How to Implement 2D and 3D Graphics in Apple® iOS Applications

Gopinath Taget – Autodesk Inc.
Adam Nagy – Autodesk Inc.

DV2751 In this class, we talk about and demonstrate Apple iOS libraries for 2D and 3D graphics programming. We demonstrate the use of native classes that the iOS SDK provides for graphics programming including the Quartz and UIKit SDK, which are useful for 2D graphics. We also introduce and demonstrate the use of OpenGL library implementation for iOS (called OpenGL ES) as well as GLKit library. Finally, we demonstrate a browser-based graphics application that uses the WebGL technology to display 2D and 3D graphics on a browser running in an Apple device. This class requires some familiarity with the Objective C language and the usage of XCode IDE.

Learning Objectives

At the end of this class, you will be able to:

- Implement 2D graphics using the native Quartz and UIKit
- Implement 2D and 3D graphics using the OpenGL ES and GLKit
- Implement 2D and 3D graphics for a browser on iOS
- Determine which of these technologies are best for your graphics needs

About the Speaker

Gopinath is a member of the Autodesk Developer Technical Services Team. He has more than nine years of experience developing and supporting AutoCAD® APIs, including ObjectARX®, Microsoft® .NET, VBA and LISP. Gopinath also has several years of experience in software development on other CAD platforms, including MicroStation®, SolidWorks®, and CATIA® mainly using C++ and technologies such as MFC and COM. Gopinath was also involved in the development of Web-based applications for Autodesk® MapGuide® and AutoCAD Map 3D. Currently Gopinath is working with AEC products (Revit, ACA) and cloud based solutions inside Autodesk.

Gopinath has master's degrees in Civil Engineering and Software Systems.
gopinath.taget@autodesk.com

Introduction

Historically, graphics programming has not been easy. You not only have to be good at general programming but also have to have specific knowledge about 2D and 3D geometry and the mathematics around them. And then there are specialized APIs and API Architectures that you need to work with on specific platforms. The Apple iOS is one such (mobile) platform. Luckily, over the years, graphics programming has moved towards standardization of the APIs and interfaces. So skills learnt on one platform can more often than not be used on other platforms. One big example of this is the OpenGL programming interface. According to Wikipedia "**OpenGL** (**Open G**raphics **L**ibrary)[2] is a cross-language, multi-platform application programming interface (API) for rendering 2D and3D computer graphics. The API is typically used to interact with a Graphics processing unit (GPU), to achieve hardware-accelerated rendering." This makes OpenGL very popular graphics and 3D modeling API interface across multiple platforms.

Another aspect that a graphics application developer should be mindful of is hardware. Graphics applications are computationally intensive (unless the app is very simple). They tend to hog both CPU as well as memory resources causing a general slow down of the computer system. This spawned a whole hardware industry specializing in hardware targeted at computer graphics. This specialized hardware reduced the burden on the main CPU and memory of the system.

This poses a special problem for mobile developers like iOS and Android developers. Mobile devices due to their small form factor can pack a lot less hardware compared to regular desktop computers or even laptops. This means smaller CPU, smaller memory and smaller graphics hardware. So graphics applications that would work well on desktops and laptops would not work very well on mobile devices. Graphics applications for mobile devices need to be designed specifically to make sure they perform optimally on constrained hardware of mobile devices. This is not to say that mobile devices do not have good hardware. Some of the modern day mobile devices are incredibly powerful. For instance here are the technical specs for the latest version (As of AU 2013) of iPhone, the iPhone 5s:

OS	iOS 7, upgradable to iOS 7.0.3
Chipset	Apple A7
CPU	Dual-core 1.3 GHz Cyclone (ARM v8-based)
GPU	PowerVR G6430 (quad-core graphics)
Internal	Accelerometer, gyro, proximity, compass

Memory	16/32/64 GB storage, 1 GB RAM DDR3

As you can see, the hardware specifications are by no means bad. However, they are still not comparable to standard desktops and laptops of the day. The impact of smaller mobile hardware is not limited to your application design. It influences the whole mobile system, including the design of the iOS operating system as well as the available SDKs. For instance, OpenGL API interface has been refactored to become a smaller subset of the original interface so it can fit the mobile environment. This refactored interface is called OpenGL ES.

