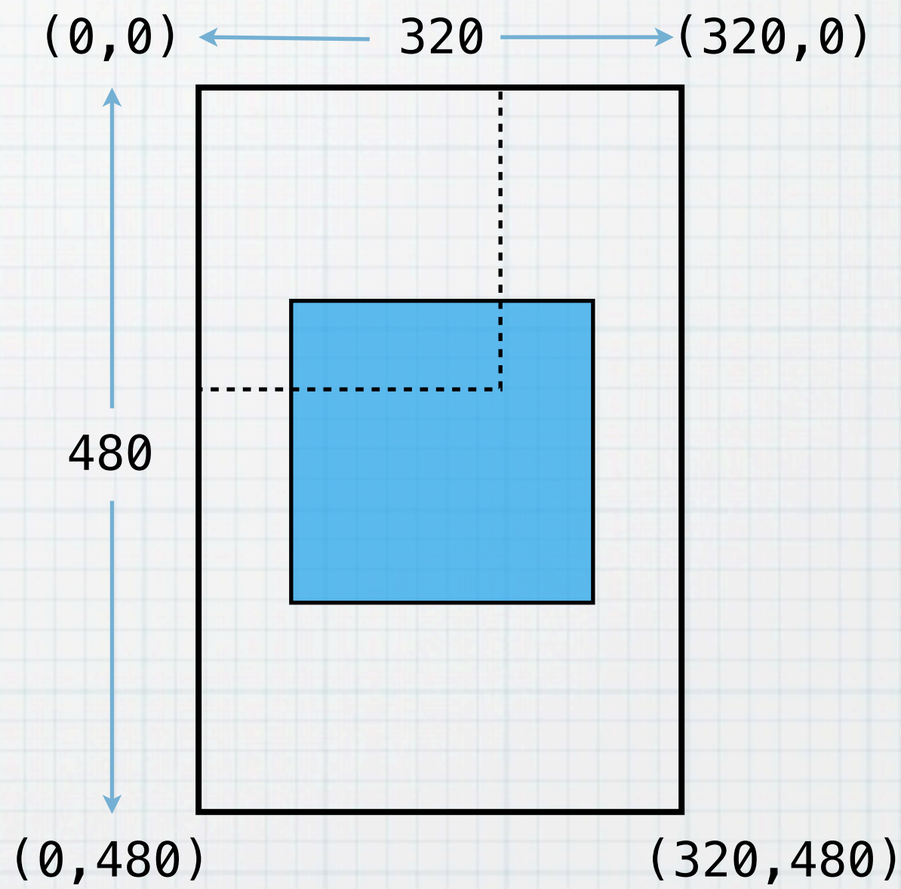
```
[square frame]  {{0,0}, {200,200}}
[square bounds] {{0,0}, {200,200}}
[square center] {100,100}
```



Translation

```
UIView *square = [[UIView alloc] initWithFrame:
    (CGRect){{0,0}, {200,200}}];
[square setBackgroundColor:[UIColor blueColor]];
[self.view addSubview:square];

// Translate
CGAffineTransform transform =
    CGAffineTransformMakeTranslation(60, 140);

[square setTransform:transform];
```

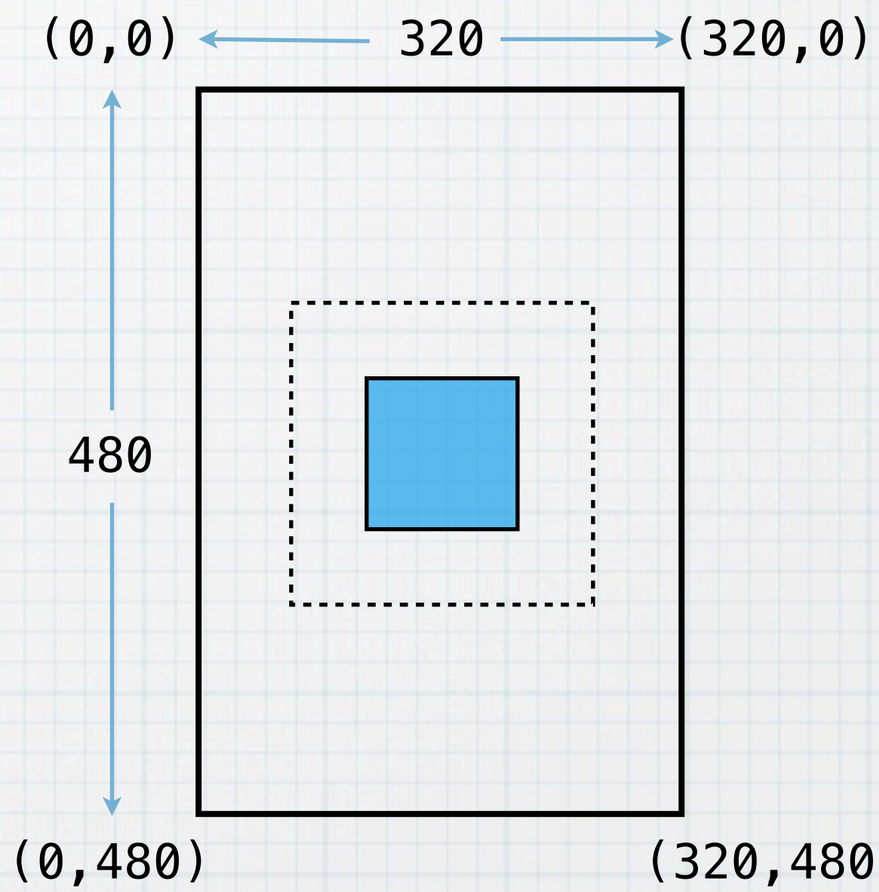
```
[square frame]  {{60,140}, {200,200}}
[square bounds] {{0,0}, {200,200}}
[square center] {100,100}
```



Scale

```
UIView *square = [[UIView alloc] initWithFrame:
    (CGRect){{0,0}, {200,200}}];
[square setBackgroundColor:[UIColor blueColor]];
[self.view addSubview:square];

// Translate
CGAffineTransform transform =
    CGAffineTransformMakeTranslation(60, 140);
// Scale
transform = CGAffineTransformScale(transform, 0.5, 0.5);

[square setTransform:transform];
```

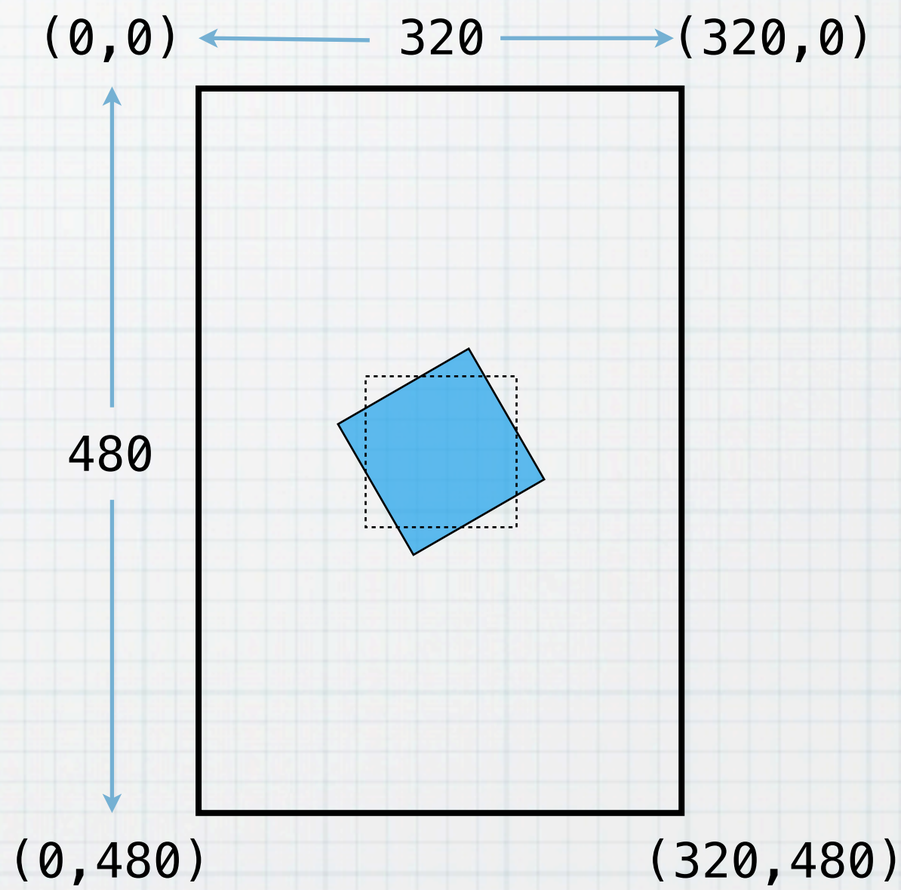
```
[square frame]  {{110,190}, {100,100}}
[square bounds] {{0,0}, {200,200}}
[square center] {100,100}
```



