



UNIVERSITAS INDONESIA



UNIVERSITÉ DE BRETAGNE SUD

**STUDY AND IMPLEMENTATION OF PORTING 3D
APPLICATION ON IPAD**

THESIS

**LANG JAGAT
0906577974**

**FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING
MULTIMEDIA AND INFORMATION NETWORK
DEPOK
JULI 2011**



UNIVERSITAS INDONESIA



UNIVERSITÉ DE BRETAGNE SUD

**STUDY AND IMPLEMENTATION OF PORTING 3D
APPLICATION ON IPAD**

THESIS

**Submitted to the Graduate Faculty of Engineering in partial fulfillment of
the requirements for the master degree**

**LANG JAGAT
0906577974**

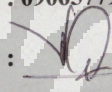
**FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING
MULTIMEDIA AND INFORMATION NETWORK
DEPOK
JULI 2011**

ORIGINALITY DECLARATION

I declare that this thesis is my own work, and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Name : Lang Jagat

NPM : 0906577974

Signature : 

Date : 4 July 2011

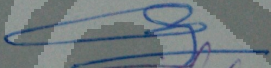


APPROVAL FORM

This Thesis is proposed by:

Name : Lang Jagat
NPM : 0906577974
Study Program : Electrical Engineering
Title : Study and Implementation of Porting 3D Application on iPad

Has been officially approved, supervised and finally examined by the examiners board authorized by Université de Bretagne-Sud, in partial fulfillment of the requirements for the degree of Magister Teknik at Electrical Engineering Department, Technical Faculty, Universitas Indonesia.

EXAMINERS BOARD

Supervisor : Nicolas COURTY ()
Tutor : Sylvie GIBET ()
Examiner : Gildas MÉNIER ()

Approved in : Vannes, France
Date : 4 July 2011

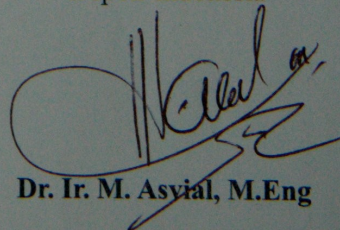
Acknowledged,

The Director of Formation M2 IIR,
Université de Bretagne-Sud,
Vannes-France



Nicolas COURTY

The Head of Electrical Engineering
Department, Universitas Indonesia,
Depok-Indonesia



Dr. Ir. M. Asyial, M.Eng

ACKNOWLEDGMENT

Thank's God for His blessings and His mercy, I can complete a thesis entitled "Study and Implementation of Porting 3D Application on iPad". Writing a thesis is done in order to meet one of the requirements to achieve the degree Master of Engineering, Department of Electrical Engineering Universitas Indonesia. I can not realize that, without the help and guidance from various parties, from the lecture to the preparation of this thesis. Therefore, I would like to thank:

1. Dr. Nicolas COURTY, as my supervisor who giving me the opportunity to work in their respective research groups for my project of internship in VALORIA laboratory, *Université de Bretagne Sud*.
2. Prof. Sylvie GIBET as my tutor, for all of her advice, support and enthusiasm during my internship.
3. Prof. Irwan Katili as Coordinator DDIP (Double Degree Indonesia Perancis) .
4. All the teachers in Department of Electrical Engineering University Indonesia.
5. All the teachers of UBS which are sometimes very severe, for teaching me and gave me valuable lessons during my studies.
6. To my family, my parents and my brothers and my sisters, my great family who are always close to me and gave me the support all the times.
7. My colleague that always stay for assisting me with technique.

Finally, I hope that God Almighty is pleased to reply to all the good of all parties who have been helped. Hopefully this thesis brings benefits to the development of science.

Vannes, 4 July 2011

**DECLARATION APPROVAL OF THESIS PUBLICATION
FOR ACADEMIC INTEREST**

As an academic community of Universitas Indonesia, the undersigned below:

Name : Lang Jagat
NPM : 0906577974
Study Program : Multimedia and Information Network
Department : Electrical Engineering
Faculty : Engineering
The kind of work : Thesis

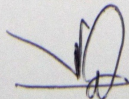
for the development of science, give non-exclusive royalty-free right to Universitas Indonesia this thesis with the title :

STUDY AND IMPLEMENTATION OF PORTING 3D APPLICATION ON IPAD

With this non-exclusive royalty-free right, Universitas Indonesia have permission to reproduction, storage in retrieval system, or transmission in any format or by any means, electronic, mechanical, photocopying, recording, or likewise with the copyright remain by the author.

Depok, 29 July 2011

Author



Lang Jagat

ABSTRACT

Name : Lang Jagat
Study Program: Multimedia and Information Network
Title : Study and Implementation of Porting 3D Application on iPad

A virtual character is graphics simulation of real or imaginary persons that enable natural can be interact with the user by means of voice, facial expression, gesture, and gaze direction. Many research has developed 3D virtual character animation not only in PC platform but also in Mobile device.

This Theses present an implementation of a 3D virtual character animation and rendering engine running on iPad using OpenGL-ES with Objective-C and C++. The Objective of this project is a port of the library SMR that has developed by the team SAMSARA from Valoria laboratory at *Université du Bretagne Sud (UBS)*. This library used for developing applications that make use of real time 3D graphics for visualization and animation of 3D virtual human character use sign language in French. Real time animation of virtual character on iPad represent a challenging task since many limitation must be taken into account with respect the processing power like graphics capabilities, disk spaces, and memory size. The original source code has been modified to make SMR library work on the iPad platform, and further optimizations have been done to increase the runtime performance as well.

Keyword : OpenGL 2 ES, Objective-C++, SMR Library

ABSTRAK

Nama : Lang Jagat
Program Studi: Jaringan Informasi dan Multimedia
Judul : Study and Implementation of Porting 3D Application on iPad

Karakter virtual merupakan simulasi grafik yang bisa berupa objek manusia riil maupun imajiner yang memungkinkan objek berinteraksi dengan pengguna baik melalui suara, ekspresi wajah, isyarat, dan pandangan. Banyak penelitian telah dilakukan dalam pengembangan karakter animasi 3D yang tidak hanya pada platform PC melainkan juga pada platform mobile.

Pada thesis ini membahas mengenai implementasi dari karakter animasi virtual 3D dan *3D engine* di iPad dengan menggunakan library graphics OpenGL ES dan bahasa pemrograman Objective-C dan C++. Tujuan dari proyek ini adalah *porting* library SMR yang telah dikembangkan oleh tim SAMSARA dari laboratorium SAMSARA di *Universtité de Bretagne Sud* (UBS). Library ini digunakan untuk mengembangkan aplikasi 3D *real time* untuk visualisasi dan animasi karakter manusia yang menggunakan bahasa isyarat Perancis (Sign Language on French). Animation *real time* dari karakter virtual di iPad harus memperhitungkan beberapa keterbatasan yang dimiliki oleh iPad seperti kemampuan grafik, ruang disk, dan ukuran memori. Source Code library SMR harus dilakukan beberapa modifikasi sehingga memungkinkan untuk berjalan dengan baik di platform iPad.

Kata Kunci : OpenGL 2 ES, Objective-C++, Library SMR

RÉSUMÉ

Un personnage virtuel est la simulation graphique de personnes réelles ou imaginaires qui permettent naturels peuvent être d'interagir avec l'utilisateur au moyen de la voix, l'expression du visage, le geste et la direction du regard. Beaucoup de recherches a développé animation de personnages 3D virtuels, non seulement dans la plate-forme PC, mais aussi dans l'appareil mobile. Ce rapport présente une mise en œuvre d'une animation de personnage virtuel en 3D et le moteur de rendu des vues fonctionne en utilisant OpenGL iPad-ES avec Objective-C et C ++.

L'objectif de ce projet est portage du SMR bibliothèque qui a mis au point par l'équipe SAMSARA du laboratoire Valoria à l'Université du Bretagne Sud (UBS). Cette bibliothèque utilisée pour développer des applications qui font usage de vrais graphics en 3D temps réel pour la visualisation 3D et d'animation de langue signe français (LSF). Animation en temps réel du personnage virtuel sur iPad représentent une tâche difficile car de nombreuses limites doivent être prises en considération à l'égard de la puissance de traitement, comme les capacités graphiques, espaces disques, et la taille de la mémoire. Le code source original a été modifié pour faire fonctionner la bibliothèque de SMR sur la plateforme iPad, et d'autres optimisations ont été faites pour augmenter les performances d'exécution aussi bien.

