Best Practices for OpenGL ES Apps on iOS

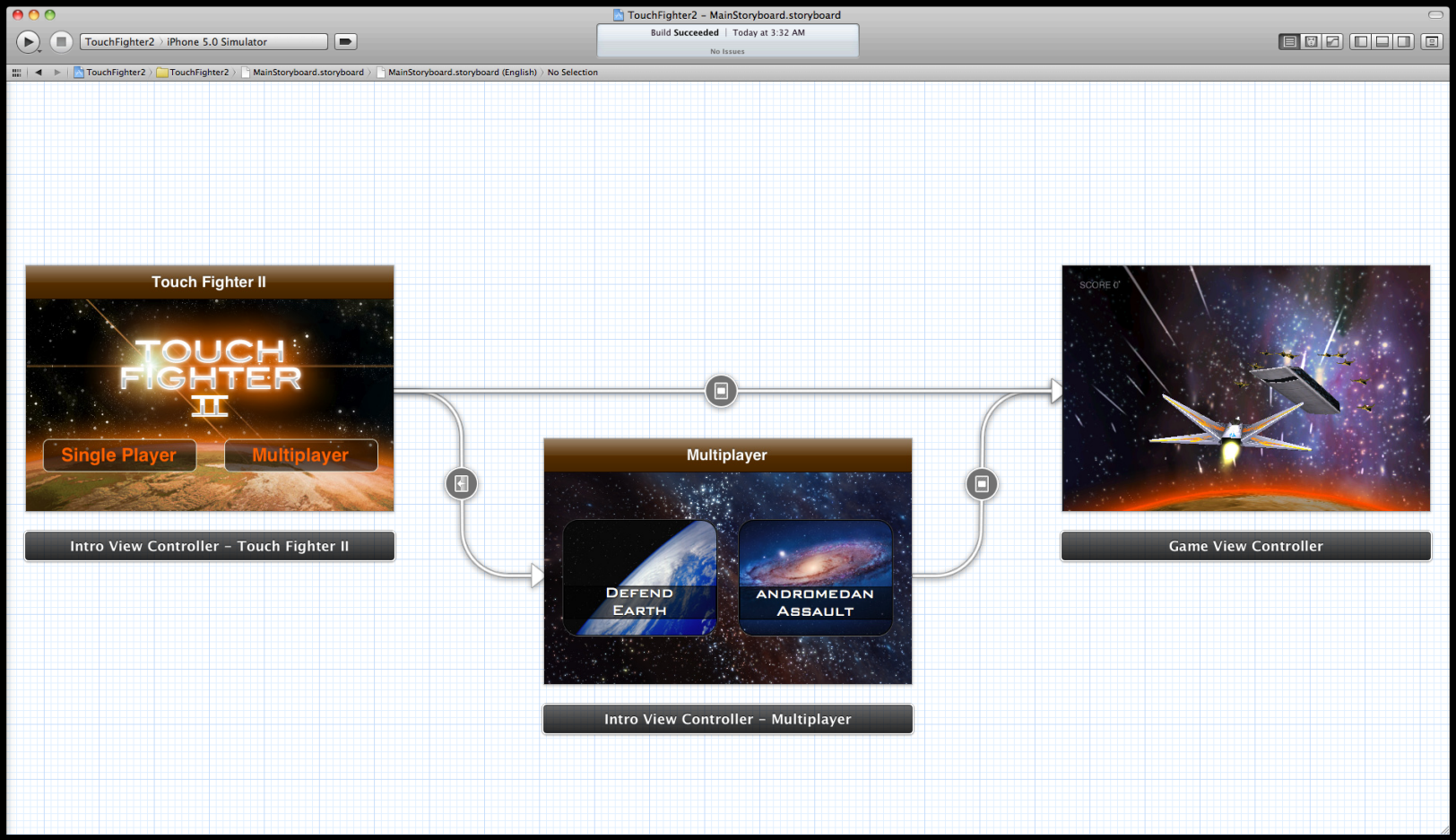
Session 418

Richard Schreyer
GPU Software

Agenda

- View Controllers
- Multithreading
- Screen sizes
- Taking advantage of iPad 2

View Controllers


TouchFighter2 – MainStoryboard.storyboard
TouchFighter2 › iPhone 5.0 Simulator
Build Succeeded | Today at 3:32 AM
No Issues
TouchFighter2 › TouchFighter2 › MainStoryboard.storyboard › MainStoryboard.storyboard (English) › No Selection
Touch Fighter II
TOUCH FIGHTER II
Single Player
Multiplayer
Intro View Controller – Touch Fighter II
Multiplayer
DEFEND EARTH
ANDROMEDAN ASSAULT
Intro View Controller – Multiplayer
SCORE 0
Game View Controller

