2004

Exploring the possibilities of three dimensional image manipulations on mobile devices

Thanuja Nuwan Hettiarachchi
Edith Cowan University

Edith Cowan University

Copyright Warning

Exploring the Possibilities of Three Dimensional Image Manipulations on Mobile devices

Dissertation submitted in Partial requirement for

Bachelor of Science (Computer Science) Honours

Edith Cowan University

By: Thanuja Nuwan Hettiarachchi

Student No: [redacted]

Supervisor: Dr. Leisa Armstrong

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

4th January 2005
Dated

Table of Contents

Table of Contents

Table of Contents

List of Figures

List of Tables

DECLARATION

I declare that this thesis does not incorporate without acknowledgment any material previously submitted for a degree in any institution of higher education, and that, to the best of my knowledge and belief, it does not contain any material previously published or written by any other person except where due acknowledgment is made.

Signature

Date: 15 October 2004

ACKNOWLEDGEMENTS

I would like to thank my honours project supervisor Dr. Leisa Armstrong for her commitment of time, for the guidance and for being patient and understanding. I would also like to thank the Edith Cowan University Honours coordinator Mr. Michael Collins for his guidance and support.

Thank you to everyone who shared their opinions, ideas and reviews.

I specially would like to thank my parents for their understanding, guidance and support, and my father whose help and encouragement made this possible.

Abstract

The rapid evolution of computer graphics is largely the result of increase in hardware capabilities. The improvements made in processors and graphic cards and the reduction in their price ensure powerful graphics systems can be built at low costs. With the introduction of more powerful mobile microprocessors and colour screen technology, complex image manipulations on various mobile devices such as mobile phones and handheld devices have become a reality. As a consequence of these improvements, there has been an increasing demand by users for interactive computer games which produce complex graphics by utilizing these advanced hardware technologies.

Three dimensional (3D) graphics have been used to produce realistic interactive imaging for computer games during recent years. Java, through its mobile device programming platform, provides the framework for such complex image manipulations in computer games deployed on Java compatible mobile devices. However, the lack of a standard 3D application-programming interface (API), supported by mobile phone manufactures, has resulted in the need for program developers to use custom APIs to create 3D programs such as the WGE (Wireless Graphics Engine) API produced by TTPcom. There is some evidence that the use of custom APIs to develop 3D graphic images may result in poor compatibility and performance across different mobile platforms and devices

This study initially examines the proposed Sun Microsystems specification for the Java 2 Micro Edition (J2ME) "Mobile 3D API" for the development of 3D graphics programming of mobile devices. These specifications have been designed to create an Industry standard Mobile 3D API. In addition, this study investigates the current specification for the Java 2 Micro Edition (CDDC1.0.3), to ascertain to what extent the development of 3D gaming on mobile devices is effected by the deficiencies in the current specification. These deficiencies include a lack of support of for a floating point data type and the specification's reliance on fixed-point number calculations for developing 3D graphics. An assessment will be made to determine how these deficiencies influence the performance, stability of 3D algorithms deployed on different mobile device platforms.

Investigations carried out on 3D graphics algorithm implementations on Java 2 Standard Edition (J2SE) platform suggests that the implementations rely on float data type and that the CLDC 1.0.3 configuration layer does not support the float data type. Experiments were conducted to determine whether fixed-point number methods can be used effectively to conduct precision calculations. These calculations are required to implement the 3D algorithms for the J2ME platform. In order to assess this, a simulation study was conducted on a number of emulators released by Nokia, Motorola and Siemens mobile phone manufactures. In addition, the algorithms were tested on a Java compatible Nokia 6610 mobile phone to ascertain if findings from emulator studies could be replicated on phones.

The emulator study findings suggest that 3D algorithm implementations using fixed-point methods are compatible on Java compatible mobile handsets released by Nokia, Motorola and Siemens. Further more, it was shown that the fixed-point methods are suitable for implementing simple 3D algorithms (Rotation, Scaling and Translation). However, it was found that these methods were not suitable for extreme precision calculations such as Cartesian curve generations.

1.0 INTRODUCTION

1.1 Background

While the principal use of the mobile phone still remains that of voice communication, mobile phone technology has now evolved into a mobile multipurpose platform, incorporating such facilities as messaging, mobile internet access and mobile entertainment. With the introduction of various technologies such as Java 2 Micro Edition, Symbian, and WAP, the mobile phone has become an integral component of business and entertainment industries (Introduction to Mobile Entertainment Solutions, 2003)

There has been an increasing usage of mobile technologies during the last decade. Nokia Corporation, a major industry leader in mobile technologies has released sales figures that show an enormous and expanding mobile phone industry (107.5 million mobile phones sold in the year 2003 (1Q 2003 phone sales figures, 2003)). Mobile phone technologies are now established as the main criteria for top selling personal entertainment devices. Furthermore, Nokia has stated that in 2002, sales of new Java enabled phones have exceeded one million phones. (Introduction to Mobile Entertainment Solutions, 2003).