TABLE OF CONTENTS

PAGE OF TITLE.....	i
APPROVAL FORM.....	ii
ACKNOWLEDGMENTS.....	iii
ABSTRACT.....	v
TABLE OF CONTENTS.....	ix
LIST OF FIGURES.....	x
CHAPTER 1.....	1
INTRODUCTION.....	1
1.1 Presentation of Valoria Laboratory.....	1
1.2 Introduction of Project 3D on iPad.....	3
1.2.1 Objectives.....	4
1.2.2 Scope of Project.....	4
1.2.3 Requirement.....	5
1.2.4 Software Development Plan.....	7
1.3 Thesis Outline	8
CHAPTER 2	9
BACKGROUND	9
2.1 iOS SDK	9
2.2 Concept 3D Engine	11
2.2.1 Resource Handling	11
2.2.2 Scene Graphs	16
2.2.3 Rendering	16
CHAPTER 3	18
PROGRAMMING WITH OBJECTIVE-C ++ AND OPENGL ES.....	18
3.1 Using Objective-C with C++	19
3.2 OpenGL ES Library	19
CHAPTER 4	21
IMPLEMENTATION	21
4.1 Description of SMR Library.....	21
4.2 SMR Library on iPad	23
4.3 Objective-C and C++	26
4.4 OpenGL to OpenGL ES	29
CHAPTER 4	33
CONCLUSION AND FUTURE WORK.....	33
5.1 Conclusion	33
5.2 Future Work	33
BIBLIOGRAPHY	34

LIST OF FIGURES

Figure 1.1 Project Timeline.....	7
Figure 2.1 iPhone Programming Stacks	10
Figure 2.2 Model Human Object in Wireframe Mode.....	12
Figure 2.3 Texture with u,v Dimension	13
Figure 2.4 File BVH Format	14
Figure 4.1 Cross Platform SMR Library for OpenGL ES and MacOS/Wind..	22
Figure 4.2 Class Diagram generally of Project	22
Figure 4.3 Class Diagram for Objective-C	23
Figure 4.4 Code Snippets definition of Class GLView	23
Figure 4.5 Code Snippets implementation of Class GLView	24
Figure 4.6 Class Diagram SmrApplication and SmrHumanoid in C++.....	29
Figure 4.7 Code Snippets from SmrRenderer in OpenGL.....	30
Figure 4.8 Code Snippets from SmrRenderer in OpenGL ES.....	31

CHAPTER 1

INTRODUCTION

In this chapter, described the introduction of the these work. Includes a presentation of laboratory Valoria, introduction the background of the project topic, problem formulation, objectives, constraints problems, working methodology.

1.1 Presentation of VALORIA

Valoria is the computer science laboratory at *Université de Bretagne Sud*. It develops research activities in the scope of Ambient Intelligence (AmI), addressing research and development topics along three complementary research activities:

1.1.1 Interaction and Intelligence

This first activity aims at providing end-users with technological, innovative means for greater user-friendliness, more efficient services support and user-empowerment, while contributing to user-friendly, dependable, adaptive and non-intrusive hardware/software environments.

1.1.2 Software Architecture

The second activity is dedicated to architecting/refining, testing/refactoring and maintaining/evolving dynamic, distributed, mobile and context-aware systems considered as the background support to ambient computing.

1.1.3 Middleware

The third activity focuses on providing middleware for distributed mobile and communicating systems as a support to ubiquitous and pervasive computing.

There are some group that work in these domain, one of them is Group SAMSARA. Group SAMSARA (synthesis and analysis of motion for the simulation and animation of realistic agents) has research activities on model gesture and motion in a computer graphics context. The objectives are centered on real time simulation of biomimetic gestures, with a particular focus on realism and reactivity to the environment. The finality is to build interactive, communicative and artificial humanoids endowed with human-like gestural abilities.

The aim of the research conducted by SAMSARA is to model and simulate motion, with respect to different application fields mainly to design virtual humanoids endowed with communicative behaviors. The build synthetic characters can communicate with the user in a human-like way. This supposes to have a better understanding of the inherent mechanisms of sensorimotor control for communication gestures and skilled motion, as well as to build animation methods to embody the virtual character. Realism can be achieved by developing synthesis models with respect to sensory-motor principles, or to integrate within animation system properties of human motions extracted from data captured on real subjects. Communication and interaction can be supported by high-level description languages and real-time control mechanisms.

Some of them are dedicated to the development of software tools and methods for creating and animating communicative virtual characters, which can be embedded in immersive devices or in mobile communication systems (PDA, etc.). This application domain fits with some objectives of Ambient Intelligence, in the sense that (intelligent) interactions might be improved by using humanoid representations as a mean of communication (humanisation of interface). The principles of the different animation prototypes might also interest the video games community, 3D oriented CAO tools, and robotics (new paradigms to control robots). Other virtual reality applications are related to the design, evaluation and adaptation of workplaces dedicated to physically disabled persons. In the context of mobile communication such as in-vehicle systems, gestural

interaction was studied based on gestural interaction techniques.

The Valoria laboratory is essentially located on the Campus of Tohannic, in the south of Vannes. Valoria offices are in ENSIbs building.

Postal address :

Valoria Laboratory

Université de Bretagne-Sud, Campus de Tohannic

BP 573

56017 Vannes cedex

Telephone : 02 97 01 35

Fax 02 97 01 72 79

1.2 Introduction of Project 3D on iPad

The research activities of the SAMSARA group cover a wide variety of applications. Some of them are dedicated to the development of software tools and methods for creating and animating communicative virtual characters, which can be embedded in immersive devices or in mobile communication systems. SAMSARA actively design and implement a generic platform for humanoid animation on C++ code along with an OpenGL based visualization.

The first one is dedicated to motion database handling. The second one, called Animation Engine , is responsible for managing actuators (which compute new motions or play captured motions) in combination with blenders (which blend several types of motion together). Finally the Rendering Engine is responsible for displaying the scene along with different virtual actors. This engine uses modern graphics features for enhancing the quality of rendering like, for instance, the OpenGL Shading Language.

This platform used to visualization and animation virtual actor for realization of avatar with French sign language (*LSF*). *LSF* is an effective method that used for communication between normal people to communicate with the deaf and hearing

impaired communities used in France regional area and countries who use a French language. There are many research that has developed in the domain of computer graphics to developing and proposed technique to facilitate process of communication.

The platform that was developed in computer based will be run on iPad device. Ipad with iOS constitute an interesting hardware platform for developers an application since so many people use it. However, Ipad are generally not specifically designed to support application 3D for animation, which poses problems for developers. One of these problems has been that Ipad have provided comparatively simple graphics.

1.2.1 Objectives

This project aims to porting the SMR library from PC platform to platform iPhone OS. SMR is an project for the creation of 3D real time applications that official release supports many operating systems, like Windows, Linux and Mac OS. On this project used OpenGL ES 2.0, objective-C and C++. This project also evaluate the effective process, the difficulty of that process conversion and evaluate the limitation of this platform. On the other hand, this project evaluate the graphics capabilities of current and upcoming iPad, specifically focused on 3D graphics using the OpenGL ES graphics programming interface version 2.0 (Opengl ES 2.0).

1.2.2 Scope of Project

This project focuses on the OpenGL ES API, and especially on versions OpenGL ES 2.0 and also mainly focuses on iPhone OS. Because of iPad use same operating system developed by Apple like on the iPhone. iPad device which supports OpenGL ES 2.0, was released during the creation of this project. The performance of the solution will tested with the real device.

The first step was of this project started by studying the concept OpenGL 2.1 on the project SMR and OpenGL ES 2.0 in order to understanding the principle of

OpenGL and shader. The goals were to examine how to develop iPad application with OpenGL ES 2.0 and how three-dimensional graphics and shaders can be successfully used in iPad.

The second step was integrated the code source of C++ on the SMR library with objective-C code, this is done because the on iPad development using the programming language Objective-C as a primary while the SMR code using C++. On other side to be an adjustment code from OpenGL to OpenGL ES. this is done because for embedded systems using the graphics library OpenGL ES.