I will discuss OpenGL ES and other graphics SDKs on iOS in the following sections. Before we dive in though, a few points to keep in mind. Programming for Apple iOS requires knowledge of the Objective C language and XCode IDE (Integrated Development Environment) programming interface. If you are new to iOS programming, I would recommend the 2013 AU class "DV1732 - Introduction to Apple® iOS Mobile Development" as a good starting point. There are several other resources for learning iOS development. Here is a very good one.

Another thing to note is that mobile platforms tend to get updated much more frequently than desktop/laptop platforms. This means mobile developers have to relearn and upgrade their knowledge and migrate their applications to conform to the latest platforms at least every two or three of releases. The latest iOS version as of this writing is iOS 7.

SDK and APIs for 2D Graphics

You can implement 2D graphics a couple of ways in iOS. You can use either the native iOS SDK called the Quartz 2D SDK or the generic OpenGL ES SDK. The Quartz 2D SDK is a relatively simple SDK that allows you to draw on a canvas represented by the iOS view (UIView) object or the image (UIImage) object. It allows you to draw simple 2D primitives like lines and curves. This drawing happens in relation to or inside a Context object. This context object represents not just the canvas on which the drawing happens. It also holds inside itself graphical tools used to draw. Like the size and color of the pen used to draw, the color palette for draw and fill colors for patterns, the background color or image and more. You can think of the context as the convas as well as the color palette and tools used to draw.

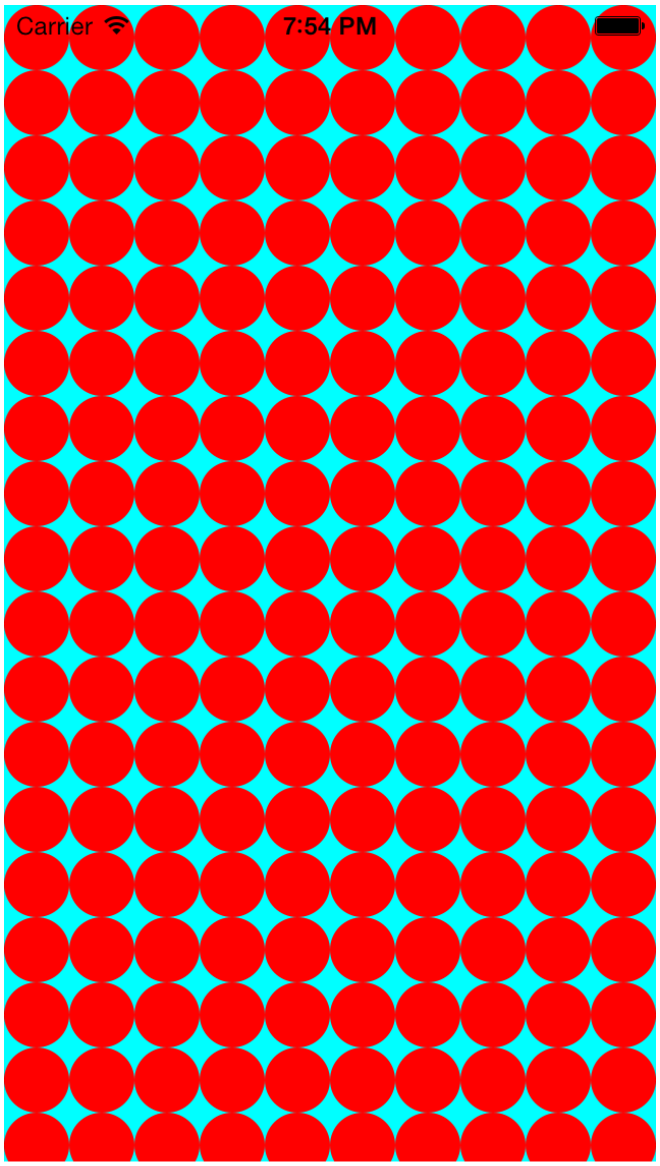
So before drawing anything, a context object needs to be instantiated/acquired, initialized with the palette and tools and then used to draw. That is basically it. To summarize 2D iOS graphics:

1) Instantiate the context
2) Initialize the context with the color, pen and other settings
3) Begin drawing

