iPhone Graphics

Kevin Cathey
iPhone Graphics

acm.uiuc.edu/macwarriors/devphone
iPhone Graphics

Frameworks

acm.uiuc.edu/macwarriors/devphone
iPhone Graphics

Frameworks

Common graphics concepts
iPhone Graphics

Frameworks

Common graphics concepts

CoreGraphics

acm.uiuc.edu/macwarriors/devphone
iPhone Graphics

Frameworks
Common graphics concepts
CoreGraphics
OpenGL ES
CoreAnimation

acm.uiuc.edu/macwarriors/devphone
iPhone Graphics

Frameworks

Common graphics concepts
CoreGraphics
OpenGL ES
CoreAnimation

acm.uiuc.edu/macwarriors/devphone
Why custom graphics?

• Most of the time UIKit classes are ample.
• But sometimes we want our own custom drawing.
Three graphics frameworks
Three graphics frameworks

Software

Hardware
Three graphics frameworks

- Software: Slow & simple
- Hardware: Fast & complex
Three graphics frameworks

CoreGraphics

Software

Slow & simple

Hardware

Fast & complex
Three graphics frameworks

- **CoreGraphics**
  - Software
  - Slow & simple

- **OpenGL ES**
  - Hardware
  - Fast & complex
Three graphics frameworks

- CoreGraphics: Software, Slow & simple
- CoreAnimation: Hardware, Fast & complex
- OpenGL ES: Hardware
iPhone Graphics

- Frameworks ✓
- Common graphics concepts
  - CoreGraphics
  - OpenGL ES
  - CoreAnimation

acm.uiuc.edu/macwarriors/devphone
Graphics contexts
Graphics contexts

• Graphics data should be independent from destination.
Graphics contexts

• Graphics data should be independent from destination.
• Contexts store state of your drawing — they are a state machine.
Graphics contexts

• Graphics data should be independent from destination.
• Contexts store state of your drawing — they are a state machine.
• Draw into a context by telling it to “add a line” or “add circle”
Graphics contexts

• Graphics data should be independent from destination.
• Contexts store state of your drawing — they are a state machine.
• Draw into a context by telling it to “add a line” or “add circle”
• Change state by changing colors, translating, scaling, …
  ▪ Changes persist until you change them again.
Graphics Contexts
Graphics Contexts

• Drawing commands in context change its backing store.
Graphics Contexts

• Drawing commands in context change its **backing store**.
• When you want to display, take data and send it somewhere.
  ▪ Windows (on screen)
  ▪ Images
  ▪ PDF documents
Graphics Contexts

- Drawing commands in context change its **backing store**.
- When you want to display, take data and send it somewhere.
  - Windows (on screen)
  - Images
  - PDF documents
- Different context types live at different levels
  - Software
  - Hardware
Transformation Matrices
Transformation Matrices

- Graphics is linear algebra intensive.
Transformation Matrices

• Graphics is linear algebra intensive.
• Every operation is representable by a matrix.
Transformation Matrices

- Graphics is linear algebra intensive.
- Every operation is representable by a matrix.
- All contexts have a transformation matrix.
Transformation Matrices

- Graphics is linear algebra intensive.
- Every operation is representable by a matrix.
- All contexts have a transformation matrix.
- Every drawing operation is multiplied by the transformation matrix to produce final location, angle, color, etc.
Transformation Matrices

- Graphics is linear algebra intensive.
- Every operation is representable by a matrix.
- All contexts have a transformation matrix.
- Every drawing operation is multiplied by the transformation matrix to produce final location, angle, color, etc.
- Powerful because you can focus on local drawing.
Transformation Matrices

- Graphics is linear algebra intensive.
- Every operation is representable by a matrix.
- All contexts have a transformation matrix.
- Every drawing operation is multiplied by the transformation matrix to produce final location, angle, color, etc.
- Powerful because you can focus on local drawing.
- Learn this and save tons of time and complexity.
Transformation matrices

• Stack based (kind of like autorelease pools)
  ▪ Stack of matrices is multiplied together.
  ▪ **Push** (adds new identity matrix to stack)
  ▪ **Pop** (pulls top matrix off stack)
  ▪ Again — worry about local drawing.
Transformation Matrices

Example — some something 40 pixels up & left

\[
\begin{bmatrix}
1 & 0 & 0 & -40 \\
0 & 1 & 0 & 40 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \times \begin{bmatrix}
20 \\
0 \\
0 \\
1 \\
\end{bmatrix} = \begin{bmatrix}
-20 \\
40 \\
0 \\
1 \\
\end{bmatrix}
\]
iPhone Graphics