1.2.3 Requirements

a. Hardware

On this project use iPad as environment where Apple was released iPad since April 2010 with runs the same operating system as the iPhone and the new released with support Wi-Fi + 3G has the following specification [6]:

- a) Device size and weight : Height: 9.50 inches (241.2 mm)
 Width: 7.31 inches (185.7 mm)
 Depth: 0.34 inch (8.8 mm)
 Weight: 1.35 pounds (613 g) (Wi-Fi + 3G model)
 Weight: 1.34 pounds (607 g (Wi-Fi + 3G for Verizon model)
- b) Display : 9.7-inch (diagonal) LED-backlit glossy widescreen Multi-Touch display with IPS technology, 1024-by-768-pixel resolution at 132 pixels per inch (ppi)
- c) Storage: 16 GB to 64 GB
- d) Processor : 1GHz dual-core Apple A5 custom-designed, high-performance, low-power system-on-a-chip
- e) Power and Battery : Built-in 25-watt-hour rechargeable lithium-polymer battery.

b. Programming Environment

Apple provide Xcode as an Integrated Development Environment (IDE) as a tool to developing software for Mac OSX and iPhone Software Development Kit (SDK) to development program in IOS. Xcode contains an iPhone simulator that

lets developers test their software without deploying it on the device. The Xcode includes version of free software GNU Compiler Collection (GCC apple-darwin) and support C, C++, Objective-C, Objective C++. It also contains various tools for debugging and measuring memory and CPU consumption [20].

In this project used the iOS SDK version 4.2, it is the core software of iPad as mobile Platform that developed by Apple. The iOS SDK is available only for OS X. it can be downloaded on Apple's iPhone developer site. With only the free SDK, it is possible to develop complex applications and even test them on the iPhone Simulator, because it includes with Xcode IDE and iPhone simulator. The iPhone simulator allows to run, test, and debugging of iPhone application directly on Mac. It also provides support for running both OpenGL ES 1.1 and OpenGL ES 2.0 based application.

Because of The iPhone OS is not open source, so for a developer, Apple provided 2 kinds of licenses, namely: The Standard Program, for an independent developer, which costs \$99 and The Enterprise Program, for a business developing team, which costs \$299. For iPhone application development software:

- a) The iPhone SDK, which contains the frameworks needed to successfully integrate your application as an iPhone app
- b) iPhone emulator, to test your application live on Mac.
- c) XCode, the developing environment which has a customization module for iPhone app development .
- d) Cross Compiler, that allow to the developer to programmed application in a different language than Objective-C, you can sometimes find a cross compiler for it, which will translate the binary code into Objective-C binary code .

1.2.4 Software Development Plan

a. Job Description

In the table list of jobs below show the description job with estimation and allocation time of in the internship in accordance with the time that has been set by

the university namely during the 22 weeks 3 days starting from 26 January to 1 July 2011.

No	Job Description	Estimation and Allocation Time		
		Duration	Begin Date	End Date
1.	Job Description	2 days	26-01-2011	26-01-2011
2.	Tool Preparation	2 days	28-01-2011	01-02-2011
3.	Study Concept Development	12 days	01-02-2011	17-02-2011
4.	iOS and OpenGL ES	16 days	17-02-2011	11-03-2011
5.	Objective-C and C++	14 days	11-03-2011	31-03-2011
6.	Study Code SMR	10 days	31-03-2011	14-04-2011
7.	Software Development 1	24 days	14-04-2011	18-05-2011
8.	Software Development 2	15 days	18-05-2011	08-06-2011
9.	Test	5 days	08-06-2011	15-06-2011
10.	Report	10 days	15-06-2011	29-06-2011

Table : List of Job with Estimation and Allocation Time

b. Project Timeline

In the figure 1.1 below show the timeline internship in accordance with the time that has been set as show in table above.

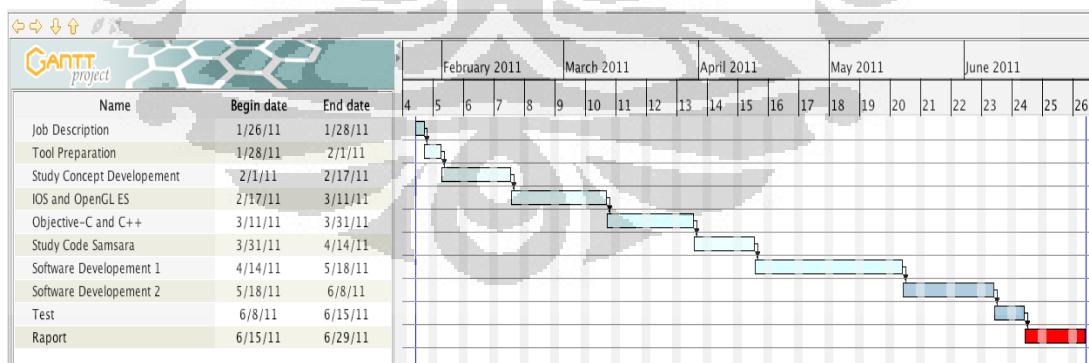


Figure 1.1 Project Timeline

1.3 Thesis Outline

The thesis can be divided into three main parts: Introduction, background, implementation/ evaluation, and Conclusions and Further Work.

1.3.1 Introduction

Introduction describes the introduction of this work. Includes a presentation of Laboratory Valoria, introduction of the project topic, objectives, requirements and Software development plan.

1.3.2 Background

Background provides some explanation of the concept of application development on the iOS platform. Some things that will be discussed in this section are about iOS SDK and concept of 3D engine, graphics libraries. This chapter describes the use of OpenGL ES graphic libraries.

1.3.3 Implementation/evaluation

In the part of Implementation/evaluation describes the details of implementation of the engine and prototype, as well as an evaluation of the results. The chapters of this part are as follows: Approach, Implementation details, focusing on the 3D graphics engine. An evaluation of results as well as some more general discussion about 3D graphics and animation for mobile phones and how the limitations of the hardware affect the design of engines for iOS platforms.

1.3.4 Conclusions and Further Work

This part provides information about work conclusions and suggestions for further study.

CHAPTER 2

BACKGROUND

In this chapter will explain the concept of application development on the IOS platform. Some things that will be discussed in this section is about IOS SDK and concept of 3D engine.

2.1 iOS SDK

The iOS platform consists of an operating system and a set of tools developed by Apple. The iPhone platform targets portable, multi-touch devices and is currently only available on the Apple iPhone, iPod touch and iPad. The iPad has its very own operating system, It is means that there is a maximization of the software-hardware compatibility. Apple neatly organizes all of the iPad's public APIs into 4 layers including the core OS : Cocoa Touch, Core Services, Media service, and core OS.

Most of the programs in this report were implemented in Cocoa Touch Layer and media layer as show in Figure 2.1. This layer is very important for application developers, because it contains the key frameworks that provide the infrastructure that need to implement applications. Cocoa Touch layer with UIKit frameworks, and Media layer. UIKit Framework is the frameworks that mainly used for the implementation of the graphical user interface, and the Foundation Framework, mainly used to communicate with the Core Foundation Framework. Media layer has 3 frameworks : Core Graphics, OpenGL ES, Quartz Core frameworks.

2. 1.1 Core Graphics frameworks

Core Graphics is Vanilla C interface to Quartz core Frameworks. The Core Graphics framework is a C-based API that is based on the Quartz advanced drawing engine. It provides low-level, lightweight 2D rendering with unmatched output fidelity. This framework used to handle path-based drawing, transformations, color management, off screen rendering, patterns, gradients and

shadings, image data management, image creation, masking, and PDF document creation, display, and parsing[2].

2.1.2 OpenGL ES frameworks

OpenGL ES frameworks is Low-level hardware-accelerated C API for rendering 2D or 3D graphics. In OpenGL ES has EAGL , tiny glue API between OpenGL ES and UIKit. Some EAGL classes (such as CAEGLLayer) are defined in Quartz Core framework, while others (such as EAGLContext) are defined in the OpenGL ES framework[13].

EAGL is derives from UIView class abstract, since all rendering on the iPhone must take place within a view. It class controls a rectangular area of the screen, handles user events, and sometimes serves as a container for child views. Almost all standard controls such as buttons, sliders, and text fields are descendants of UIView . EAGL is a small Apple-specific API that links the iPhone operating system with OpenGL..