View Controllers

View Controller responsibilities

- Creates and manages a collection of views
- Transitions to other View Controllers
- OpenGL-specific considerations
  - When to animate
  - Device rotation

View Controllers

Visibility

- Start and stop animation as the view's visibility changes

```
@interface UIViewController

- (void)viewWillAppear:
- (void)viewDidAppear:
- (void)viewWillDisappear:
- (void)viewDidDisappear:

@end
```

View Controllers

GLKViewController

- Drives an animation loop on the main thread
- Determines appropriate frame rate
- Stops animation when the view is obscured


Welcome back, LYRICYST
Rochambeau
PLAY
LEADERBOARD
ACHIEVEMENTS

View Controllers

Rotation

- View Controllers provide support for device rotation
  - Also handles status bar, banners, volume HUD, etc.
- View Controller rotation is free on iOS 4.3 and later
  - Now the best way to rotate OpenGL views
  - Remove error-prone manual rotation code

View Controller Rotation

Step 1: Declare ability to rotate

```
@implementation MyViewController

- (BOOL)shouldAutorotateToInterfaceOrientation:
            (UIInterfaceOrientation)interfaceOrientation
{
  //  Support all 4 orientations
  return YES;
}

@endif
```

View Controller Rotation

Step 2: Allocate correctly sized framebuffers

```
@implementation MyOpenGLView

//  Respond to changes in view dimensions by reallocating the framebuffer
- (void)layoutSubviews
{
  [self deleteFramebuffer];
  [self createFramebuffer];
}

@endif
```

