Technology for creating 3D realtime applications in Android OS

Polotnyanschikov I.S.
mjollneer@gmail.com

Scientific advisor: Zalogova L.A.
zalogova.la@gmail.com

Software and Computing Systems Mathematical Support Perm State University
Perm, Russia

Abstract— **This article discusses the development of technology for creating 3D realtime applications for OS Android. Described tools selection, data domain analysis and realization. Also described process of creating new light model for luminous segment. The results of the work are illustrated with screenshots of real application.**

Keywords-Androd; 3D graphics; Opengl ES; OOP; shaders; lighting model

I. INTRODUCTION

Creation of 3D applications for Android OS became especially actual after devices using this platform spread over the world, outstripping its rival Apple iOS.

Devices with Android OS belong to differing classes of performance and appointment – from a MP3 players and a watches to tablet computers. Users of this devices often require high-quality 3D visualization in real time.

The purpose of the work is to develop technology of creation of interactive applications for the mobile devices using 3D graphics. Achievement of this purpose require to solve the following problems:

- investigate features of development 3D applications in Android OS,
- prove use of an object-oriented paradigm,
- design and implement hierarchy of classes for creation of 3D appendices.

II. CURRENT STATE OF 3D GRAPHICS IN ANDROID

Nowadays best engines of 3D graphics in Android presented by following two categories:

A. Commercial engines

Best commercial engines are Corona SDK and UNITY 3D [9, 10]. No doubt, they offer very powerful cross platform solutions. Library of different multimedia resources, music subsystem, physics emulation, strong animation and most modern visual effects. Customers also get professional support and many tools for development 3D applications. And the cost from $200 to $1500 annually.

B. Free engines

Best free engines are AndEngine and LibGDX [11, 12]. They both corresponds big difficult systems with many functions. Both of them contains many subsystems, like sound, file i/o, animation, physics. LibGDX is crossplatform, AndEngine is 2D-only. Support here is forums and manuals.

Note now, that cross platform source code is noticeably slower than native, because of different wrapping technologies. Also high complexity made mentioned engines difficult to understand and use.

So, taking into account all of the above, seems urgent to develop easy-to-use technology for creating android-only hardware accelerated real-time 3D graphic.

III. INTERFACE CHOICE

There are different approaches to render graphics in Android [7]. OpenGL ES library was chosen as the hardware-software interface. It is recommended by developers of Android for creation of high-efficiency applications [3]. Besides, OpenGL ES allows to reach the most qualitative result.

At present the majority of mobile devices work under control of Android 2.3 or more senior version. In these devices OpenGL ES 1.0 and 2.0 is supported at the same time. So there is a question of a choice.. Each subsequent OpenGL version for the personal computer comprises all functionality of the previous versions. At the same time senior and younger OpenGL ES versions contain essentially different functionality. Therefore, they solve identical problems in qualitatively different ways. For example, the programmer should create a part of functionality of ES 2.0 by means of special programs – shaders. It is impossible to recognize the senior version unequivocally better than the younger. It is necessary to choose the version allowing in the best way to achieve the object of work. This choice will directly affect structure of the developed technology.

IV. OPENGL ES VERSION CHOICE

During comparison of OpenGL ES versions distinctions in their syntax [1,2] and functionality were analysed. Results presented in tab. 1.

TABLE I. RESULT OF COMPARING OF OPENGL ES VERSIONS

Criterion	OpenGL ES 1.0	OpenGL ES 2.0
Code amount	Less	More
Program structure	More easy	More difficult
Result	Worse	Better
Number of supported effects	Less	More
Perfomance	Less	More
Rendering setup	By means of parameters of rasterization, texturing, lighting, etc.	By means of shaders

OpenGL ES 1.0 is suitable when speed of development is more important than image quality. But in this work performance and quality of result interests us first of all. So version 2.0 corresponds us in the best way. In ES 2.0 growth of number of geometrical objects cause the size of the program considerably increases.

Note, that there are different paradigms available for anfroid-developers. For example procedural paradigm and even workable bindings for OpenGL and LISP (functional paradigm) [5,6]. But, for effective management of large amount of a code the object-oriented paradigm was selected.

V. CLASS HIERARCHY DEVELOPMENT

In the course of creation class hierarchy it is necessary to consider features of data domain and the instrument of implementation. The data domain (a 3D graphics) is described in such terms as the camera, geometrical object, a material, a light source. The developed hierarchy includes the classes corresponding to terms specified above. However, their fields, methods and relations substantially depend on features of OpenGL ES 2.0.