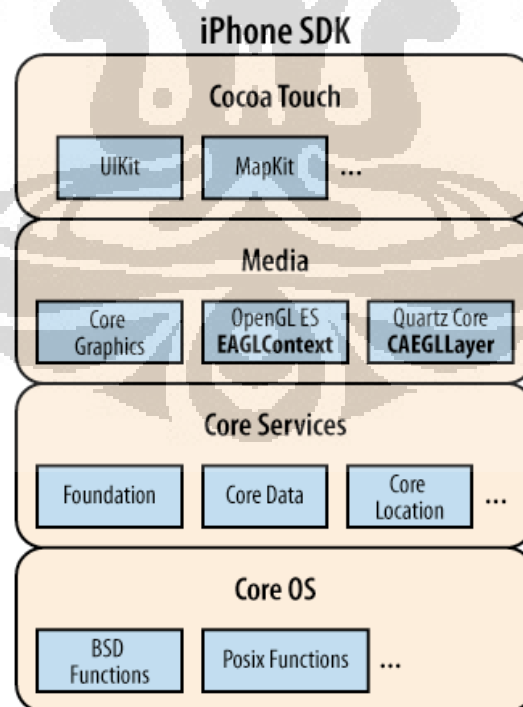


Figure 2.1 iPhone Programming stacks [13]

2.1.3 Quartz Core Frameworks

Quartz Core frameworks has Core animation that facilitates complex animations[13]. Quartz core has vector-based graphics library that supports alpha blending, layers, and antialiasing.

2.2 Concept of 3D Engine

This platform was originally developed to run on top of a Linux or Windows operating system. It would now be interesting to carry on a system like Android, which would include the use of devices like smart-phones.

Generally the core functionality typically provided by a 3D engine includes a rendering engine for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, and a scene graph. 3D Engine is a software which is intended to perform certain tasks, such as handling resources, scene graphs and rendering.

2.2.1 RESOURCE HANDLING

Resources are the content of the scene, i.e. what eventually is drawn on the screen. Resources may include models, textures, shaders, materials and animations.

a. Model

Models are the geometries of the scene, which are bound to objects in the scene graph. A model is essentially a list of polygons and associated information, such as texture coordinates or vertex normals[19]. Models can be generated through code, but most often they are read from a file, which usually has been exported from a modeling application, such as Autodesk's Maya or Blender and imported to an application using model loader[6][19]. The geometry consists of vertex data, as well as a surface description which describes how vertices are connected to create primitives such as polygons, triangle strips, lines or points. Figure 2.1 shows an example of model.

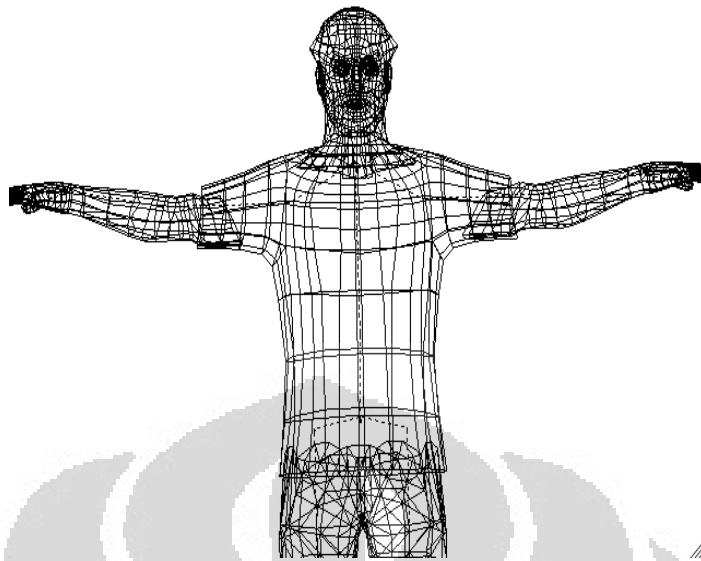


Figure 2.2 : Model Human Object in wireframe mode

b. Texture

To create 3D object becomes more attractive and more look like a real object usually add texture. The texture differentiates the model from others and provides more detailed information generally texture is image in 2D still with many format file like .bmp, .jpeg, .png, .gif, .tga. Files can be applied to a 3D surface (a triangle) by adding a pair of coordinates (U,V) to each vertex of the 3D mesh[14].

We use UV mapping method for mapping images onto mesh faces. We use several images from different angles to cover whole mesh. The UV means, that each vertex of a face has its UV coordinates in the texture space. The texture is interpolated during the mapping. Faces have an associated texture. The texture mapping is an affine transformation defined by the vertex UV coordinates. Textures can be one-, two- or three-dimensional, though OpenGL ES 2.0 only supports two-dimensional textures. When data is read from a texture it is usually filtered to make the output continuous and reduce aliasing. Figure 2.2 shows an example of texture in u,v dimension.



Figure 2.3 : Texture with U,V Dimension

c. Shaders

Shaders are short programs that are usually executed on graphics hardware or graphics processing unit (GPU)[6][19]. In the OpenGL ES 2.0 specification, there are two kinds of shader programs: vertex shaders and fragment shaders[6]. Shaders receive data in the form of attributes and uniforms. Attributes vary with every element that is processed and are provided either from the previous shader in the pipeline or by the engine. Uniforms on the other hand vary at most once per draw call. Typical examples of uniforms are object properties such as position, texture bindings and material properties. On Embedded system, there are 2 type of shaders available: vertex shaders and fragment shaders[6].

d. Animations

There are some of methods for animation 3D, one of the most common methods for is bone animation. This method can be implemented using either forward kinematics or inverse kinematics. The former is a technique for directly interpolating bone positions between certain positions, defined in keyframes[6]. Bone animations modifies the position, rotation or scale of certain nodes known as bones. Vertices in a model are linked to one or many of these bones in order to follow their movement. This technique is known as skinning. Bones are usually linked in a hierarchical manner to affect each other, mimicking the behavior of, for example, a skeleton[5].

Another common animation method is blend shapes, where multiple versions of a model (each with the same number of vertices) are loaded, and animation performed by interpolating between these variations[6].

We prefer to use motion capture data for real representation of human movements. For motion capture data we use the BVH format (BioVision Hierarchy). This format is widely used by animation software and can be imported into Blender or Maya. The BVH file format was originally developed by Biovision, a motion capture services company. The name BVH stands for Biovision hierarchical data. Its disadvantage is the lack of a full definition of the rest pose (this format has only translational offsets of children segments from their parent, no rotational offset is defined). The BVH format is built from two parts, the header chapter with joint definitions and the captured data chapter. Figure 2.3 shows an example of BVH file.

```

HIERARCHY
ROOT root
{
  OFFSET 4.68667 86.1523 3.06052
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT l_hip
  {
    OFFSET 10.9995 -8.42114 0
    CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
    JOINT l_knee
    {
      OFFSET 3.52859e-005 -38.8488 0
      CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
      JOINT l_ankle
      {
        OFFSET -0.00230834 -31.8504 0
        CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
        JOINT l_midfoot
        {

```

```

OFFSET 0 -6.23656 8.85409
CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
JOINT l_forefoot_tip
{
  OFFSET 0 0 7
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  End Site
  {
    OFFSET 1 0 0
  }
}
}
}
}
}
}
}
JOINT abdomen
.....
MOTION
Frames: 924
Frame Time: 0.01
2.15495 86.1264 3.67291 -0.306075 2.54431 -0.773478 10.9995 -8.42114 -1.65526e-006 3.15998
-11.4707 2.85534 3.52859e-005 -38.8488 -6.51212e-012 -0.0549957 18.9671 5.6765 -0.00230834
-31.8504 -5.93801e-011 -3.59907 1.98522 -0.534339 5.12871e-008
.....

```

Figure 2.4 File BVH format

The start of the header chapter begins with the keyword HIERARCHY. The following line starts with the keyword ROOT followed by the name of the root segment. The hierarchy is defined by curly braces. The offset is specified by the keyword OFFSET followed by the X,Y and Z offset of the segment from its parent. Note that the order of the rotation channels appears a bit odd, it goes Z rotation, followed by the X rotation and finally the Y rotation. The BVH format uses this rotation data order. The world space is defined as a right handed coordinate system with the Y axis as the world up vector. Thus the BVH skeletal segments are obviously aligned along the Y axis (this is same as our skeleton

model).