View Controller Rotation

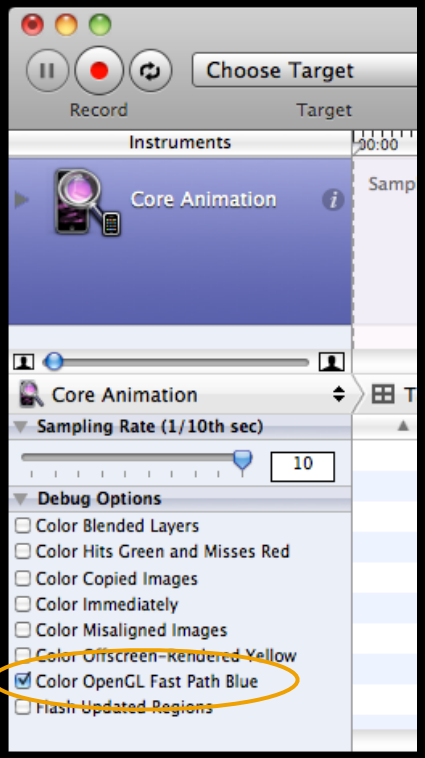
Step 3: Remove legacy rotation support

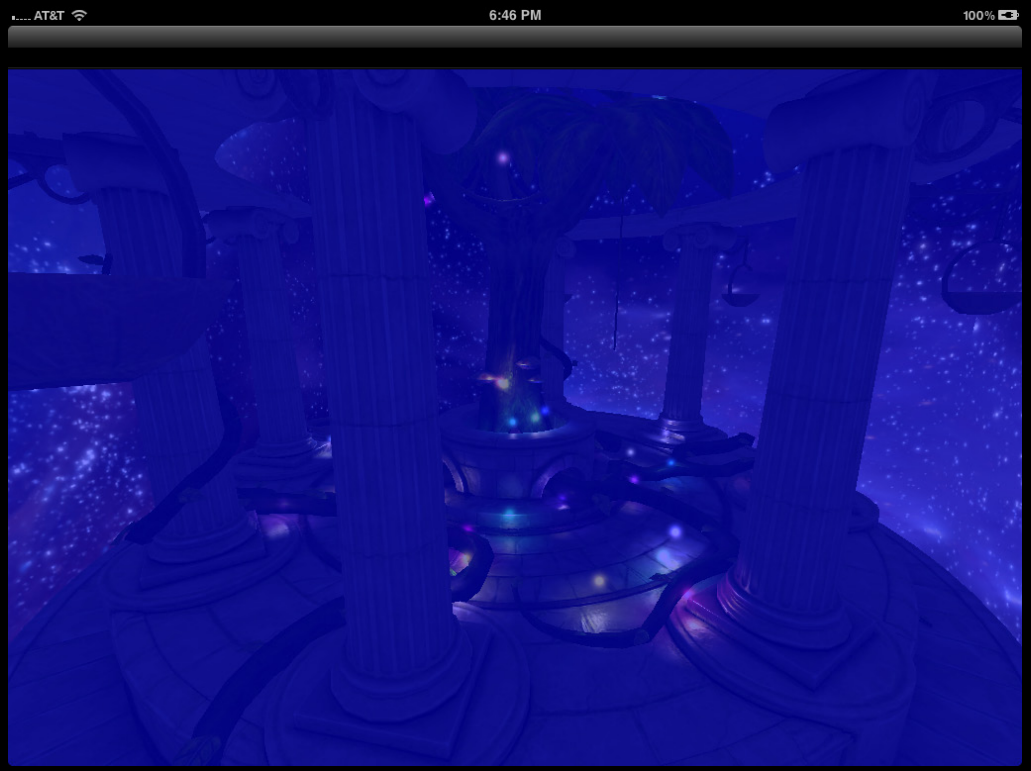
```
if (portrait)
  glViewport(0, 0, width, height);
else
  glViewport(0, 0, height, width);

if (landscape) {
  glTranslatef(160.0f, 240.0f, 0.0f);
  glRotatef(90.0, 0.0, 0.0, 1.0);
}
```

View Controller Rotation

Step 4: Verify efficient compositing





View Controllers

Summary

- Update OpenGL apps to use View Controllers
- GLKViewController manages animation timing
- View Controller rotation is free on iOS 4.3
- Use Instruments to verify performance

Multithreading

OpenGL and Multithreading

Threaded programming models for OpenGL applications

- Asynchronous resource loading
- Move entire game away from main thread

OpenGL and Multithreading

The current context

- The current context receives OpenGL commands
- The current context is per-thread state

```
[EAGLContext setCurrentContext: myContext];
glDrawArrays(...);    //  issued to myContext

[EAGLContext setCurrentContext: nil];
glDrawArrays(...);    //  illegal
```

OpenGL and Multithreading

OpenGL contexts are not thread safe

Loading Thread

```
[EAGLContext setCurrentContext: ctxA];
glTexImage2D(...);
```

Rendering Thread

```
[EAGLContext setCurrentContext: ctxA];
glDrawArrays(...);
```

