Core Animation in Practice

Part 2

John Harper

Goal of This Session

To dive deeper into Core Animation

What You'll Learn

- New and overlooked APIs
- Mental model of CA performance
- High-DPI details

Selected APIs

Drop Shadows

- Shadows provide crucial depth cues in 2D user interfaces
- iPhone OS 3.2 supports the full set of CALayer shadow APIs
- New API for more efficient shadows
  - @property CGPathRef shadowPath
- Defines the opaque region of the layer
- Lets the compositor cache a shadow bitmap

Shadow Example

```
CALayer *sublayer = [CALayer layer];
sublayer.bounds = sublayer_bounds;
sublayer.backgroundColor = random_color();
```

```
sublayer.shadowOpacity = shadowsEnabled ? .5 : 0;
sublayer.shadowRadius = 10;
sublayer.shadowOffset = CGSizeMake(0, 10);
```

```
CGPathRef shadowPath
  = [UIBezierPath bezierPathWithRect:sublayer_bounds].CGPath;
sublayer.shadowPath = pathEnabled ? shadowPath : nil;
```

Shape Layers

- Most layers use bitmaps to provide their content
  - Doesn't scale well, doesn't animate well
- Use a CAShapeLayer with path for scalable/animatable content
- Performance tradeoffs
  - Uses little memory
  - Uses more CPU to render
  - No cost for transparent areas
- Best for a few large elements

Arrows Example

```
CGPathRef path0 = random_arrow();
CGPathRef path1 = random_arrow();
```

```
CGRect shape_bounds
 = CGRectUnion(CGPathGetBoundingBox(path0),
               CGPathGetBoundingBox(path1));
```

```
CAShapeLayer *sublayer = [CAShapeLayer layer];
sublayer.fillColor = random_color();
sublayer.position = random_point(…);
sublayer.bounds = shape_bounds;
```

Arrows Example

```
CABasicAnimation *anim
 = [CABasicAnimation animationWithKeyPath:@"path"];
```

```
anim.fromValue = (id)path0;
anim.toValue = (id)path1;
```

```
anim.duration = random_float() * 3 + 1;
anim.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
anim.autoreverses = YES;
anim.repeatCount = HUGE_VAL;
```

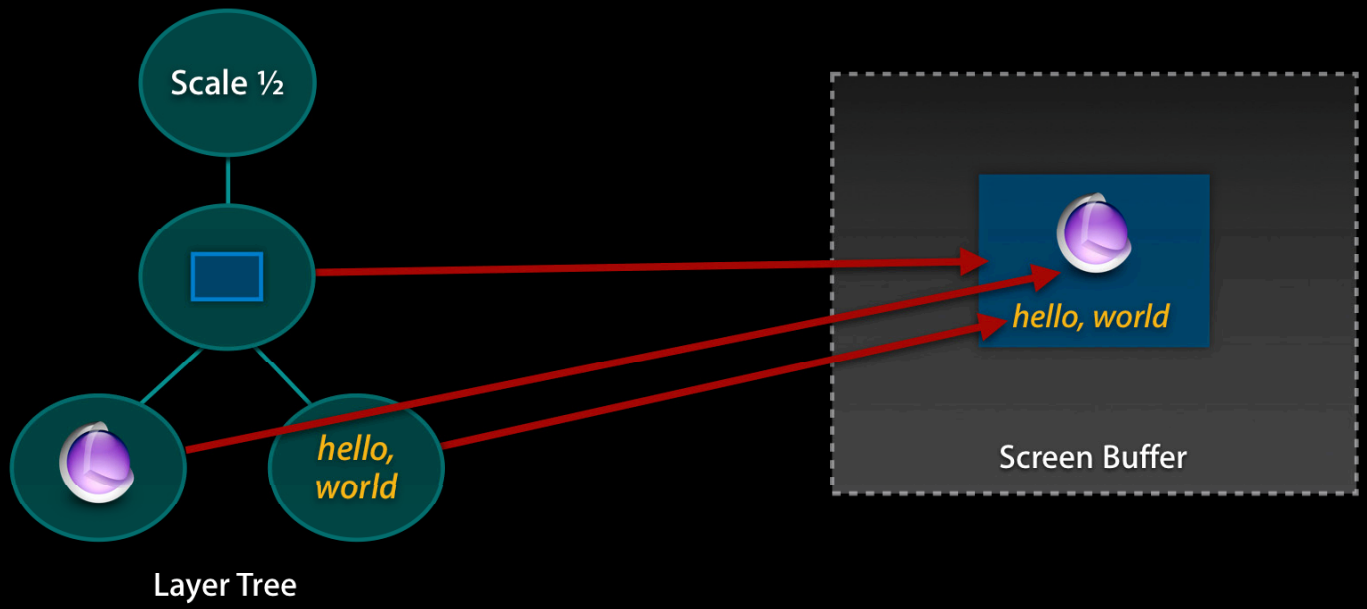
```
[sublayer addAnimation:anim forKey:nil];
```

