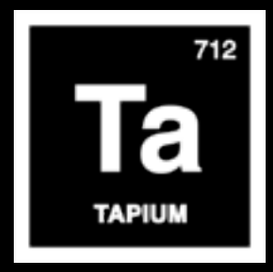



# iPhone OpenGL ES Crash Course

Richerd Chan

# Today's Agenda

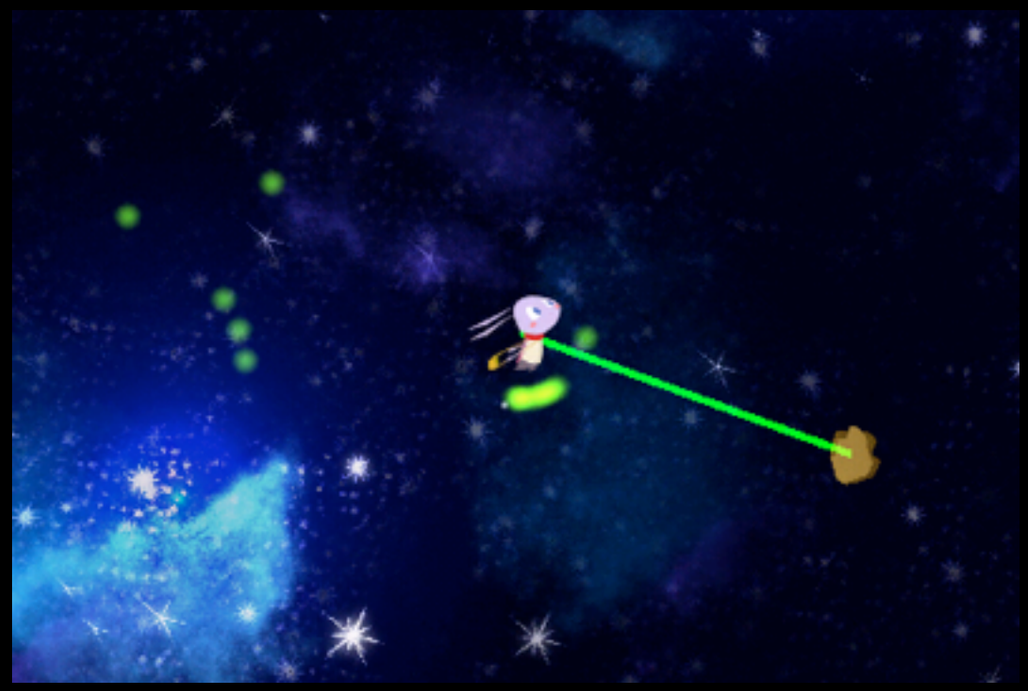
1. OpenGL
2. Graphics Theory
3. OpenGL ES Tutorial
4. iPhone OpenGL ES
5. Resources
6. Questions

# OpenGL

# OpenGL
**Open G**raphics **L**ibrary

Cross platform API for creating 2D/3D graphics

Standard API that define a set of graphic functions



Grow • Tapium

2D


Real Racing • Firement

3D

# Open GL

## What does it do?

- Rendering and Display to a screen
- Graphics math, calculations, and optimizations
- Provides a standard programming interface to the GPU
- A very powerful way to create graphics

# OpenGL ES
**Open G**raphics **L**ibrary **E**mbedded **S**ystems

- A Subset of Open GL
- Designed for Mobile Devices:

  Low Processing Power, Limited Memory, Limited Battery, Low Resolution
- Found on:

  Phones - iPhone, Android

  Consoles - Playstation 3
- Current Versions 1.0, 1.1, 2.0
- Not Backwards Compatible

# Open GL vs Open GL ES

- Immediate Mode Removed - glBegin/glEnd
- Fixed function - no shaders
- No GLUT - GL Utility Toolkit
- See Khronos OpenGL vs Open GL ES - API Walk through for the full detail of differences

# Open GL ES for the iPhone

## Information

- Open GL ES 1.1
- GPU: PowerVR MBX Lite 3D
- UIView Subclass
- Higher Speed/Performance/Control over Quartz, Core Animation, UIKit
- Biggest optimization you can use for Rendering