Rotation

```
UIView *square = [[UIView alloc] initWithFrame:
    (CGRect){{0,0}, {200,200}}];
[square setBackgroundColor:[UIColor blueColor]];
[self.view addSubview:square];

// Translate
CGAffineTransform transform =
    CGAffineTransformMakeTranslation(60, 140);
// Scale
transform = CGAffineTransformScale(transform, 0.5, 0.5);
// Rotate
transform = CGAffineTransformRotate(transform, radians(60));

[square setTransform:transform];
```

```
[square frame]  {{92,172}, {137,137}}
[square bounds] {{0,0}, {200,200}}
[square center] {100,100}
```



CGAffineTransform

Applying Transforms

* CGPointApplyAffineTransform
* CGSizeApplyAffineTransform
* CGRectApplyAffineTransform

CGAffineTransform
Evaluating Transforms

* CGAffineTransformIsIdentity
* CGAffineTransformEqualToTransform

DEMO

CGAffineTransform

CATransform3D

CATransform3D

* 3D
* CALayer transform property (animatable)
* can convert to CGAffineTransform if 2D (z =0)

CATransform3D

```
/* Homogeneous three-dimensional transforms. */

struct CATransform3D
{
  CGFloat m11, m12, m13, m14;
  CGFloat m21, m22, m23, m24;
  CGFloat m31, m32, m33, m34;
  CGFloat m41, m42, m43, m44;
};
```

CATransform3D

* Transforming 3D points

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \times \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{21}y + m_{31}z + m_{41} \\ m_{12}x + m_{22}y + m_{32}z + m_{42} \\ m_{13}x + m_{23}y + m_{33}z + m_{43} \\ m_{14}x + m_{24}y + m_{34}z + m_{44} \end{bmatrix}$$

CATransform3D

* m34 Skew – creates 3D perspective

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \times \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{21}y + m_{31}z + m_{41} \\ m_{12}x + m_{22}y + m_{32}z + m_{42} \\ m_{13}x + m_{23}y + m_{33}z + m_{43} \\ m_{14}x + m_{24}y + m_{34}z + m_{44} \end{bmatrix}$$

CATransform3D
Identity

* CATransform3DIdentity

```
[layer setTransform:CATransform3DIdentity];
```

CATransform3D
Creating Transforms

* CATransform3DMake
* CATransform3DMakeTranslation
* CATransform3DMakeScale
* CATransform3DMakeRotatation

CATransform3D
Modifying Transforms

* CATransform3DTranslate
* CATransform3DScale
* CATransform3DRotate
* CATransform3DInvert
* CATransform3DConcat

CATransform3D

Creating vs. Modifying

Create

CATransform3DMake

CATransform3DMakeTranslation

CATransform3DMakeScale

CATransform3DMakeRotation

Modify

CATransform3DConcat

CATransform3DTranslate

CATransform3DScale

CATransform3DRotate

CATransform3DInvert

CATransform3D

Converting to/from Affine

* CATransform3DGetAffineTransform
* CATransform3DMakeAffineTransform

CATransform3D

Evaluating Transforms

* CATransform3DIsIdentity
* CATransform3DEqualToTransform
* CATransform3DIsAffine

DEMO

CATransform3D

Basic Animations

Basic Animations

1. Create a CGAffineTransform
2. call setTransform on UIView within an animation block
3. Done!

Basic Animations

```
- (void)rotateView:(UIView *)view byAngle:(CGFloat)angle
{
    [UIView animateWithDuration:0.5
                          delay:0
                        options:UIViewAnimationCurveEaseInOut
                     animations:^{
                         CGAffineTransform transform = [view transform];
                         transform = CGAffineTransformRotate(transform, angle);
                         [view setTransform:transform];
                     }
                     completion:nil
    ];
}
```

DEMO

Basic Animations

Keyframe Animations

Keyframe Animations

Q: When do you need them?

A: When you want to animate a CGPathRef

A: When the built-in timing curves don't suit your needs

Keyframe Animations

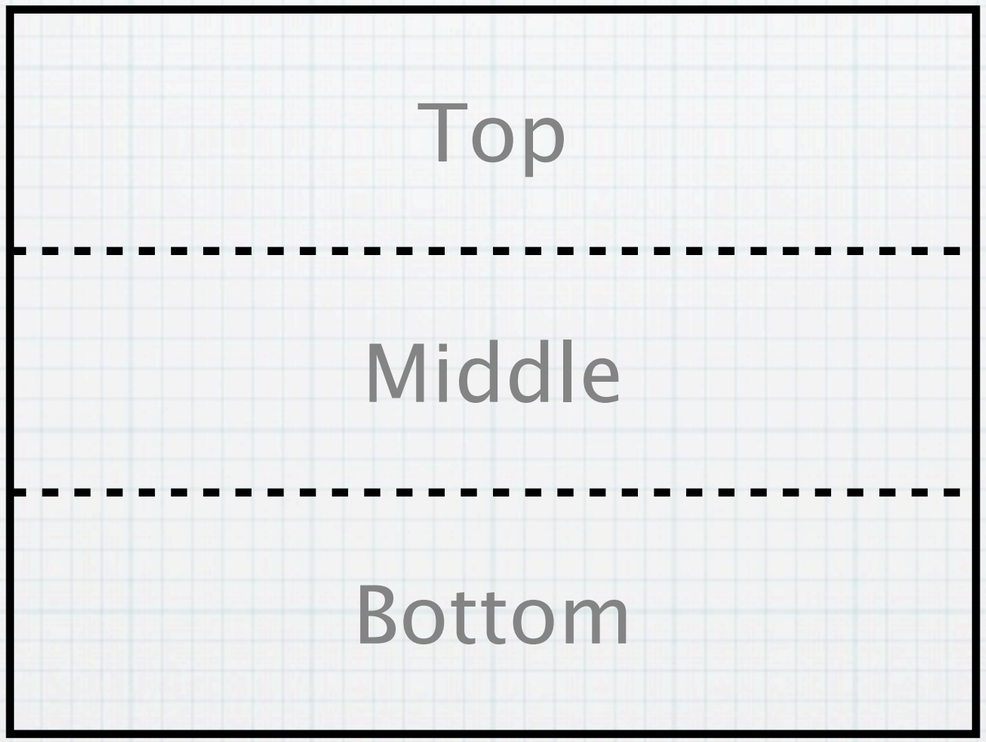
```objc
// Create the animation (we're animating transform property)
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:@"transform"];

// set our keyframe values
[animation setValues:[NSArray arrayWithArray:array]];
[animation setDuration:duration];
[animation setTimingFunction:
    [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionDefault]];
[animation setRemovedOnCompletion:YES];

// add the animation
[self.arc.layer addAnimation:animation forKey:@"transform"];

// set final state
NSValue* toValue = [animation.values lastObject];
[self.arc.layer setTransform:[toValue CATransform3DValue]];
```