The motion chapter begins with the keyword MOTION followed by a line indicating the number of frames (Frames: keyword) and frame rate. The rest of the file contains the motion data. Each line contains one frame and the tabulated data contains data for channels defined in header chapter.

2.2.2 Scene Graphs

A scene graph is a data structure that arranges the logical and often spatial representation of a graphical scene that contains a number of linked nodes, usually in such a way that a node can have multiple child nodes, but only one parent, thus making it a directed graph. A scene graph consists of nodes connected in a directed, acyclic fashion. Geometry lies at the leaves of the scene graph while internal nodes support notions such as grouping, transformation, selection, and sequencing as well as special operations such as level-of detail switching, and morphing.

Nodes can be divided into two categories, group nodes, which may have children, and leaf nodes, which may not. There can be numerous types of nodes, for instance transform nodes, object/model nodes, light nodes, camera nodes and emitter nodes for particle systems.

A transform node is a group node which represents a transform relative to its parent node. This arranges the scene in a hierarchical structure, which is useful for numerous reasons, such as moving a complex object by only moving the parent node. An object node, or a model node, is a leaf node that represents a graphical object that can be rendered. It has references to a mesh and a material resource. All leaf nodes, such as those for objects, lights, cameras and emitters receive transforms from a parent transform node.

2.2.3 Rendering

Rendering is the process of generating an image from a model (or models in what collectively could be called a *scene* file), by means of computer programs. A scene

file contains objects in a strictly defined language or data structure; it would contain geometry, viewpoint, texture, lighting, and shading information as a description of the virtual scene. The data contained in the scene file is then passed to a rendering program to be processed and output to a digital image or raster graphics image file.

There are several ways of rendering a scene, such as ray-tracing and radiosity. Such methods allow for advanced lighting effects and global illumination. Global illumination takes the environment into consideration so that effects such as reflections, refractions and light bleeding are possible. However, global illumination is generally considered too slow to be applied to games. This section will hence focus on using hardware accelerated rasterisation, which is the most common method employed by games. Although rasterisation only directly supports local lighting effects, which only considers the actual surface and light sources, modern games include many global effects such as shadows and reflections. This, however, makes the process of rendering a modern game a very complex task.

CHAPTER 3

PROGRAMMING WITH OBJECTIVE-C++ AND OPENGL ES

In this project, used legacy code of SMR library that use C++ as programming language. Rideout.P [13] discuss in his book about using C++ code and Objective-C code to create application on iPhone, while the iPhone-specific glue is written in Objective-C. As show below in the Figure 3.1, Rendering Engine uses portable C++ with the classes where most of the work takes place and all calls to OpenGL ES are made from here. The variation on the right separates the application engine (also known as game logic) from the rendering engine.

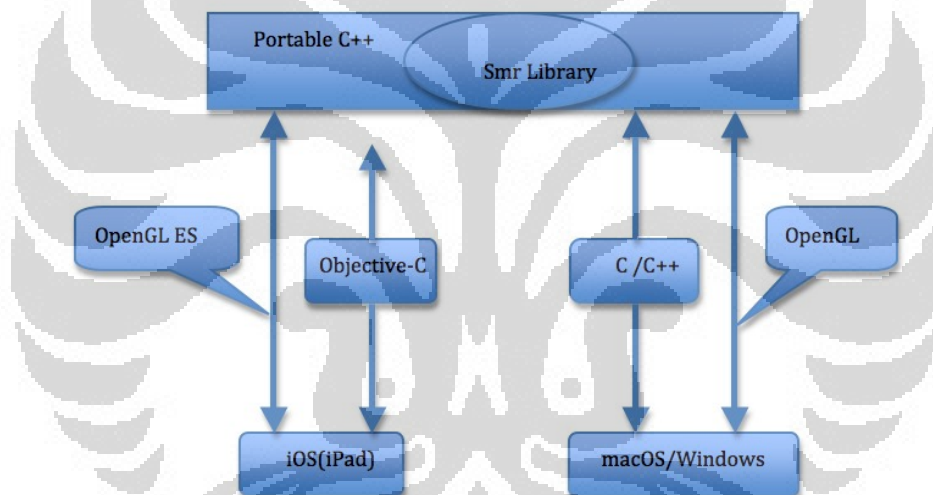


Figure 3.1 Cross Platform Smr Library for OpenGL ES and MacOS/Windows.

Olson [6] in his thesis use OpenGL ES 2.0 with shaders to create application cross-platform for mobile phones, where one of that is for iPhone. Nystrom[19], Olsson[6] used OpenGL ES 2.0 for create 3D animation engine and run application in mobile hardware with many optimizations for 3D desktop application can be directly applied to mobile application .

Apple use the Objective-C language within their development framework. Objective-C is an objective oriented programming language that is superset of C

thus it's allows us to write portions of code in C [4]. Instead of Objective-C is the primary language used on the iPad development, but we can actually use pure C or C++ for much of your application logic, if it does not make extensive use of UIKit[13]. This is especially true for OpenGL development because it is a C API. Files in Objective-C use the .m as file extension and used for the bridge code between the iPhone/iPad operating system and OpenGL ES.

3.1 Using Objective-C with C++

Objective-C can be seen as a thin layer on top of C that adds object orientation to the language. The object syntax is derived from Smalltalk and all of the non-object related syntax is identical to that of C. Its allows to use mixed Objective-C and C and C++. Objective-C compiler allows to mix C++ and objective-C in the same source file that call objective-C++, language hybrid between objective-C and C++. That mean it use the existing C++ libraries in Objective-C application. On Objective-C++, files that contain C++ implementations should use the extension .mm for to be enabled by the compiler instead of .m file extension[17] [8].

In objective-C++, we can call methods from either language in C++ code and in Objective-C methods, for this purpose we need interface between both of them. The interface between the Objective-C class and the other C++ classes called C++ Wrapper object. The method is include pointer to objective-C objects as members of C++ class, and includes pointers to C++ objects as instance variables of objective-C classes[2][12] .

3.2 OpenGL ES Library

There are many different programming libraries for the purpose of rasterizing images in computer graphics. One of the common used library is OpenGL (Open Graphics Library). A graphics library is a low level library that provide an interface against a lower level graphics library[6]. OpenGL is a standard specification defining a cross-platform application programming interface (API) for rendering 2D and 3D object. For targeted at handheld on embedded system, the

Khronos Group released OpenGL ES, simplified version of OpenGL that eliminates redundant functionality to provide a library that is both easier to learn and easier to implement in mobile graphics hardware[2][12], with some functionality was removed in order to make the library smaller and simpler. However, fixed-point functionality was added since few embedded systems efficiently handle floating-point calculations[8].

OpenGL ES 1.0 does not include direct mode for rendering, which means one cannot draw polygons by sending vertices to OpenGL one vertex at a time [6] Instead of must use vertex arrays. It also introduces functions for doing fixed-point math, as Embedded system devices often do not have dedicated floating-point math hardware. Many features that can make development easier but are not strictly needed have also been removed, such as pushing and popping the OpenGL state.

In the 2004, OpenGL ES 1.1 was released based on OpenGL 1.5. The most important additions compared to OpenGL ES 1.0 are automatic mip-mapping, Vertex Buffer Objects (VBO), and the support of all advanced texture (support multi texturing) environment settings in OpenGL 1.5[6]. In March 2007, the Khronos Group released the OpenGL ES 2.0 as a subset of the OpenGL 2.0 specification which entailed a major break in backward compatibility by ripping out many of the fixed function features and replacing them with a shading language thus includes a programmable pipeline[12][5]. OpenGL ES is only one of many graphics technologies supported on the iPad[6].

Unlike OpenGL 2.0 that backward compatible with OpenGL previous version, GLES 2.0 completely removes all support for the fixed function pipeline. As a result, GLES 2.0 is not backward compatible with previous versions of GLES, as a result for handles multi texturing and lighting in favor of a more modern shader-based pipeline. Shaders are specified in a version of GLSL called GLSL ES. GLES 2.0 only supports the Common profile, and the fixed point support has been limited to vertex arrays only.

CHAPTER 4

IMPLEMENTATION

This project was started by studying the concept of OpenGL 2.1 from the SMR Library in order to understanding the principle of OpenGL and shaders. The goals were to examine how to develop iPhone application with OpenGL ES 2.0 and how three-dimensional graphics and shaders can be successfully used in iPad.

