

# iPhone/iPad DevCon 2010



## USING CORE ANIMATION

Nathan Eror - Free Time Studios  
@neror

WHO AM I?



<http://www.freetimestudios.com>

WTF?

# WTF?

- Core Animation in detail
- Lots of sample code
- Get the code at github: <http://github.com/neror/CA360>
- FTUtils - open source Core Animation utility library
- <http://github.com/neror/ftutils>

# WHAT IS CORE ANIMATION?

# WHAT IS CORE ANIMATION?

- Compositing and animation API created for Cocoa Touch
- Hierarchical collection of layers
- Highly optimized for speed (only draws when necessary)
- Interpolates animations automatically and asynchronously
- Simple API (most of the time)
- Hardware accelerated for high performance

# WHEN SHOULD I USE IT?

- Always try to use UIKit drawing and animation first
- Use Core Animation if you need:
  - More granular control over animation and timing
  - More flexible styling options without resorting to images
  - To move your views around in 3D space
  - To draw simple vector images quickly (and animate them)
  - To have more fun :)

# FUNDAMENTALS

# CORE ANIMATION & UIKIT

## UIView.h

```
UIKIT_EXTERN_CLASS @interface UIView : UIResponder<NSCoding> {  
    @package  
    CALayer *_layer;  
}  
  
+ (Class)layerClass;  
  
@property(n nonatomic, readonly, retain) CALayer *layer;
```

- Core Animation is at the heart of UIKit
- A UIView is essentially a CALayer with touch event handling
- A UIView is composed of a single CALayer and is the layer's drawing delegate

# LAYER TREE

## Layer Tree Methods

```
@property(readonly) CALayer *superlayer;  
- (void)removeFromSuperlayer;  
@property(copy) NSArray *sublayers;  
  
- (void)addSublayer:(CALayer *)layer;  
- (void)insertSublayer:(CALayer *)layer atIndex:(unsigned)idx;  
- (void)insertSublayer:(CALayer *)layer below:(CALayer *)sibling;  
- (void)insertSublayer:(CALayer *)layer above:(CALayer *)sibling;  
- (void)replaceSublayer:(CALayer *)layer with:(CALayer *)layer2;  
  
@property CATransform3D sublayerTransform;
```



- Layers are organized in a tree
- The layer tree hierarchy is analogous to the UIView hierarchy
- A layer is positioned, sized and transformed relative to its parent's coordinate system and sublayerTransform
- By default, a layer does not clip its children to its bounds

# LAYER TREE HIERARCHY

## DEMO Layer Tree

# GEOMETRY & TRANSFORMS

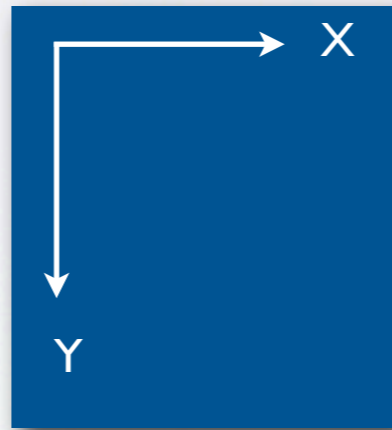
# GEOMETRY & TRANSFORMS

## Quartz2D Data Structures

```
struct CGPoint {
    CGFloat x;
    CGFloat y;
};
typedef struct CGPoint CGPoint;

struct CGSize {
    CGFloat width;
    CGFloat height;
};
typedef struct CGSize CGSize;

struct CGRect {
    CGPoint origin;
    CGSize size;
};
typedef struct CGRect CGRect;
```