DEMO

Keyframe Animation

Fold Animation

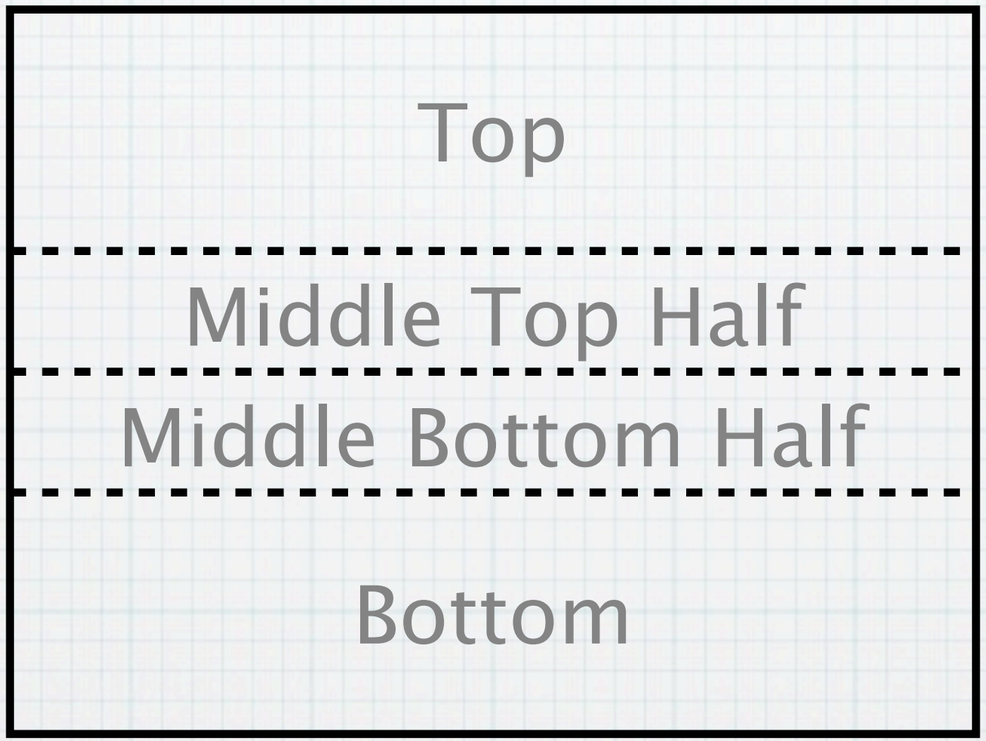
Fold Animation



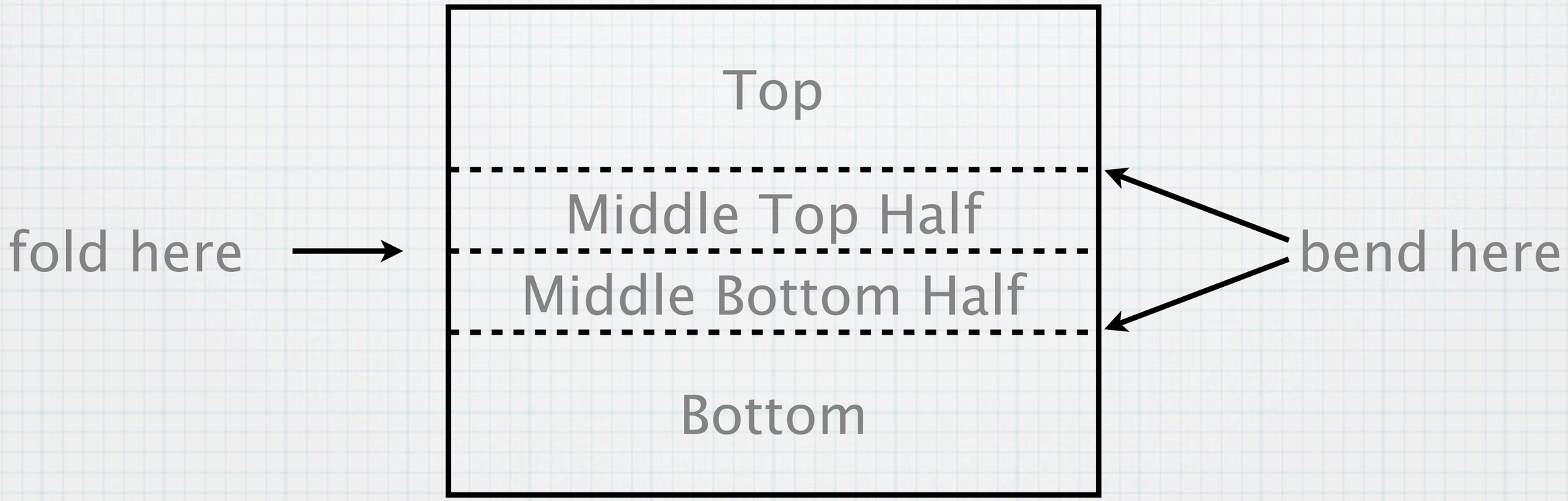
Fold Animation

"cut" view into
4 pieces

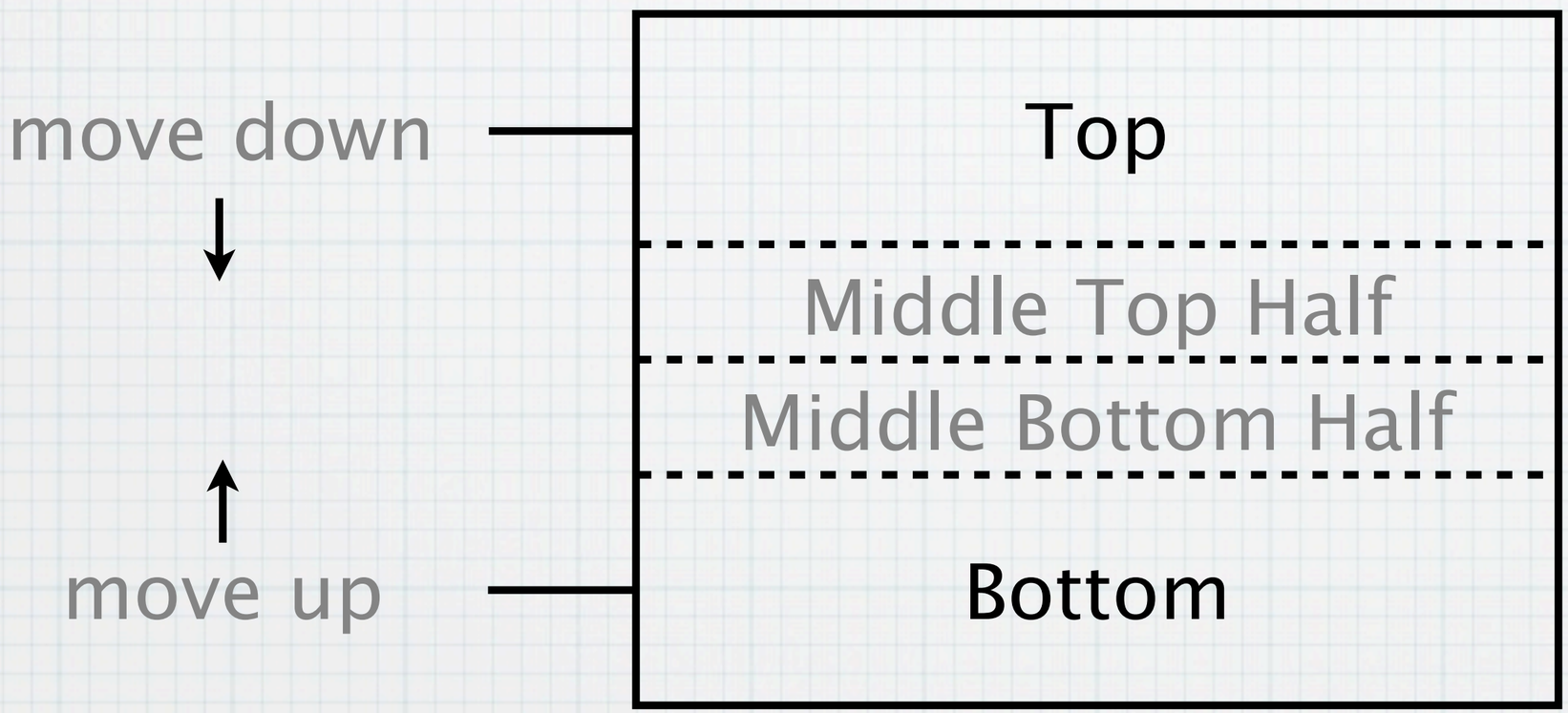
render each
