The Architecture of a J2ME-based OpenGL ES 3D Library

Cheng-Han Tu* and Bing-Yu Chen†
National Taiwan University
*toshock@cmlab.csie.ntu.edu.tw †robin@ntu.edu.tw

Abstract

In this paper, we propose a new framework for developing 3D graphic applications on portable devices such as cellular phones or personal digital assistants (PDAs). Our framework is based on J2ME (Java 2 Platform, Micro Edition[1]) environment, an application environment that specifically addresses the needs of developing programs on embedded systems using Java, and we leverage the advantages of J2ME to build our platform-independent 3D graphic library on mobile devices. Adopting this framework, the 3D program developers can not only implement a cross-platform 3D program easily as they did using OpenGL ES[2] library but also gain the powerful features including efficient memory usage and shading functionalities.

1. Introduction

The demands for showing 3D graphics on mobile devices increases drastically in recent years as a result of those portable devices such as personal digital assistants (PDAs) and cellular phones become more and more prevalent today. Using 3D technologies, a small hand-held device will become to a powerful one on which the user can see 3D animations, use 3D graphics editing tools, playing 3D games, etc.

However, so far there is no standard 3D graphics library which the 3D program developers can use for such devices. Moreover, the applications written with traditional programming languages such as C/C++ are highly platform-dependent, i.e., the program might run well on a certain machine while it probably crashes on the other.

Most people encounter the problem that they have to port their codes to different platforms over and over again on different machines or operating systems which have different capabilities as well as different instruction sets.

To solve these problems, we present a new platform-independent 3D library framework specialized for mobile devices based on J2ME (Java 2 Platform, Micro Edition) environment and has a similar programming architecture with OpenGL ES. With our library, the programmers feel a little difference of the programming style between our library and OpenGL ES, and can smoothly transfer their codes to a cross-platform 3D program. To provide such a software 3D library on mobile devices, performance and memory limitation are the great challenges; hence we put our most efforts on efficient memory usage and enhance the performance. Besides, since this 3D engine is developed for pure J2ME environment, it also has backward capability for different hardware architectures and some old hand-held devices.

The paper is organized as following. In Section 2, the related work is introduced. In Section 3, we detail the architecture, implementation issues, and performance evaluations of this library. Section 4 illustrates the experimental results of our work and a short discussion on the future work of this on-going project.

2. Related Work

2.1. OpenGL ES

OpenGL ES [1] provides a *de facto* standard for 3D graphics programming on embedded systems, including hand-held devices, appliances, and vehicles. OpenGL ES is a subset of OpenGL[3] which creates a low-level interface between software applications and hardware or software graphics engines. Currently, OpenGL ES includes Common and Common-Lite profiles for floating-point and fixed-point systems and also the EGL (OpenGL ES Native Platform Graphics Interface) specification for portably binding to native window systems.

[1] http://java.sun.com/j2me/
[2] http://www.khronos.org/opengles/
[3] http://www.opengl.org/

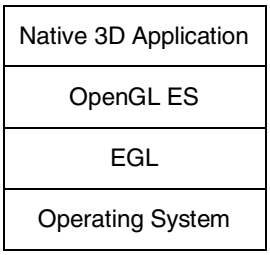

Figure 1: Hierarchy of OpenGL ES

Because of the widespread desktop version of OpenGL, OpenGL ES programming interface indeed provides a well-structured and intuitive design for programmers to use. Although OpenGL ES itself can be cross-platform via providing different EGL components, applications developed with traditional languages such as C/C++ are highly platform-dependent, i.e., the program might run well on a certain machine while it probably crashes on others. The hierarchy of OpenGL ES is shown in Figure 1.

2.2. J2ME

For cross-platform and memory usage requirements of mobile devices, J2ME has been presented. By adding a virtual machine between applications and operating systems, J2ME programs are inherently platform-independent and thus increase productivity for the programs running on mobile devices.

The virtual machine of J2ME, Kilobyte Virtual Machine (KVM), is much smaller than traditional Java Virtual Machine (JVM). It is designed to run under the constraints of the limited resources available on micro devices. Currently, J2ME has been one of the most widely-adopted developing platforms for mobile devices. The hierarchy of J2ME is depicted in Figure 2.