OpenGL and Multithreading

OpenGL contexts are not thread safe

Loading Thread

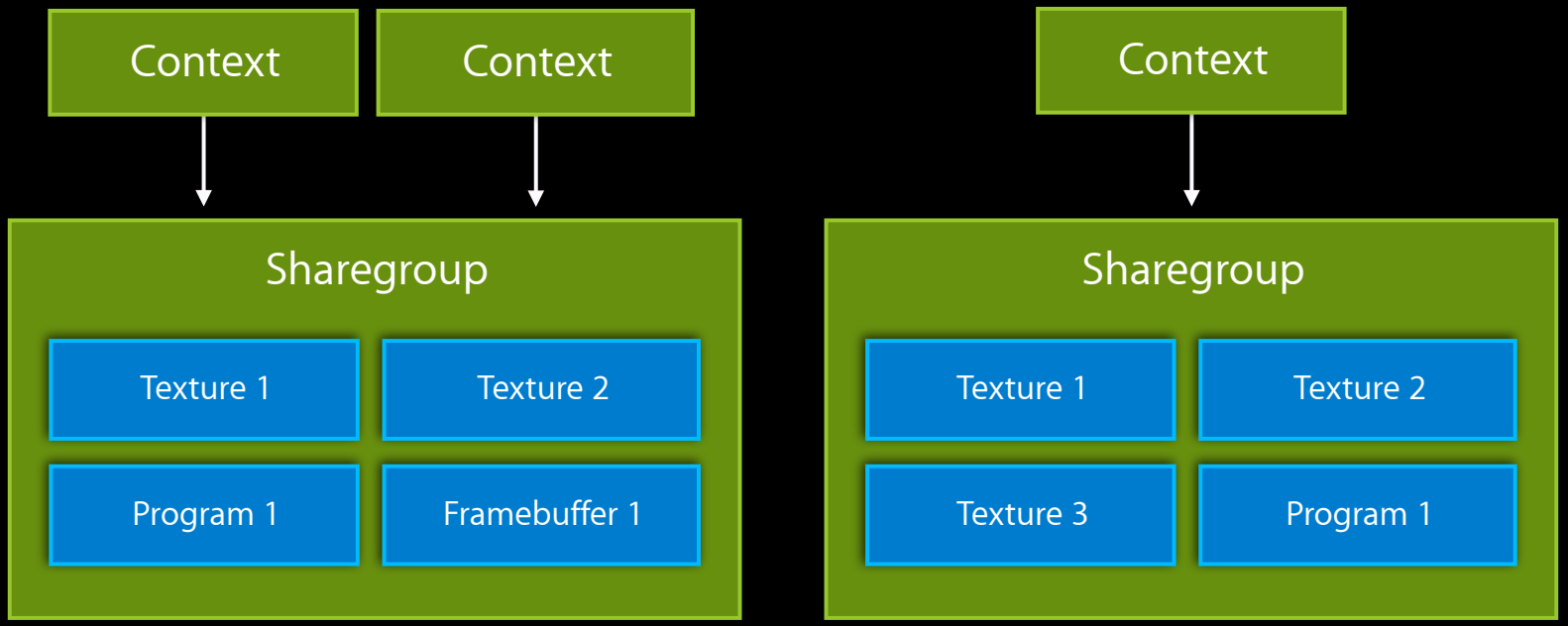
```
[EAGLContext setCurrentContext: ctxA];
glTexImage2D(...);
```

Rendering Thread

```
[EAGLContext setCurrentContext: ctxB];
glDrawArrays(...);
```

OpenGL and Multithreading

Shared contexts



OpenGL and Multithreading

Shared contexts

- A sharegroup is a collection of OpenGL objects
  - Textures, buffers, framebuffers, renderbuffers, shaders, programs
- Two contexts can read concurrently from an object

```
context1 = [[EAGLContext alloc]
            initWithAPI: kEAGLRenderingAPIOpenGLES2];

context2 = [[EAGLContext alloc]
            initWithAPI: kEAGLRenderingAPIOpenGLES2
            sharegroup: [context1 sharegroup]];
```

OpenGL and Multithreading

Shared contexts

- Some OpenGL commands modify objects
  - `glTexImage2D`, `glBufferSubData`, etc.
- After modifying an object
  - The modifying context calls `glFlush`
  - The observing contexts call `glBind*`

OpenGL and Multithreading

Shared contexts

- Other ways to modify objects
  - Changing a uniform of a GLSL program
  - Render to a framebuffer
  - EAGLContext renderbuffer methods
    - `-[EAGLContext renderbufferStorage: fromDrawable:]`
    - `-[EAGLContext presentRenderbuffer:]`

OpenGL and Multithreading

Asynchronous resource loading

Loading Thread	Rendering Thread
`[EAGLContext setCurrentContext: loaderCtx];`	`[EAGLContext setCurrentContext: renderCtx];`

`glGenTexture(1, &tex)`
`glBindTexture(GL_TEXTURE_2D, tex);`
`glTexImage2D(...);`
`glFlush();`	

	`glBindTexture(GL_TEXTURE_2D, tex);` `glDrawArrays(...);`

OpenGL and Multithreading

Asynchronous resource loading with Grand Central Dispatch

```
dispatch_async(loaderQueue,
^{
  [EAGLContext setCurrentContext: loaderContext];
  GLuint tex = CreateOpenGLTexture(...);
  glFlush();
  [EAGLContext setCurrentContext: nil];

  dispatch_async(renderQueue,
  ^{
     printf(“Texture %d Loaded”, tex);
     //  must bind ‘tex’ before using it
  });
}
```

OpenGL and Multithreading

Asynchronous resource loading with Grand Central Dispatch

```
dispatch_async(loaderQueue,
^{
  [EAGLContext setCurrentContext: loaderContext];
  GLuint tex = CreateOpenGLTexture(...);
  glFlush();
  [EAGLContext setCurrentContext: nil];

  dispatch_async(renderQueue,
  ^{
     printf("Texture %d Loaded", tex);
     //  must bind 'tex' before using it
  });
}
```

