# TEACHING GRAPHICS PROGRAMMING ON MOBILE DEVICES

Michael Werner
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115
617-989-4143 wernerm@wit.edu

## ABSTRACT

This is an experience report on teaching graphics programming using Android mobile devices. The goals were to cover standard introductory graphics programming topics and to pique interest by deploying graphics apps on mobile phones and tablets.

## INTRODUCTION

This is an experience report on teaching graphics programming with applications deployed on mobile devices. A bit of background: We have been teaching an introductory graphics programming course for several years using the OpenGL libraries. Students programmed a variety of lab exercises written in C and executed on PC computers using a rudimentary windowing system (usually GLUT). Some students exercised an option to substitute an advanced project for required labs. They were free to choose their own platform and programming language. One popular choice was Microsoft's DirectX and C#.

The syllabus was aligned with popular textbooks by Edward Angel "Interactive Computer Graphics" and Hearn & Baker "Computer Graphics" [1, 2]. The syllabus topics included:

- 2-D Programming
- Event-Driven Graphics
- Mathematics for graphics
- 3-D Programming
- Model Transformations
- Lights and materials
- Texture mapping
- Hierarchical Modeling
- Curves and surfaces
- Molecular Visualization

It should be noted that although the primary focus of the course has been displaying 3D images on the screen, there has always been a secondary goal of supporting user interaction, i.e. allowing the user to view an object from various sides and to manipulate objects by applying model transformations.

While the course was usually a popular elective some students complained that the laboratory programming environment was a bit dated, particularly in the use of GLUT [3]) for providing a

surface to draw on and primitive event handling. In response to student concerns and to spark interest in the course we decided to move the required labs to a more current platform, namely Android apps using OpenGL. The same graphics topics would be covered but the programming would be shifted dramatically:

- C was replaced by Java
- Microsoft Visual Studio was replaced by Eclipse
- OpenGL was replaced by OpenGl ES 1 (embedded systems).
- GLUT was replaced by use of an Android emulator and Android phones and tablets.

There were also changes in the methods used for user interaction. Mobile devices have touch screens supporting new modes of interaction including taps, swipes, and gestures such as pinching. More traditional means of interaction such as making menu choices and entering text are cumbersome. Assigning different meanings to right and left mouse buttons are impossible on mobile devices. Our hope was to make the course more attractive while at the same time building student competence in the core theory and practice of computer graphics.

## EXPERIENCE

This section recounts a preliminary experience of teaching graphics on an Android platform. The choice of Android was motivated by the openness of the Android platform and the extensive support tools available to developers. The chief competitor to Android is currently iOS [4] (iPhone), where App development is usually done in Objective-C. Our students currently learn Java and C++, so programming in Objective-C would present a needless obstacle for them.

The course was offered in the spring of 2012 to a section of 15 computer science majors. They had all taken courses in C++ and Java, and the majority had also taken a course in linear algebra. The students were mostly upper level though there were 2 freshmen and 1 sophomore. Roughly half the students had already done some development in graphics and games, which is why they took the course. The other students were looking to fulfill an elective requirement.

At the time there were no textbooks available for teaching graphics on Android. To fill this void two inexpensive trade books by Angel and Zechner, respectively [3, 5] were recommended. Angel's primer contains the rudiments of graphics theory but none of the code is applicable to Android. Zechner has a few chapters on graphics as well as useful information and code examples on Android apps in general including user interaction. The students were also provided with draft chapters of a book being prepared by the professor.

Android development is usually done in Java using a free Android SDK (Software Development Kit). The preferred IDE is Eclipse enhanced by the ADT (Android Development Tools) plugin. This environment supports logging and debugging. Testing is done using an emulator provided with the SDK, or on an Android device such as a tablet or smart phone. An extensive

development guide and other resources are available at the Android Developers web site [6]. Google makes it relatively easy for developers to market their apps on the Google Play Store.