# OpenGL ES for iPhone

## When to use

- Games 2D/3D
- Custom Effects
- Custom Graphics / UI
- Cross Platform Applications / Porting
- Whenever you need graphics performance and speed

# Graphics Theory

# Graphics Topics
## For OpenGL

Coordinate Systems

Points / Vertex

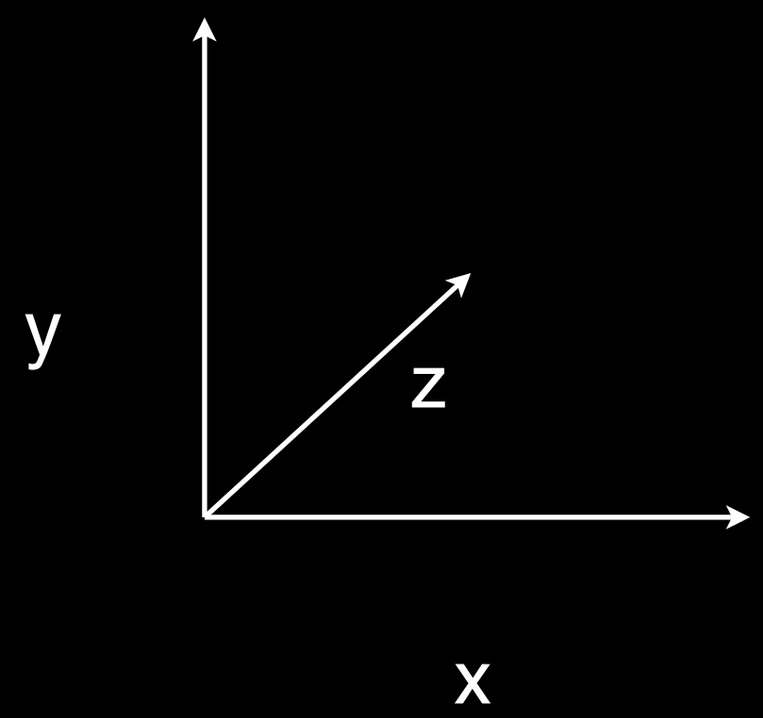
Triangles

View Ports

# Coordinate System

## Cartesian coordinate system

3D Space defined by an x, y, z axis with an origin being the intersection of the three axis (0, 0, 0)



# Point / Vertex

Point - A Location in space relative to the origin defined by its distance x, y, z.  eg. (x, y, z) = (1, 0, 1)

In OpenGL a vertex is the term used to define a point

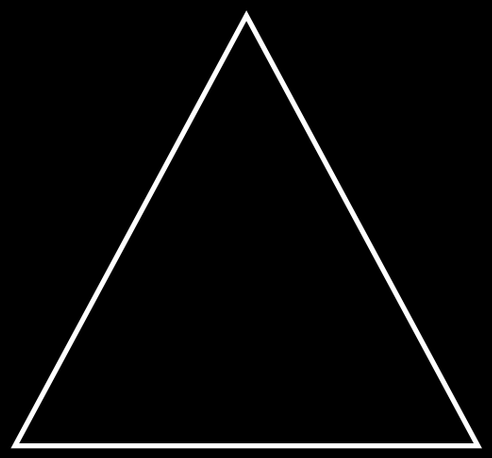
A Vertex is usually a corner point of a triangle

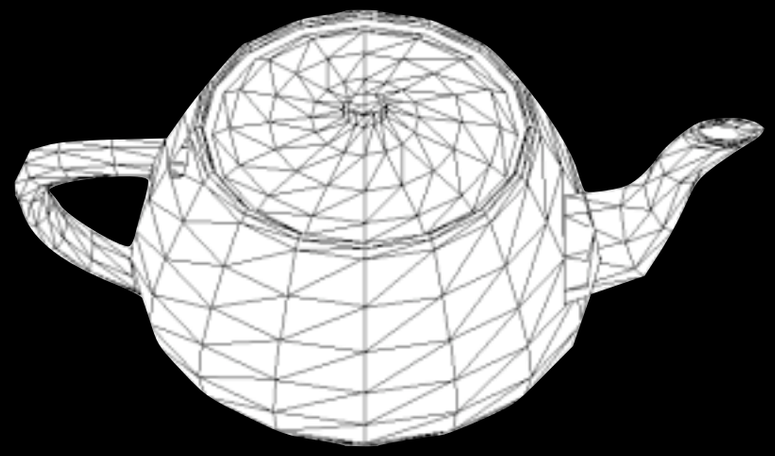
# Triangles

Area defined by 3 vertices - smallest amount of data required to create a surface