as UIImageView



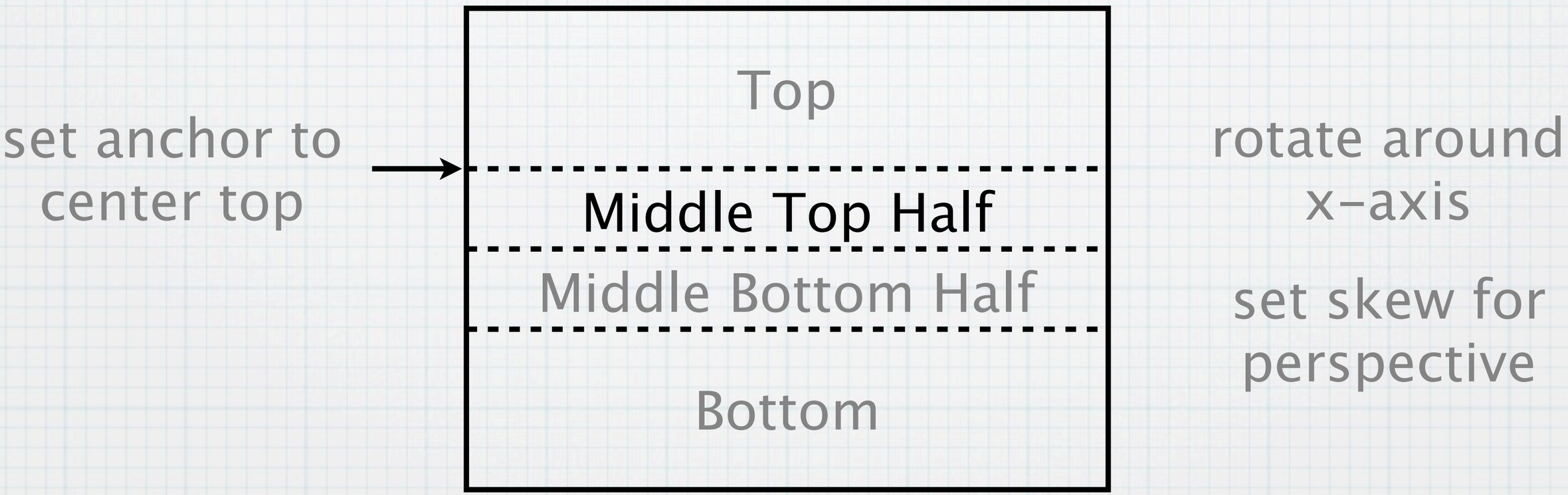
Fold Animation



Fold Animation

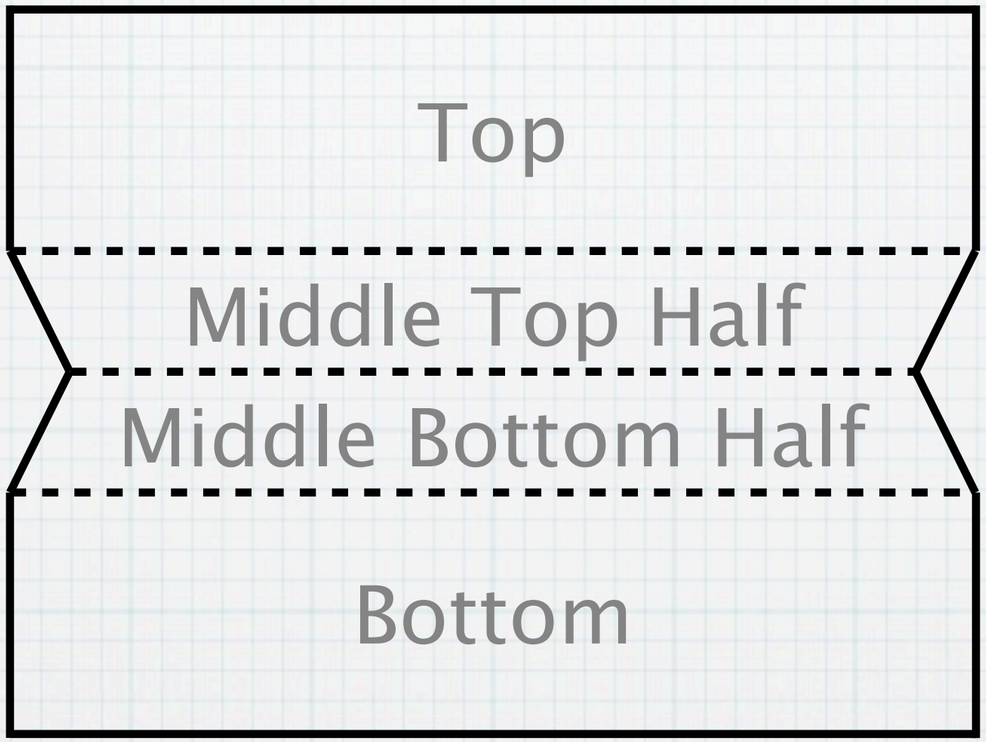


Fold Animation

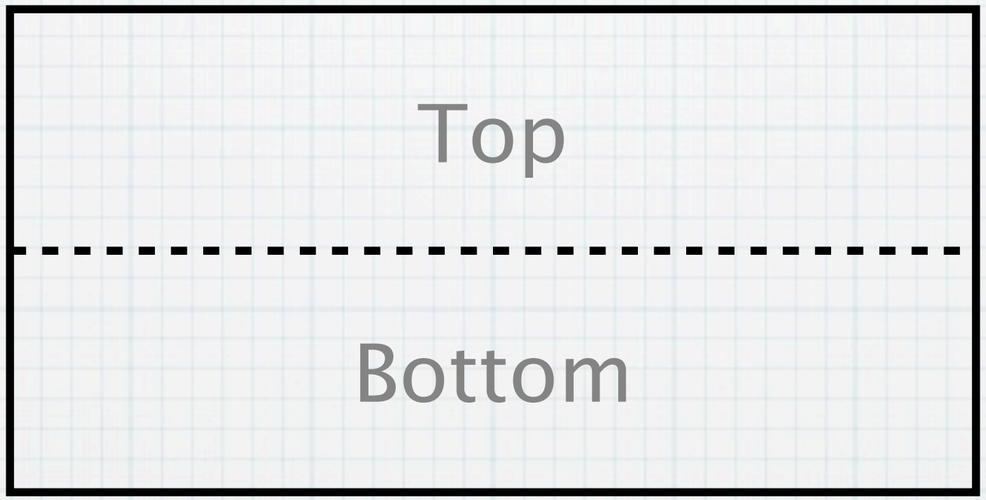


Fold Animation

Fold Animation



Fold Animation