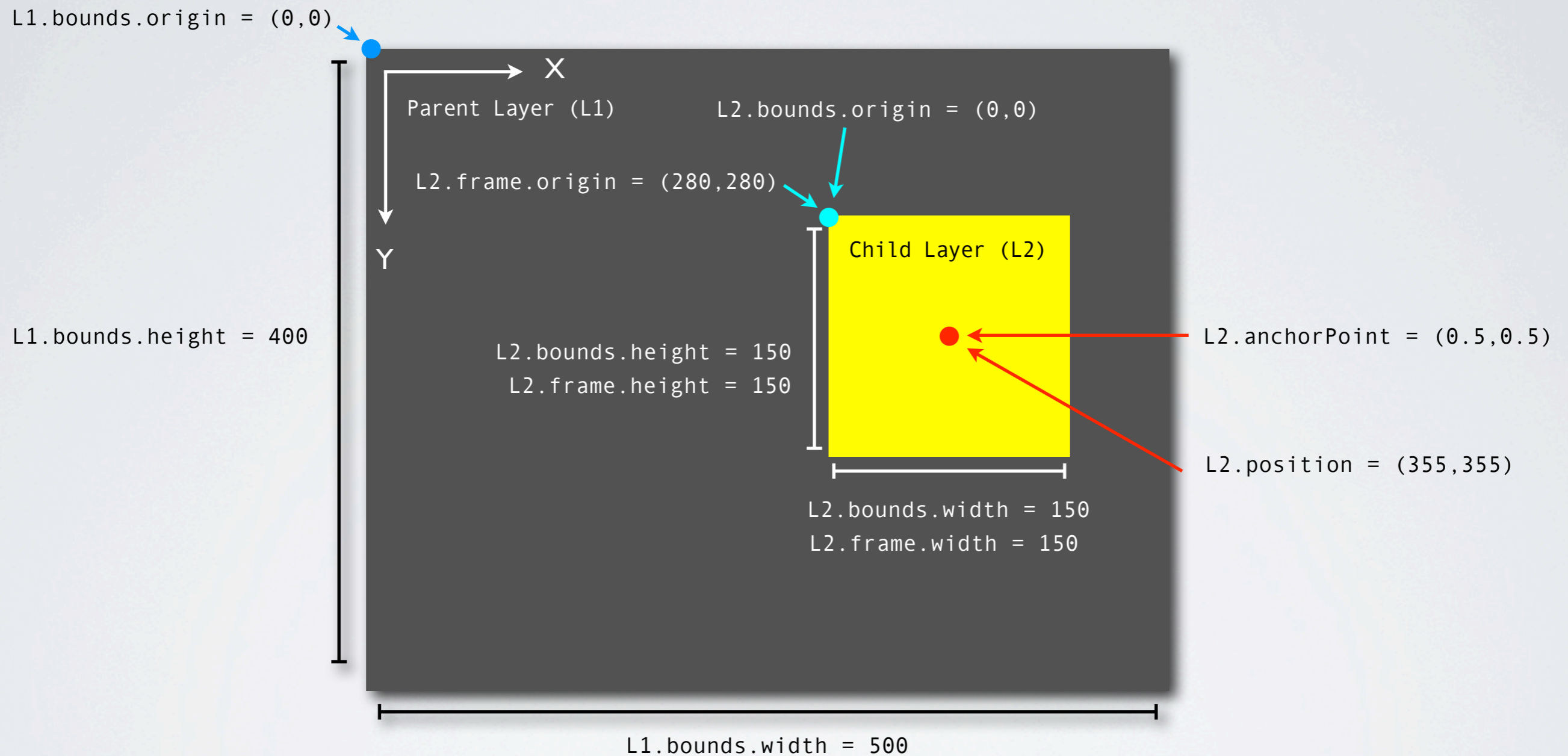


## CALayer Geometric Properties

```
@property CGRect bounds;
@property CGPoint position;
@property CGFloat zPosition;
@property CGPoint anchorPoint;
@property CGFloat anchorPointZ;
@property CATransform3D transform;
@property CGRect frame;
```

- 2D Cartesian coordinate system with the origin in the top left
- Uses Quartz2D primitive data structures

# GOMETRY & TRANSFORMS



## 3D Transform Properties

```
L1.transform = CATransform3DIdentity
L1.sublayerTransform = CATransform3DIdentity
L2.transform = CATransform3DIdentity
```



- Each layer has its own coordinate system
- A layer is a 2D plane projected in 3D
- The 'frame' property is calculated using the 'bounds', 'position', and 'anchorPoint' properties

# GEOMETRY & TRANSFORMS

## DEMO

### Geometric Properties

# GEOMETRY & TRANSFORMS

## 3D Transformation Matrix Data Structure

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

## CALayer's 3D Transformation Properties

```
@property CATransform3D transform;

@property CATransform3D sublayerTransform;

@property CGPoint anchorPoint;

@property CGFloat anchorPointZ;
```

## CATransform3D Matrix Operations

```
CATransform3D CATransform3DMakeTranslation(CGFloat tx, CGFloat ty, CGFloat tz);

CATransform3D CATransform3DMakeScale(CGFloat sx, CGFloat sy, CGFloat sz);

CATransform3D CATransform3DMakeRotation(CGFloat angle, CGFloat x, CGFloat y, CGFloat z);

CATransform3D CATransform3DTranslate(CATransform3D t, CGFloat tx, CGFloat ty, CGFloat tz);

CATransform3D CATransform3DScale(CATransform3D t, CGFloat sx, CGFloat sy, CGFloat sz);

CATransform3D CATransform3DRotate(CATransform3D t, CGFloat angle, CGFloat x, CGFloat y, CGFloat z);

CATransform3D CATransform3DConcat(CATransform3D a, CATransform3D b);

CATransform3D CATransform3DInvert(CATransform3D t);
```



- CATransform3D is the what graphics chip expects from OpenGL ES
- The transform property describes the 3D transform for its layer
- The sublayerTransform property is the projection matrix used to add perspective to sublayers
- The transform and sublayerTransform are applied to the geometry before rendering
- The transforms only affect the final rendering of the geometry.
- The actual geometric values (the “model” values) are not affected by the transforms
- The affects things like hit testing and collision detection
- This is a very important distinction to understand once we start using the CAAnimation classes