Basic building block in Open GL

Any model / shape can be built from a collection of triangles





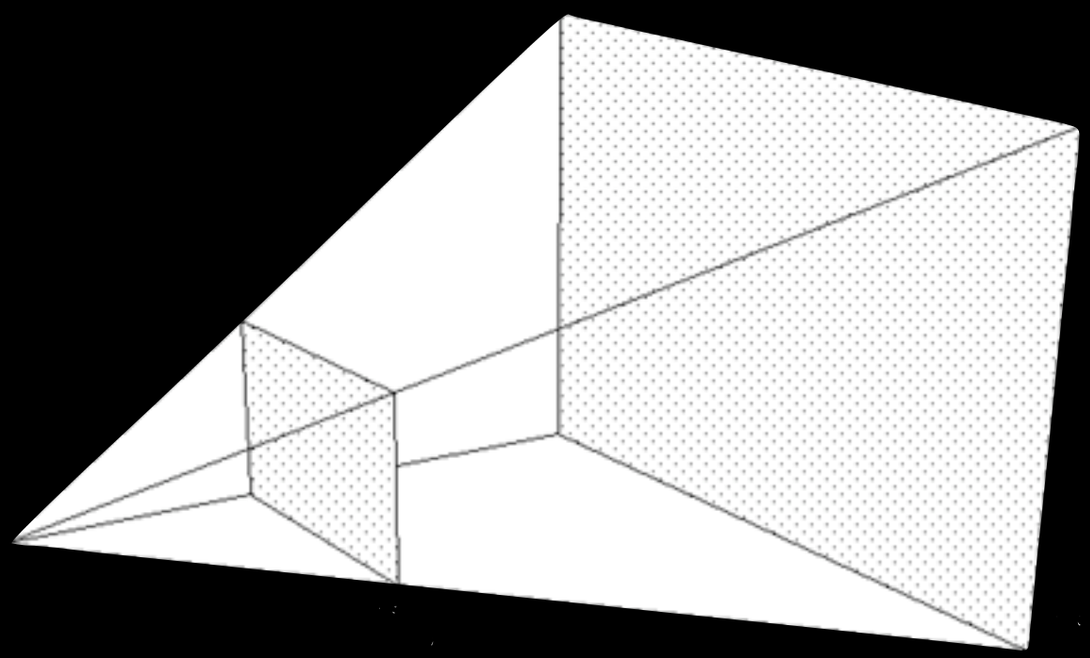
# View Ports

Defines how a scene is viewed on screen

The projection of a 3D image onto a 2D surface

Think of it like a Camera

OpenGL Modes: Orthographic, Perspective

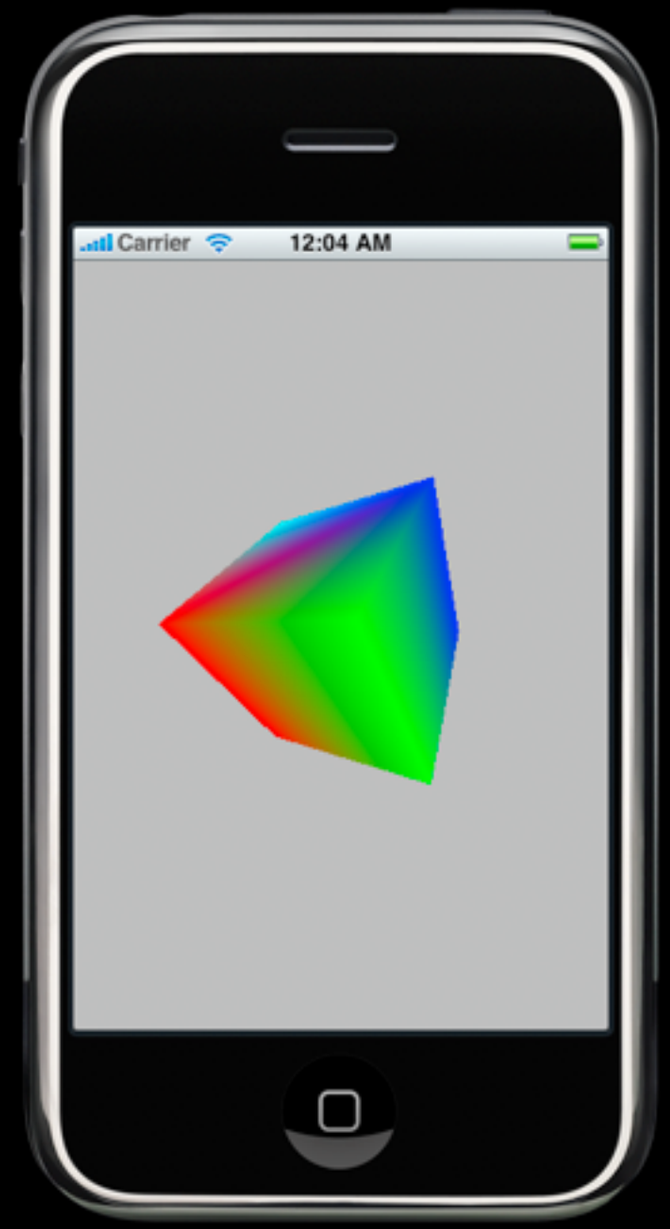


# Tutorial Time

# Tutorial Objectives

## Render a 3D Scene - Spinning Cube

- Setup Open GL ES Project in Xcode
- Introduction to Graphics Programming Concepts
- Learn about Open GL ES on the iPhone
- Display a 3D Scene