With the introduction of more powerful mobile microprocessors and colour screen technology, complex image manipulations on various mobile devices, such as mobile phones and handheld devices have become a reality. As a consequence of these improvements in hardware capabilities, there has been an increasing demand by users for interactive computer games which produce complex graphics by utilizing these advanced hardware technologies. Three dimensional (3D) graphics have been used to produce these realistic interactive imaging for computer games during recent years.

Java through its mobile device programming platform (J2ME), provides the framework for the development of computer games for Java compatible mobile devices. However, the lack of a standard 3D application-programming interface(API), supported by mobile phone manufactures, has resulted in the need for

program develcpers to use custom APIs, such as the WGE (Wireless Graphics Engine) API produced by TTPcom, to create 3D graphics based programs. However, some evidence suggests that the use of these custom APIs to develop 3D graphic images may result in poor compatibility and performance across different mobile platforms and devices (Beardow, 2002). The introduction of Nokia 3410, the first Global System for Mobile communication (GSM) phone by Nokia with 3D graphics software, demonstrates the possibility of 3D animation on mobile devices. Nokia has predicted that 3D visual effects will help boost the mobile entertainment industry in the near future (Visuals Attract People to Games, 2003).

1.2 Significance

Mobile phone technology is now being targeted as a medium for 3D graphics, as it provides facilities for games, interactive menus, customizable interfaces and interactive messaging (Kewney, 2002). As a consequence, the mobile communications industry through corporations such as Nokia and Motorola, is working towards standardizing the implementation of 3D graphics on mobile devices.

A proposed 3D specification, developed through the Java Community Process, is aimed at addressing the deficiencies associated with implementing 3D graphic content on various mobile platforms and devices. However, in order to utilize the 3D Graphics API library, floating point functionality is required. The newly released CLDC version 1.1 supports floating point functions and is expected to be used alongside MIDP version 2.0 to provide better functionality.

To date, the majority of Java compatible mobile devices is based on the CLDC 1.0.3 and MIDP 1.0 J2ME architecture, and therefore, cannot implement 3D Graphics using the standardized 3D Graphics API for the newly released CLDC version 1.1. By introducing an alternative method to develop 3D graphics imaging for the current widely used configuration layer (CLDC 1.0.3) and the current profile (MIDP 1.0), which does not rely on floating point data types, the majority of mobile phone users will be provided with the capability to use 3D graphics content. This will result in an expansion in the potential market for Java based computer games.

1.3 Purpose

The purpose of this study is to provide a mechanism to allow 3D graphics to be implemented on all Java enabled mobile phones, some of which do not provide support for the standardized 3D API developed through the Java community process. In order to establish whether this is achievable, a determination needs to be made as to whether 3D graphic algorithms can be supported on the current Java mobile platform (CLDC 1.0.3 and MIDP 1.0), which is limited by its lack of support for floating point data types. It is proposed that this can be achieved through the alternative use of fixed-point methods to implement 3D graphics. By finding an alternative mechanism to carry out 3D graphics algorithms, there will be an increase in the possible number of Java enabled phones which can then use 3D graphics and therefore an overall expansion in the Java games industry.

In order to establish whether this alternative implementation of 3D graphics can be used, an assessment is required to determine whether differences in stability and compatibility of the graphics algorithms (using fixed-point methods) written for the current configuration layer, occur across the different mobile phone platforms.

Previous studies have made such assessments based on testing, by carrying out simulation studies. It is proposed that an appropriate assessment can be achieved using similar experimental techniques.

1.4 Statement of Research Question

Can the current J2ME configuration (CLDC 1.0.3) and profile (MIDP 1.0) facilitate 3D graphics algorithm implementations?

Components of the above question:

1. What techniques are the most appropriate for the development of 3D graphics animation using the current J2ME configurations?

    a. Can the precision calculations required for 3D graphics animations be achieved with the use of fixed-point methods?

and,

2. Are there differences in the compatibility and stability of 3D animations run on various mobile phone models, using the current J2ME configuration?

1.5 Definition of Terms

API - Application Programming Interface: An interface between the operating system and the application programs.

ARM Chips - A family of RISC-based microprocessors and microcontrollers from ARM Inc. ARM chips are high-speed CPUs which require low power, they are widely used in PDAs and other handheld devices

CLDC - Connected Limited Device Configuration: Provides the programming interface for wireless applications.

Emulator - A hardware device or a program that pretends to be another device or program with which other components can interact.

Fixed-Point - A way of storing and calculating numbers that have fixed decimal points.