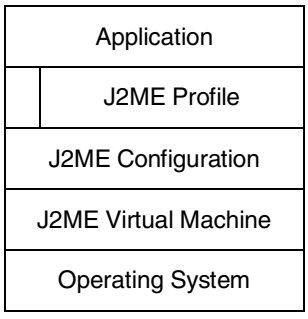

Figure 2: Hierarchy of J2ME

2.3. jGL

jGL[4] [2][3] is a 3D graphics library written in pure Java (J2SE[5]; Java 2 Platform, Standard Edition) on normal PC environments. It defines the application programming interface (API) in a manner quite similar to that of OpenGL. Unlike most similar products like Java3D[6]; jGL does not need any native codes or pre-installed libraries, it is developed with pure Java only. Users who run any 3D programs developed with it do not need to install any package before using them, but all required codes will be downloaded at run time instead. The scheme of the proposed 3D library for mobile devices originates from jGL.

2.4. Vincent OpenGL ES Library

Vincent[7] is an open source project that aims to provide an OpenGL ES 1.0 software implementation. A key feature of Vincent is that it provides a runtime compilation infrastructure that creates optimized rasterizer code for the current graphics context settings. Currently, Vincent can only run on the limited operating systems and processors including Microsoft Windows Mobile operating system using the Intel XScale PA2xx processor.

3. Architecture

3.1. Motivation

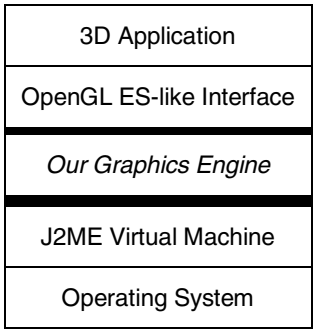

Figure 3: The hierarchy of the proposed 3D library on mobile devices.

In order to solve the cross-platform problem addressed in the previous section, a J2ME-based OpenGL ES 3D library for mobile devices is provided, which follows the industrial standard for embedded system and is platform-independent. The proposed library plays a role as a bridge for 3D graphics between the

[4] http://graphics.im.ntu.edu.tw/~robin/jGL/
[5] http://java.sun.com/j2se/
[6] http://java.sun.com/products/java-media/3D/
[7] http://ogl-es.sourceforge.net/

programmers and the native operating systems. The overall relationship among the proposed graphics engine, J2ME virtual machine and the native operating systems is illustrated in Figure 3. Hence, the programmer could only faces an OpenGL ES-like programming interface. The detailed rendering pipeline is hidden in our graphics engine.

3.2. Application Programming Interface

Since OpenGL ES is a well-known and widely-adopted 3D graphics library for mobile devices by most 3D programmers, we followed the specifications of OpenGL ES 1.0 [1] to develop our library. Therefore, the programmers who want to use the proposed library to develop their own mobile 3D programs do not need to learn how to use the new library; they can find a one-to-one mapping function in the proposed library and that in OpenGL ES 1.0.

A peep of the proposed library coding style is shown in Figure 4. As the figure shows, the proposed library provides a similar interface as OpenGL ES for programmers, thus they can use the proposed library like OpenGL ES except adding a prefix such as "myGLES".

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import opengles.GLES;
import opengles.GLfixed;

public class Sample extends Canvas{
  GLES myGLES = new GLES();

  private void init() {
    myGLES.glViewport(0, 0, iScreenWidth, iScreenHeight);
    myGLES.glClearColorx(ZERO,ZERO,ZERO,ZERO);
    myGLES.glMatrixMode(GLES.GL_PROJECTION);
    myGLES.glLoadIdentity();
    myGLES.glOrthox(ZERO,ONE,ZERO,ONE,ZERO,ONE);
  }