# GEOMETRY & TRANSFORMS

## DEMO

### Layer Transforms

# LAYER CONTENT & STYLE

# LAYER CONTENT & STYLE

## Content Properties

```
@property(retain) id contents;  
@property CGRect contentsRect;  
@property(copy) NSString *contentsGravity;  
@property CGRect contentsCenter;  
@property(getter=isOpaque) BOOL opaque;  
@property BOOL needsDisplayOnBoundsChange;
```

## Content Methods

```
- (void)display;  
- (void)setNeedsDisplay;  
- (void)setNeedsDisplayInRect:(CGRect)r;  
- (BOOL)needsDisplay;  
- (void)displayIfNeeded;  
- (void)drawInContext:(CGContextRef)ctx;
```

## Style Properties

```
@property CGColorRef backgroundColor;  
@property CGFloat cornerRadius;  
@property CGFloat borderWidth;  
@property CGColorRef borderColor;  
@property float opacity;  
@property(retain) CALayer *mask;
```

## Drawing Delegate Methods

```
- (void)displayLayer:(CALayer *)layer;  
- (void)drawLayer:(CALayer *)layer  
  inContext:(CGContextRef)ctx;
```

# LAYER CONTENT & STYLE

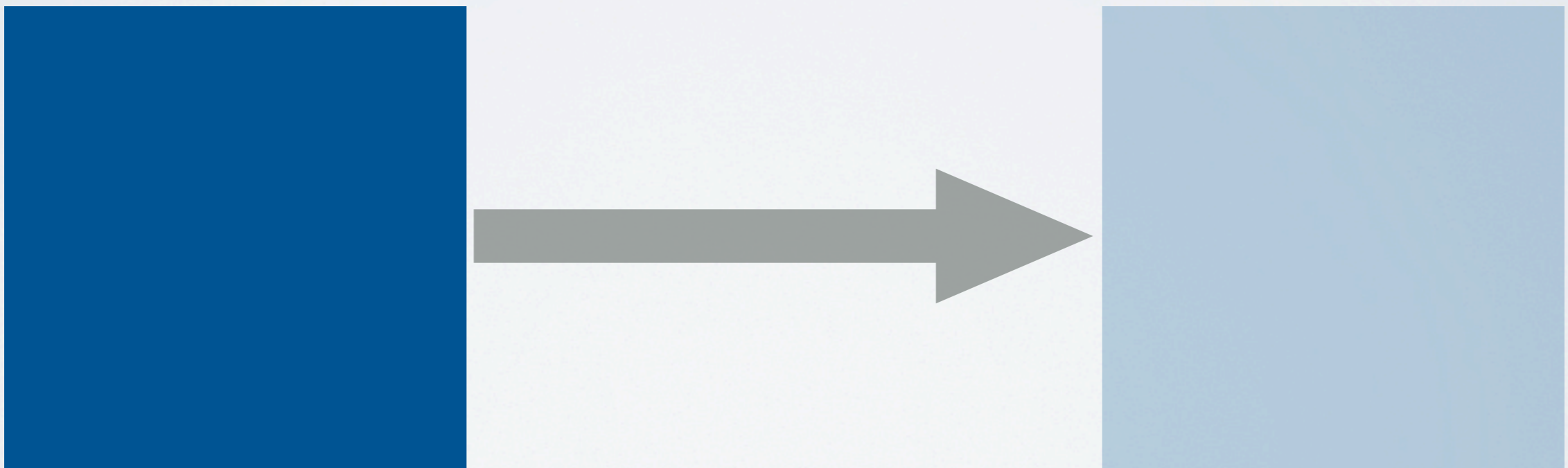
## DEMOS

Image Content  
Delegate Drawing  
Style Properties

# ANIMATION

# ANIMATION

Move the box from the left position to the right position over 2 seconds distributing the interpolated values along a curve that slopes slowly at the beginning, quickly through the middle and slowly again at the end.



- An animation is defined by the following 3 things:
  1. The property it is animating
  2. The time it takes
  3. The pacing of the interpolation
- All animatable properties have a default implicit animation
- All animatable properties can be explicitly animated as well
- Animations do not effect underlying geometric and style properties, only the rendering (presentation) of the layer
- All animation classes descend from CAAAnimation
- A CAAAnimation notifies an optional delegate at the start and end of its animation