In order to do this we decided to porting code an application 3D animation in PC Platform with shader to iOS Platform. Here we will porting 3D engine SMR that used OpenGL to OpenGL ES 2.0 . We investigated if it would be possible to adapt an available PC OpenGL engine but were concerned that they were too complex and large to be easily adapted to a hand-held device. However, we also realized that we did not have enough time to develop a complete 3D engine from scratch as part of the project. We decided to handle this problem by trying to keep the engine as small and simple as possible.

The implementation in this written with Objective-C++ legacy from SMR Library. In the iPhone SDK 4.2 platform which supports OpenGL ES 2.0. Because of the iPhone Platform use some Objective-C code had to be written but the actual OpenGL ES implementation is only C++ making it more portable. This section explains of the implementation project.

4.1 Description of SMR Library

The Library SMR is developed within the research team of SAMSARA from Valoria Laboratory of Université du Bretagne Sud. It propose a feature of animation human virtuals with the skeleton movement. This library also allows to loading Biovision Hierarchy (BVH) file that contain the data of skeleton and his movement that recorded from movement of real people. Generally SMR project has some Library such as newmath, SmrMath Library, SmrAnimation, SmrCore, SmrEngine, SmrRenderer, SmrInterprocess. As Show in the figure 4.1

SmrApplication is used to link engine and offer a fast solution to create a client application. Here we begin, update and end the scene.

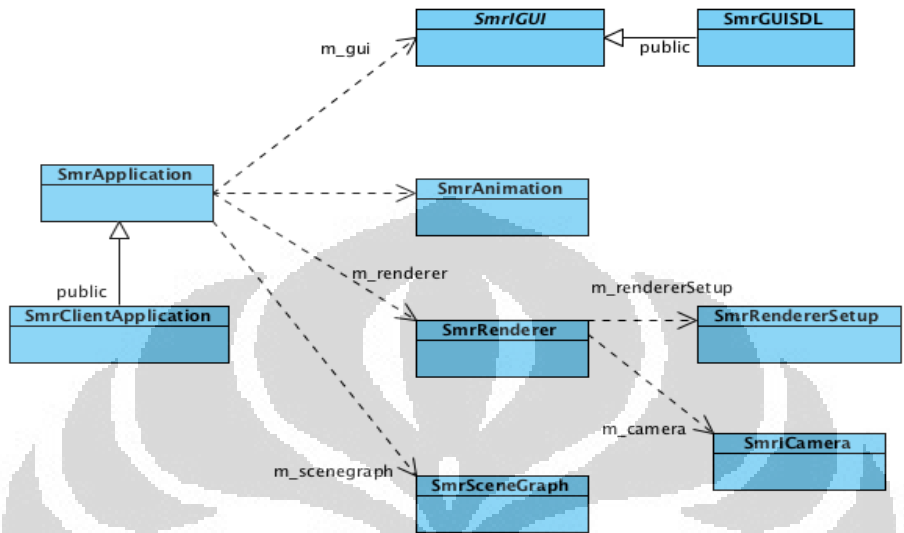


Figure 4.1 Class diagram general of project

There are some libraries dependencies that needed in the Smr Project such as FreeType2, GLEW, Boost, SDL, fbxSDK show in figure 4.2. Along with their purpose on the code, some details on how the port process was done are described next.

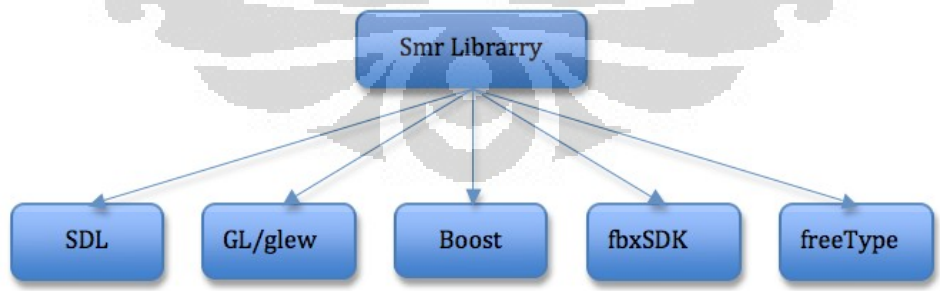


Figure 4.2 Dependency Library SMR

4.1.1 Simple DirectMedia Layer (SDL)

The class SmrGUI is class interface that include Library SDL (SDL.h) defines

creation windows, camera management, keyboard, and joystick. It containing declaration functions used to managed SDLkey and SDL_Event for managed mouse events structure used to retrieve data related to the mouse.

4.1.2 Boost C++ Library

Boost C++ Library usually use to implementation of frequent programming tasks beyond the standard C++ Library such as asynchronous I/O, threading, serialization, date and time, smart pointer, delegate functions, etc. On the other side Boost Library compatible with the STL. Boost used by class TransformationSender to send transformation in the memory.

4.1.3 fbxSDK

fbxSDK is a C++ software development kit (SDK) that allows to import and export scenes in the .fbx file format. It also allows to create, access, or modify various elements such as meshes, parametric surface, skeleton, lights, cameras, and animation data.

4.1.4 OpenGL Extension Wrangler Library (GLEW)

GLEW is a cross-Platform C++ library that helps in querying and loading OpenGL extensions. It provides efficient run-time mechanism for determining which OpenGL extensions are supported on the target platform. All OpenGL extension are expose in a single header file, which is machine-generated from official extension list, on the other hand GLEW available for a variety of operating system.

4.1.5 FreeType

FreeType is a software library written in C that implements a font rasterization engine. it used to render text on to bitmaps and provide support for other font related operations, by using the FreeType library we can create anti-aliased text that look better than text made us bitmap fonts

4.2 SMR Library on iPad

The first task performed to make SMR Library available for the iOS platform was to port all the dependencies of SMR Library that do not have a working version for the iOS platform, and extend the libraries that supported it to supply the needed functionalities, such as a more complete file handling. These dependencies are libraries used by the engine for specific tasks, like texture loading, shaders, and file management. The libraries that officially did not support version for the iOS platform were SDL, FreeType2, glew, Boost, fbxSDK.

4.2.1 SDL iOS

For handle the windows interface Apple provide UIKit Frameworks. This Framework provides the classes needed to construct and managed an application's user interface for iOS. It also provides an application object, event handling, drawing model, view, and controls specifically designed for a touch screen interface. On the other side, we can use library SDL instead of UIKit Frameworks. The application can be ported to other platforms or other operating system more easily without having to replace much of code. SDL 1.3 provide two headers that allows us to chose OpenGL ES 1 or OpenGL ES 2 (SDL_opengles.h and SDL_opengles2.h).

In this project use SDL Library version 1.3 that combined with Cmake as build tool. Cmake can automatically generates Xcode projects, so you can use Xcode for the actual development if you like to do that. In fact, keeping xcode open for compiling to automatically runs the iPhone simulator. Steps to make library SDL for iOS:

- a. Download SDL version 1.3 from site
- b. Navigate to the « X-Code-iPhoneOS/SDL » folder and open the SDLiPhoneOS.Xcodeproj.
- c. During the local developement , changing 3 « iPhone Simulator» we can change SDK by going to Project -> Edit Project Setting and Build and Run.
- d. libSDL.a file can find in the « build/Debug-iphonesimulator » folder with the

- headers, the headers are in a full canonical folder structure (usr/local/include
- e. instead of SDL)
 - f. Moved and rename a bunch of files libSDL.a to libs/SDL/Debug/libSDL.a and usr/local/include to libs/SDL/include.
 - g. For use OpenGL ES 2.0 change SDL_opengles.h to SDL_opengles2.h

4.2.2 Boost iOS

In iOS project Boost library allows us to write platform independent C++ code which relies on high performance. Many of the boost libraries can be used directly without requiring any compilation. In this project use Boost version 1.44, for building the binary boost libraries for iOS can be done from XCode project, some steps for create library for iOS :

- a. The boost sources can be download from the site[19].
- b. Open the XCode project on the boost group and select the root folder of the boost sources.
- c. Open the project settings and change the BOOST_ROOT setting to match the location where you extracted the boost archive.
- d. Select the boost_all target. This is a dummy target that depends on all libraries so you can build everything at once. Select the target SDK and build configuration. Build. Alternatively you can build only the libraries you require.
- e. To use the libraries you can either compile them and copy them to the project or add your code to this project.