For detection of features of ES 2.0 the test program consisting of several geometrical objects, shader objects and attributes of vertexes was written. Specific organization of this program allowed to select repeating parts of a code and data with similar behavior. This features became fields and methods of classes.

The developed object model were named "Lit Engine". In paragraphs A-D classes of LitEngine grouped by implication and described in outline.

A. Creation of geometrical objects

Containers of data – the abstract class LitDataContainer and its successors. They intended to storage attributes of any dimensionality and assignment.

Factory of data – the abstract class LitDataFactory and its successors. They intended to filling data containers with the attributes describing one geometrical object.

A transformation matrix – the class ModelViewMatrix. It is intended to storing and processing matrix in terms of transformation of coordinates.

The universal object – the class Universal3DObject. It is intended for storage all information describing one geometrical object.

B. Light setup

Material – the abstract class LitAppearance and its descendants. They intended to storing settings for the specific shader program.

The effect manager – LitSpecialEffect and its descendants. Effect manger can tune one shader program for one universal 3D object using one specified material."Material".

C. Camera setup

Projection matrix - the class ProjectionMatrix. It is intended to storing and processing matrix in terms of projection of coordinates.

The camera – class LitCamera. It is intended to operate a projection matrix in terms of setup of the camera.

D. Rendering

Shader program – class glslProgramm. It is intended to encapsulate all operation with one shader: loading, compilation, linking, setup and activation.

VI. DEVELOPMENT OF NOT POINTWISE LIGHT SOURSE MODEL

Let’s take a close look on process of development new light model.

A. Formulation of the problem

Lighting calculation by means of shaders traditionally use models of pointwise light sources. For example Ward's models, Lambert, Gooch, Blinn and Phong [13]. These models appeared preferentially to simulate more and more difficult materials.

However, obviously, it is not enough in those situations when it is impossible to neglect the form or the sizes of a source – a lamp shade occupies essential volume and creates dim shadows, the lamp of day lighting shall illuminate like shining cylinder and be mirrored like a bright straight line. The screen of the computer shall be processed as rectangle occupying a certain fixed space and creating an adequate flare on smooth surfaces.

Let's call such objects not pointwise, i.e. consisting of more than one point.

This task is certainly solved by means of global illumination models in which emit and reflect light can any polygon. Such decision provides the high-quality image, generated for the long time every frame. For real-time applications similar physically accurate solution was not found by the author.

Within this work the problem of simulation of not pointwise light sources was posed and solved.

B. *Solution*

First of all let's describe, how lighting from a luminous segment empirically shall be created.

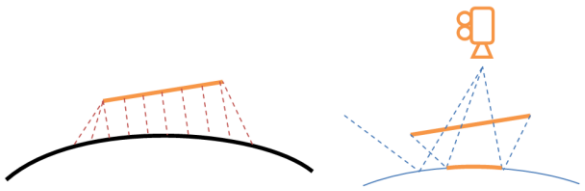

Figure 1. Desirable light for luminous segment

On fig. 1 we see how diffuse (left) and specular (right) light should spread in space near luminous segment.

Note that diffuse lighting is the brighter where the perpendiculars lowered from a segment to a surface.

Author made the assumption that in case of calculation diffuse lighting of each separate fragment it is possible to replace a luminous segment without loss with one pointwise light source. This pointwise source at the same time should belong to a segment and be placed as close to a lighted fragment as possible. Searching of such points on a segment for different surface fragments is schematically figured by orange dotted lines.

Let's note that the mirror flare is brightest on fragments which reflect a vector of a look *v* as precisely into a segment as it is possible (blue dotted lines).

We will make the assumption, based on reasons of common sense, that in case of calculation specular lighting for each surface fragment it is possible to replace a luminous segment with one pointwise light source without loss. This source shall belong to a straight line passing through a segment. At the same time it must be as close to reflected eye vector *v* as possible.

In other words, summary it is necessary to calculate coordinates of two light sources – the first will give us diffusion component, the second – specular.

It is necessary to calculate these coordinates for each surface fragment for every frame.

Specific values of diffusion and specular components can be calculated by means of any pointwise illumination model, for example Blinn.

Below illustrated (fig. 2) a model of diffusion light for segment and one surface fragment.