# ANIMATION TIMING

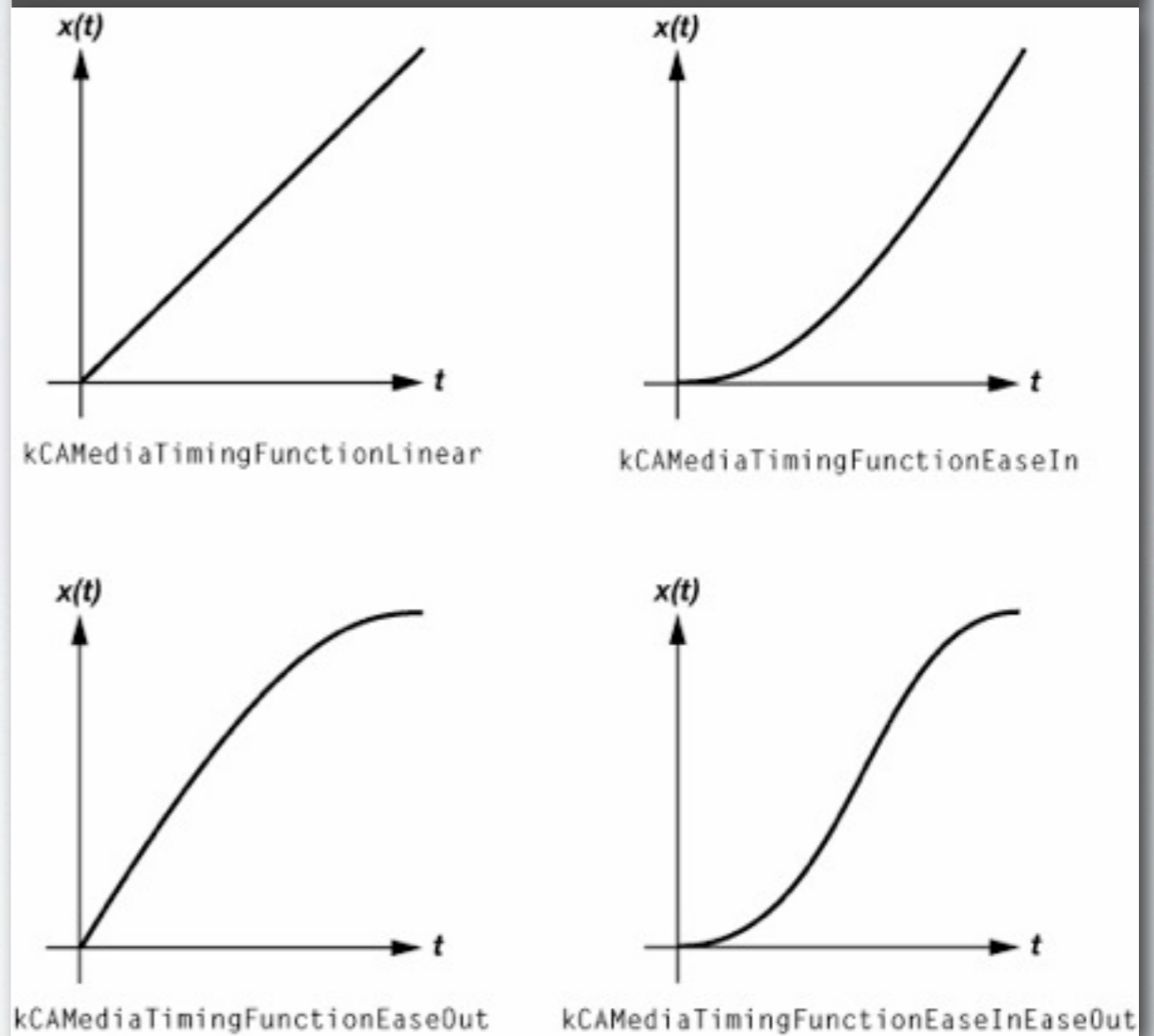
## Timing Properties

```
@property CTimeInterval beginTime;  
@property CTimeInterval duration;  
@property float speed;  
@property CTimeInterval timeOffset;  
@property float repeatCount;  
@property CTimeInterval repeatDuration;  
@property BOOL autoreverses;
```

## Fill Mode Options

```
@property(copy) NSString *fillMode;  
NSString * const kCAFillModeForwards;  
NSString * const kCAFillModeBackwards;  
NSString * const kCAFillModeBoth;  
NSString * const kCAFillModeRemoved;
```

## Default Timing Functions



- The CAMediaTiming protocol is responsible for the duration and repeating of animations
- An animation's timing is defined relative to its parent layer's timespace (controlled by the speed property)
- The fillMode property defines the animation's behavior outside of its active duration
- The CAMediaTimingFunction describes the pacing of the animation as a cubic bezier curve

# PROPERTY ANIMATION

- Use `CABasicAnimation` to explicitly animate a layer property
  - `toValue`: The final value of the property
  - `fromValue`: The optional starting value
  - `byValue`: Add `byValue` to the current value

## One Property

`CABasicAnimation`

`CAKeyframeAnimation`

## *n* Properties

`CAAnimationGroup`

## Layer Tree Actions

`CATransition`

# PROPERTY ANIMATION

## BasicAnimation.m

```
CABasicAnimation *pulseAnimation =  
    [CABasicAnimation animationWithKeyPath:@"transform.scale"];  
pulseAnimation.duration = .5;  
pulseAnimation.toValue = [NSNumber numberWithFloat:1.1];  
pulseAnimation.timingFunction =  
    [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];  
pulseAnimation.autoreverses = YES;  
pulseAnimation.repeatCount = MAXFLOAT;  
  
[pulseLayer_ addAnimation:pulseAnimation forKey:nil];
```