FPS - Frames per Second: A number of frames rendered on the screen per second.

GSM - Global System for Mobile communication: A digital cellular phone technology that was developed in the 1980s.

J2ME - Java 2 Platform Micro Edition: A version of Java for small devices such as mobile phones, PDAs and consumer appliances.

JVM - Java Virtual Machine: Software that converts the Java intermediate language into machine language and then executes it.

KVM - K (Kilo Byte) Virtual Machine: A version of the Java virtual machine for small devices with limited memory.

MIDP - **Mobile Information Device Profile**: A profile which provides a graphical interface, networking and storage.

OpenGL - **Open Graphics Language**: A 3D graphics language developed by SGI, OpenGL can be implemented as an extension to an operating system or a window system.

RISC - **Reduced Instruction Set Computer**: A computer architecture that reduces chip complexity by using simpler instructions.

Symbian - An operating system for data-enabled mobile phones from Symbian Ltd.

WAP - **Wireless Application Protocol**: A standard for providing cellular phones, pagers, and other handheld devices with secure access to e-mail and text-based web pages.

Embedded Systems - Any electronic system that uses a CPU chip, but that is not a general purpose workstation, desktop or laptop computer. Such systems generally use microprocessors or they may use custom designed chips or both; they are used for example in mobile phones, PDAs, automobiles.

2G - Second generation of mobile cellular communication: These devices were developed in the 1990s as a second-generation system using digital encoding.

3G - Third generation of mobile cellular communication, expected to be available by 2005 time frame. 3G is designed for high-speed multimedia data and voice.

2.0 REVIEW OF LITERATURE

2.1 General Literature

2.1.1 3D Graphics on Embedded Systems

Although 3D graphics is common on PC platforms, 3D graphics is not effectively used on the mobile platforms. 3D graphics requires higher levels of processing power, which results in increased power consumption and high volume of graphics data being required to transmit over a slow mobile phone link. (Kewney, 2002). However, there are a number of companies, such as Superscape, which are in the process of developing 3D engines for the ARM mobile processors. In addition, more companies, such as Japanese NTT DoCoMo, provide high performance mobile links to transfer high volume data faster. When combined together, these services will enable high quality 3D graphics to be used on mobile devices (Donelan, 2003).

2.1.2 Java programming for Mobile Devices

The Java language was established as a result of Sun Microsystems internal research project then called "Green". The Green project started in 1991 which led to development of a new C and C++ based language named "OAK", by its creator James Gosling, but was later renamed Java (Deitel, Deitel, 1995, p. 12). Java has ventured into other areas beyond desktop machines, since its release in 1995. Two years after the release of Standard Edition (J2SE), the Enterprise Edition (J2EE) was released, targeting server based application development. The latest introduction to the Java family is the Micro Edition (J2ME), targeting micro device applications (Muchow, 2002).

The J2ME platform is divided into two categories that target the high end and low end consumer markets. The high-end consumer devices use the Connected Device Configuration (CDC). Examples of devices in this category include TV set-boxes, Internet TVs and auto mobile navigation systems. These devices feature high range of user interface capabilities, two to four megabytes of memory and high-bandwidth network connections. Low-end consumer devices use the Connected Limited Device Configuration (CLDC). Devices that fall into this category include cell phones,

pagers and personal organizers (PDAs). These devices have simple user interface capabilities compared to the high-end consumer devices, for example a minimum memory size of 128-256 kilobytes, low bandwidth communications and most of these devices are battery operated. Figure 2.1 illustrates the J2ME platform.

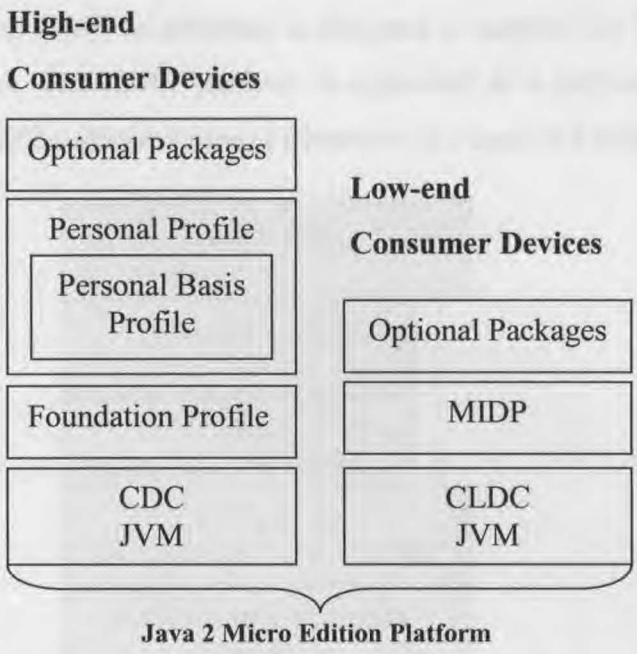


Figure 2.1 Java 2 MicroEdition Platform (Riggs, Taivalsaari, VandenBrink, 2003, p. 8).