# Tutorial Steps

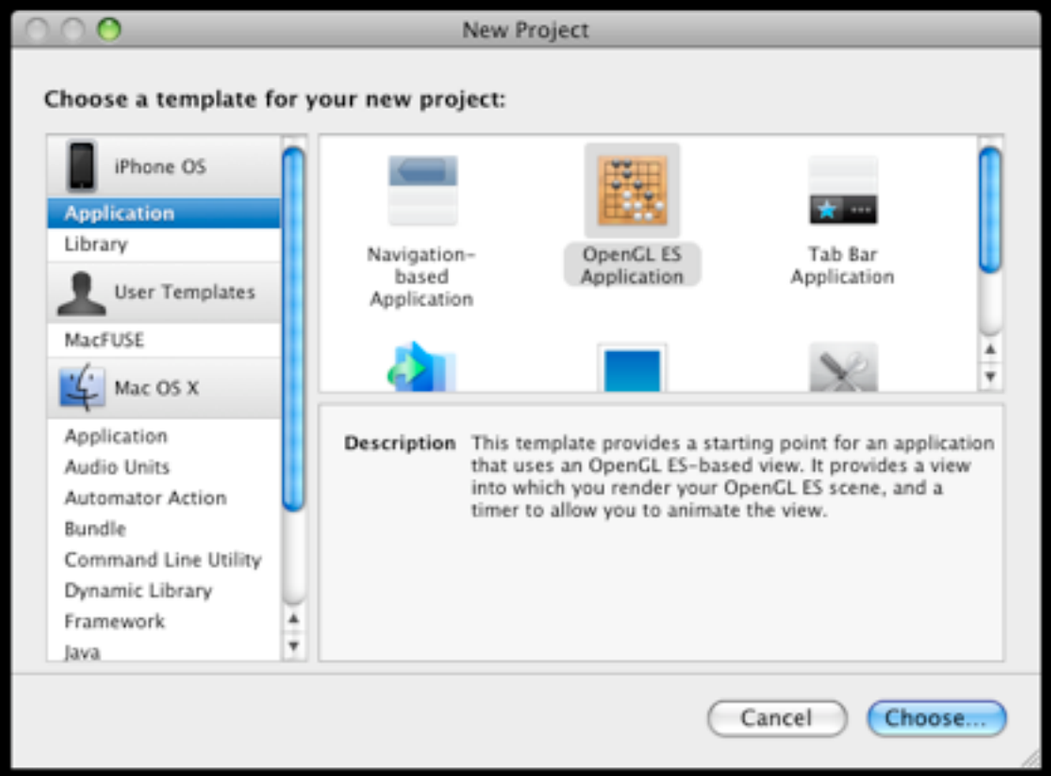
## Render a 3D Scene - Spinning Cube

1. Setup Xcode Project
2. Render a Triangle
3. Render a Multi-Colored Triangle
4. Render a Square
5. Render a Cube
6. Rotate the Cube

# Project Setup

## 1. Setup an Open GL ES Xcode Project

1. Launch Xcode
2. File > New Project
3. iPhone OS Application > Open GL ES Application
4. Choose...
5. Save As ‘Cube’
6. Build & Go!



# Build and Go!

## 1. Setup an Open GL ES Xcode Project

- 2D Square that Spins
- Our first Open GL ES Application (not really)
- Lets Examine!

# How It Works

## 1. Setup an Open GL ES Xcode Project

AppDelegate.h / AppDelegate.m

- Loads Window and View from MainWindow.xib
- Sets Render Loop Timer

EAGLView.h / EAGLView.m

- UIView subclass that sets up OpenGL & Draws to Screen
- All of the work is done Here

MainWindow.xib

- Setup Window and EAGLView view through IB
- Standard IB behavior (nothing special)

# EAGLView.m

## 1. Setup an Open GL ES Xcode Project

EAGLView.m

```
+ (Class)layerClass;
- (id)initwithcoder:(NSCoder*)coder;
- (void)drawview;
- (void)layoutSubviews;
- (BOOL)createFramebuffer;
- (void)destroyFramebuffer;
- (void)startAnimation;
- (void)stopAnimation
- (void)setAnimationTimer:(NSTimer *)newtimer;
- (void)setAnimationInterval:(NSTimeInterval)interval
```

# Rendering Basics
## 2. Render a Triangle

1. Set Models, Data, Geometry
2. Set View Port
3. Put Data into the Pipe
4. Draw (to Buffer)
5. Present (Swap buffers)

# Set Model / Data

## 2. Render a Triangle

```
-(void) render;
```

```
const GLfloat triangle[] = {