Bitmap Caching

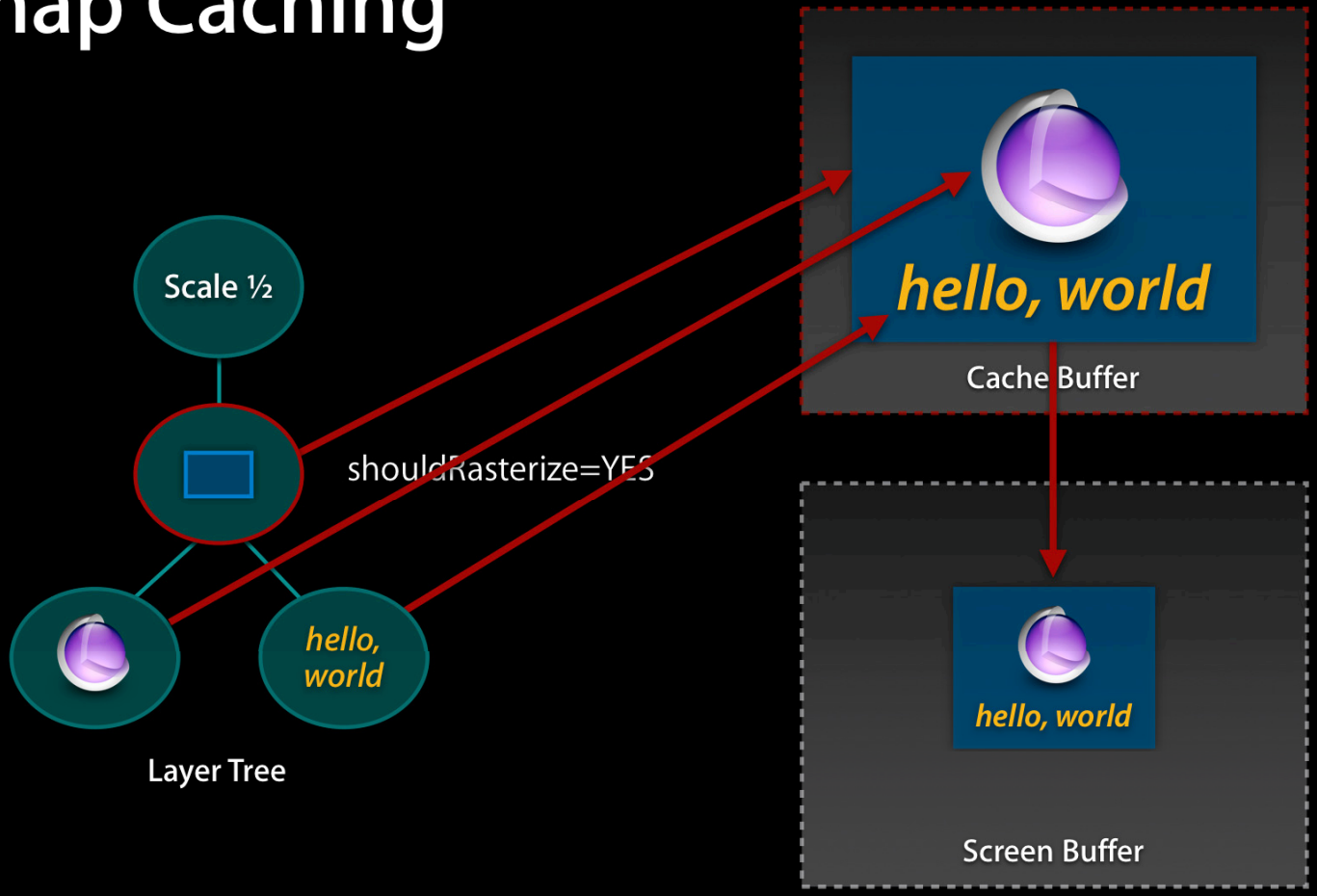
iPhone OS 3.2 or later

- Animated UIs on embedded devices can be challenging
- Can now request that a layer subtree is flattened to a bitmap
  - layer.shouldRasterize = YES
- Bitmap version will be reused when possible
  - May significantly improve performance

