6.1-4

jGL ES - a J2ME-based 3D Library for Mobile Devices

Cheng-Han Tu and Bing-Yu Chen, *Member, IEEE*
National Taiwan University

Abstract**--To provide 3D programs for all kinds of mobile devices, platform-dependent is still a big problem. In this paper, we present a platform-independent 3D graphics library, jGL ES, which is based on J2ME and has a similar programming environment as OpenGL ES. With jGL ES, the programmers can follow the same programming style of the framework of OpenGL ES and smoothly transfer their source codes to be cross-platform applications.**

I. INTRODUCTION

The demand for showing 3D graphics on mobile devices increases drastically in recent years as a result of portable devices become more and more prevalent today. For this purpose, OpenGL ES is adopted widely as a *de facto* industry standard for embedded systems or mobile devices. However, different mobile devices have different operating systems as well as programming environments at this moment. Hence, the 3D applications written by C/C++ with OpenGL ES library on portable devices are highly platform-dependent, i.e., the programs might run well on certain machines but probably crash on others. Therefore, in this paper we present jGL ES, a platform-independent 3D graphics library based on J2ME (Java 2 Platform, Micro Edition) environment and has a similar programming architecture with OpenGL ES for mobile devices, since J2ME can provide a cross-platform environment for almost all kinds of mobile devices.

II. ARCHITECTURE

From users' viewpoint, jGL ES is an interface between 3D applications and native operating systems. The system hierarchy and architecture of the graphics engine of jGL ES is shown in Fig. 1.

A. *Application Programming Interface*

jGL ES is implemented by following the specifications of OpenGL ES 1.0 [1], which is a well-known and widely-adopted industry standard. However, since OpenGL ES and its ancestor, OpenGL, are not designed for J2ME or Java platform, to develop jGL ES using pure Java on J2ME environment, we have to redesign its application programming interface (API). Because Java is an object-oriented programming (OOP) language, we encapsulate all functions in various functional objects and provide a GLES class for programmer use. Each function in OpenGL ES has a corresponding one in jGL ES. Hence, the users can use jGL ES in the same way as using OpenGL ES except they have to use these functions in an object-oriented manner as shown in Fig. 2.

This work was partially supported by the National Science Council of Taiwan under the numbers: NSC93-2622-E-002-033.

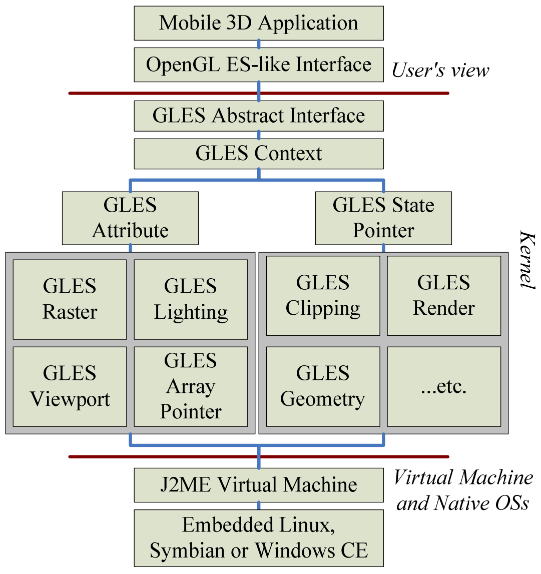


Fig. 1. The system hierarchy and architecture of the jGL ES graphics engine.

```
GLES myGLES;
private void paint() {
    myGLES.glClear(GLES.GL_COLOR_BUFFER_BIT |
                         GLES.GL_DEPTH_BUFFER_BIT);
    myGLES.glPushMatrix();
    myGLES.glRotatex(ROTATE_ANGLE, R_X, R_Y, R_Z);
    myGLES.glEnableClientState(GLES.GL_VERTEX_ARRAY);
    myGLES.glVertexPointer(3, GLES.GL_INT, 0, MODEL_VERTEX_ARRAY);
    myGLES.glDrawArrays(GLES.GL_TRIANGLES, 0, 100);
    myGLES.glPopMatrix();
}
```

Fig. 2. A part of an example code of jGL ES.

B. *Architecture of Graphics Engine*

The architecture of jGL ES is originated from jGL [2], which is a 3D graphics library for Java (J2SE) with an OpenGL-like API. Since the J2ME environment has more constraints than J2SE platform, we modify several parts of the architecture to fit the requirements of developing 3D applications in mobile environments.

To enhance the performance of jGL ES, we utilize the class inheritance to redesign the system hierarchy of jGL ES as illustrated in Fig. 1. The graphics context of jGL ES can be divided into two parts. One is for changing the information and states stored in the graphics context without performing real actions; the other performs real actions and results in changes directly. Since jGL ES is defined as a state machine like OpenGL ES, the states of jGL ES have been classified into several categories including flat or smooth shading, with or without depth test (z-buffer), etc. Hence, different clipping, geometric and rendering routines are pointed and called by the GLES State Pointer on the fly in different states.

C. Performance and Memory Usage Issues

Currently, most mobile devices use low-cost processors and rather limited memories. Consequently, to provide better performance with lower memory requirements is a great challenge for supporting 3D graphics on mobile devices. Based on our experience, we design the following strategies to speed up the performance of jGL ES but use less run-time memory:

1) Support floating point data type

Most mobile devices are lack of floating-point accelerator, thus J2ME CLDC 1.0 does not support the data type of floating-point. In order to handle the calculation using this data type for 3D graphics, we provide a well-tuned class GLfixed in jGL ES which emulates floating-point data type and use table-lookup method for acceleration, including the calculation of trigonometric functions.

