iPhone
Graphics
Programming
Let’s Get Started!

FULL SPEED AHEAD
Multiple Graphics API’s

• There are many ways to do the same thing.
• Use the right tool for the job.
• Some API’s are easy to use, but slow (UIKit)
• Some API’s are hard to use, but fast (OpenGL ES)
### The Big Picture

<table>
<thead>
<tr>
<th></th>
<th>Toolkit</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard GUI</td>
<td>UIKit</td>
<td>Easy</td>
</tr>
<tr>
<td>Custom GUI</td>
<td>Quartz2D</td>
<td>Medium</td>
</tr>
<tr>
<td>2D Graphics</td>
<td>Quartz2D</td>
<td>Medium</td>
</tr>
<tr>
<td>2D Animation</td>
<td>CoreAnimation</td>
<td>Medium</td>
</tr>
<tr>
<td>2.5D Graphics</td>
<td>CoreAnimation</td>
<td>Medium</td>
</tr>
<tr>
<td>3D Graphics</td>
<td>OpenGL ES</td>
<td>Hard</td>
</tr>
<tr>
<td>3D Animation</td>
<td>OpenGL ES</td>
<td>Hard</td>
</tr>
</tbody>
</table>
First up API
UIKit

The ‘heart’ of Cocoa Touch

- User interface
  (Buttons, Sliders, Pickers)
- Events
  (User touch inputs)
- Views
  (Displaying content)
UIKit

- **Highest-level** interface available (easy to use)
- Drawing images
- Animate your UI:
  - Move, Scale, Rotate
  - Fade in, Fade out
  - Transitions
- Custom drawing
UIKit - Typical Uses

- Make a rectangle, fill colour (`UIView`)
- Display a string (`UILabel`)
- Image drawing (`UIImageView`)
- Limited Capability, but very easy to use
Demo

Available User Interface elements
in iPhone OS 3.0:
UICatalog

Demonstrate the User interface elements that are available to iPhone developers in the UICatalog demo

1–UICatalog
2D Graphics - Views

- View is a rectangular region you can place content inside

- What is content?
  Text, Images, Video, User interface controls
Displaying 2D Images

- A couple different ways to do this:
  - UIImageView class
  - UIView class

UIImageView – (Set with a UIImage)
UIView – (set background colour with an image, dodgy way)
Event handling (touch input from the user):
User interaction is disabled by default.
Need to set `userInteractionEnabled` property to YES after initializing the object.
`myView.userInteractionEnabled = YES;`
Here we have two UIImageView’s that have been added as sublayers to a UIView.
Display the Image

- Add your image (PNG) to Xcode
- Create a UIImageView (in Code or Interface Builder)

```objective-c
[myImageView setImage:[UIImage imageNamed:@"stars.png"]];
```

Write code or use Interface Builder.
Demo for: “How to display an image in a view”

Steps:
* Open Xcode, create a new UIView based Project
* Open ViewController.xib file
* Drag new UIImageView, set size: 550x550px
* Right click on Resources folder, add file: “stars.png”
* Create new UIImageView IBOutlet. Synthesize.
* Write code for ImageView: `[myStarImage setImage:[UIImage imageNamed:@"stars.png"]];`
* Connect the serialized IBOutlet to the pointer in code. Run :)
Demo Recap: Displaying an image in a view

* Xcode: Create a new UIView based Project.
* Xcode: Open 'ViewController.xib' file: Interface builder opens.
* Xcode: Drag a new UIImageView instance, set size: 550x550px.
* Xcode: Right click on Resources folder: add file: 'stars.png'.
* Obj-C: Create pointer for an IBOutlet UIImageView. Synthesize.
* Obj-C Set the image in -(void)viewDidLoad: delegate method.
* IB: Connect the UIImageView to the IBOutlet pointer in code.

Demo for: “How to display an image in a view”

Steps:
* Open Xcode, create a new UIView based Project
* Open ViewController.xib file
* Drag new UIImageView, set size: 280x280px
* Right click on Resources folder, add file: “stars.png”
* Create new UIImageView IBOutlet. Synthesize.
Custom drawing in a view

- Subclass `UIView`
- Override `-drawRect:`
- `drawRect:`
  Automatically called by UIKit (so long as you implement it)
**DrawRect:**

- If for some reason the UI doesn’t call your method frequently enough, you need to call `setNeedsDisplay`:
  - This informs UIKit to call `DrawRect:` as soon as possible
  - Calling `DrawRect:` yourself directly is unnecessary and can decrease performance.

Let the UIKit framework manage the calls to the drawing method.
Graphics APIs

UIKit  Quartz2D  CoreAnimation  OpenGL ES
First up API
Quartz 2D

- Best choice for 2D graphics
- C-based API
- A way to draw custom graphics
  - Vector graphics
  - Bitmap images
  - PDF content
How Quartz 2D works

- ‘Painter’ based drawing model
- You draw onto a Context
- Context maintains state
Graphics Context

- Quartz 2D can draw to many locations depending on the context
- Window or Layer
- PDF
- Bitmap
Painter’s Model

- Objects are ‘painted’ to a ‘canvas’
- Canvas is known as a context
- The draw order matters
- Objects blend automatically if transparent
Painter’s Model

Draw Order

Result

1 2

1 2
Drawing destinations

• Quartz 2D can draw to many locations depending on the context
• Window, Layer, PDF, Bitmap
Context

Getting a Context:

```c
CGContextRef myContext = UIGraphicsGetCurrentContext();
```

Saving a Context:

```c
CGContextSaveGState (myContext);
```

Restoring a Context:

```c
CGContextRestoreGState (myContext);
```
Coordinate System

- Flipped Cartesian
- (0,0) is top left corner

![Diagram showing a flipped Cartesian coordinate system with points (0,0), (-3,0), (10,2)]
Quartz 2D Capabilities