    0.0, 0.5, 0.0,   //vertex 1
    0.25, 0.0, 0.0,  //vertex 2
    -0.25, 0.0, 0.0  //vertex 3
};
```

# Setup View
## 2. Render a Triangle

EAGLView.m

```
// View Port Setup
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrthof(-1.0f, 1.0f, -1.5f, 1.5f, -1.0f, 1.0f);
```

View Port Options

```
// Orthographic
glOrthof(-1.0f, 1.0f, -1.5f, 1.5f, -1.0f, 1.0f);

// Perspective
glFrustumf(-1.0f, 1.0f, -1.5f, 1.5f, 1.0f, 10.0f);
```

# Put Data into the Pipe

## 2. Render a Triangle

```
-(void) render;
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glClearColor(0.7, 0.7, 0.7, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
glEnableClientState(GL_VERTEX_ARRAY);
glColor4f(1.0, 0.0, 0.0, 1.0);
glVertexPointer(3, GL_FLOAT, 0, triangle);
```

glVertexPointer

- Points to Vertex Data
- Reads information Serially

# Draw
## 2. Render a Triangle

`glDrawArrays`

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

glDrawArrays

- Drawing Command,  Draws based off of Vertex Pointer

# All together
## 2. Render a Triangle

EAGLView.m

```
- (void) render {
   const GLfloat triangle[] = {
      0.0, 0.5, 0.0,
      0.25, 0.0, 0.0,
      -0.25, 0.0, 0.0 };

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

   // Render Verticies
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glClearColor(0.7, 0.7, 0.7, 1.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glEnableClientState(GL_VERTEX_ARRAY);
   glColor4f(1.0, 0.0, 0.0, 1.0);
   glVertexPointer(3, GL_FLOAT, 0, triangle);
   glDrawArrays(GL_TRIANGLES, 0, 3);
   glDisableClientState(GL_VERTEX_ARRAY);
}
```