  private void paint(){
    myGLES.glClear(GLES.GL_COLOR_BUFFER_BIT |
                   GLES.GL_DEPTH_BUFFER_BIT);
    myGLES.glRotatex(ANGLE,ROTATE,ROTATE,ROTATE);
    my-
GLES.glEnableClientState(myGLES.GL_VERTEX_ARRAY);
    myGLES.glVertexPointer(3,myGLES.GL_INT,0, MODEL);
    myGLES.glDrawArrays(myGLES.GL_TRIANGLES,0,100);
    myGLES.glPopMatrix();
  }
}
```

Figure 4: Coding style of the proposed library

3.3. Architecture of the Graphics Engine

The architecture of the proposed library originates from jGL, an OpenGL-like 3D library for Java (J2SE). We modify the architecture of jGL accordingly to make it fit the requirements of the mobile devices.

To enhance the performance of the proposed library, we utilize the class inheritance to redesign the system hierarchy of the proposed library as shown in Figure 5. The graphics context of the proposed library can be divided into two parts. One is for changing some information stored in the graphics context without performing real actions and the other performs some actions and directly produces some changes.

The proposed library is defined as a state machine like OpenGL ES, and the states of the proposed library have been classified into several categories including flat or smooth shading, with depth test or without and so on. Therefore, running in different states will need different clipping, geometry and rendering routines.

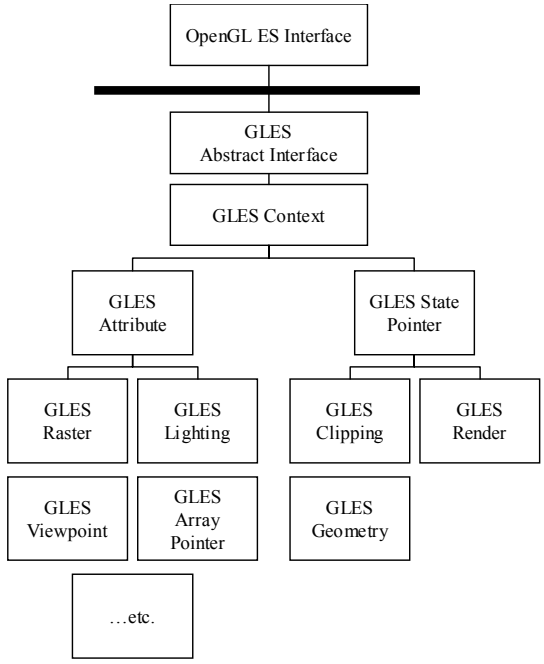

Figure 5: The kernel architecture

3.4. Performance and Memory Usage Issues

Currently, most mobile devices support much weaker processors than PCs and are equipped with limited size of memory. Thus performance and extreme memory limitation are great challenges for 3D graphics on mobile devices. Based on our experience, we develop the following design strategies to speed up the proposed 3D library's performance and run-time memory usage:

3.4.1. Support floating point data type on mobile devices. Most mobile devices are lack of floating point accelerator, thus J2ME CLDC[8] (Connected, Limited Device Configuration) 1.0 does not support floating point data type. In order to handle floating point requirement for 3D graphics, we provide a well-tuned class *GLfixed* which emulates the behavior of floating point and use table-lookup scheme whenever possible (ex. sin/cos) to speed up run-time calculation.

3.4.2. Reduce the frequency of object construction and destruction. Frequently constructing and cleaning up objects via garbage collector process make the graphics engine hopelessly slow. Therefore, we precreate a set of objects which are then dispatched on demand by the graphics engine. Through this mechanism, we self-manage the memory usage of the proposed library in order to reduce runtime object creation and deletion operation especially the *GLfixed* floating point objects.

3.4.3. Avoid deep class inheritance. It would be a great overhead on J2ME environment to have a too deep class inheritance, thus we redesign the class inheritance hierarchy of the proposed library to avoid such situation.

3.4.4. Minimize total number of object creation. Memory is a limited resource in embedded environments, and consequently we adopt object-recycling scheme by which different data can share the same object to minimize total amount of created objects in our library and hence can reduce the memory usage.

4. Results