# ANIMATION GROUPS

- Use `CAAnimationGroup` to group multiple property animations to be run concurrently on the same layer
- The `delegate` and `removedOnCompletion` properties of the animations are ignored
- The `duration`, `fillMode` and `beginTime`

## One Property

`CABasicAnimation`

`CAKeyframeAnimation`

## $n$ Properties

`CAAnimationGroup`

## Layer Tree Actions

`CATransition`

# ANIMATION GROUPS

## AnimationGroups.m

```
CABasicAnimation *pulseAnimation = [CABasicAnimation animationWithKeyPath:@"transform.scale"];
pulseAnimation.duration = 2.;
pulseAnimation.toValue = [NSNumber numberWithFloat:1.15];

CABasicAnimation *pulseColorAnimation = [CABasicAnimation animationWithKeyPath:@"backgroundColor"];
pulseColorAnimation.duration = 1.;
pulseColorAnimation.fillMode = kCAFillModeForwards;
pulseColorAnimation.toValue = (id)[UIColorFromRGBA(0xFF0000, .75) CGColor];

CABasicAnimation *rotateLayerAnimation =
    [CABasicAnimation animationWithKeyPath:@"transform.rotation"];
rotateLayerAnimation.duration = .5;
rotateLayerAnimation.beginTime = .5;
rotateLayerAnimation.fillMode = kCAFillModeBoth;
rotateLayerAnimation.toValue = [NSNumber numberWithFloat:DEGREES_TO_RADIANS(45.)];

CAAnimationGroup *group = [CAAnimationGroup animation];
group.animations = [NSArray arrayWithObjects:pulseAnimation, pulseColorAnimation,
                                             rotateLayerAnimation, nil];

group.duration = 2.;
group.timingFunction =
    [CAMediaTimingFunction functionName:kCAMediaTimingFunctionEaseInEaseOut];
group.autoreverses = YES;
group.repeatCount = MAXFLOAT;

[pulseLayer_ addAnimation:group forKey:nil];
```

# KEYFRAME ANIMATION

- `CAKeyframeAnimation` allows more fine control over the animation of a layer property
  - `values`: Array of `toValues`
  - `path`: Animate along a `CGPathRef` (alt. to `values`)
  - `calculationMode`: Interpolation style between keyframes

## One Property

`CABasicAnimation`

`CAKeyframeAnimation`

## *n* Properties

`CAAnimationGroup`

## Layer Tree Actions

`CATransition`

# KEYFRAME ANIMATION

## KeyframeAnimation.m

```
CGSize viewSize = self.view.bounds.size;
[marioLayer_ removeAnimationForKey:@"marioJump"];

CGMutablePathRef jumpPath = CGPathCreateMutable();
CGPathMoveToPoint(jumpPath, NULL, 0., viewSize.height);
CGPathAddCurveToPoint(jumpPath, NULL, 30., 140., 170., 140., 170., 200.);

CAKeyframeAnimation *jumpAnimation =
    [CAKeyframeAnimation animationWithKeyPath:@"position"];
jumpAnimation.path = jumpPath;
jumpAnimation.duration = 1.;
jumpAnimation.timingFunction =
    [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
jumpAnimation.delegate = self;

CGPathRelease(jumpPath);

[marioLayer_ addAnimation:jumpAnimation forKey:@"marioJump"];
```

# TRANSITIONS

- A `CATransition` animates changes to the layer tree
- Canned animations that effect an entire layer
- The type and subtype properties define the canned transition animation
- The `CATransition` is added to the *parent layer* and animates the hiding, showing, adding and removing of its child layers

## One Property

`CABasicAnimation`

`CAKeyframeAnimation`

## *n* Properties

`CAAnimationGroup`

## Layer Tree Actions

`CATransition`

# TRANSITIONS

## LayerTransitions.m

```
CATransition *transition = [CATransition animation];  
transition.duration = .5;  
transition.timingFunction =  
    [CAMediaTimingFunction functionName:kCAMediaTimingFunctionEaseInEaseOut];  
transition.type = kCATransitionPush;  
transition.subtype = kCATransitionFromRight;  
  
[containerLayer_ addAnimation:transition forKey:nil];
```

# SPECIAL LAYER TYPES

- `CATiledLayer`: For displaying very large layers (bigger than 1024x1024) at multiple levels of detail
- `CAShapeLayer`: Draws an animatable `CGPath` (since 3.0)
- `CAGradientLayer`: Draws a gradient over its background color (since 3.0)
- `CATransformLayer`: A container layer for true 3D hierarchy (since 3.0)
- `CAReplicatorLayer`: Creates copies of its sublayers with each copy potentially having geometric, temporal and color transformations applied to it (since 3.0)
- `CATextLayer`: Renders text using Core Text. Supports loadable fonts and `NSAttributedString`. (since 3.2)