# A Triangle
## 2. Render a Triangle

# Colors

## 3. Render a Mutli colored Triangle

GL_COLOR_ARRAY

```
static const GLfloat colors[] = {
  1.0f, 0.0f, 0.0f, 1.0f,  //vertex 1
  0.0f, 1.0f, 0.0f, 1.0f,  //vertex 2
  0.0f, 0.0f, 1.0f, 1.0f   //vertex 3
};

glEnableClientState(GL_COLOR_ARRAY);
glColorPointer(4, GL_FLOAT, 0, colors);
```

# A Colored Triangle

## 3. Render a Mutli colored Triangle

# A Square - Add More Data

## 4. Render a Square

```
-(void) render
```

```
static const GLfloat square[] = {
    -0.5f, 0.5f, 0.0f,      // vertex 1
    0.5f, -0.5f, 0.0f,      // vertex 2
    0.5f, 0.5f, 0.0f,       // vertex 3
    -0.5f, 0.5f, 0.0,       // vertex 4
    -0.5f, -0.5f, 0.0f,     // vertex 5
    0.5f, -0.5f, 0.0f       // vertex 6
};

static const GLfloat colors[] = {
    1.0f, 0.0f, 0.0f, 1.0f,  // vertex 1
    0.0f, 1.0f, 0.0f, 1.0f,  // vertex 2
    0.0f, 0.0f, 1.0f, 1.0f,  // vertex 3
    1.0f, 0.0f, 0.0f, 1.0f,  // vertex 4
    0.0f, 1.0f, 1.0f, 1.0f,  // vertex 5
    0.0f, 1.0f, 0.0f, 1.0f   // vertex 6
};

glVertexPointer(3, GL_FLOAT, 0, square);
glDrawArrays(GL_TRIANGLES, 0, 6);
```

# A Square

## 4. Render a Square

# Not Efficient!

## 4. Render a Square

```
static const GLfloat square[] = {
  -0.5f, 0.5f, 0.0f,      // vertex 1
  0.5f, -0.5f, 0.0f,      // vertex 2
  0.5f, 0.5f, 0.0f,       // vertex 3
  -0.5f, 0.5f, 0.0,       // vertex 4 = vertex 1
  -0.5f, -0.5f, 0.0f,     // vertex 5
  0.5f, -0.5f, 0.0f       // vertex 6 = vertex 2
};

static const GLfloat colors[] = {
  1.0f, 0.0f, 0.0f, 1.0f,  // vertex 1
  0.0f, 1.0f, 0.0f, 1.0f,  // vertex 2
  0.0f, 0.0f, 1.0f, 1.0f,  // vertex 3
  1.0f, 0.0f, 0.0f, 1.0f,  // vertex 4 = vertex 1
  0.0f, 1.0f, 1.0f, 1.0f,  // vertex 5
  0.0f, 1.0f, 0.0f, 1.0f  // vertex 6 = vertex 2
};
```