Here is the drawRect method of a custom UIView (the commented code is also useful to perform different operations):

```
// Only override drawRect: if you perform custom drawing.
// An empty implementation adversely affects performance during animation.
- (void)drawRect:(CGRect)rect
{
    // Drawing code


    //CGPoint drawingTargetPoint = CGPointMake(0,0);
    //[backgroundImage drawAtPoint:drawingTargetPoint];

    CGContextRef    context = UIGraphicsGetCurrentContext();

    // Pattern
    PaintMyPattern(context, rect);
    // Pattern End

    // Gradient
    //Gradient related variables
    CGGradientRef myGradient;
    CGColorSpaceRef myColorSpace;
    size_t locationCount = 3;
    CGFloat locationList[] = {0.0, 0.5, 1.0};
    CGFloat colorList[] = {
        1.0, 0.0, 0.5, 1.0, //red, green, blue, alpha
        1.0, 0.0, 1.0, 1.0,
        0.3, 0.5, 1.0, 1.0
    };
    myColorSpace = CGColorSpaceCreateDeviceRGB();
    // CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
    myGradient = CGGradientCreateWithColorComponents(myColorSpace, colorList,
                                                     locationList, locationCount);

    //Paint a linear gradient
    CGPoint startPoint, endPoint;
    startPoint.x = 0;
    startPoint.y = 0;
    endPoint.x = CGRectGetMaxX(self.bounds)/2;
    endPoint.y = CGRectGetMaxY(self.bounds)/2;

    //CGContextDrawLinearGradient(context, myGradient, startPoint, endPoint,0);
    //Radia Gradient Rendering
    float startRadius = 20;
    float endRadius = 210;
    //CGContextDrawRadialGradient(context, myGradient, startPoint, startRadius,
    //                    endPoint, endRadius, kCGGradientDrawsAfterEndLocation);
    // Gradient

    CGContextSetLineWidth(context,4);
    CGContextSetStrokeColorWithColor(context, currentColor.CGColor);


    CGContextMoveToPoint(context,fromPoint.x , fromPoint.y);
    CGContextAddLineToPoint(context, toPoint.x, toPoint.y);
    CGContextStrokePath(context);
}
```

The above method demonstrates different 2D drawing tasks like setting up the context with color, drawing a line between two points, drawing patterns and gradients (some of the code is commented). The result of this code is the geometry pattern below. I will walk through the code in detail during the AU class:



The code above is a slightly modified version of the tutorial available in [this codeproject article](). This article gives a very friendly introduction to the iOS 2D library.

SDK and APIs for 3D Graphics

3D graphics on iOS is typically performed with OpenGL ES. OpenGL ES has been used on mobile devices for several years now. The first version of OpenGL ES available on mobile devices is OpenGL ES 1.0. There have been several versions since then including 1.1, 2.0 and

3.0. For an overview of the differences between these versions, please refer to the "Versions" section of the Wikipedia article on OpenGL ES.

For a long time OpenGL ES 1.1 was the de facto library for 3D programming on mobile devices. It was (relatively) easy to use (compared to OpenGL ES 2.0) and provided all features necessary for a professional 3D application or game on iOS. The web-based tutorial OpenGL ES from the Ground Up by Jeff LaMarche is an excellent tutorial introduction for OpenGL ES 1.1. It not only gives you a very good introduction on OpenGL ES 1.1, it also gives you a beginners introduction to the OpenGL architecture. I highly recommend this tutorial.

iOS is known to support multiple versions of OpenGL ES including 1.1 and 2.0. The latest version of iOS (7.0.4 as of this writing) also supports OpenGL ES 3.0.

One point to note here is that because OpenGL ES is a cross platform SDK, it can be coded using pure C++ on iOS. There are a couple of advantages with using C++:

1) You will find a lot more people in the developer community with C++ skills than Objective C skills.
2) You can use existing C++ based OpenGL applications written for desktop and reuse it for the mobile (refactoring it for OpenGL ES of course); i.e., you do not have to rewrite your desktop based OpenGL modules for mobile. You can reuse them.