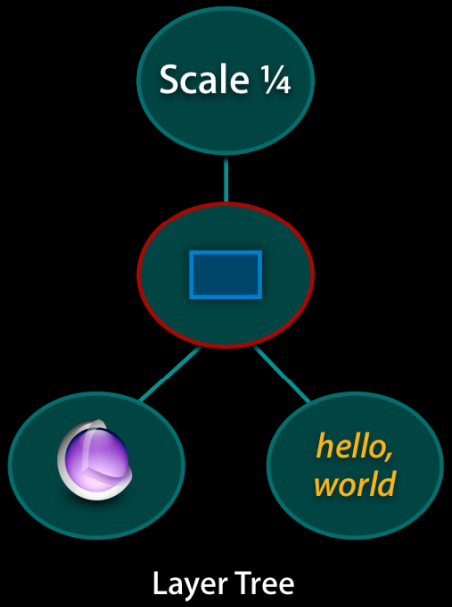
Bitmap Caching

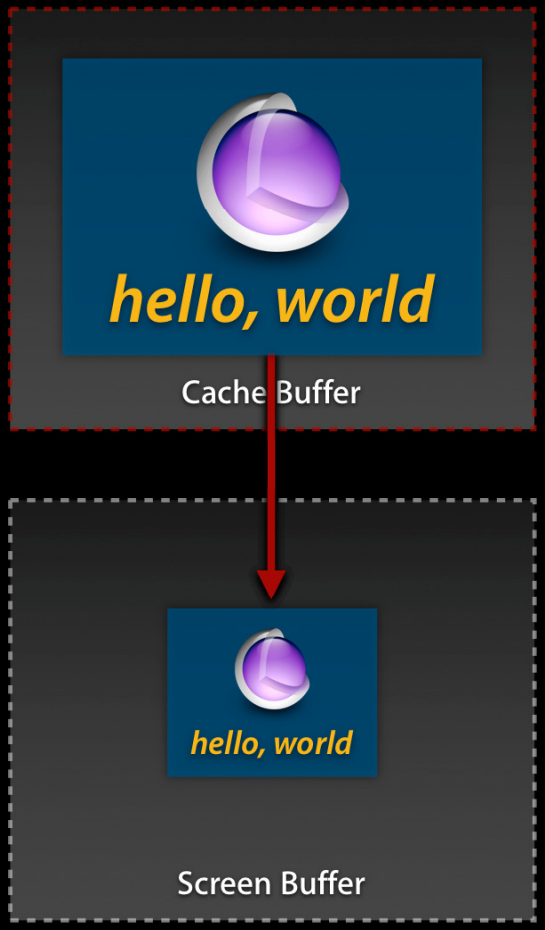


Bitmap Caching



Bitmap Caching





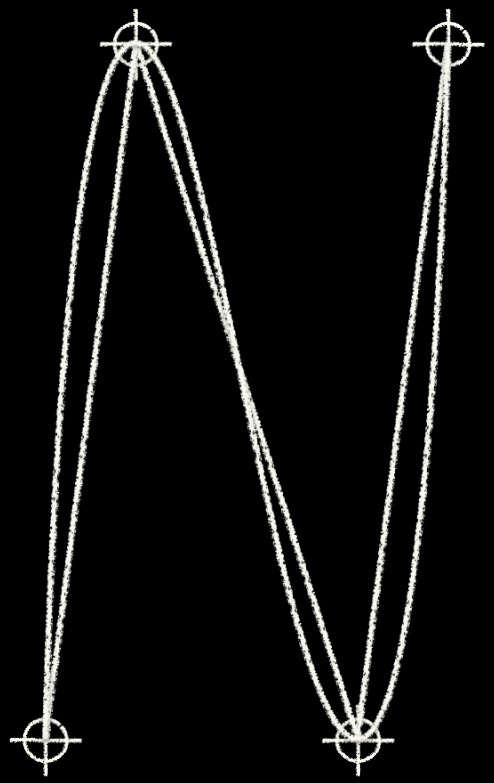
Bitmap Caching Caveats

- Devices are memory-challenged, so cache space is limited
- Caching and not-reusing is more expensive than not caching
- Rasterizing locks the layer image to a particular size
- Rasterization occurs before the mask is applied