# GRADIENT LAYERS

## GradientLayers.m

```
gradientLayer_.backgroundColor = [[UIColor blackColor] CGColor];
gradientLayer_.bounds = CGRectMake(0., 0., 200., 200.);
gradientLayer_.position = self.view.center;
gradientLayer_.cornerRadius = 12.;
gradientLayer_.borderWidth = 2.;
gradientLayer_.borderColor = [[UIColor blackColor] CGColor];
gradientLayer_.startPoint = CGPointZero;
gradientLayer_.endPoint = CGPointMake(0., 1.);
gradientLayer_.colors = [NSArray arrayWithObjects:
    (id)[[UIColor whiteColor] CGColor],
    (id)[UIColorFromRGBA(0xFFFFFFFF, .1) CGColor],
    nil];
```

# SHAPE LAYERS

## ShapeLayers.m

```
shapeLayer_.backgroundColor = [[UIColor clearColor] CGColor];
shapeLayer_.frame = CGRectMake(0., 0., 200., 200.);
shapeLayer_.position = self.view.center;

CGMutablePathRef path = CGPathCreateMutable();
CGPathAddEllipseInRect(path, NULL, rect);
shapeLayer_.path = path;
CGPathRelease(path);

shapeLayer_.fillColor = [[UIColor blueColor] CGColor];
shapeLayer_.strokeColor = [[UIColor blackColor] CGColor];
shapeLayer_.lineWidth = 4.;
shapeLayer_.lineDashPattern = [NSArray arrayWithObjects:
                                [NSNumber numberWithInt:8],
                                [NSNumber numberWithInt:8],
                                nil];
shapeLayer_.lineCap = kCALineCapRound;
```



- Shape layers allow for optimized drawing of simple CGPaths
- The changing of the 'path' property can be animated

# TEXT LAYERS

## TextLayers.m

```
CTFontRef font = CTFontCreateWithName(CFSTR("Courier"), 16.f, NULL);
normalTextLayer_ = [[CATextLayer alloc] init];
normalTextLayer_.font = font;
normalTextLayer_.string = @"This is just a plain old CATextLayer";
normalTextLayer_.wrapped = YES;
normalTextLayer_.foregroundColor = [[UIColor purpleColor] CGColor];
normalTextLayer_.fontSize = 20.f;
normalTextLayer_.alignmentMode = kCAAlignmentCenter;
normalTextLayer_.frame = CGRectMake(0.f, 10.f, 320.f, 32.f);
CFRelease(font);
```

- 'string' property can be either an NSString or and NSAttributedString
- If using NSAttributedString, the text styling properties (fontSize, foregroundColor, etc) are ignored (see next slide)

# TEXT LAYERS

## TextLayers.m

```
NSDictionary *fontAttributes = [NSDictionary dictionaryWithObjectsAndKeys:
    @"Courier", (NSString *)kCTFontFamilyNameAttribute,
    @"Bold", (NSString *)kCTFontStyleNameAttribute,
    [NSNumber numberWithFloat:16.f], (NSString *)kCTFontSizeAttribute,
    nil];

CTFontDescriptorRef descriptor =
    CTFontDescriptorCreateWithAttributes((CFDictionaryRef)fontAttributes);
CTFontRef courierFont = CTFontCreateWithFontDescriptor(descriptor, 0, NULL);
CFRelease(descriptor);

NSDictionary *stringAttributes =
    [NSDictionary dictionaryWithObject:(id)courierFont
    forKey:(NSString *)kCTFontAttributeName];
NSMutableAttributedString *attrString =
    [[NSMutableAttributedString alloc] initWithString:EXAMPLE_STRING
    attributes:stringAttributes];

NSRange rangeOfClassName = [[attrString string] rangeOfString:@"CATextLayer"];
[attrString addAttribute:(NSString *)kCTForegroundColorAttributeName
    value:(id)[UIColor redColor] CGColor]
    range:rangeOfClassName];

CFRelease(courierFont);
attributedTextLayer_.string = attrString;
```