- Frameworks ✓
- Common graphics concepts ✓
- CoreGraphics
- OpenGL ES
- CoreAnimation

acm.uiuc.edu/macwarriors/devphone
CoreGraphics

• Software based rendering.
• Relatively simple, but (relatively) slow too.
• Very high sample rate produces beautiful vector graphics.
• In straight C.
• Default graphics framework when drawing in UIView instances.
CoreGraphics

• Transformation matrix is only 2D (affine transformations)
  ▪ Translate
  ▪ Scale
  ▪ Rotate

• Drawing primitives
  ▪ Lines
  ▪ Rectangles
  ▪ Ellipses (arcs)
  ▪ Quadratic bezier paths
CoreGraphics

• Steps to drawing with CoreGraphics
  (1) Create a custom view subclass.
  (2) Override drawRect:
  (3) Get current graphics context
  (4) Change transformation matrix (if necessary)
  (5) Do drawing commands
• Drawing automatically sent (flushed) to screen when run loop runs back around.
CoreGraphics

Graphics context concepts

• Two kinds
  ▪ Bitmap
  ▪ PDF (for drawing vector graphics)

• Part of current state is a path
  ▪ Drawing commands act on the current path
CoreGraphics
Primitive Drawing Functions

• **Getting the graphics context**
  • `UIGraphicsGetCurrentContext` — gets current CGContextRef graphics context

• **Setting colors**
  • `CGContextSetRGBFillColor` — sets current fill color with RGBA
  • `CGContextSetRGBStrokeColor` — sets RGBA color as current color stroking (outlining) color
CoreGraphics

Primitive Drawing Functions

• Filling and stroking (outlining) current path
  - CGContextFillPath — fills the current path with the color set by CGContextSetRGBFillColor (or variant)
  - CGContextStrokePath — strokes (outlines) the current path with the color set by CGContextSetRGBStrokeColor (or variant)
CoreGraphics

Primitive Drawing Functions

• Adding to the current path
  • `CGContextAddRect` — adds a rectangle to the the context's current path
  • `CGContextAddLineToPoint` — adds a line to the context's current path
  • `CGContextAddQuadCurveToPoint` — adds a bezier curve to the current path
  • `CGContextClosePath` — draws lines back to the start of the current path
• You can also create a CGPath and add it to the context using `CGContextAddPath`. 
CoreGraphics

Non-primitive Drawing Functions

• Drawing images
  • CGContextDrawImage — draws a CGImage in the context,
  • Can also use UIImage convenience methods.

• Drawing gradients
  • CGContextDrawLinearGradient — draws a linear (uni-directional) gradient
CoreGraphics

Transformation matrix functions

- **CGContextConcatCTM** — multiples current transformation matrix (CTM) by another transformation matrix.
- **CGContextTranslateCTM** — convenience function for translating the current CTM
- **CGContextScaleCTM** — convenience function for scaling the current CTM
- **CGContextRotateCTM** — convenience function for rotating the current CTM
CoreGraphics

UIKit convenience functions/methods

• **UIRectFill** — fills a rectangle
• **UIRectFrame** — draws a line around a rectangle
• `[UIColor set]` — sets given color as current color
OpenGL

A few notes on OpenGL

• Hardware rendering.
• Very complex, but very fast and powerful.
OpenGL

Multiple transformation matrices

• **Model** — how model relates to the world
• **View** — how the world relates to the origin
• **Projection** — How the eye perceives the world.
• **Window to viewport** — How the graphics card generates pixels from window coordinates
• These get multiplied together to create the graphics pipeline
OpenGL

Graphics pipeline

• Model coordinates
• World coordinates
• Perspective
• Clipping
• Hardware window coordinates
• Viewport coordinates — actual pixels on screen
OpenGL

• But all these matrix - matrix multiplies, isn’t that slow?
• No. Hardware has many many more processing units than CPUs.
• These are built for matrix - matrix and matrix - vector multiplies.
OpenGL on the iPhone — OpenGL ES
OpenGL on the iPhone — OpenGL ES

• Subset of OpenGL.
OpenGL on the iPhone — OpenGL ES

- Subset of OpenGL.
- Optimized for mobile GPU performance.
OpenGL on the iPhone — OpenGL ES

• Subset of OpenGL.
• Optimized for mobile GPU performance.
• Missing convenience functions and capabilities
  ▪ No anti-aliasing.
  ▪ No glu utility functions (camera movement or perspective). You have to write these yourself.
  ▪ No automatic texture generation. Have to write environment mapping (pseudo reflections) by yourself.
  ▪ Only two simultaneous textures on GPU.
  ▪ All drawing commands done with vertex, coordinate, and color arrays — more at one instance is better.
OpenGL ES