Cubic Keyframe Interpolation

iOS 4

- Keyframe animations move properties through multiple points
- Smooth curves need timing functions or a path
- iOS 4 adds new calculation modes
  - anim.calculationMode = kCAAnimationCubic
- Will interpolate continuously through all points
  - Catmull-Rom spline, but you can customize this



Animation Functions

- Animating layer rotation has been problematic
  - Using “transform” property—angle is modulo 360°
  - Using “transform.rotation.z” property—Euler angle issues
- CAPropertyAnimation now has a valueFunction property
  - Animated property is set to a function of the interpolated value
  - layer.transform = *makeRotationMatrix*(t), where t ∈ [0…2π]

Animation Functions

```
CABasicAnimation *anim
 = [CABasicAnimation animationWithKeyPath:@"transform"];
```

```
anim.fromValue = [NSNumber numberWithDouble:0];
anim.toValue = [NSNumber numberWithDouble:2*M_PI];
```

```
anim.valueFunction
 = [CAValueFunction functionWithName:kCAValueFunctionRotateZ];
```

```
anim.duration = 2;
[layer addAnimation:anim forKey:nil];
```

Animation Completion

- Animations created explicitly can use a delegate
- Implicit animations can use a block

```
[CATransaction setCompletionBlock:^{
    // block that runs when animations have completed
    [CATransaction setDisableActions:YES];
    [layer removeFromSuperlayer];
}];
```

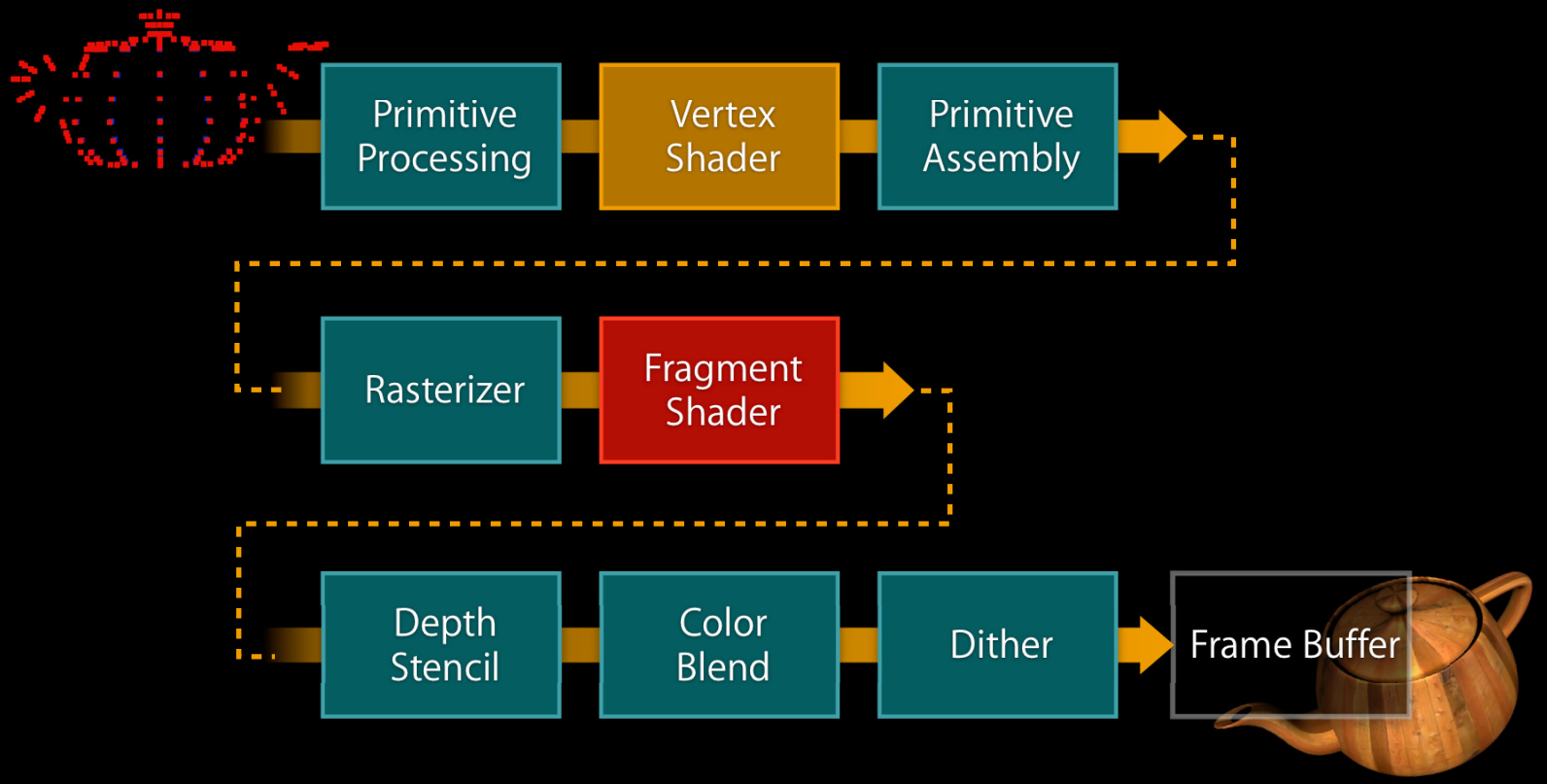
```
layer.opacity = 0;
layer.position = CGPointMake (2000, layer.position.y);
```