# PERFORMANCE

# PERFORMANCE

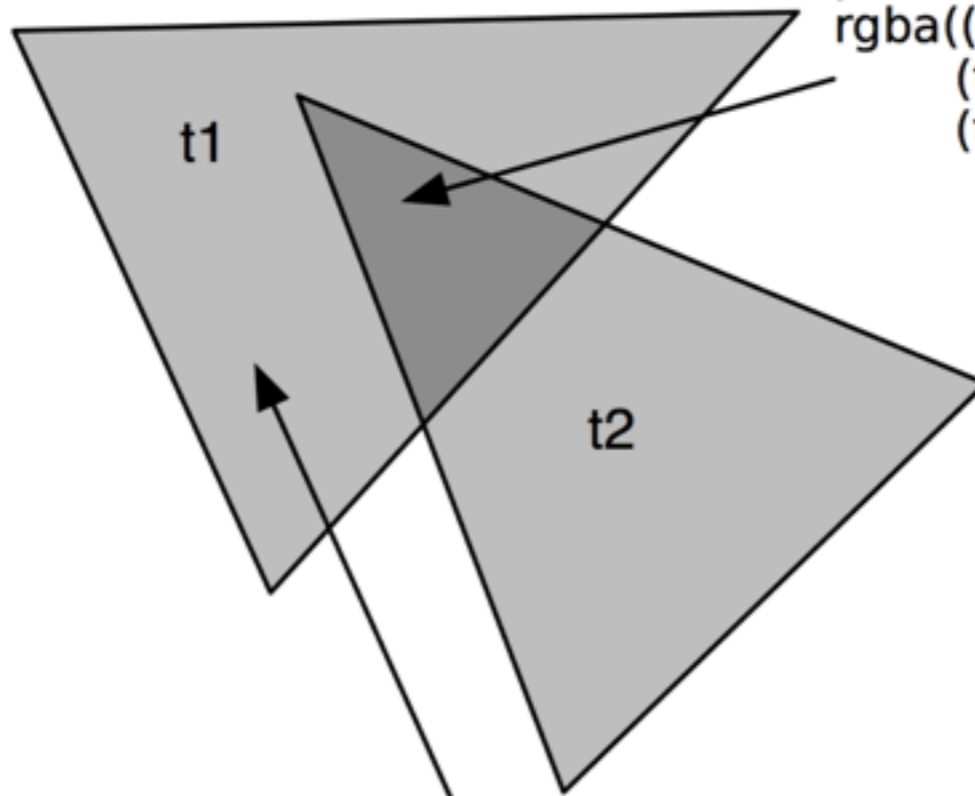
- Core Animation is hardware accelerated
- It takes advantage of the features of the GPU
- It works very well as long as we stay out of its way
- We should know how the GPU is used by Core Animation

# THE GPU

## Blending

t1 color = rgba(1.0, 1.0, 1.0, 0.25)

t2 color = rgba(1.0, 1.0, 1.0, 0.25)



pixel color = blend(t1, t2) =  
rgba((t2r \* t2a) + (1 - t1a) \* t1r,  
(t2g \* t2a) + (1 - t1a) \* t1g,  
(t2b \* t2a) + (1 - t1a) \* t1b)

pixel color = t1 color = rgba(1.0, 1.0, 1.0, 0.25)

### The GPU:

1. Takes a bunch of triangles as input
2. Transforms those triangles into pixels
3. Renders those pixels into a pixel buffer.

Each triangle sent to the GPU is filled with a color or an image. The GPU examines the contents of each triangle, and renders a pixel to the appropriate position in the buffer. When two or more translucent triangles overlap, the GPU must combine the color and alpha values of each triangle for every pixel in which they overlap. This is called "blending".