OpenGL and Multithreading

Asynchronous resource loading with Grand Central Dispatch

```
dispatch_async(loaderQueue,
^{
  [EAGLContext setCurrentContext: loaderContext];
  GLuint tex = CreateOpenGLTexture(...);
  glFlush();
  [EAGLContext setCurrentContext: nil];

  dispatch_async(renderQueue,
  ^{
     printf("Texture %d Loaded", tex);
     //  must bind 'tex' before using it
  });
}
```

OpenGL and Multithreading

Asynchronous resource loading with Grand Central Dispatch

```
dispatch_async(loaderQueue,
^{
  [EAGLContext setCurrentContext: loaderContext];
  GLuint tex = CreateOpenGLTexture(...);
  glFlush();
  [EAGLContext setCurrentContext: nil];

  dispatch_async(renderQueue,
  ^{
     printf("Texture %d Loaded", tex);
     //  must bind 'tex' before using it
  });
}
```

OpenGL and Multithreading

Why render on a background thread?

- Event loop control
- Avoid blocking the main thread
- Utilize multiple CPU cores

OpenGL and Multithreading

Rendering on a background thread

- Use a UIViewController subclass on the main thread
  - GLKViewController animation timer not required
  - Signal background thread to start/stop animation
- Issue all OpenGL commands from one thread
  - Watch for unexpected GL usage on the main thread

OpenGL and Multithreading

Watch for OpenGL calls on the main thread

Main Thread

```
@implementation MyGLView

- (void)layoutSubviews
{
   [self deleteFramebuffers];
   [self createFramebuffers];
}

@end
```

Rendering Task

```
animation loop
{
   glClear(...);
   glDrawElements(...);
   [ctx presentRenderbuffer];
}
```

OpenGL and Multithreading

Manage display renderbuffers on the rendering thread

Main Thread

```
@implementation MyGLView

- (void)layoutSubviews {
   needsResize = true;
}

@end
```

Rendering Task

```
animation loop {
   if (needsResize) {
      DeleteFramebuffers();
      CreateFramebuffers();
      needsResize = false;
   }
   glClear(...);
   glDrawArrays(...);
   [ctx presentRenderbuffer];
}
```

OpenGL and Multithreading

Threads and Grand Central Dispatch

- If using a simple animation loop, create one thread

```
while (!done) {
  update();
  render();
}
```

- Use Grand Central Dispatch for wider multithreading
  - Separate tasks evaluating physics, skinning, game logic, etc.
  - One task to perform all rendering

Related Sessions

Blocks and Grand Central Dispatch in Practice	Pacific Heights Wednesday 10:15AM
Mastering Grand Central Dispatch	Pacific Heights Thursday 10:15AM