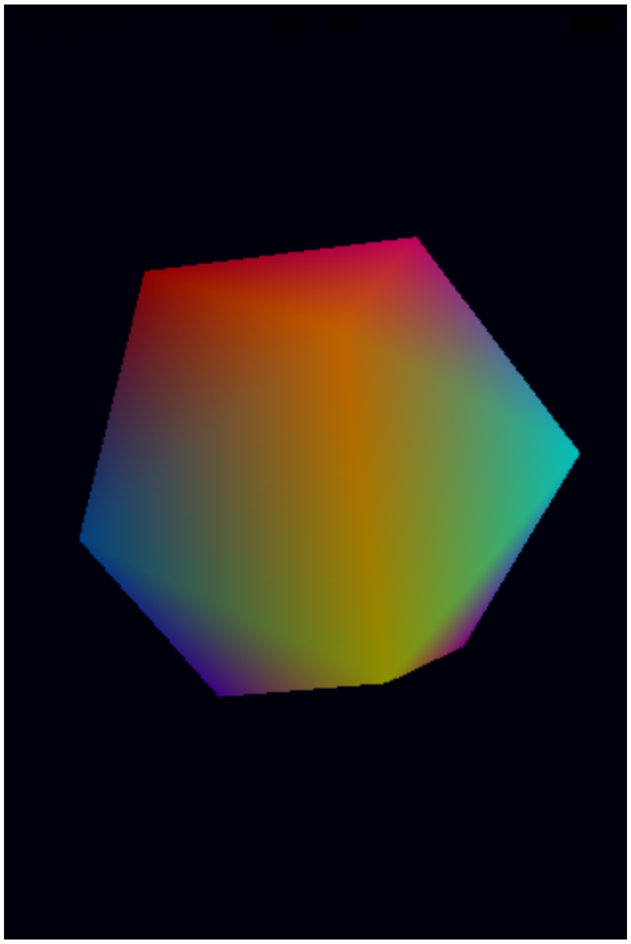
OpenGL ES 1.1

Like iOS 2D graphics, OpenGL ES 1.1 needs you to setup contexts. In the case of OpenGL, they are referred to as Buffers. Typically you instantiate, Frame buffers and Render buffers (like color and depth render buffers). Please don't worry about what these really mean for now. I will explain them in greater detail during the AU class presentation (Or you could go through the documentation and tutorial links I provided in the previous sections). Just think of the buffers as the canvas used to draw your graphics on.

Once the buffers are instantiated, you can define your geometry. One point to note is that all graphics in OpenGL is represented as a collection of triangles. For instance, a sphere surface is broken down into an approximation of a collection of triangles. You can breakdown the triangles to as small a size as you want. As you can imagine, the smaller the size, greater the smoothness of a curved surface (like that of a sphere).

While setting up the geometry, you can also setup visual information like lighting and color of the lines used by the triangles.

You will find an example of everything I explained so far in the GL11Sample I am providing for my AU class. All the relevant code in the GLView and GLViewController classes. Please take a look at the GLView.CreateFrameBuffer and GLViewController.setupView (for setting up geometry). Here is a snapshot of the resultant graphics:



This sample (GL11Sample) is modified version of the 1.1 sample by Jeff LaMarche. You will find his code on GitHub here. I modified the code to work with iOS 7 and XCode 5.