There are currently two flavors of OpenGL supported on the Android platform:

1. OpenGL ES 1.X is designed for fixed function hardware. It supports the notion of a state machine piping all vertices through the same transformation and lighting calculations.
2. OpenGL ES 2.X requires programmer supplied vertex and fragment shaders using the OpenGL ES Shading Language. It does not support a fixed function transformation pipeline architecture. Programmers may not mix ES 1.X and ES 2.X code. [7]

The Android Developers site recommends "*Developers who are new to OpenGL may find coding for OpenGL ES 1.0/1.1 faster and more convenient*" (8). This advice was confirmed by several students who had tried both. For an introductory course it was felt using ES 2.X would distract from the goal of learning fundamental graphics theory. Hence, all formal instruction was in ES 1.X although a few students on their own learned how to use ES 2.0.

The course ran with 3 hours of lecture and 2 hours of lab per week. At the beginning there were specific coding examples to get students started in the development environment and in the requirements of the target platform. The students needed a robust testing environment. Many of them used their own Android phones or tablets. They needed to connect with a cable and load a suitable device driver. The department also had several "loaner" tablets available. The fallback was to use an Android device emulator supplied with the SDK. But the emulators respond slowly and are limited in the interactions they support. From the beginning the students were using a variety of devices running different versions of the Android operating system. While this presented a classroom challenge it was also a good simulation of the problems commercial Android App developers face. As a class we quickly decided not to use the latest Android version since this would severely limit the number of devices that could run our Apps. We settled on a minimum of version 2.3 (Gingerbread). Even so, we noticed some hardware dependencies, i.e. the same App running on different 2.3 devices sometimes behaved differently.

The first job was to get something to show on the screen. The Android plug-in for eclipse has a wizard for setting up a new Android project. It generates a subclass of the Activity class. This class is used for starting, ending, pausing and resuming the App. It also handles some user interaction such as menus. Drawing requires a surface to draw on and a renderer to do the drawing. These are created by sub-classing android.opengl.GLSurfaceView and implementing the Renderer interface.

The first few labs used OpenGL for 2D graphics apps. (See Figure 1) There was work on simple interaction including making menu choices and listening for and handling user taps and gestures. For anyone who has done OpenGL work on a computer be aware that some things are different in OpenGL ES. The glBegin ..glEnd blocks are gone, there are no more polygons and quads, only triangles, lines and points. Data for vertices, indices, normals, etc. is stored into Byte

Buffers to meet the requirements of the graphics hardware. Generating the data for drawing is typically separated from the rendering code.

After a few weeks we began to discuss 3D modeling including the use of OpenGL calls to set the projection matrix. The students initially produced 3D versions of some of the 2D models already covered. For example, a 2D Sierpinski Triangle was expanded to a 3D one. See Figure 2.

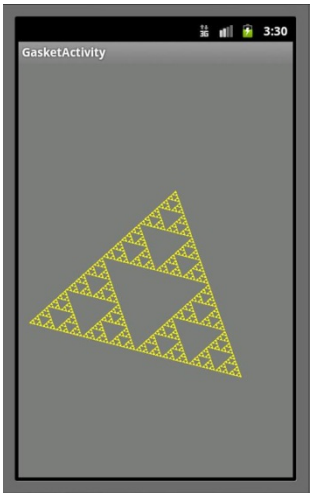

Figure 1. A 2D Sierpinski Triangle

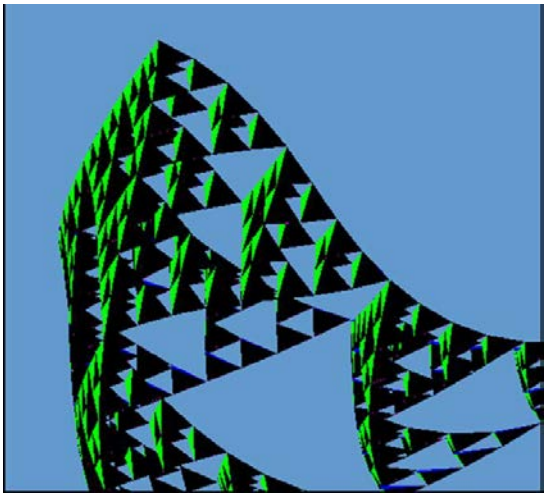
Figure 2. A 3D Sierpinski tetrahedron that has been twisted