Summary

- Use shadowPath for high-performance shadows
- Use CAShapeLayer for scalable, animatable vector content
- Use shouldRasterize=YES for cached layers

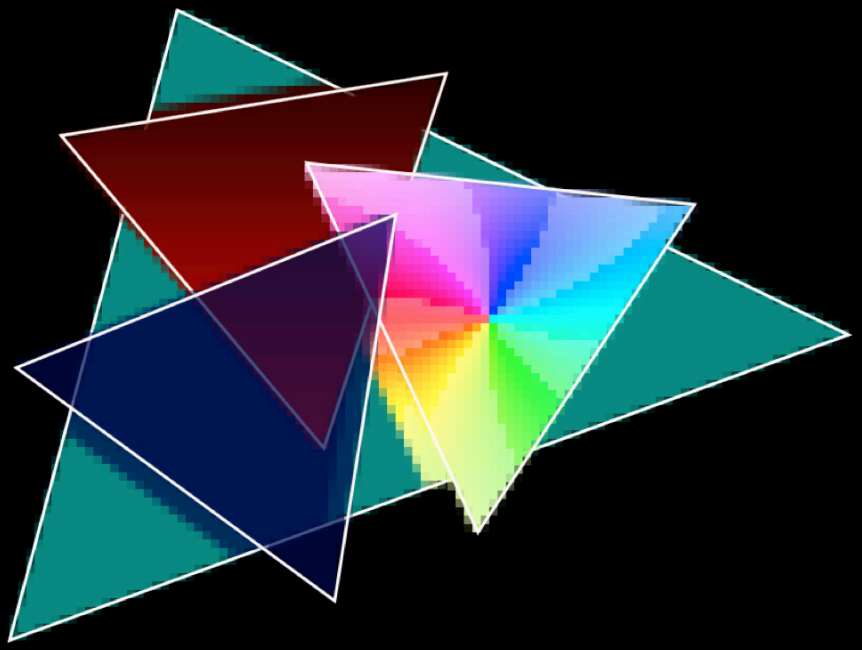
Performance

How Do GPUs Work?