OpenGL ES 2.0

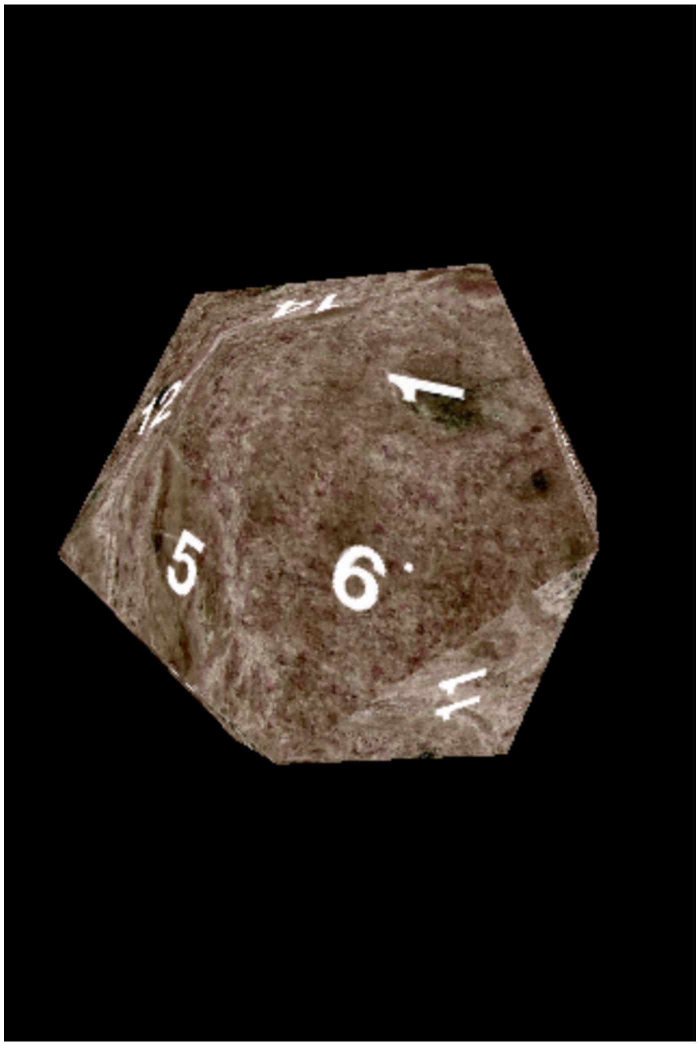
OpenGL WS 2.0 is the next major version of OpenGL ES following 1.1. The biggest difference between between 1.1 and 2.0 is how 2.0 allows developers to define “shaders”. Shaders are text based files that are “compiled” by OpenGL ES 2.0 library at runtime to define custom material and lighting for geometry elements. This provided very high level of control on visualization of models, something that was not possible with 1.1. Here is an excerpt from Wikipedia about this:

OpenGL ES 2.0 was publicly released in March 2007.[2] It is based roughly on OpenGL 2.0, but it eliminates most of the fixed-function rendering pipeline in favor of a programmable one in a move similar to transition from OpenGL 3.0 to 3.1.[3] Control flow in shaders is generally limited to forward branching and to loops where the maximum number of iterations can easily be determined at compile time.[4] ***Almost all rendering features of the transform and lighting stage, such as the specification of materials and light parameters formerly specified by***

the fixed-function API, are replaced by shaders written by the graphics programmer*. As a result, OpenGL ES 2.0 is not backward compatiblewith OpenGL ES 1.1. Some incompatibilities between the desktop version of OpenGL and OpenGL ES 2.0 persisted until OpenGL 4.1, which added the`GL_ARB_ES2_compatibility` extension.[5]*

The actual workflow of the 2.0 application is similar to that of 1.1; i.e., you create the appropriate buffers and define the geometry. The only thing extra that you would do would be to compile the shaders before you define the geometry.

The sample "Simple OpenGL ES 2.0 Example" provided with this AU class demonstrates the OpenGL ES 2.0 library. This is also a slightly modified version of Jeff LaMarche's samples that I obtained from GitHub. I modified it to work with iOS 7 and XCode 5. Here is the result of this sample:



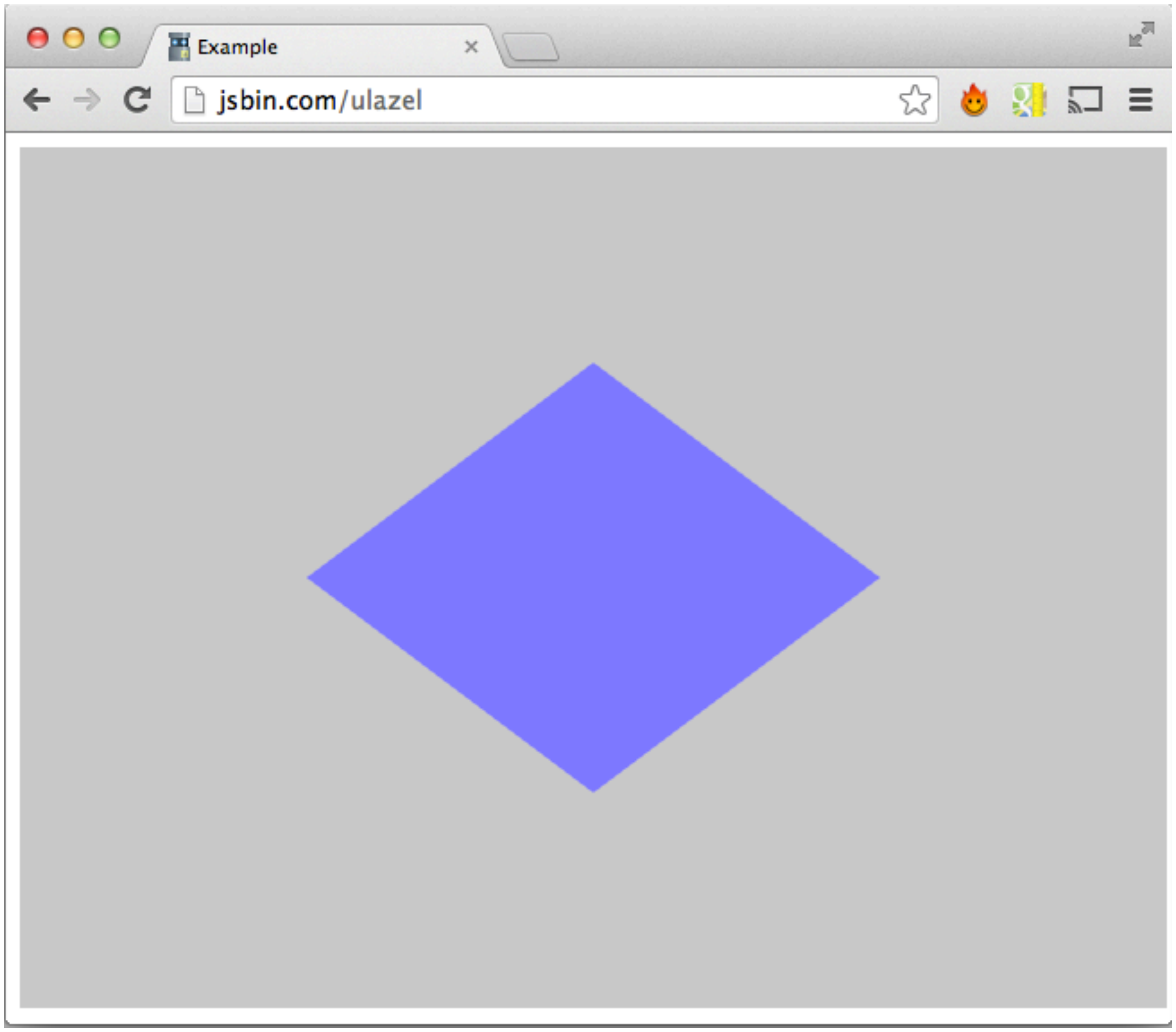
OpenGL ES 3.0

OpenGL ES 3.0 is very new specification that was published just about a year ago in August 2012 and not many devices support it yet. The iOS 7.0 devices are among the first few to support it. There are therefore very few applications that use it and support it out there. Further, OpenGL ES 3.0 is backward compatible with OpenGL 2.0 so 2.0 applications will work with 3.0 library. You will find more information on OpenGL ES 3.0 specification here and iOS specific information on OpenGL ES here. You will find sample usage of OpenGL ES 3.0 here.

Graphics programming on the browser using WebGL

WebGL according to Wikipedia is "*a JavaScript API for rendering interactive 3D graphics and 2D graphics*[2] *within any compatibleweb browser without the use of plug-ins. WebGL is integrated completely into all the web standards of the browser allowing GPU accelerated usage of physics and image processing and effects as part of the web page canvas. WebGL elements can be mixed with other HTML elements and composited with other parts of the page or page background.*[3] *WebGL programs consist of control code written in JavaScript and shader code that is executed on a computer's Graphics Processing Unit (GPU). WebGL is designed and maintained by the non-profit Khronos Group.*[4]"

It is currently based on the OpenGL ES 2.0 specification. One point to note here is that all WebGL code (like javascript code) run in the **browser** on the **client device**. So even though the web page (that contains the Javascript code) lives on a web server, the rendering is performed on the client browser. So in many ways, it is very similar to an OpenGL ES application. The only difference is that the application runs inside a browser. Here is a sample that shows a web page with simple WebGL script. Here is the result of loading the webpage with the script:



Conclusion

You can use Quartz 2D for 2D graphics, OpenGL ES 1.1, 2.0 or 3.0 for shaded and textured 3D graphics in a native iOS application. If you are open to using a web browser to host your application, you can create a web page that use the WebGL library in a Javascript application inside the web page to show your 2D and 3D graphics. I will demonstrate all of this in the AU class. See you there.