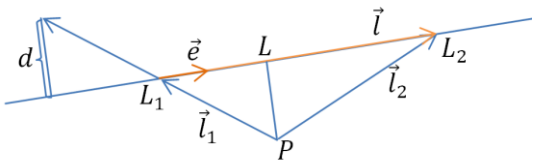

Figure 2. Geometrical model of luminous segment

P – point for which intensity computation is made (surface fragment)

L_1, L_2 – coordinates of the ends of a luminous segment

L – required pointwise diffuse light source

d – distance to a straight line passing through a segment

Computation of required coordinates without intermediate formulas:

$$\vec{e} = \frac{\vec{l}}{|\vec{l}|} \quad (1)$$

$$L = L_1 + \vec{e} \cdot |L_1 L| \quad (2)$$

The received formula is correct, only if the point *L* belongs to a segment. If point *L* is outside of segment, it is necessary to replace it with one of the segment ends.

For computation of coordinates of the second light source it is necessary to find a point on segment close to reflected *v* vector.

$$L = L_1 + m \cdot \vec{l} \quad (3)$$

Equation for coefficient *m* will not be shown in this article. If *m* is not between [0,1], it must be replaced with L_1 or L_2.

Described model was realized via GLES shaders and included in LitEngine.

VII. Example of application of LitEngine

Fig. 3 and 4 is the screenshots of demo application running on mobile device. By means of LitEngine the scene consisting of a set of cubes and illuminated with one luminous segment is rendered in real-time (about 20fps and 600 polygons on Adreno 205 GPU). It is important that luminous segment create adequate diffuse and specular lighting.

Figure 3. Example of use of LitEngine

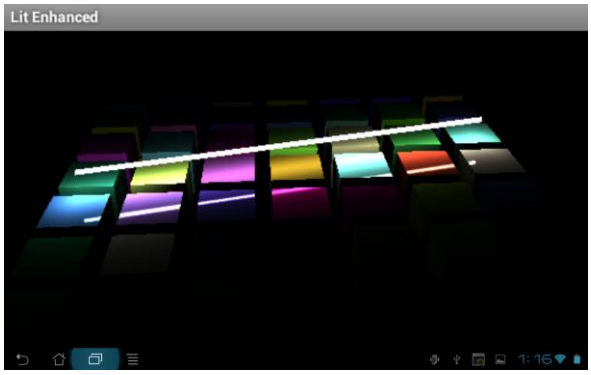

Figure 4. Example of use of LitEngine

VIII. CONCLUSION

Possibility of a reuse of the developed hierarchy was proved in practice by a means of several demonstration applications with a different set of geometrical and shader objects.

Extensibility of the developed hierarchy is proved by one of probable scenarios of extension. For example support of new geometry – the torus was added. Torus was completely incapsulated in one new class inherited from the abstract factory of data. For use of new geometry in the user application its requiring just to rewrite one line of code.

Thus, the developed hierarchy can be reused in case to solve tasks from different data domain (game, simulation modeling, advertizing, editors, GIS, augmented reality, etc.). Further it is necessary to expand model with additional special effects, use JNI [8], include support for external models and create more demo applications.

REFERENCES

[1] Leech J. OpenGL ES Common Profile Difference Annotated Specification 2.0.25. URL: http://www.khronos.org/registry/gles/specs/2.0/es_cm_spec_2.0.25.pdf

[2] Munshi A. OpenGL ES 2.0 Programming Guide. Boston: Addison-Wesley, 2009. 457 c.

[3] Android DevGuide. Graphics URL: http://developer.android.com/guide/topics/graphics/index.html

[4] Imagination Technologies Ltd. Migration from OpenGL ES

[5] Spare time projects: OpenGL and Lisp URL: http://www.mindstab.net/spare-time-projects-opengl-lisp/

[6] Lisp and Android SDK URL: http://stackoverflow.com/questions/5683543/lisp-and-android-ndk

[7] Android Developer. Graphics URL: http://developer.android.com/guide/topics/graphics/index.html

[8] Java Native Interface URL: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/index.html

[9] UNITY URL: http://unity3d.com

[10] Corona URL: http://www.anscamobile.com/corona/index.html

[11] AndEngine - http://www.andengine.org/

[12] LibGDX - http://libgdx.badlogicgames.com/documentation.php

[13] Light models URL: http://steps3d.narod.ru/tutorials/lighting-tutorial.html