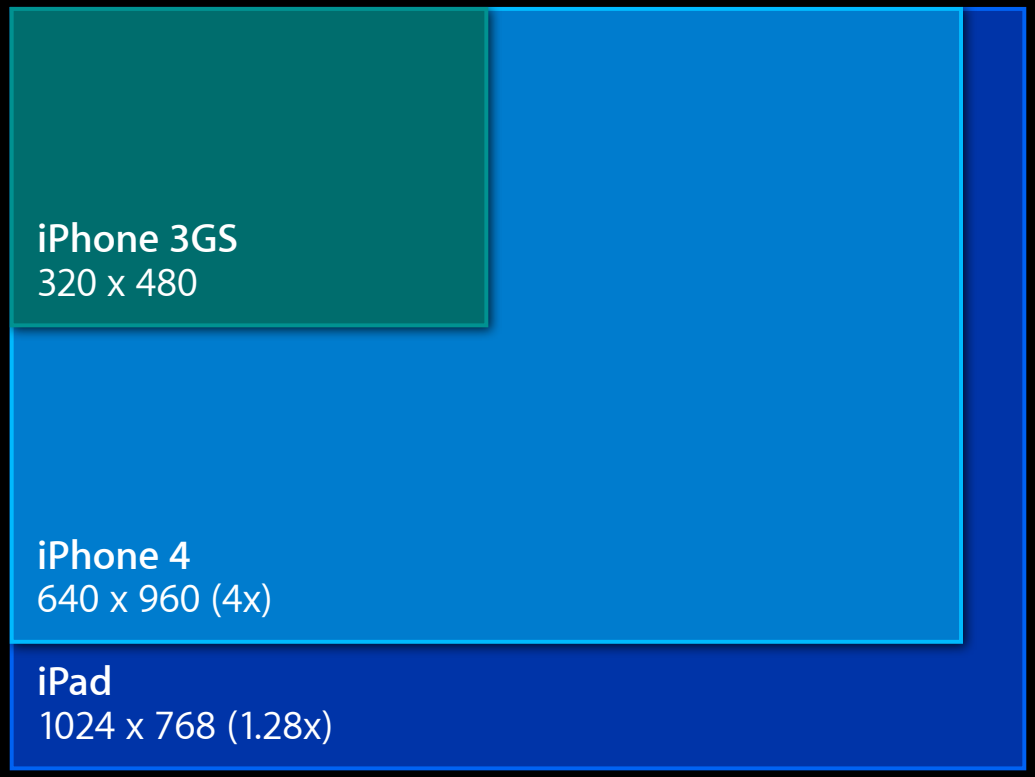
OpenGL and Multithreading

Summary

- OpenGL can be used from any thread
- Minimize concurrent usage of OpenGL
  - Render with a single GCD serial queue or thread
  - Use GLKTextureLoader for asynchronous texture loading