4.2.3 FreeType 2[18]

For library Freetype on OpenGL ES use project Freetype [20] , project that primarily designed for iPhone as portable code. To use font, TrueType font (i.e. the .tff file) should include in the resource project. The principle class is FontAtlas, that retrieves the size of each character from the FreeType library ,than use its bin packing algorithm to fit them into OpenGL ES texture [3].

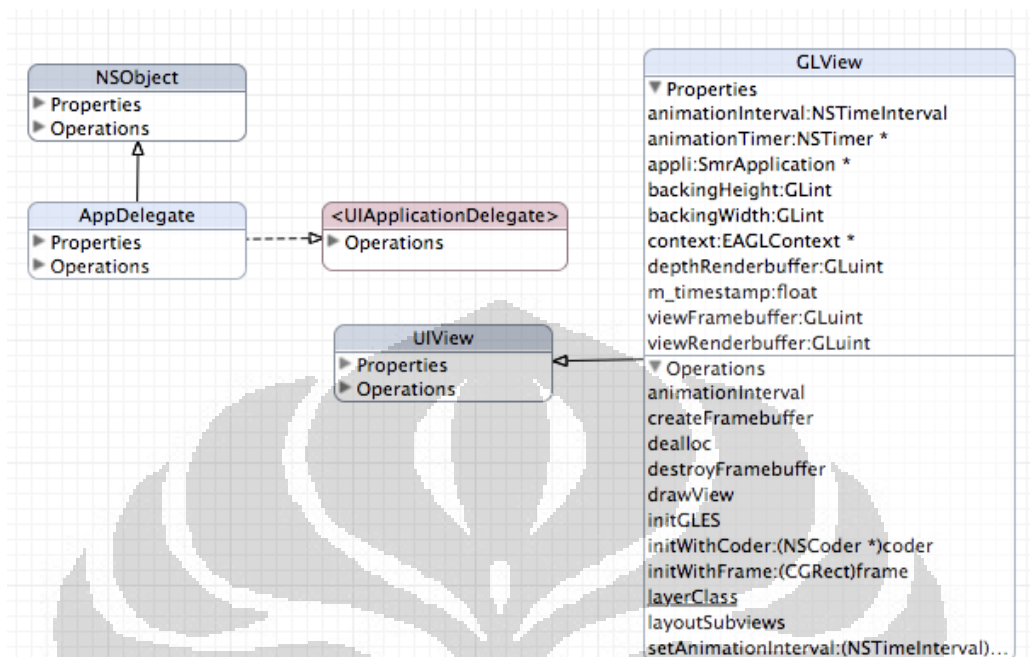
The distribution of the Freetype 2 library that includes an Xcode project can build the Freetype 2 library into a static binary for the iOS operating system. The project

includes the Freetype modules for processing TrueType and OpenType fonts. Steps to make library SDL for iOS:

- a. Download freetype2-iOS from site.
- b. Navigate to the freetype2-iOS Xcodeproj.
- c. During the local development, changing to « iPhone Simulator» we can change SDK by going to Project and then Edit Project Setting and Build and Run.
- d. libfreetype2.a file can find in the « build/Debug-iphonesimulator » folder with the headers.
- e. Add to the project file workspace of an existing project by dragging freetype2.xcodeproj to the appropriate place in Xcode and link target with the libFreetype2.a static library. From there, you will need to tell you target project where the Freetype 2 headers are located.
- f. Add a recursive link to the "include/" path of this Freetype source code distribution, to the "Header Search Paths" entry of Xcode project. The "Header Search Path" entry needs to be relative to the target project's location on your file system.

4.3 Objective-C and C++

In the section 3 has mentioned how to use objective-C++, we can call methods from either language in C++ code and in Objective-C methods. The method is include pointer to objective-C objects as members of C++ class, and includes pointers to C++ objects as instance variables of objective-C classes[2][4]. As show in the figure 4.3 and figure 4.4 the source Objective-c was defined on class GLView a variable appli as a pointer to class SmrApplication on the source C++.



4

Figure 4.3 Class Diagram for Objective-C

```

#include "SmrApplication.h"

@interface GLView : UIView
{
@private
    // The pixel dimensions of the backbuffer
    GLint backingWidth;
    GLint backingHeight;

    EAGLContext *context;
    GLuint viewRenderbuffer, viewFramebuffer;
    GLuint depthRenderbuffer;

    NSTimer *animationTimer;
    NSTimeInterval animationInterval;

    SmrApplication *appli;
}

-(void)drawView;
  
```

Figure 4.4 Code Snippets definition class GLView

```

- (void)drawView
{
    // Make sure that you are drawing to the current context
    [EAGLContext setCurrentContext:context];

    glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);

    appli = new SmrHumanoidDemo();

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    appli->SetUpWindow()->SetSize(768, 1024);
    appli->SetUpRenderer()->SetClearColor(112.0f/255.0f, 117.0f/255.0f, 130.0f/255.0f, 1.0f);

    appli->SetUpRenderer()->m_drawCube =true;
    appli->SetUpRenderer()->m_drawGround =true;
    appli->SetUpRenderer()->m_groundSize=60.0f;

    try {
        // On initialise l'appli: GUI et renderer
        appli->Inif();
        // lancement de la boucle de rendu
        appli->Run();
    }
    catch (const std::exception& E) {
        std::cerr << E.what() << std::endl;
    }

    appli->Release();
    delete appli;

    glBindRenderbufferOES(GL_RENDERBUFFER_OES, viewRenderbuffer);
    [context presentRenderbuffer:GL_RENDERBUFFER_OES];
}

```

Figure 4.5 Code Snippets implementation of class GLView

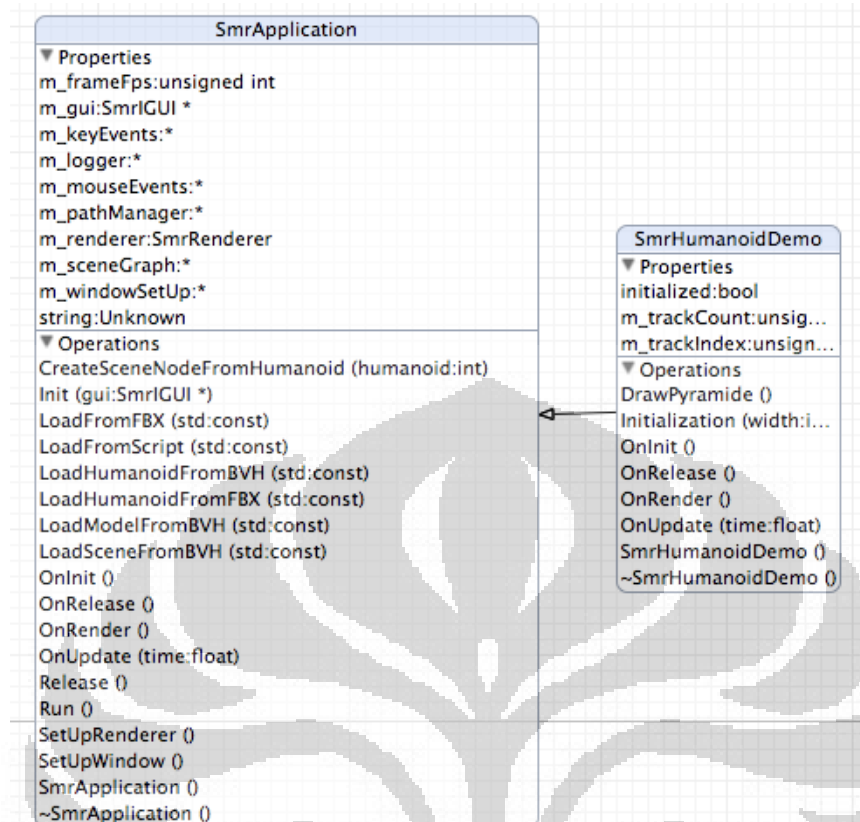


Figure 4.6 Class

Diagram SmrApplication and SmrHumanoid in C++

4.3 OpenGL to OpenGL ES

The initial OpenGL on SMR Renderer project has been modified in turn to OpenGL ES. Many of the OpenGL ES function signatures are equivalent to OpenGL, contributing to decrease the number of changes needed in the original code. Some function names have little differences, like `glClearDepth` (OpenGL) and `glClearDepthf` (OpenGL ES).