Duplication of Vertex Data

Duplication of Color Data

# A Better Way
## 4. Render a Square

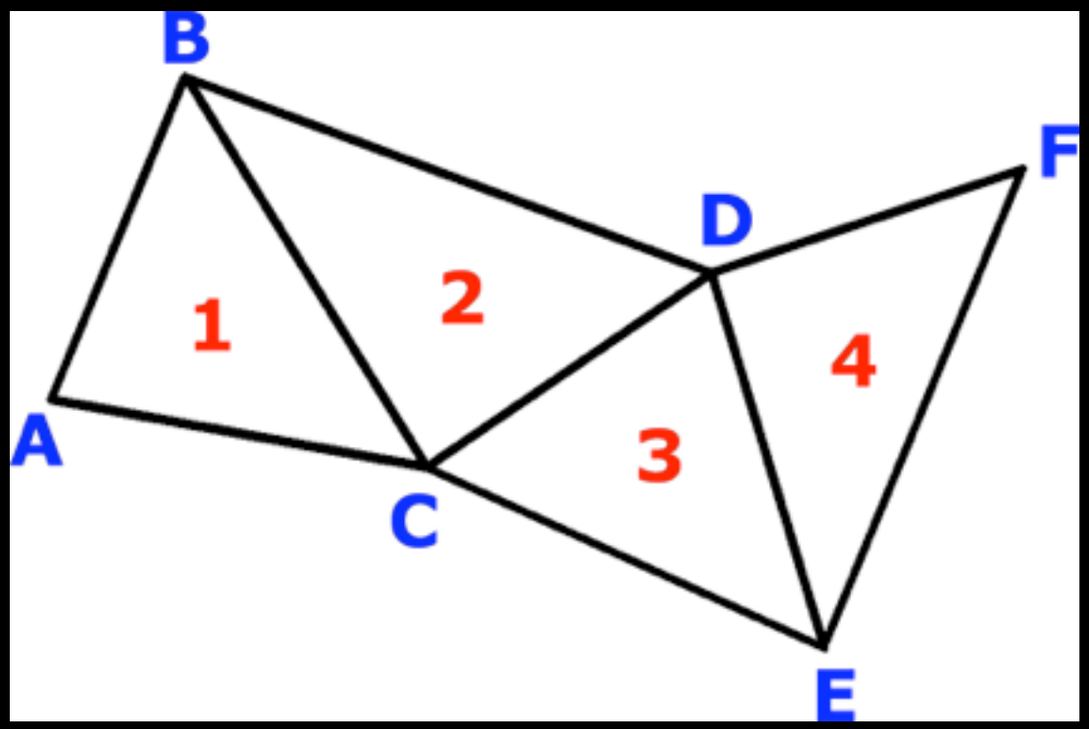
```
-(void) render
```

```
    static const GLfloat square[] = {
   -1.0f, 1.0f, 1.0f,    // vertice[0]
   1.0f, 1.0f, 1.0f,    // vertice[1]
   1.0f, -1.0f, 1.0f,    // vertice[2]
    -1.0f, -1.0f, 1.0f }; // vertice[3]

static const GLfloat colors[] = {
        1.0f, 0.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 1.0f };

glVertexPointer(3, GL_FLOAT, 0, square);
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

# GL_TRIANGLE_STRIP

## 4. Render a Square

Uses last three vertex points in an array to render a triangle




What about more complex geometry?

# Even Better!

## 4. Render a Square

`glDrawElements`

```
static const GLfloat square[] = {
 -0.5f, 0.5f, 0.0f,       // vertex 1
 0.5f, 0.5f, 0.0f,        // vertex 2
 -0.5f, -0.5f, 0.0f,      // vertex 3
 0.5f, -0.5f, 0.0       // vertex 4
};

static const GLfloat colors[] = {
 1.0f, 0.0f, 0.0f, 1.0f,
 0.0f, 1.0f, 0.0f, 1.0f,
 0.0f, 0.0f, 1.0f, 1.0f,
 0.0f, 1.0f, 1.0f, 1.0f
};

static const GLubyte triangles [] = {
 0, 1, 2,
 1, 2, 3
};

glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, triangles);
```

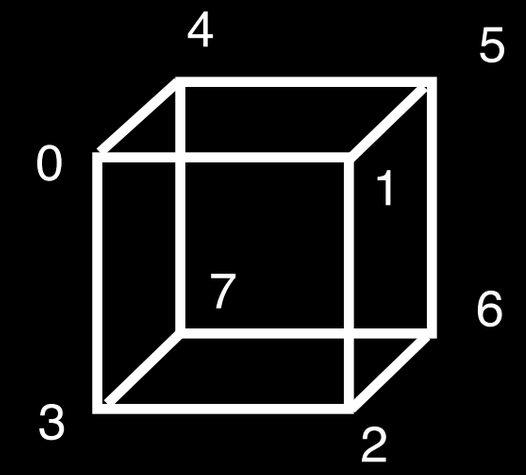
# Into 3D

## 5. Render a Cube

- Add another dimension
- Our Scene is 3D but we are just looking at it from a 2D perspective - Taking a piece of paper and looking at it straight on
- Add more data to introduce depth - z component
- Change the view

# Add More 3D Data!

## 4. Render a Cube




### 3D Cube Data