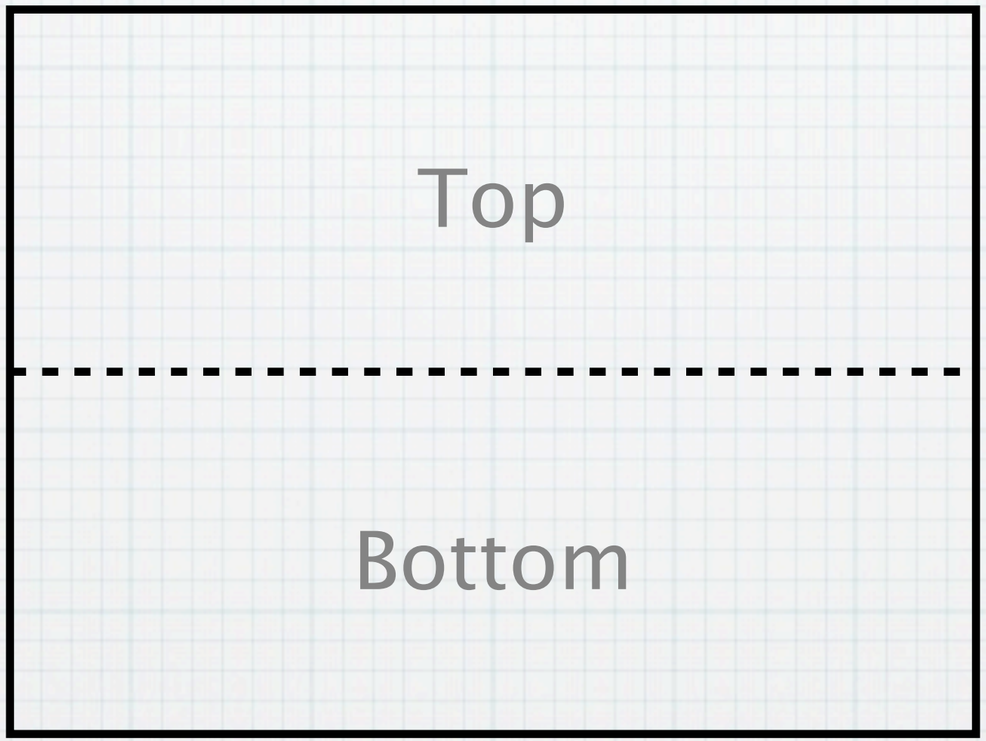


DEMO

Fold Animation

Flip Animation

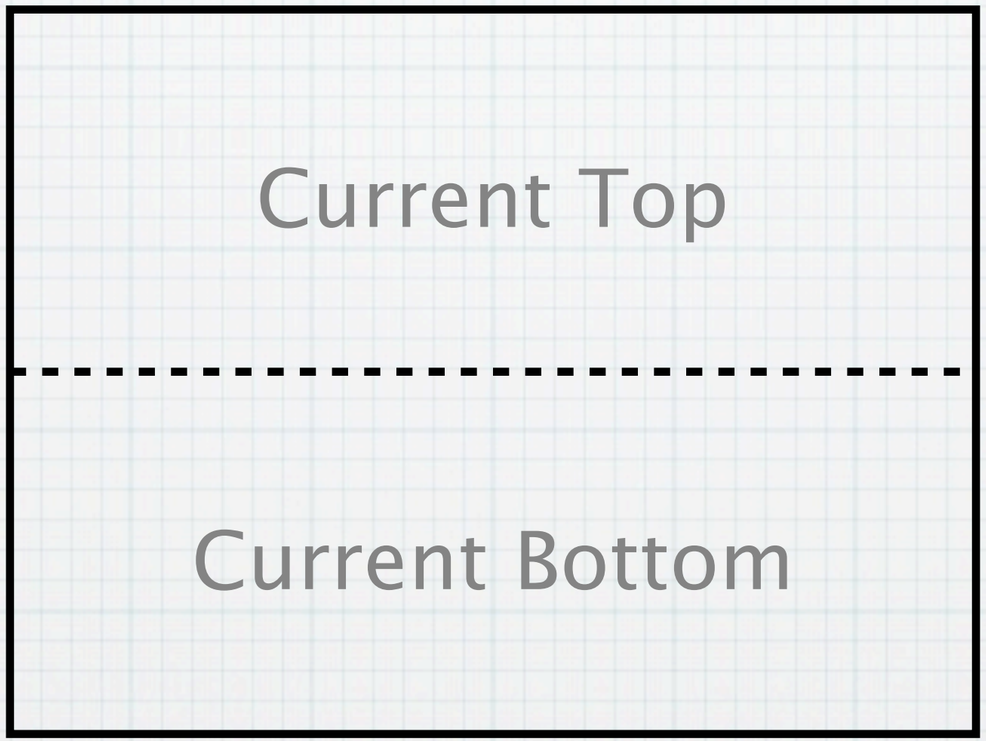
Flip Animation



Flip Animation

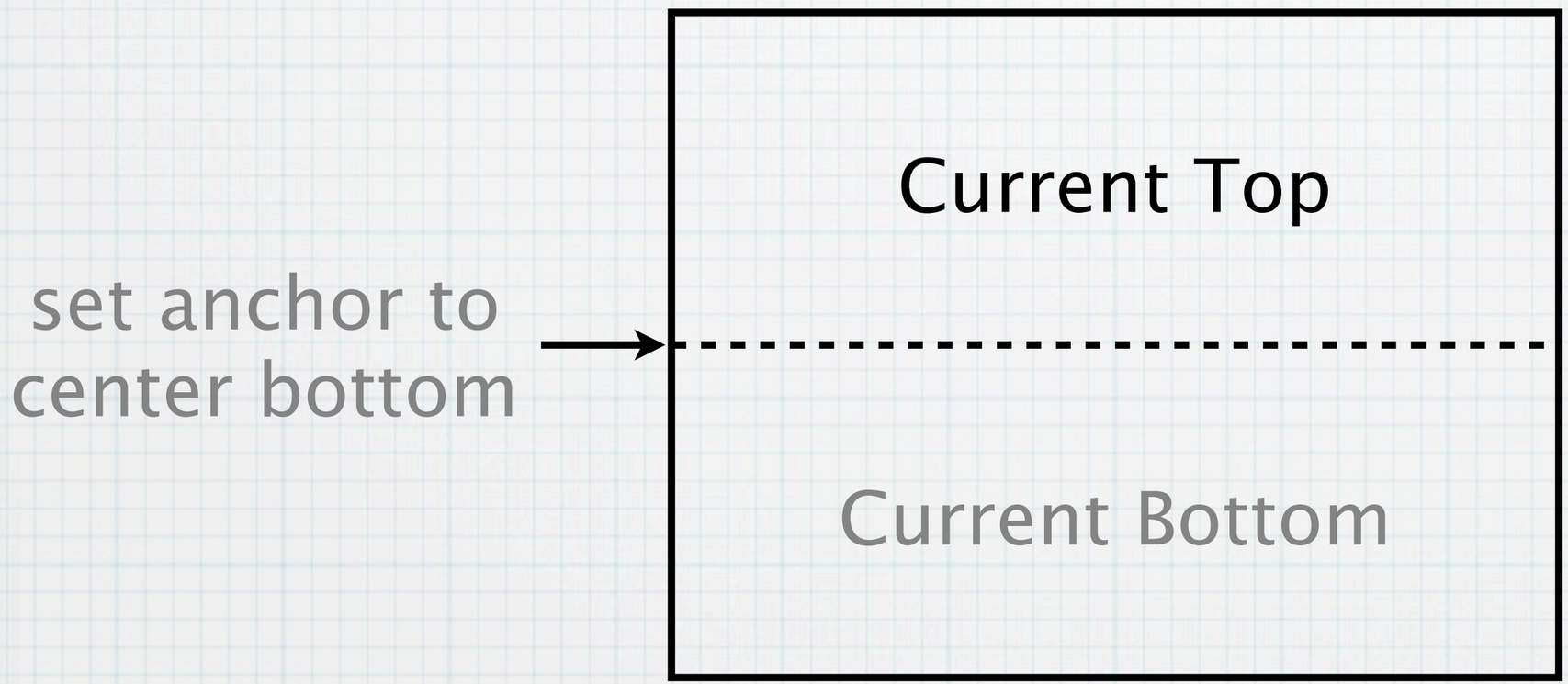
"cut" current and next views into 2 pieces each