The SMR Renderer also used extensions to OpenGL, like GL ARB, which were substituted for the equivalent OpenGL ES functions. OpenGL 2.0 that backward compatible with OpenGL previous version, GLES 2.0 completely removes all support for the fixed function pipeline. As a result, GLES 2.0 is not backward compatible with previous versions of GLES, as a result for handles multitexturing and lighting in favor of a more modern shader-based pipeline. Shaders are specified in a version of GLSL called GLSL ES. GLES 2.0 only supports

theCommon profile, and the fixed point support has been limited to vertex arrays only.

The most noteworthy functionality in OpenGL ES 2.0 removed from OpenGL 2.0 is

- a. Begin/end-paradigm.
- b. Specific vertex arrays for attributes such as positions, normals and texture coordinates. These functions have been removed in favor of the general function `glVertexAttribPointer`.
- c. Quad, quad-strip and polygon drawing primitives.

Figure 4.7 and 4.8 below show the method `DrawGround` for in class `SmrRenderer`, in this method change from OpenGL to OpenGL ES, there saw the different of to call `vertex`, on OpenGL ES call `vertex` from array and change `GL_QUADS` to `GL_TRIANGLES`.

```
void SmrRenderer::DrawGround(float size)
{
    //double _size = 8;
    const double step = size/40; //0.2

    glDisable(GL_LIGHTING);
    glColor3d(0.6,0.6,0.6);
    unsigned int count = 0;
    for (double i = 0 ; i <= 2.0*size ; i=i+step, count++)
    {
        if (!(count%5)) glLineWidth(3.0f);
        else glLineWidth(1.0f);
        glBegin(GL_LINES);
        glVertex3d(-size+i,0.001,-size);
        glVertex3d(-size+i,0.001,size);
        glVertex3d(-size,0.001,-size+i);
        glVertex3d(size,0.001,-size+i);
        glEnd();
    }
}
```

Figure 4.7 Code Snippets from `SmrRenderer` in OpenGL

```

void Renderer::DrawGround(float size)
{
    //double _size = 8;
    const double step = size/40; //0.2
    glDisable(GL_LIGHTING);
    glPushMatrix();
    glColor4f (0.3,0.3,0.3,1.0);
    unsigned int count = 0;
    for (float i = 0 ; i <= 2.0*size ; i=i+step, count++)

    {
        if (!(count%4)) glLineWidth(5.0f);
        else glLineWidth(1.0f);

        GLfloat ground_lines[] = {
            -size+i,0.001,-size,
            -size+i,0.001,size,
            -size,0.001,-size+i,
            size,0.001,-size+i
        };

        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0,ground_lines);
        glDrawArrays(GL_LINES, 0 ,4);
    }
    glPopMatrix();

    // draw some plane
    glPushMatrix();

    glColor4f(.4,0.4,0.4,1.0);
    GLfloat ground_quads[] = {
        -size,-0.01,-size,
        -size,-0.01,size,
        size,-0.01,size,
        size,-0.01,-size
    };
    glVertexPointer(3, GL_FLOAT, 0, ground_quads);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
    glDisableClientState(GL_VERTEX_ARRAY);
    glPopMatrix();
}

```

Figure 4.8 Code Snippets from SmrRenderer in OpenGL ES

- d. All functionality that modifies the current matrix transforms. The model-view and projection matrices are removed, programmers choose which transforms are needed and pass these as uniforms to shaders.
- e. Automatic texture coordinate generation. This can be done in vertex shaders.
- f. All functionality that handles lighting and material state. Programmers implement a custom lighting solution with shaders and feed data to the shaders with generic uniforms and vertex attributes.
- g. One- and three-dimensional textures, leaving two-dimensional textures and cube maps.

- h. Texture environment settings, these are not needed since shaders are a direct substitute and much more powerful.
- i. Fog settings, handled by shaders instead.
- j. Display lists.



CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The Objective of this project is a port of the SMR library, this library used for developing applications that make use of real time 3D graphics for visualization and animation of 3D virtual human character use sign language on French. Real time animation of virtual character on iPad represent a challenging task since many limitation must be taken into account with respect the processing power like graphics capabilities, disk spaces, and memory size.

The original source code has been modified to make SMR library work on the iPad platform, and further optimizations have been done to increase the runtime performance as well. The SMR port for the iOS platform brings to the embedded devices developer the opportunity to take advantage of the high-level features of 3D rendering engines.

5.2 Future Work

As future work, there are plans to implement a SMR Library on real device to monitor hardware acceleration and performance. Another practice for optimizing the runtime speed use different type processors and provides a great performance improvement.

Some bug fixing is still needed, since some features are not working on the current version of the engine. Finally, it is going to be applied some usability tests to evaluate how good is the user interaction with the system and how it can be improved.

The implementation of a render system based on OpenGL ES opens up the possibility of having SMR library ported to other mobile platform environments, like the android mobile base.

BIBLIOGRAPHY

- [1] Freescale Semiconductor Application Note. 2010. *High-End 3D Graphics with OpenGL ES 2.0*, Document Number: AN3994 Rev. 0, 01/2010.
- [2] <http://developer.apple.com/library/iOS/DOCUMENTATION> .
- [3] <http://www.codeproject.com/KB/iPhone/iPhoneFreeType.aspx>
- [4] Hwanyong Lee, Nakhoon Baek and James K. Hahn, *OpenGL ES 1.1 software implementation on mobile phones*, IEICE Electron. Express, Vol. 7, No. 12, pp.880-885, (2010). 2010.
- [5] Khronos Group, OpenGL ES 2.0 Specification. <http://www.khronos.org>.
- [6] Mans Olson , *On the Use of OpenGL ES 2.0 Shaders for Mobile Phones using Cross-Platform Middle-Ware*, Thesis , Sweden 2010.
- [7] Martin Nystrom , *Master of Science Thesis, Inter-Device Multiplayer and Performance Optimization in Games for Modern Mobile Device*, Stockholm , Sweden, 2010.
- [8] Mikael Gustavsson , *3D Game Engine Design for Mobile Phones with OpenGL ES 2.0*, Master Thesis, KTH Computer science and Community, 2008.
- [9] *Multimedia Application Division Freescale Semiconductor*, Inc.Austin, TX 2010. High-End 3D Graphics with OpenGL ES 2.0 Document Number: AN3994 Rev. 0, 01/2010.
- [10] Nakhoon Baek · Hwanyong Lee. 2009. *Implementing OpenGL ES on OpenGL*. The 13th IEEE International Symposium on Consumer Electronics (ISCE2009). 2009.
- [11] Nitin Singhal, In Kyu Park, and Sungdae Cho. 2010. *Implementation and Optimization of Image Processing Algorithms on Handheld GPU*. Proceedings of 2010 IEEE 17th International Conference on Image Processing.
- [12] Objective-C Programming Language, Apple.Inc, 2010.
- [13] Rideout.P, "Iphone 3D Programming" First Edition. O 'Reilly Media,

Inc. 2010.

- [14] Sammi Hawalani, *Arabic Sign Language Translation system on Mobile Devices*, IJCSNS *International Journal of Computer Science and Network Security*, VOL.8 No.1, January 2008.
- [15] Steve HILL, Mathieu ROBART, and Emmanuel TANGUY. *Implementing OpenGL ES 1.1 over OpenGL ES 2.0* STMicroelectronics, Bristol, UK. IEEE. 2008.
- [16] Tae-Young Kim , Jongho Kim , and Hyunmin Hur. 2007. *A Unified Shader Based on the OpenGL ES 2.0 for 3D Mobile Game Development*. Springer-Verlag Berlin Heidelberg.
- [17] Zirkle.P and Hogue.J. *Iphone Game Development* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. 2010.
- [18] <https://github.com/cdave1/freetype2-ios>
- [19] <http://sourceforge.net/projects/boost/files/boost/1.46.1>
- [20] Martinsson Johannes, Trost Reimund. *Implementation of Motion capture support in smartphones*. these 2010. Departement of computer Sciene and Engineering . Chalermers university of Technology.