Due to the technological advances made in computer, telecommunication, consumer electronics and entertainment industries, the categorization between high-end and low-end consumer devices is not distinct. The line between the two categories is defined in practice by the memory budget, bandwidth, battery power consumption and physical screen size of the device rather than functionality and connectivity features (Riggs, Taivalsaari, VandenBrink, 2003).

2.1.3 Java 2 Micro Edition Architecture

With the introduction of J2ME, developers now have the capability to develop custom applications, without the constraints of memory limitations and operating system constraints. The J2ME architecture is designed to support the flexible needs of different consumers. The J2ME platform is organized as a software stack. The organization of the J2ME software layers is illustrated in Figure 2.2 following.

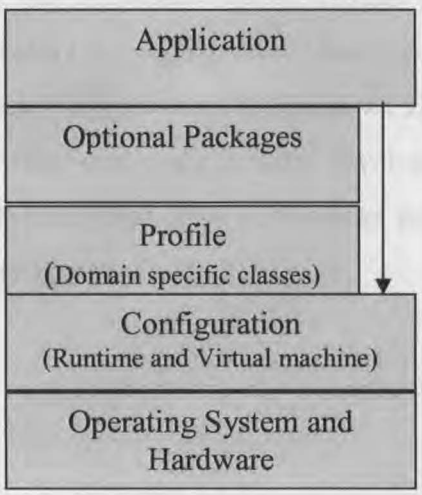


Figure 2.2 Software layers in a J2ME device (Riggs et al., 2003, p. 12).

Since the introduction of the original configuration (CLDC) and profile (MIDP) standards, it was realized that there was a need for general-purpose libraries that were not unique to one category of devices. For example, the location API that provides geographical positioning functionality can be used in both high-end and low-end consumer devices. Optional packages introduce specific functionality and are not bound to any profile. The optional packages are APIs that can be used to extend a profile. New APIs are created as optional packages and new versions are created as needed through the Java Community Process (JCP). These can be subsequently incorporated into the profile (Riggs et al., 2003)

Vendor-specific APIs that are not part of the standard profile or configuration, and are not shown in Figure 2.2. An example is the Nokia UI API, where sound and graphics transparency functions are added as optional vendor specific APIs (Nokia cooperation, 2003). These vendor-specific APIs are also optional extensions to profiles, however unlike J2ME optional APIs, they are developed by manufactures for a specific device or range of device brands (Riggs et al., 2003).

The J2ME development environment was introduced to provide an application development platform for the embedded market (Riggs et al., 2003). As a result, J2ME enables content creators to deploy new ideas to customers worldwide, on many platforms and devices. Furthermore, J2ME allows J2ME-compatible hardware manufactures to provide third-party application development and downloads on different J2ME compatible platforms. This in turn can provide better products and services for the consumer (Riggs et al., 2003).

a) Configurations

The configuration defines the core Java libraries and the specific virtual machine. Each range of devices with similar capabilities is grouped under a configuration. Currently, there are two configurations defined by Sun Microsystems, the Connected Device Configuration (CDC) and the Connected, Limited device Configuration (CLDC) (Riggs et al., 2003). The specifications of each configuration are shown below in Table 1.

Table 1 J2ME Configurations (Riggs et al., 2003, p. 31).

Configuration Name	Description	Target Device
Connected, Limited Device Configuration (CLDC)	Targeted at low end devices with 128KB for running Java, 32 KB for running memory allocation and with limited connectivity. At the heart of this configuration is a VM like the KVM with some J2SE capabilities removed	Devices with 16-bit or 32-bit processors with at least 160KB of non-volatile memory and 32KB of volatile memory and low bandwidth network connectivity, devices such as cell phones, two- way pagers and low-end PDAs.
Connected Device Configuration (CDC)	Targeted at less restricted high end-devices with 512KB memory for running Java. At the heart of this configuration is a VM like the CVM with full J2SE capabilities	Devices with 32-bit processors, at least 2MB of total memory and high bandwidth network connectivity, devices such as high-end PDAs and TV set-top boxes

The CDC includes all the classes included in the CLDC and new classes not included in J2SE (Ortiz, 2002). The relationship between the CDC and the CLDC is illustrated in Figure 2.3.