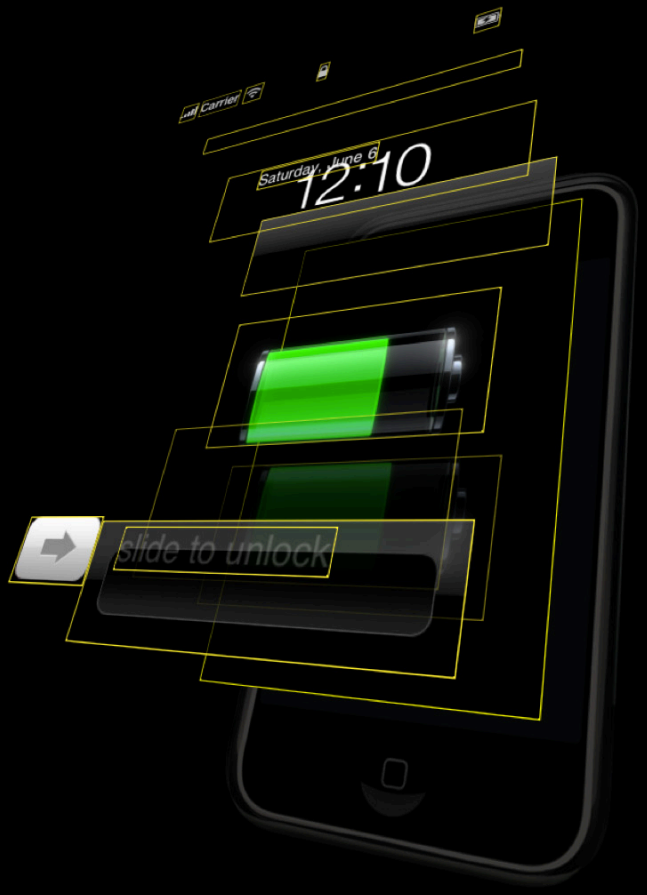
How Do GPUs Work?

- GPU converts triangles to pixels
  - Each is filled with a color or image
  - Each can “blend” over background
- Destination can also be an image



How Do We Use the GPU?

- CA translates your layers into triangles
  - "backgroundColor" is two colored triangles
  - "contents" is two triangles with an image
- Cached or masked layers draw offscreen
- Areas under opaque regions are ignored



GPU Performance Model

- What are the costs?
  - How many destination pixels?—Write bandwidth
  - How many source pixels?—Read bandwidth
  - How many buffers?—Rendering passes
- Too much non-opaque content → limited by write bandwidth
- Too many large images → limited by read bandwidth
- Too many masked layers → limited by rendering passes

Demo

Write Bandwidth

- Minimize alpha-blended pixels
- Use “Color Blended Layers” option, or CA_COLOR_OPAQUE=1
- Ensure opaque CGImageRef’s have no alpha channel
  - Set “layer.opaque = YES” for layers that draw opaque content
- Cut layers with opaque regions into multiple sublayers

Read Bandwidth

- Use images that match screen resolution
  - e.g., don't use 1024x768 image for 200x150 layer
- Use "Color Misaligned Images" option, or CA_COLOR_SUBPIXEL=1

Rendering Passes

- Ideally one rendering pass per frame
- Complex compositing features often require multiple passes
  - Masking, group opacity, filters
- Use "Color Offscreen" Instruments option, or CA_COLOR_OFFSCREEN=1
- Layer bitmap caching can hide extra passes
  - Unless the cached subtree changes during the animation!

Summary

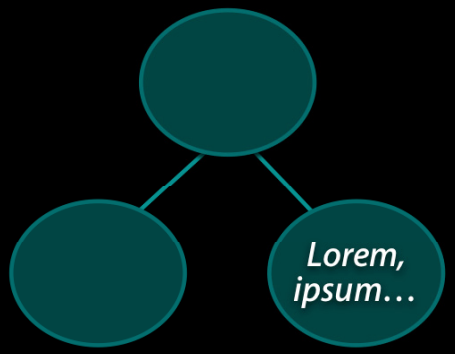
- Performance optimization algorithm

  while (fps < 60)
  - Eliminate rendering passes
  - Reduce read bandwidth
  - Reduce write bandwidth