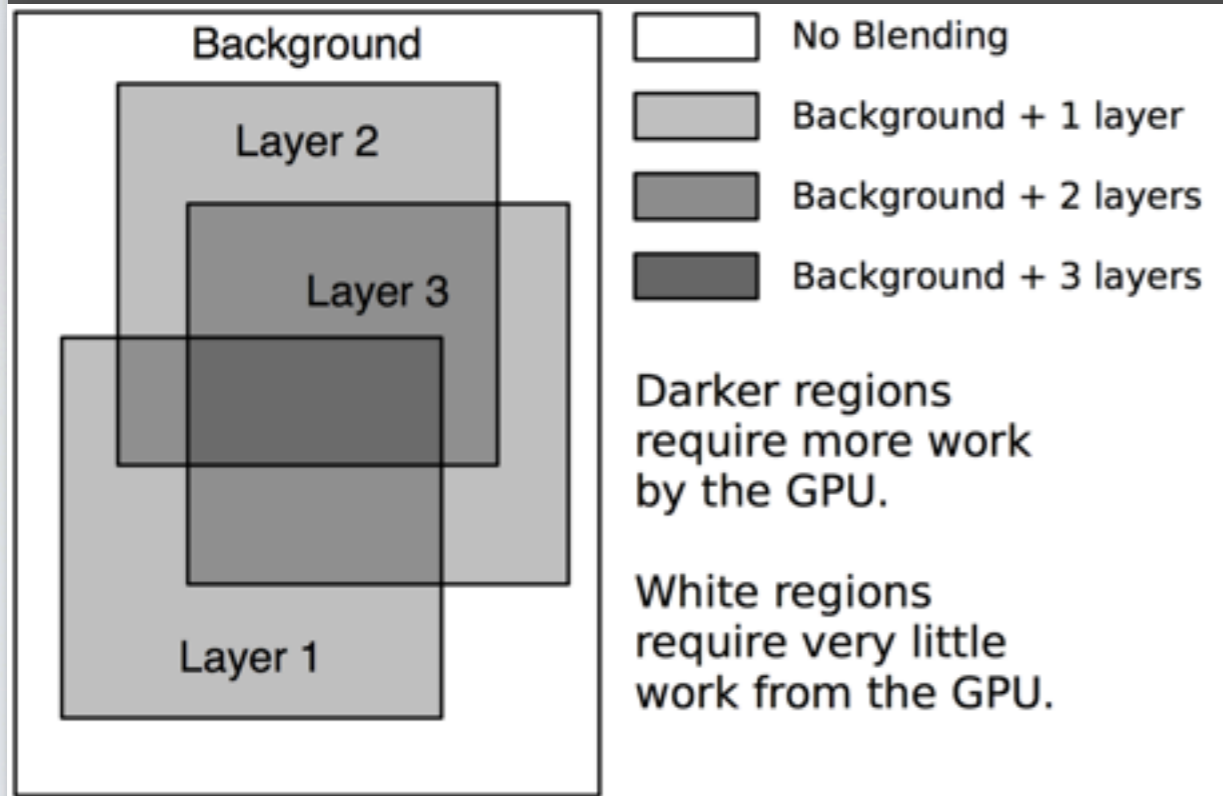
# GPU BOTTLENECKS

- **Destination Pixels** - The number of pixels the GPU must render/blend to reach the final bitmap
- **Source Pixels** - The number of pixels the GPU must read in from source images
- **Rendering Passes** - The number of rendering passes required to reach the final bitmap

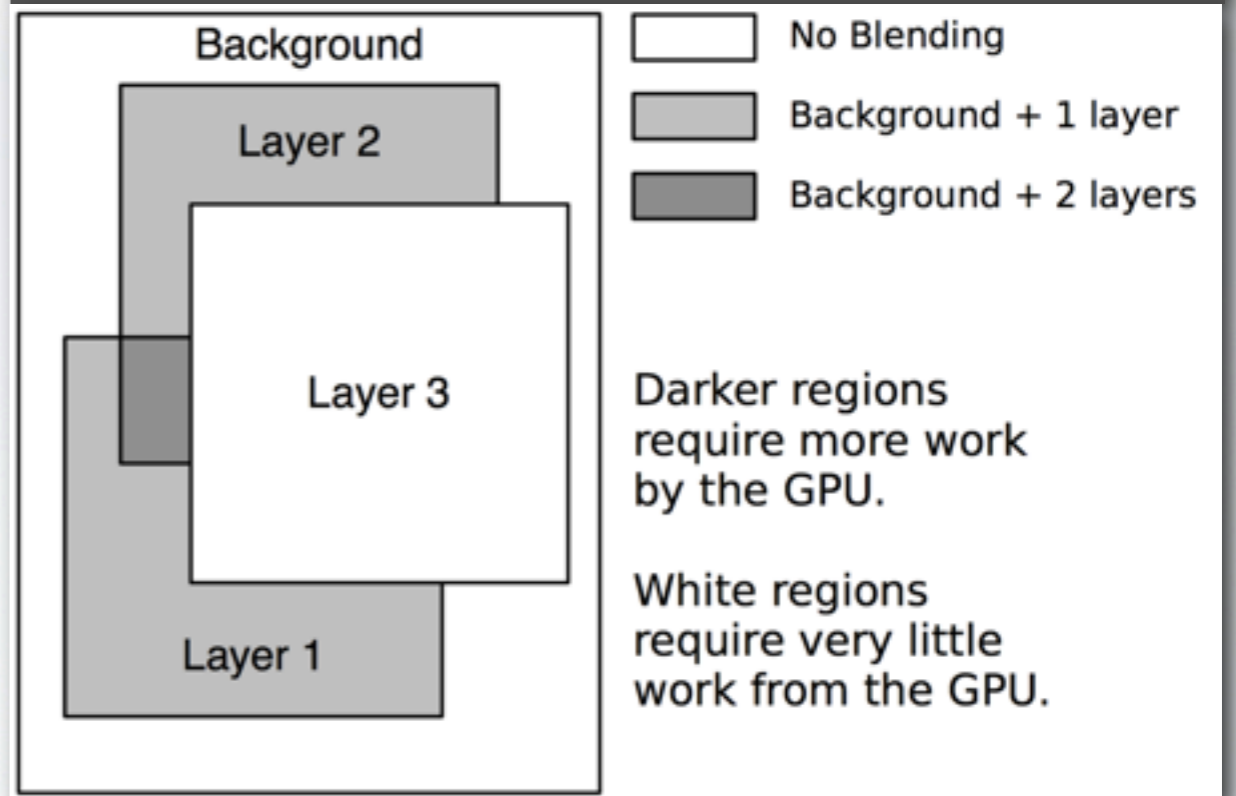
# DESTINATION PIXELS

Minimize Transparent Pixels to Minimize Blending

## All non-opaque layers



## Using opaque layers



# SOURCE PIXELS

- Pre-render properly sized images to minimize offscreen drawing. Unnecessarily large images kill performance.
- Properly sized source images allow CA to map the image pixels directly into the framebuffer (as long as the image is opaque) bypassing scaling and compositing operations
- Also results in reduced memory usage and disk i/o
- Use the “Color Misaligned Images” option in Instruments

# RENDERING PASSES

- Limit use of mask layers and group opacity
- Use layer bitmap caching carefully
- Use the “Color Offscreen” option in Instruments

# THANK YOU

Nathan Eror

[neror@freetimestudios.com](mailto:neror@freetimestudios.com)

<http://twitter.com/freetimestudios>

<http://twitter.com/neror>

<http://www.freetimestudios.com>

<http://www.neror.com>