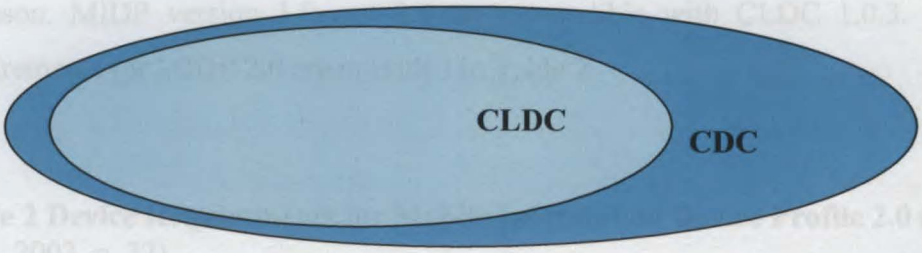


Figure 2.3 Relationship between the CLDC and the CDC (Riggs et al., 2003, p. 16)

b) J2ME Virtual Machines

The J2ME virtual machine can be customized to support a range of consumer devices. This is accomplished by providing a range of virtual machine technologies, which are optimized for different processors and memory capacity. At the centre of the CLDC is the K-Virtual Machine (KVM). The KVM is a compact virtual machine designed for devices with low memory such as mobile phones. KVM is designed specifically for 16/32 bit microprocessors and requires a minimum of 128 Kb of memory. Due to its low memory requirement, the KVM is ideal for small devices. The "K" stands for Kilo, as memory budget is measured in tens of Kilobytes. (Riggs et al., 2001). The CDC uses the Compact Virtual Machine (CVM). The CVM has full J2SE 1.3 Virtual Machine support, however, it is more portable, efficient and compact than the J2SE VM (Ortiz, 2002).

c) Profiles

The profile include classes that provide functionality which is absent from the configuration. It further extends the functionality of the configuration. Profiles can be device oriented or program oriented and it is possible for a single device to accommodate two different profiles, regardless of their type (Muchow, 2002). Profiles for the CLDC include Mobile Information Device Profile (MIDP Version 1.0 and 2.0) and Personal Digital Assistant Profile (PDAP). CDC-based profiles include the Foundation Profile (FP), the Personal Basis Profile (PBP) and the Personal Profile (PP).

i) Mobile Information Device Profile (MIDP)

The MIDP requires the CLDC configuration and is considered a standardized implementation among major device manufactures such as Nokia, Motorola and Ericsson. MIDP version 1.0 and 2.0 are compatible with CLDC 1.0.3. Device requirements for MIDP 2.0 are provided in Table 2.

Table 2 Device Requirements for Mobile Information Device Profile 2.0 (Riggs et al., 2003, p. 32)

Requirement	Description
Display	Screen Size: 96x54 pixels, Display Depth: 1-bit Pixel Shape
Input	One of the Following: One Handed Keyboard, Two handed Keyboard or Touch Screen.
Memory	256KB of non volatile memory for MIDP implementation, 128 of volatile memory for Java runtime
Networking	Two-way, wireless, with limited bandwidth
Power	Limited power (e.g. battery powered)

2.1.4 Standardization of Class Libraries

It is through the process of standardization, that the compatibility issues faced by different product manufactures can be resolved. A number of standardized class libraries are available in various languages such as C++, Java and the .Net framework. For example, in 1992 Alexander Stepanov and Meng Lee developed the Standard Template Library (STL) for C++. In their own words, they describe the STL as " well structured generic C++ components that work together in a seamless way" (Musser, 1996). The STL was developed with the intention of showing the possibility of defining the algorithms as generic as possible without losing efficiency. The STL was designed to enable the C++ programmers to do generic programming with extensive programming (Pohl, 1997). The benefits of such a generic programming idea, enables programmers to write less code faster by introducing reuse of sophisticated data structures and algorithms.

Code reuse is a prominent feature of in the Java programming language. Java, like other established languages, comprises a number of standardized class libraries (APIs). These libraries provide the functionality to a number of program component features, including the graphical user interface, database connectivity and IO components. The process to increase the functionality of Java developed software has been enabled through an independent body, called the Java Community Process.

2.1.5 Java Community Process

Adding new technology through the Java Community Process (JCP), has resulted in the rapid development of the Java language platform. JCP is an open community based organization, which defines and revises Java technology specifications. The JCP was originally created by SUN Microsystems in 1998, with any specific goals determined by Sun Microsystems. JCP aims to oversee any development of new Java software, through an established set of procedures. This will help ensure Java technology's standard of stability and cross-platform compatibility (Java Community Progress background, 2003).

The goals of the Java Community Process include:

1.) To enable the broader Java Community to participate in the proposal, selection, and development of Java APIs by establishing a means for both licensees and non-licensees to participate.

2.) To enable members of the Java Community to propose and carry-out new API development efforts without the need for Sun engineers to be actively involved.

3.) To ensure that the process is faithfully followed by all participants each time it is used by enabling auditing at key milestones.

4.) To ensure that each specification is backed by both a reference implementation and the associated suite of conformance tests.