With 3D drawing it became necessary to equip the user with the ability to rotate and scale the objects so as to get a better view. This required implementing android interactions such as swipe and pinch. On the graphics side, perspective viewing, model transformations, lighting and texture mapping were introduced in the lectures including their mathematical basis. Finally curves and surfaces were studied including Bezier curves and patches.

## OPTIONAL STUDENT PROJECTS

After completing 3 mandatory labs, students were given latitude to pursue their own projects rather than following the 8 remaining labs correlated to lecture topics. Guidelines required that their project include graphics features such as lighting and texture mapping. They also had to show originality. Finally, students doing projects had to present progress reports and demonstrate the finished project to the class. They were free to deviate from the Android platform and use advanced features from OpenGL ES 2.X, primarily shaders.

Six students chose to do projects. None was on Android devices which meant that the students were largely on their own. One abandoned his project (involving CUDA [9]) and went back to doing the labs. Two projects were very successful and incorporated advanced graphics features. In fact, one project won a cash prize in an international competition. One student attempted a cross platform graphics application. This was ambitious and only partly successful. Another student built on available graphics engines used for video games and managed to add some novel

features. Students doing projects continued to participate in the class and often were called on to present various aspects of graphics such as lighting and textures.

## ASSESSMENT

Prior to teaching the class a number of outcome criteria were specified in two areas:

1. Graphics specific – Including 2D and 3D viewing, model transformations, lights and materials, texture mapping, hierarchical modeling, event drive interactivity, curves and surfaces.
2. CAC of ABET specific – These involved the abilities to analyze a problem, define its computing requirements, design, implement and evaluate a solution using current techniques, skills, and tools.

The graphics specific outcomes were assessed in laboratory and project submissions and on a midterm and final exam. Test questions were platform neutral. They did not involve Android although a few specified OpenGl. The CAC of ABET specific outcomes were measured primarily in the 3 mandatory labs and secondarily in the remaining labs and student projects. After the course concluded a course evaluation was completed by the instructor and (following procedure in our department), submitted to peer review by a small faculty committee. The evaluation showed that all goals had been met.

A student survey was conducted in the last week of class. Students gave a 1.9/5 (disagreed) rating to the question: "I would have preferred a simpler environment", indicating they did not find Android environment too challenging. On the question "I would recommend this course to a friend" they gave a 4.4/5.

## DRAWBACKS TO THIS APPROACH

The biggest drawback is that an inordinate amount of time is spent on meeting the fast evolving requirements of the Android devices. Testing is difficult owing to the variation in hardware devices and versions of the operating system.

Another issue is the increased effort required to handle the user interface. That said, graphics on the computer also have user interaction, i.e. using context menus, keystrokes, color pickers and mouse actions. A mobile device's touch screen enables enhanced interaction but requires more effort. Finally, mobile devices have much smaller screens limiting what can be shown. To some degree this is mitigated by high resolution and allowing the user to pan and zoom.

## ADVANTAGES TO THIS APPROACH

Programming for mobile devices is very current and hence motivating for students. They could always hope to produce and market their own App. A couple of students did publish their apps for free on the Google Play Store. There are other positives as well. The programming makes a powerful case for object orientation. Due to the complexity of the environment an App of

necessity inherits most of its functionality from library classes. Hence you only code the difference between what is inherited and what is required. Eclipse provides support for generating source code to override inherited methods, implement interfaces and do refactoring. In addition, Android imposes certain coding conventions and restrictions that must be adhered to. For example, an App must implement pause and resume methods to enable an App to keep its state when interrupted by a phone call.