2) Avoid deep class inheritance

It would be a great overhead on J2ME environment to have deep class-inheritance hierarchy, thus we design the class inheritance hierarchy of jGL ES to avoid such situation.

3) Reduce the frequency of object (de-)construction

Memory allocation is a very time-consuming process, so constructing and deconstructing objects frequently would bring down the performance. Hence, we manage the memory by ourselves by pre-creating a set of objects and deliver them out on demand. Through this mechanism, we not only use the memory efficiently but also reduce the frequency of runtime object creation and deletion to have a higher performance.

4) Eliminate the redundancy of object properties

To reduce the run-time memory use, we also adopt Fly-Weight design pattern to share the objects which have common, intrinsic, and invariant information, such as the same constants.

III. EXPERIMENTAL RESULT

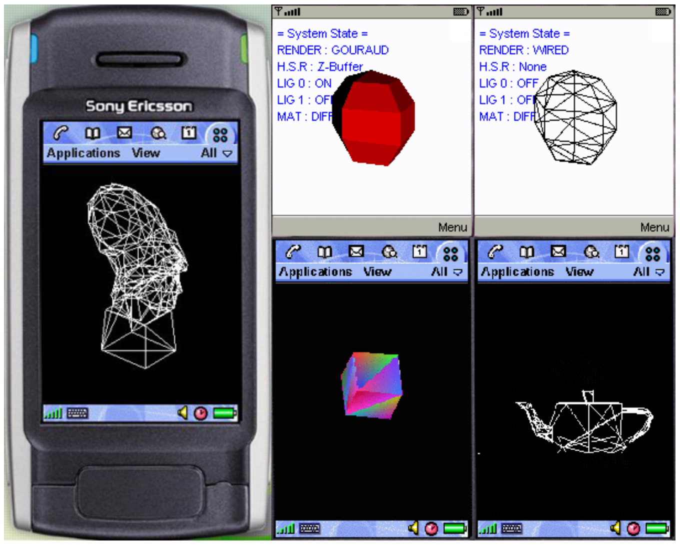
To test the performance of jGL ES, we use a Motorola A780 mobile phone with a 318MHz CPU running on Embedded Linux platform as our testing environment. Our demo programs are compiled with J2ME CLDC 1.0 / MIDP 2.0 and tested on J2ME WTK 2.1 default emulator, SonyEricsson P900 emulator and some real mobile devices. Fig. 3 shows some different rendering results on the emulators and TABLE I lists the testing results on the real devices. We also have tested jGL ES on several other mobile operating systems including Symbian and Microsoft Windows CE. Furthermore, to test the cross-platform capability, we also tested jGL ES on an old mobile phone (Nokia 6600). The result shows that the 3D program developed with jGL ES can run very well on such an old device.

IV. CONCLUSION AND FUTURE WORK

By using jGL ES, we provide a smooth way for those who are familiar with OpenGL ES or OpenGL libraries using C/C++ language to make their 3D programs to run on almost all kinds of mobile devices. Although the performance of current version is still limited due to the power of the processors, we indeed provide a convenient and efficient way to build up a 3D program on almost all kinds of portable devices. Currently jGL ES supports 2D/3D transformations, some rendering options including wired-frame, flat-shading, smooth-shading and limited lighting effects, etc.

Performance and memory limitation are still the great challenges for us to develop jGL ES. Since jGL ES provides one-to-one mapping functional objects as OpenGL ES functions, for the convenience to C/C++ programmers, it is possible to provide a translator or pre-compiler to translate the C/C++-based OpenGL ES codes to jGL ES codes automatically.

Our major contribution in this paper is not only for providing a 3D library for mobile devices, but also a cross-platform and familiar programming environment. Since many companies are developing GPUs for mobile devices, by porting the kernel of jGL ES to take the benefits of the native GPUs but providing the same jGL ES API for 3D applications, programmers can easily develop a platform-independent application and take advantages from the native devices.


= System State =
RENDER : GOURAUD
H.S.R : Z-Buffer
LIG 0 : ON
LIG 1 : OF
MAT : DIF

= System State =
RENDER : WIRED
H.S.R : None
LIG 0 : OFF
LIG 1 : OF
MAT : DIF

Fig. 3 (Screenshot from some emulators). Left: A wired-frame Athena model. Left-Upper: A flat-shaded icosahedron with a single (Red) lighting source. Right-Upper: A wired-frame icosahedron. Left-Lower: A smooth-shaded cube. Right-Lower: A wired-frame teapot.

TABLE I
TEST RESULTS OF REAL DEVICES

Test Environment: J2ME CLDC 1.0 / MIDP 2.0 **Devices:** Motorola A780

Model	Teapot	Icosahedron	
#Polygon	100	60	60
Rendering mode	Wired-frame	Wired-frame	Flat-shading + lighting (single light)
Time	0.3s	0.2s	2.6s
Model	Cube	Anthena	
#Polygon	12	400	
Rendering mode	Smooth-shading	Wired-frame	Smooth-shading
Time	0.6s	0.7s	4.9s

REFERENCE

[1] *OpenGL$^{®}$ ES Common/Common-Lite Profile Specification Version 1.0.02*, D. Blythe, Eds., 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.