5.) To help foster a good liaison between the Java Community and other bodies such as consortia, standards bodies, academic research groups, and non-profit organizations.

The JCP's current version (version 2.5) allows members of the community to implement Java Specification Requests (JSR) under open source licenses, resulting in more freedom and lesser costs when implementing specifications (Byous, 2002). Members of the community are either individuals or organizations who are interested in the Java technology. The Process Management Office (PMO) is responsible of looking after the JCP and its running within Sun (Ortiz, 2002).

Through the intervention of the JCP, new specifications are developed within an expert group. A time line view of the JCP procedures when introducing a new JSR is illustrated in Figure 2.4.

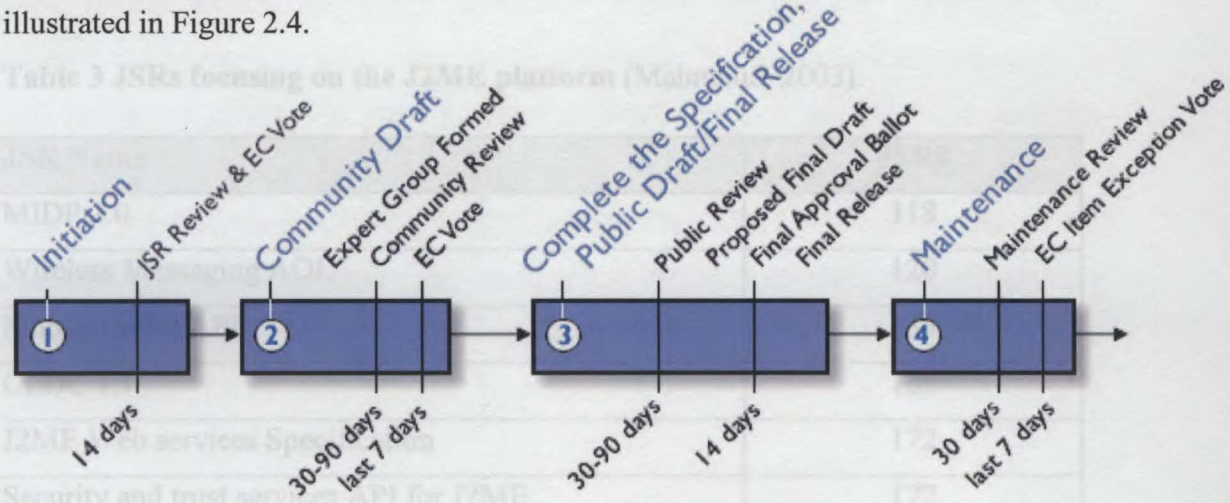


Figure 2.4 Time line View of JCP procedures when Introducing a new JSR (Java Community Progress background, 2003).

A specification starts with a request by one or more members for a new specification or a revision to an existing specification; submitting a Java Specification Request (JSR) to the PMO. There are four main stages in the JCP.

1. Initiation: Specifications are initiated by Community members and approved by the executive committee.
2. Community Draft: A group of experts is formed to develop a first draft. The draft is reviewed by the executive committee whom decides if the draft goes through to development stage.
3. Public Draft: the draft is made available to the general public. The draft is further revised with the help of the public feedback received. After making sure the implementation is complete the specification is sent to the executive committee for final approval.
4. Maintenance: The executive committee decides if requests to revise the specification can be carried out immediately or an expert group is required to carry out the revisions."

(JCP Procedures How the Process works, 2003)

Currently, there are over 200 JSRs were being processed in the JCP (Mahmoud, 2003). Some of the JSR related to J2ME new technologies are listed in Table 3.

Table 3 JSRs focusing on the J2ME platform (Mahmoud, 2003).

JSR Name	JSR#
MIDP 2.0	118
Wireless Messaging AOI	120
Mobile media API	135
CLDC 1.1	139
J2ME Web services Specification	172
Security and trust services API for J2ME	177
SIP API for J2ME	179
Location API for J2ME	180
Mobile 3D Graphics API for J2ME	184
Event tracking API for J2ME	190

2.1.6 Standardization of J2ME 3D Graphics API

The Mobile 3D graphics API for J2ME is one of the JSRs being processed by the Java Community Process, in the lead up to standardizing the mobile 3D industry. The original specification is requested by Nokia Corp. to develop a 3D API for the J2ME platform JSR 184 Mobile 3D Graphics API for J2ME, 2003.

The purpose of the JSR is to "Provide a scalable, small footprint interactive 3D API for mobile devices" (JSR 184 Mobile 3D Graphics API for J2ME, 2003). The expert group for the JSR includes mobile industry giants such as Nokia Corporation, Motorola, Sony Ericsson Mobile Communications AB, Siemens AG, HI Corporation, ARM Limited, France Telecom, Intel Corp and Superscape Ltd (JSR 184 Mobile 3D Graphics API for J2ME, 2003). The specification is currently at the public review stage and is expected to be completed in the fourth quarter of 2003 (JSR 184 Mobile 3D Graphics API for J2ME, 2003).