- Use the starting template!
- Too complex to cover in this session.
- Read the spec: http://www.khronos.org/registry/gles/specs/1.1/es_full_spec.1.1.12.pdf
OpenGL ES

Simple example — draws triangle

```c
// 200 lines of setup above this

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrthof(-1.0f, 1.0f, -1.5f, 1.5f, -1.0f, 1.0f);

glClearColor(0.0, 1.0, 0.7, 1.0);
glClear(GL_COLOR_BUFFER_BIT);

GLfloat triangle[] = {
    -0.5, -0.5,     /* lower left corner */
    -0.5, 0.5,      /* upper left corner */
    0.5, -0.5};      /* lower right corner */

glVertexPointer(2, GL_FLOAT, 0, triangle);
glEnableClientState(GL_VERTEX_ARRAY);

glColor4f(1.0, 0.7, 0.0, 1.0);

glDrawArrays(GL_TRIANGLE_STRIP, 0, 3);
```
iPhone Graphics

- Frameworks ✓
- Common graphics concepts ✓
- CoreGraphics ✓
- OpenGL ES ✓
- CoreAnimation
CoreAnimation
CoreAnimation

• Combination between CoreGraphics and OpenGL.
CoreAnimation

• Combination between CoreGraphics and OpenGL.
• Uses concept of layer hierarchy instead of view hierarchy.
CoreAnimation

• Combination between CoreGraphics and OpenGL.
• Uses concept of layer hierarchy instead of view hierarchy.
• Draw CoreGraphics content into **CALayer**.
  ▪ Graphics data then cached on the hardware.
  ▪ If you only do transformations with matrix, no need to re-draw content.
  ▪ This is how UIKit works.
CoreAnimation

• Combination between CoreGraphics and OpenGL.
• Uses concept of layer hierarchy instead of view hierarchy.
• Draw CoreGraphics content into `CALayer`.
  ▪ Graphics data then cached on the hardware.
  ▪ If you only do transformations with matrix, no need to re-draw content.
    ▪ This is how UIKit works.
• Could also draw straight OpenGL content.
CoreAnimation

- Combination between CoreGraphics and OpenGL.
- Uses concept of layer hierarchy instead of view hierarchy.
- Draw CoreGraphics content into **CALayer**.
  - Graphics data then cached on the hardware.
  - If you only do transformations with matrix, no need to re-draw content.
    - This is how UIKit works.
- Could also draw straight OpenGL content.
- Transformation matrix is true 3D.
Layers

- Layers are light-weight views.
- Have their own geometry — size and position.
- Have their own transformation matrix.
- Have basic properties like background color, outline, and corner radius.
Layers

- Layer subclasses for different purposes:
  - **CAScrollLayer** — scrolling layer like a scroll view.
  - **CATextLayer** — convenience for rendering text.
  - **CAOpenGLLayer** — render OpenGL content.
  - A few more...
Core ANIMATION
Core ANIMATION

- CoreAnimation supports automatic animation between layer states.
Core ANIMATION

- CoreAnimation supports automatic animation between layer states.
- You say move and how far, it automatically spawns threads to animate your layer independent of your run loop (non-blocking).
Core ANIMATION

• CoreAnimation supports automatic animation between layer states.

• You say move and how far, it automatically spawns threads to animate your layer independent of your run loop (non-blocking).

• You can animate the transformation matrix or any of the layer’s properties.
Core ANIMATION

• CoreAnimation supports automatic animation between layer states.
• You say move and how far, it automatically spawns threads to animate your layer independent of your run loop (non-blocking).
• You can animate the transformation matrix or any of the layer’s properties.
• This is extremely powerful — again, this is how UIKit gets such great performance.
# iPhone Graphics

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common graphics concepts</td>
<td>✓</td>
</tr>
<tr>
<td>CoreGraphics</td>
<td>✓</td>
</tr>
<tr>
<td>OpenGL ES</td>
<td>✓</td>
</tr>
<tr>
<td>CoreAnimation</td>
<td>✓</td>
</tr>
</tbody>
</table>

[acm.uiuc.edu/macwarriors/devphone](acm.uiuc.edu/macwarriors/devphone)
About These Slides

If you need to reference the content of this presentation, these slides will be posted on our website, listed below:

acm.uiuc.edu/macwarriors/devphone
macwarriors