Screen Sizes

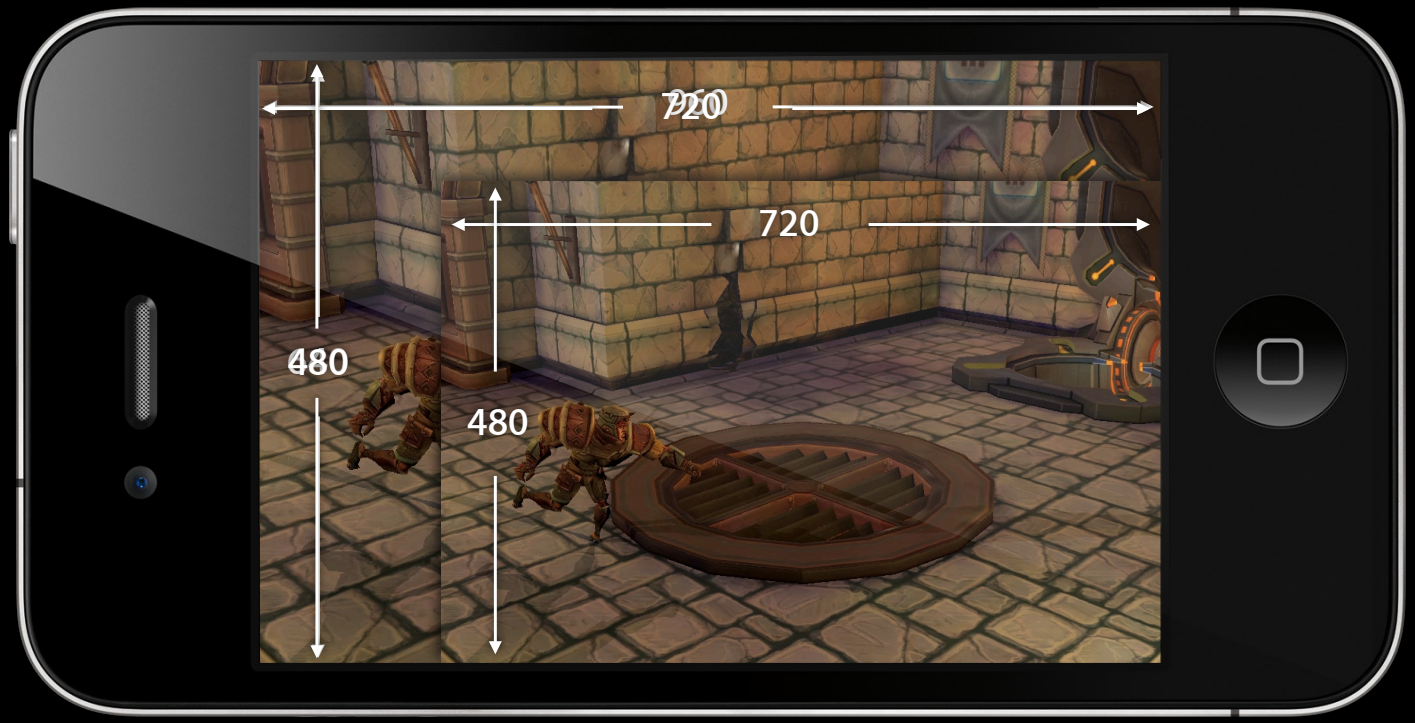
iOS Screen Sizes



Supporting Different Screen Sizes

Fill-rate optimizations

- Consider fragment shader complexity
  - Use smallest possible precision
  - Use smaller texture formats
  - Try replacing expensive math with texture lookup tables
  - Reduce use of discard, only draw after all opaque surfaces
- Consider fragment shader coverage
  - Blending defeats hidden-surface-removal features
- Check for excessive vertex load
- Reduce framebuffer width and height



56% as many pixels

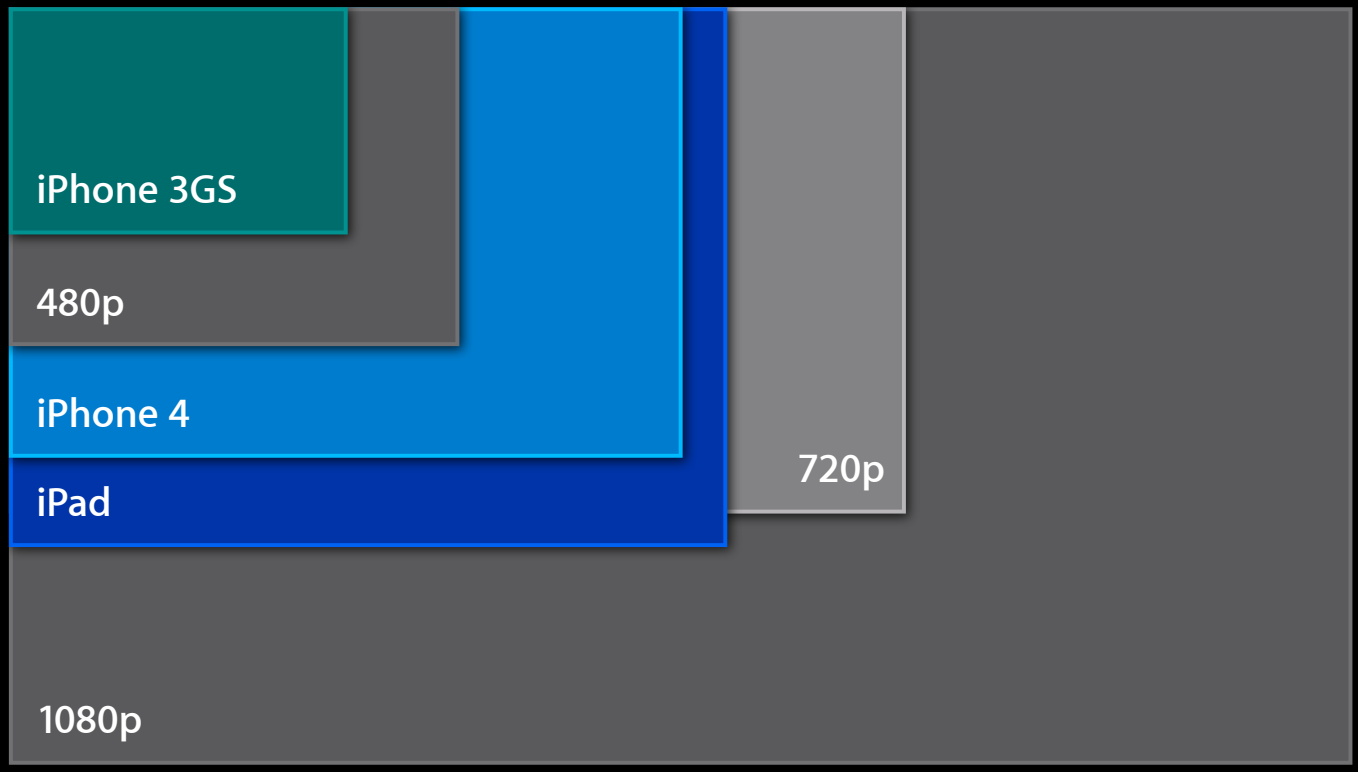
Supporting Different Screen Sizes

Scaling content

- Use Core Animation to scale up a smaller framebuffer to fill the display
- Set `UIView.contentScaleFactor` to less than `UIScreen.scale`
- Reallocate framebuffer after changing scale factor
- Scaling is efficient on iPhone 4 and iPad