The proposed Mobile 3D specification is targeted to providing interactive 3D content such as 3D games, interactive navigation menus, custom user interfaces and interactive messages on the current mobile networks. It also aims to address memory and processor constraints set by mobile manufactures to provide 3D graphics rendering with limited ROM footprint, RAM size and processor power (JSR 184 Mobile 3D Graphics API for J2ME, 2003). The API is also designed to be compatible with processors ranging from tens to hundreds of MHz. and will address issues such as colour depth and screen sizes. Furthermore, the API is designed to accommodate future technologies such as 3D acceleration for mobile devices (Mobile 3D Graphics API, 2003).

2.1.7 Other Proposed Alternative 3D Graphics APIs

The OpenGL ES (OpenGL for Extended Devices) is being developed by the Khronos Group with broad industry support. OpenGL ES is a low-level lightweight graphics API designed for embedded systems. The OpenGL ES specification is based on the OpenGL specification, which can be implemented on different platforms. The low level API is designed to act as a bridge between the software and hardware or software graphic engines (OpenGL ES overview, 2003). A number of developer advantages of the OpenGL specification have been identified by the Khronos group, These advantages include the following: it is standard and royalty free; has small footprint & low power consumption; has a seamless transition from software to hardware rendering; is extensible and evolving; is easy to use; and is well-documented (OpenGL ES overview, 2003). An agreement has been reached between the Khronos group and JCP, where by the mobile 3D specification for J2ME requested through the JCP will be incorporated with OpenGL ES. (OpenGL ES overview, 2003).

2.1.8 Current Mobile 3D Implementations

The early interest in mobile 3D graphics has resulted in some companies introducing custom 3D APIs and 3D graphics engines. Some of the prominent implementations include the i-mode network introduced by the NTT DoCoMo Company and the SWERVE 3D graphics engine created by Superscape for the ARM processor.

a) i-mode

The "i-mode" (Information-Mode) service was launched in 1999 by NTT DoCoMo Japan's leading cellular phone operator. NTT DoCoMo provides data services running parallel to the standard voice services. The success of the DoCoMo network is due to this early adoption of data services along with voice services (Donelan, 2003).

The i–mode network subscribers have access to more than 62,000 Internet sites and services like e-mail, on-line banking, on-line shopping and ticket reservations. During the company service time from 1999-2003, the number of subscribers has grown to 36 million, thus proving the popularity of the network. The network provides security for on-line transaction and by using a dedicated leased-line circuit and with the use of firewalls. (What s i-mode?, 2003).

The "i-mode" network provides compact Java based programs through the "i-appli" service launched in 2001. With "i-appli" service Java enabled i-mode mobile devices can be used to download and execute user-preferred programs from more than 533 Internet sites (i-appli: i-mode with Java, 2003). The "i-appli" platform is built on top of the CLDC 1.0.3 configuration. The MIDP (version 1.0 and 2.0) does not contain "i-mode" required APIs and is not compatible with i-appli platform.

According to Seth Reames, the marketing team manager for HI Corporation's Mascot Capsule Division, as quoted by Donelan (2003), "bringing everyone interested in the mobile 3D industry to work together is a major issue faced when trying to standardize the mobile 3D industry" (Donelan, 2003). The HI Corp. provides a 3D engine for mobile devices, which is used in 3D games and 3D virtual pets. HI Corp's services includes plug-ins that work with modeling tools, such as

NewTek's LightWave and 3D Max by Discreet (Nokia Coparation, 2003). This enables designers to create 3D content using a modeling program and export it to a format that the 3D engine understands. The Hi Corp 3D engine is embedded in to more than ten million handsets in Japan and can be downloaded from the i-mode network (Donelan, 2003). Subscribers can log into the NTT DoCoMo i-mode server and securely connect to i-appli content sites on the internet and download compatible Java applications software upgrades and perform on-line transactions. The login details are stored in the i-mode server for billing purposes (i-appli: i-mode with Java, 2003). The i-appli network diagram is shown in Figure 2.5.