High DPI

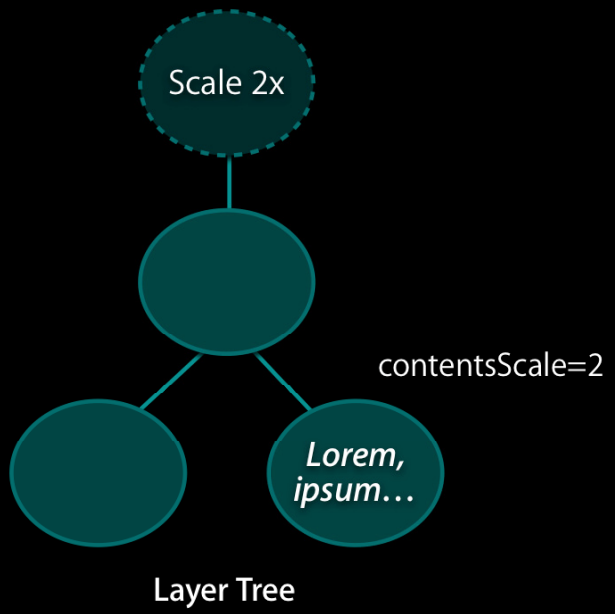
High DPI Content

iOS 4



Layer Tree

High DPI Content



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce aliquam tortor quis enim auctor id ultricies nisl consequat.

320
460

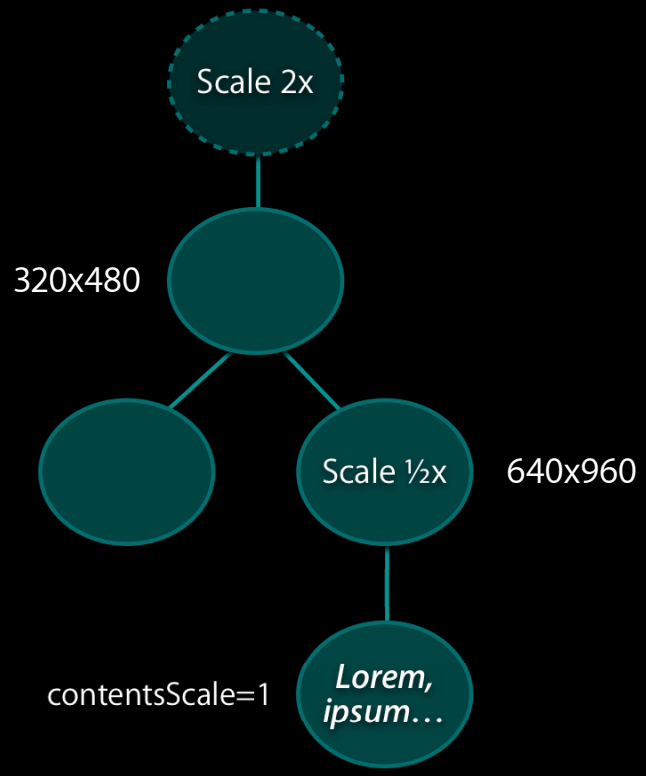
High DPI Content

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce aliquam tortor quis enim auctor id ultricies nisl consequat.

460

620

Native Pixels



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce aliquam tortor quis enim auctor id ultricies nisl consequat. Cras vel arcu vitae nunc fermentum egestas et in arcu. Etiam dictum, nunc sit amet imperdiet auctor, orci ipsum fermentum risus, et ornare nibh neque tempus nisl. Sed ligula libero, sollicitudin nec tincidunt nec, ultrices at erat. Fusce nec libero ut nibh interdum mollis. Nulla eget odio arcu, eget lacinia augue. Ut malesuada eleifend tempor.

960

640

High DPI Summary

- 2x scale factor applied to your UIWindow
  - All your view geometry remains relative to 320x480
  - Use contentsScale=2 for screen-resolution content
  - When rasterizing layers, layer.rasterizationScale=2
- To get back to the native 640x960 viewport
  - Use a scale=½ matrix to cancel the implicit scale=2 matrix

Wrap-Up

Summary

- Use shadowPath whenever possible
- Use shouldRasterize whenever necessary
- Consider what your layers mean to the GPU

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Mailing List
quartz-dev@.lists.apple.com

Documentation
http://developer.apple.com/graphicsimaging/coreanimation/

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Building Animation Driven Interfaces	Pacific Heights Thursday 9:00AM
Core Animation in Practice, Part 1	Nob Hill Thursday 11:30AM

Labs

Core Animation Lab	Graphics and Media Lab D Thursday 3:15PM
Animation Lab	Application Frameworks Lab C Thursday 4:30PM
Animation Lab	Application Frameworks Lab A Friday 9:00AM


WWDC10