- Paths
- Lines, Arcs, Curves, Rectangles, Ellipses
- Transformations
- Colour, Gradients, Shadows
Quartz 2D Paths

Open Path

Solid

Dotted

Closed Path
Building Blocks of a Path

- **Lines** - Defined by start/end points
- **Points** - XY Coordinates
- **Arcs** - Circle segments (1/2 circle = Pi)
- **Rectangles** - Path draws around its edge
- **Curves** - Bezier with control points

Points – Used for the starting location of a Path
Line – A line is defined by its endpoints
Points – Used for the starting location of a Path
Line – A line is defined by its endpoints
Points – Used for the starting location of a Path
Line – A line is defined by its endpoints
Drawing Custom 2D

- Create a UIView subclass
- Override drawRect: to draw your content
- Remember to call setNeedsDisplay to update drawing to the screen
- Add your view to another view
Open the project in “3-Quartz2DDemo”

Steps:
* Create a new UIView
Quartz 2D Demo Recap

* Xcode: Create a new UIView based application
* Xcode: Create a new class that inherits from UIView
* Xcode: Write your Quartz2D drawing code in `drawRect`
* IB: Drag a new UIView instance to the window
* IB: Set the UIView’s class type to the UIView that you created
Graphics APIs

UIKit
Quartz2D
CoreAnimation
OpenGL ES
Core Animation

First up API
Core Animation

Core Animation

- 2D Compositing Framework
- Create Layers
- Attach animations to layers
When to use Core Animation?

- When you are limited by the higher level frameworks
- Add custom animations to views
- Use Core Animation for “immersive” interfaces
Getting Started

CALayer *l = [CALayer layer];
l.frame = CGRectMake( 0,0,100,100 );
[parentLayer addSublayer:l];
l.backgroundColor = [[UIColor redColor] CGColor];
l.contents = [[UIImage imageWithContentOfFile...]
l.cornerRadius = 40;
l.opacity = 0.5;
Core Animation Demo
2.5D

- 2D is graphics on an \(X,Y\) coordinate system
- Therefore 2.5D are 2D graphics \((X,Y)\) that look like they are 3D.
- Why? Because cheating saves time
2.5D Example

• Anything that is 2D made to look 3D
Graphics APIs

UIKit  Quartz2D  CoreAnimation  OpenGL ES
First up API
OpenGL ES

- OpenGL is very powerful
- OpenGL is a complicated API
- But the assumed knowledge is high
- Almost like learning a new language like SQL due to the complexity (Matrices, etc)
OpenGL ES

- OpenGL ES stands for:
- OpenGL for Embedded Systems
- Khronos Group oversees development of OpenGL ES
- Much like Canoncial oversees Ubuntu or Redhat oversees Fedora
Khronos Group

- You see these guys to read the API docs
- Lots of help on learning the API on their site
OpenGL ES

• Don’t confuse it with ‘OpenGL’
• Not the same thing.
• C++ is a superset of C
• OpenGL is a superset of OpenGL ES
• Cross Platform:
  iPhone, Mac OS X, PlayStation 3, Wii, Windows
OpenGL ES

Low Level 3D Graphics API

- High Performance (GPU)
- Subset of OpenGL API
- Difficult to code
- Requires knowledge of some math (Matrices, Vector math)
Why should I use it?

- Use it for drawing 3D objects only
- When you need high 3D performance (Games, Simulations)
- Don’t use OpenGL ES for 2D. OpenGL ES will be overkill (although it can do both 2D+3D)
OpenGL Capabilities

- Render 3D primitives (quads, triangles)
- Render 3D models
- Lighting
- Texture mapping
- Camera (your view of the world)
Why is it difficult?

• Like any API, you need to learn:
• The available API calls. Example:
  • glColorMaterial();
  • glEnable(GL_LIGHTING);
  • glBegin(GL_TRIANGLES);
• 3D Math
  • Vectors, Matrix transformations, Dot Product, Cross product, etc.
What’s a Triangle Strip?

- Series of interconnected triangles
- Why?
- Saves vertices
What’s a Triangle Strip?

- Simple Tri-Strip in OpenGL (ES):

```gl
begin(GL_TRIANGLE_STRIP);
vertex3f( 0.0f, 0.0f, 0.0f ); //vertex 1
vertex3f( 0.0f, 1.0f, 0.0f ); //vertex 2
vertex3f( 1.0f, 0.0f, 0.0f ); //vertex 3
vertex3f( 1.5f, 1.0f, 0.0f ); //vertex 4
end();
```

Three axes: x,y,z

2D example:

Although 3D is what its used for
OpenGL ES API

Draw a Triangle Strip!

```c
glViewport(0, 0, backingWidth, backingHeight);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(0.0f, (GLfloat)(sinf(transY)/2.0f), 0.0f);
transY += 0.075f; // animation

glClearColor(0.5f, 0.5f, 0.5f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glVertexPointer(2, GL_FLOAT, 0, squareVertices);

glEnableClientState(GL_VERTEX_ARRAY);

glColorPointer(4, GL_UNSIGNED_BYTE, 0, squareColors);

glEnableClientState(GL_COLOR_ARRAY);

glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```
OpenGL Fundamentals

• Triangle Strip vs Triangle Fan

Triangle Strip

Triangle Fan
OpenGL Demo
Where can I learn OpenGL ES?

- Official Site: www.khronos.org/opengles
- Google these:
  - Lighthouse3d.com
  - Stackoverflow.com
  - iDevGames
  - NeHe (although most of his code is bad)
Graphics APIs

UIKit
Quartz2D
CoreAnimation
OpenGL ES
One More Thing...

- Video and Media
- Playing Audio
- Playing Movies