In order to test our graphics library, we use a ASUS P505 as our testing environment, which is a mobile phone with a 500MHz CPU running Microsoft Windows CE platform. The demo programs are compiled with J2ME CLDC 1.0 / MIDP[9] (Mobile Information Device Profile) 2.0. To test the cross-platform ability, besides running the demo programs on J2ME Wireless Toolkit 2.1 default emulator, we also use SonyEricsson P900 emulator and some real devices for our developing and testing environments. Figures 6 and 7 show different rendering results of the demo programs which are compiled with our proposed library and running on the emulators. Table 1 lists the experimental results while running the demo programs on ASUS P505.

[8] http://java.sun.com/products/cldc/
[9] http://java.sun.com/products/midp/

Figure 6 (Screenshot from Emulator): Left: Smooth shading of a cube, Right: Flat shading of a cube with scaling along y-axis.

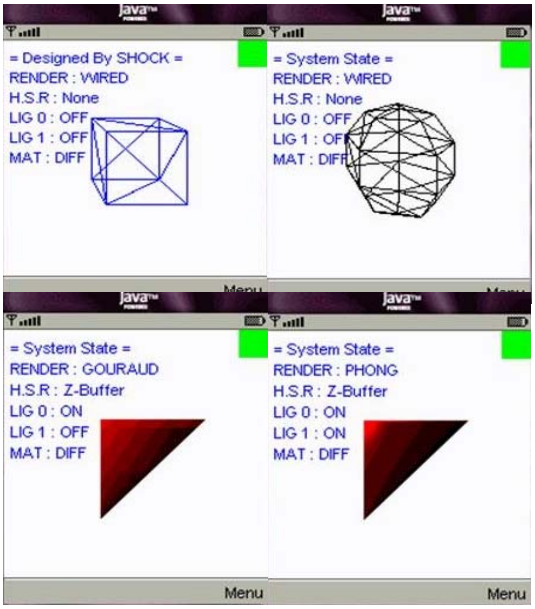

Figure 7 (Screenshot from Emulator): Left-upper: Wired-frame of a cube. Right-upper: Wired-frame of an icosahedron. Left-lower: Smooth shading of a single triangle. Right-lower: Phong shading of a single triangle.

Table 1: Test results of real devices.
Test Environment: J2ME CLDC 1.0 / MIDP 2.0
Test Devices: ASUS P505

Model	Cube	Icosa-hedron
#Polygon	12	60

Render Effect	Wired-frame	Flat shading	Smooth shading	Wired-frame
Time	0.03s	0.2s	0.3s	0.17s

Though the performance of our 3D graphics library is limited by the power of processors, we indeed provide a convenient and efficient way to build up a 3D program on portable devices. Currently, the 3D graphics library supports transformation, rendering options including wired-frame, flat-shading, smooth-shading and limited lighting effects, etc.

In addition, we also have tested the proposed 3D graphics library on several mobile operating systems including Embedded Linux, Symbian and Microsoft Windows CE. Furthermore, backward capability is taken into consideration and we have tested the demo programs on a 2-years-old mobile phone (Nokia 6600) and they still run very well.

5. Conclusion and Future Work

We aim to provide a smooth way for people who are already familiar with OpenGL ES library for C/C++ language to apply our whole new platform-independent 3D graphics library for developing 3D applications on mobile devices. This is an ongoing research project and in our current version, the performance of our library is limited due to some redundant codes. We would continue to optimize the proposed library and provide more functionality including full lighting functions, texture mapping, fog, etc. As for the convenience to C/C++ programmers, we are going to develop a translator which can translate C/C++-based OpenGL ES codes to our proposed library codes automatically to increase the productivity.

6. Acknowledge

We would like to thank Ting-Hao Huang for creating the models for our testing. This work was partially supported by the National Science Council of Taiwan under the contract number: 93-2622-E-002-033.

7. References

[1] D. Blythe, *OpenGL ES Common/Common-Lite Profile Specification Version 1.0*, Khronos Group, 2004.

[2] B.-Y. Chen and T. Nishita, "jGL and Its Applications as a Web3D Platform", *ACM Web3D 2001 Conference Proceedings*, pp. 85-91, 2001.

[3] B.-Y. Chen, T.-J. Yang, and M. Ouhyoung, JavaGL - a 3D Graphics Library in Java for Internet Browsers, *IEEE Transactions on Consumer Electronics*, Vol. 43, No. 3, pp. 271-278, 1997.