Supporting Different Screen Sizes

External displays



Supporting Different Screen Sizes

External displays

- Mirrored display
  - Do nothing
- Second display
  - Choose a maximum display mode when developing your app (usually 720p)
  - Query `UIScreen.availableModes` for best match
  - Assign best match to `UIScreen.mode`

Related Sessions

AirPlay and External Displays in iOS Apps	Presido Tuesday 3:15PM

Texture Size Selection

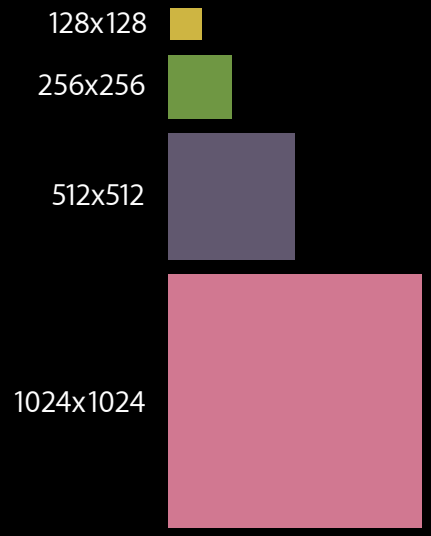
Selecting texture size for user interfaces

- Create image for specific target
- Can use UIImage naming convention
  - image~iphone.png
  - image@2x~iphone.png
  - image~ipad.png

Texture Size Selection

Selecting texture size for mipmapped textures

- Generate full set of mipmap images offline
- When loading, skip loading most detailed mipmap level
  - Drop levels based on memory or screen size
- May want to do this only to some less important textures



Visible
Mipmap Levels

320 x 480

1024 x 768

Supporting Different Screen Sizes

Summary

- Three internal screen sizes
- Larger framebuffers are more expensive
  - Use `UIView.contentsScaleFactor` to reduce resolution, if needed
- If supporting external displays, set the desired mode explicitly
- Drop base mipmap level to save memory

Taking Advantage of iPad 2

Taking Advantage of iPad 2

OpenGL feature additions in iOS 5

- Float16 texture filtering
- Float16 render targets, with blending
- Binary occlusion query
- Shadow filtering, 4-sample PCF
- 4096 x 4096 textures

- Use `GL_EXTENSIONS` string to check for features

Taking Advantage of iPad 2

OpenGL performance improvements

- Huge increase in GPU performance compared to iPad 1
  - Shader instruction rate, texture sample rate, memory bandwidth, etc.
- Higher performance when using medium and high GLSL precision
- Increased shader instruction/texture ratio

Taking Advantage of iPad 2

OpenGL performance improvements

- There is no query for OpenGL performance
- Best proxy is GPU model, queried with `glGetString(GL_RENDEDER)`

  `“PowerVR SGX 535”`

  `“PowerVR SGX 543”`

Taking Advantage of iPad 2

Scene complexity

- Enable multisample anti-aliasing, even at full resolution
- Enable more overdraw and blending
  - Higher density particle systems
  - More trees and foliage
- Larger number of draw calls in the scene
  - Faster CPU enables more draw calls, especially if multithreaded

Taking Advantage of iPad 2

Improved lighting and shadowing

- Per-pixel lighting
  - Use greater precision in shaders
  - High-resolution normal and gloss maps
  - Light probes
- Parallax mapping
- Shadows maps
  - Sample with `shadow2DProjAPPLE()`

Demo

Summary

- Updating OpenGL applications to use View Controllers
- OpenGL multithread design patterns
- Handling internal and external displays
- Using the performance of iPad 2

More Information

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

Documentation
OpenGL ES Programming Guide for iOS
http://developer.apple.com/iphone/

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

Labs

OpenGL ES Lab	Graphics, Media & Games Lab A Thursday 9:00AM
OpenGL ES Lab	Graphics, Media & Games Lab A Thursday 2:00PM


WWDC2011