```
static const GLfloat cube[] = {
  -0.5f, 0.5f, 0.5f,   // vertex[0]
  0.5f, 0.5f, 0.5f,    // vertex[1]
  0.5f, -0.5f, 0.5f,   // vertex[2]
  -0.5f, -0.5f, 0.5f,  // vertex[3]
  -0.5f, 0.5f, -0.5f,  // vertex[4]
  0.5f, 0.5f, -0.5f,   // vertex[5]
  0.5f, -0.5f, -0.5f,  // vertex[6]
  -0.5f, -0.5f, -0.5f  // vertex[7] };

static const GLfloat colors[] = {
  1.0f, 0.0f, 0.0f, 1.0f,
  0.0f, 1.0f, 0.0f, 1.0f,
  0.0f, 0.0f, 1.0f, 1.0f,
  0.0f, 1.0f, 1.0f, 1.0f,
  1.0f, 0.0f, 0.0f, 1.0f,
  0.0f, 1.0f, 0.0f, 1.0f,
  0.0f, 0.0f, 1.0f, 1.0f,
  0.0f, 1.0f, 1.0f, 1.0f  };
```

```
static const GLubyte triangles [] = {
    1, 0, 2, // front
    3, 2, 0,
    6, 4, 5, // back
    4, 6, 7,
    4, 7, 0, // left
    7, 3, 0,
    1, 2, 5, //right
    2, 6, 5,
    0, 1, 5, // top
    0, 5, 4,
    2, 3, 6, // bottom
    3, 7, 6
  };

glVertexPointer(3, GL_FLOAT, 0, cube);

glDrawElements(GL_TRIANGLES, 36,
GL_UNSIGNED_BYTE, triangles);
```

# 3D Cube

## 5. Render a Cube


It really is 3D!

# Rotation

## 6. Rotate the Cube

```
glRotatef(angle, x, y, z)
```

```
glRotatef(3.0f, 0.0f, 1.0f, 1.0f);
```

- x, y, z - vector to rotate around
- Why does it keep rotating?
- Multiplies current matrix by a rotation matrix
- glLoadIdentity() reset current matrix

# 3D Cube

## 6. Rotate the Cube


Almost....

# Culling
## 6. Rotate the Cube

```
glEnable(GL_CULL_FACE);
```

- OpenGL Iterates though the vertex array and draws to the buffer in order, triangles later in the buffer are drawn over top of triangles earlier
- Typically 3D objects are closed surfaces Culling is a way to speed up rendering by only drawing what can be seen
- Specify Triangle in a counter clockwise order

# Done!

## 6. Rotate the Cube

# Open GL ES

# iPhone

# General

## iPhone OpenGL ES Pro Tips

- Avoid transforming a UI View
- Landscape View > Rotate in OpenGL
- Avoid Placing UIKit elements above an OpenGL View
- If your OpenGL view isn’t visible, disable frame updates
- OpenGL Support Classes by Apple:

  EAGLView.h / Texture2D.h / PVRTexture.h

# Textures
## iPhone OpenGL ES Pro Tips

- Texture Limit: 24 MB
- Max texture size: 1024 x 1024
- Use Power VR Texture Compression
- Batch Textures together in a Texture Atlas
- Preload textures before using

# Performance
## iPhone OpenGL ES Pro Tips

- Instruments > OpenGL ES profiler
- Don’t use Fixed Point Arithmetic
- Minimize Open GL Calls
- Batch Drawing Calls
- Minimize State Changes

# Resources

# Resources

- iPhone Developer Library: Application Programing Guide
- Stanford CS 139P: iPhone Application Programming
- Khronos Group
- Red Book

# Questions?

richerd@tapium.com