Aside from learning the basics of graphics programming, students more generally increase their proficiency at writing programs that need to run in a complex and highly structured environment. For marketing Apps, students need to undertake thorough testing on a variety of devices. This "broadening" of their programming skills should better equip graduates to obtain and thrive in their first career placement.

## CONCLUSION AND FUTURE DIRECTIONS

The graphics programming course used Android devices for the first time in the spring of 2012. It was well received by the students who made some useful suggestions for next time. Test results showed that students performed at least as well and often better than in previous offerings of the course. The plan is to repeat the course this way in spring 2013. One change will be to include some instruction on vertex and fragment shaders [10]. "Shaders are programming patterns that are now mandatory in any recent real-world graphics API ". [11] OpenGL ES 2.X (based on shaders) is available in Android versions 2.2 and up. While it presents more challenges, it will also bring it more into conformity with "modern" graphics methods.

## REFERENCES


[1] Angel, Edward, *Interactive Computer Graphics*, Addison Wesley, 2008.

[2] Hearn, Donald & Baker, M. Pauline, *Computer Graphics with OpenGL*, Prentice Hall, 2004.

[3] Angel, Edward, *OpenGL: A Primer*, Addison Wesley, 2007.

[4] iOS Dev Center, https://developer.apple.com/devcenter/ios/index.action, retrieved: Nov 5, 2012.

[5] Zechner, Mario, *Beginning Android Games*, APress, 2011.

[6] Android Developers, http://developer.android.com/index.html, retrieved: May 8, 2012.

[7] Khronos Group, OpenGL ES., http://www.khronos.org/opengles/, retrieved: May 8, 2012.

[8] Android Developers, Choosing Version, http://developer.android.com/guide/topics/graphics/opengl.html#choosing-version, retrieved May 8, 2012.

[9] CUDA Parallel Computing Platform, http://www.nvidia.com/object/cuda_home_new.html, retrieved Nov 10, 2012.

[10] Angel E., Shreiner, Teaching a Shader-Based Introduction to Computer Graphics. D., *IEEE Explore*, Mar 2011.

[11] Heinrich Fin, Thomas Webe and Michael Wimmer, Teaching a Modern Graphics Pipeline Using a Shader-based Software renderer, *EUROGRAPHICS 2012*.

Mobile phones are already a huge part of our lives, and combining them with the power of machine learning is something that, in theory, can create user experiences that delight and impress users. But do we really need to add machine learning to our apps? And if so, what tools and platforms are currently at our disposal? That's what we'll talk about in this article. 1980: â€œI Bet There Will Be Flying Cars in the Futureâ€; 2017. To employ machine learning, you need to have a trained model. Letâ€™s see how you can use a model inside your app. My program also has a lot of scientific computing and I know C++ is best/fastest for that I've never done anything with the Android before so I'm really helpless right now. If I'm just going about it the wrong way, please give me other suggestions...Â  3,0941515 gold badges5151 silver badges9292 bronze badges. Typically the NDK doesn't incur a performance penalty, but rather a complexity penalty. Using the NDK to write C/C++ is the sort of the defacto way to do non-UI graphics on Android. â€“ Doug Stephen Jun 16 '11 at 16:38. Using the NDK will only slow down development (steeper learning curve). It teaches programmable rendering for beginning graphics programmers, from the ground up. This book also covers some important material that is often neglected or otherwise relegated to â€œadvancedâ€ concepts.Â  Graphics programming can be a daunting task when starting out. The rendering pipeline involves a large number of steps, each dealing with a variety of math operations. It has stages that run actual programs to compute results for the next.Â  The space of normalized device coordinates is essentially just clip space, except that the range of X, Y and Z are [-1, 1]. The directions are all the same. The division by W is an important part of projecting 3D triangles onto 2D images; we will cover that in a future tutorial. 13. Introduction.