render each as UIImageView

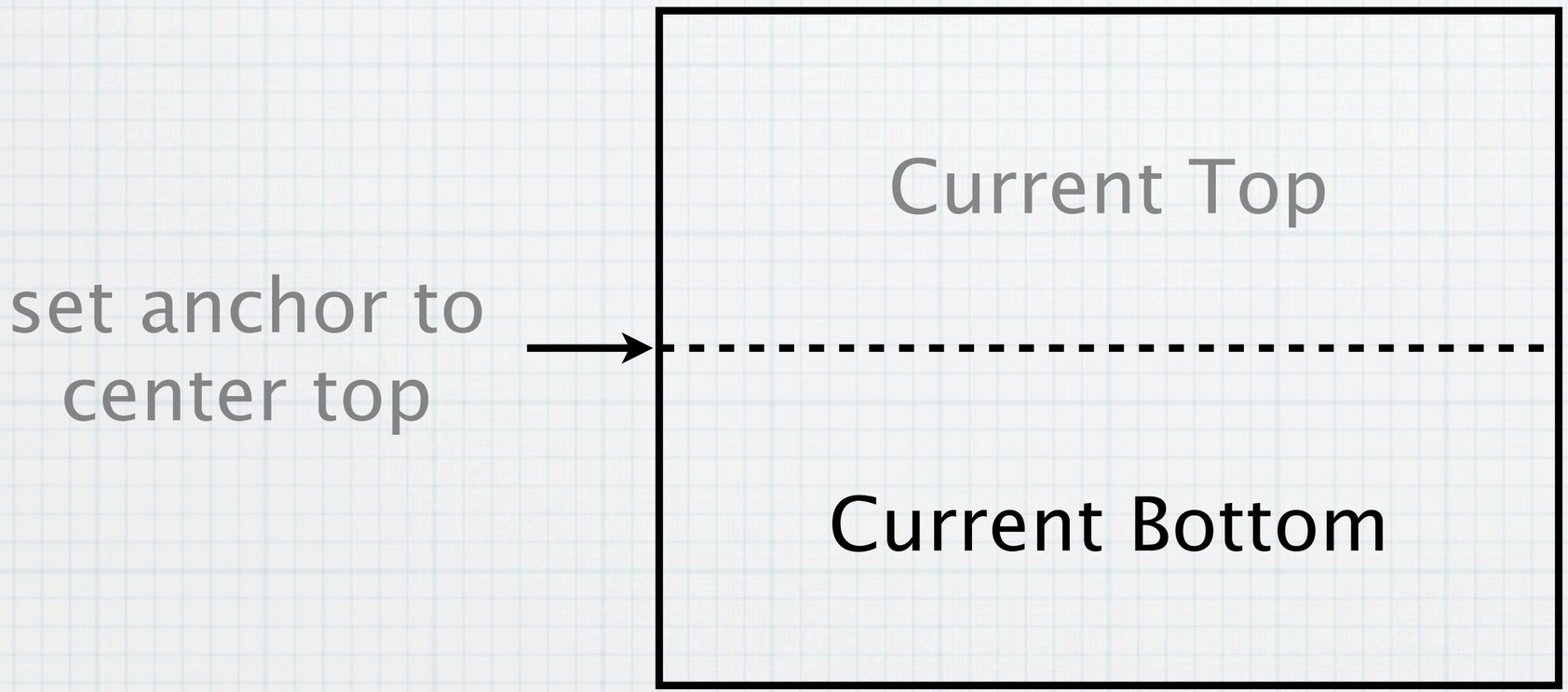


I used only 1 half of next view

Flip Animation



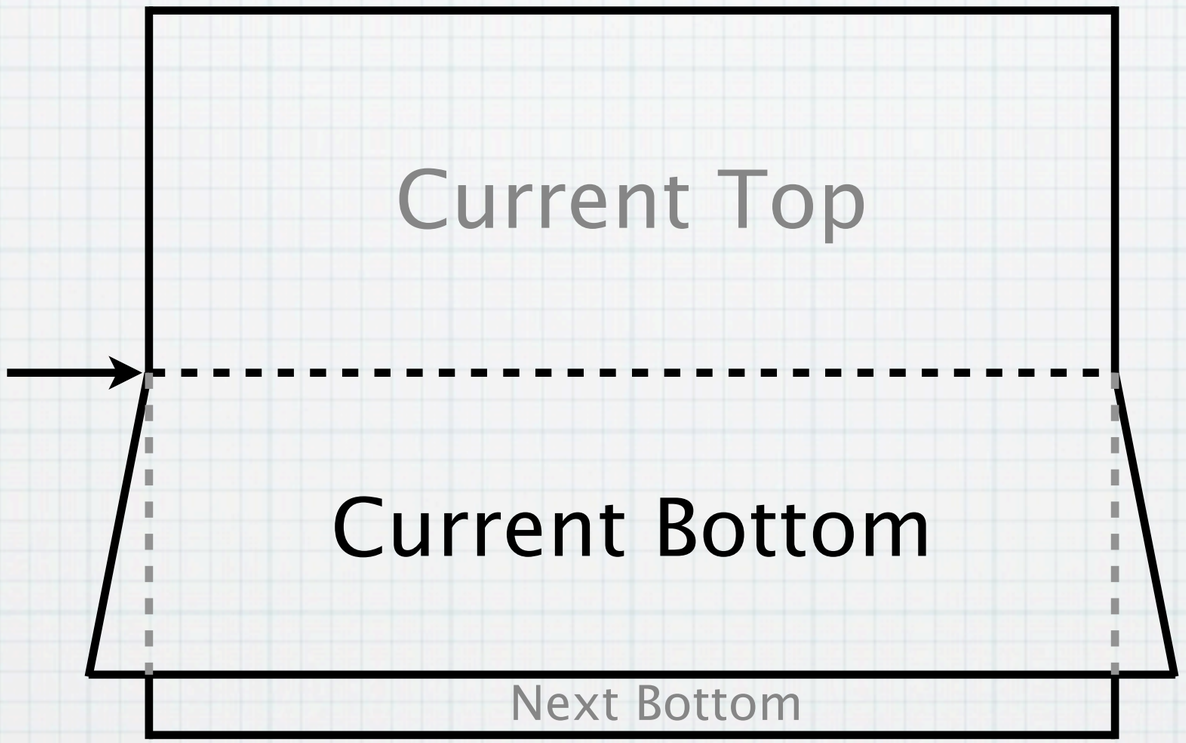
Flip Animation



Flip Animation

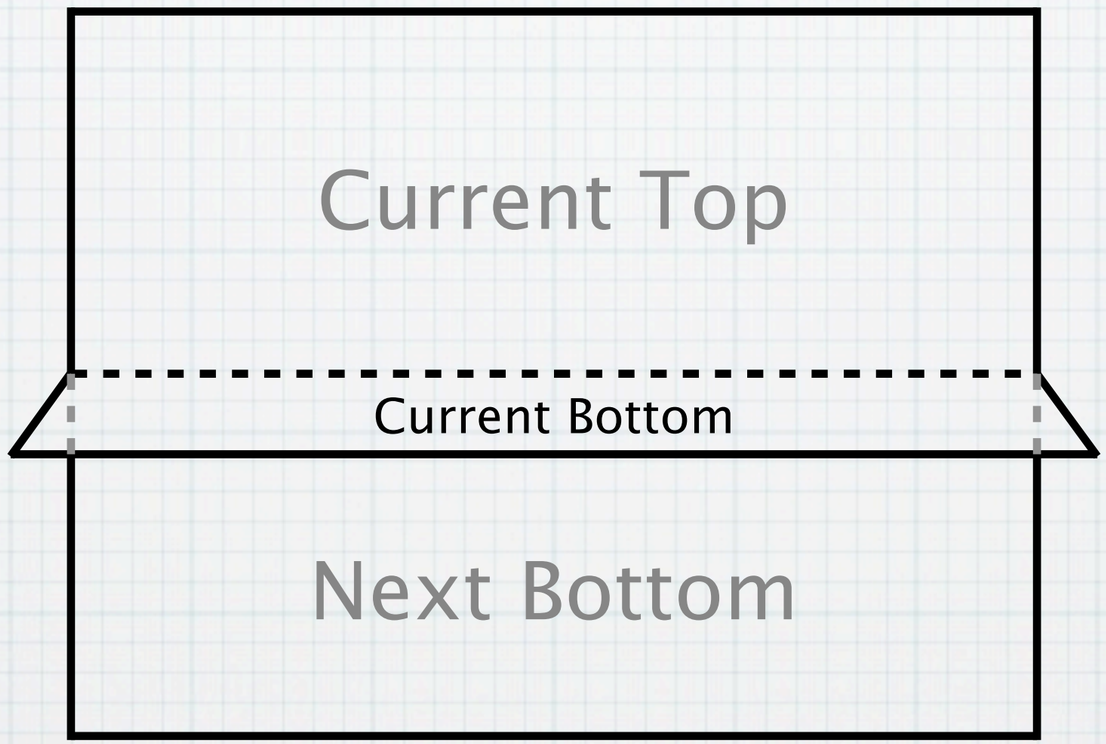
rotate Bottom around x-axis

set skew for perspective



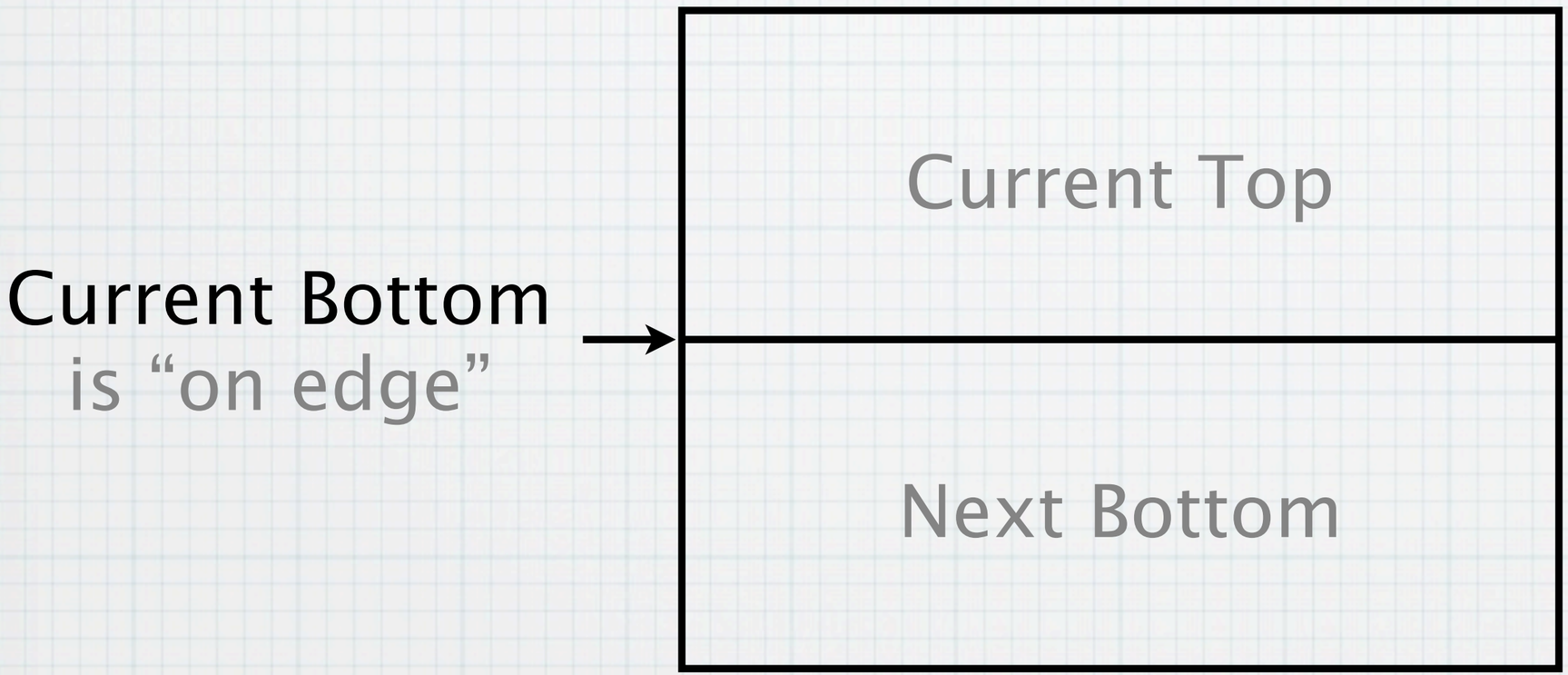
next view being revealed

Flip Animation



first half of animation almost complete

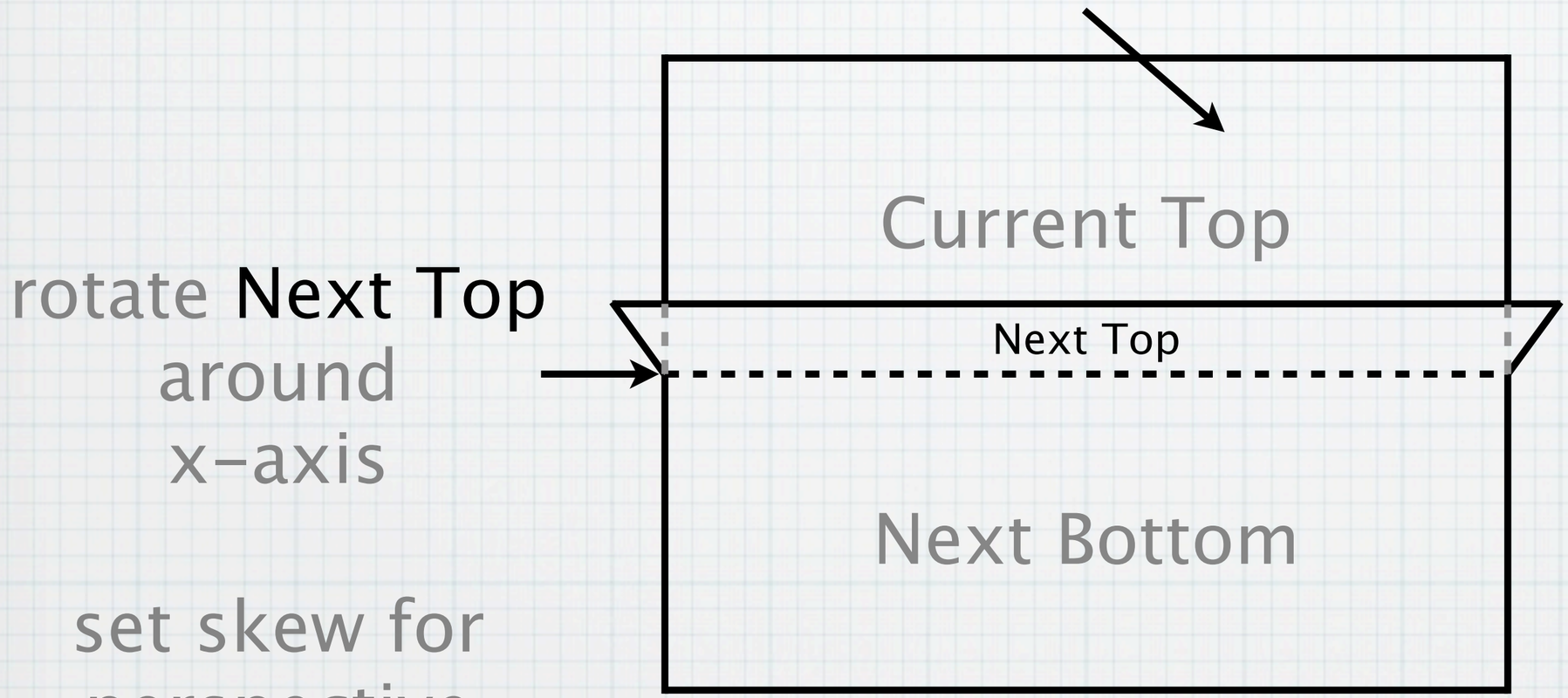
Flip Animation



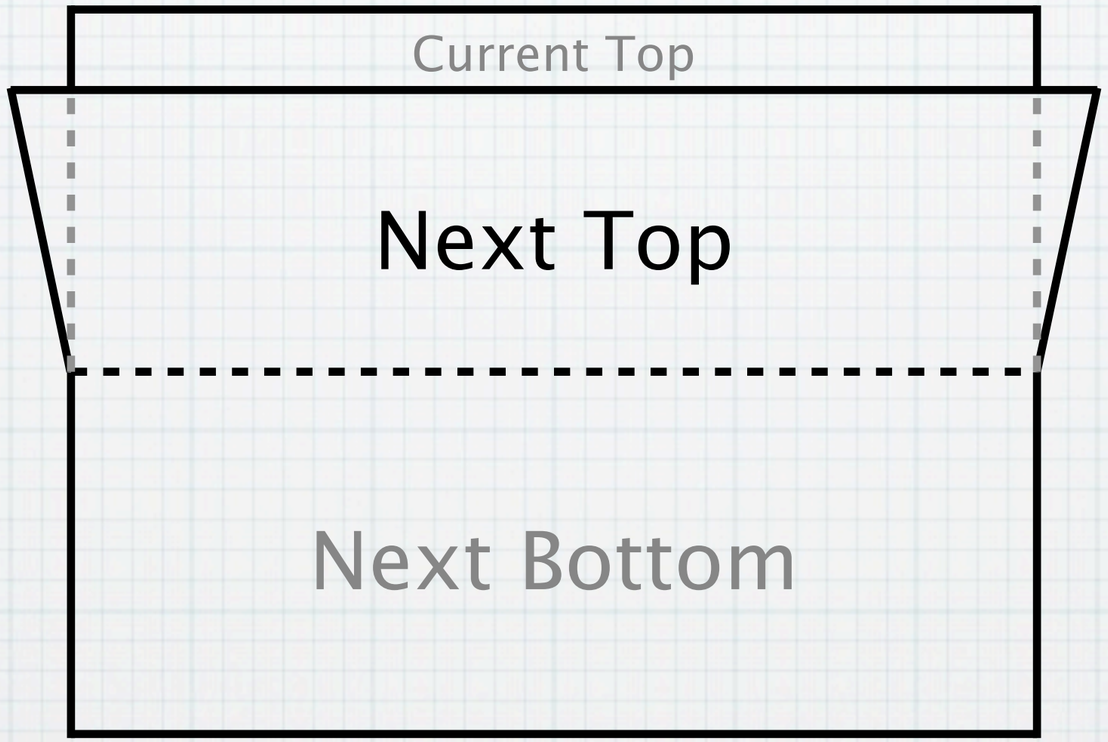
1/2 way done!

Flip Animation

previous view being obscured



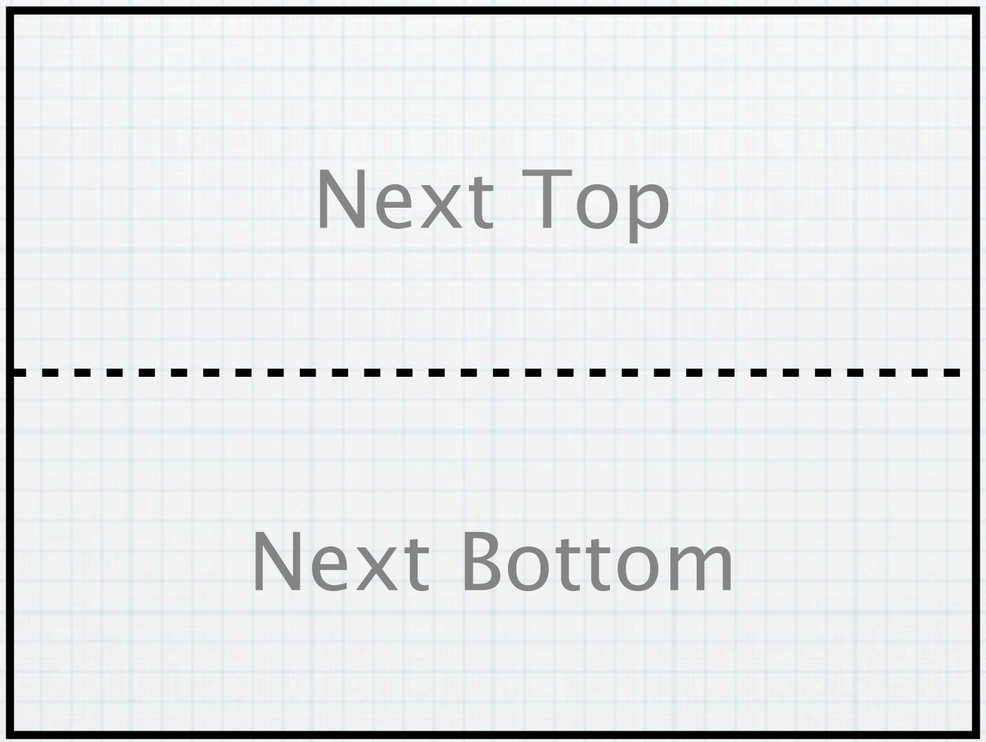
Flip Animation



second half almost done

Flip Animation

Remove the temporary images and show the Next view



Animation complete!

DEMO

Flip Animation

Further Reading

* “Transforms”, Quartz 2D Programming Guide (Apple)
* “Layer Geometry and Transforms”, Core Animation Programming Guide (Apple)
* “Graphics and Drawing in iOS”, Drawing and Printing Guide for iOS (Apple)

Questions?

Odyssey Computing, Inc.
5820 Oberlin Dr, Suite 202
San Diego, CA 92121

Twitter: @mpospese
Email: mark@odysseyinc.com
Phone: 914.282.3519
Web: www.odysseyinc.com

Contact

Odyssey Computing, Inc.

5820 Oberlin Dr, Suite 202

San Diego, CA 92121

Twitter: @mpospese

Email: mark@odysseyinc.com

Phone: 914.282.3519

Web: www.odysseyinc.com