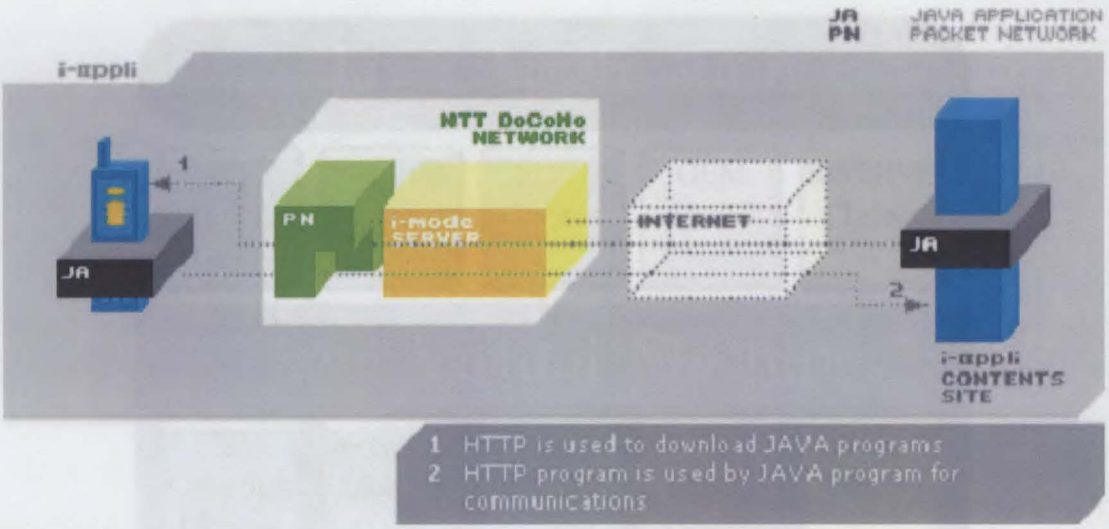


Figure 2.5 i-appli Network Diagram (i-appli: i-mode with Java, 2003)

b) SWERVE

SWERVE is the new 3D engine introduced by Superscape for the mobile platform. Superscape is a company dedicated to the development of innovative, high quality interactive 3D applications. Superscape is currently involved in projects to introduce multimedia on wireless networks, called Interactive 3D, which includes 3D browsing, menu systems, multimedia messaging, 3D multiplayer gaming and video playback. Their aim is to make these services available for current bandwidth constrained networks such as the 2G and 2.5G networks (Beardow, 2002). The 3D engine is targeted for the ARM range of processors. Figure 2.6 illustrates SWERVE's integration into the handset software environment.

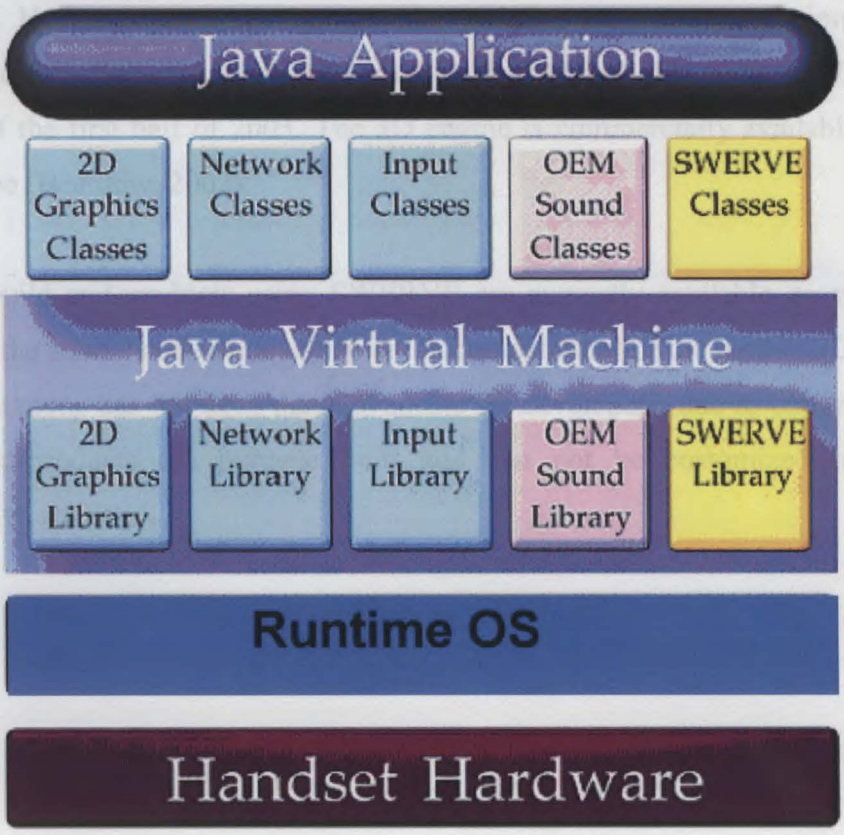


Figure 2.6 Integration of SWERVE into the handset software environment (Beardow, 2002).

As shown in Figure 2.6, the 3D engine (SWERVE) is expected to be integrated into the JVM. As a consequence, it is expected to operate independently of the hardware and the operating system, thus allowing the SWERVE 3D engine to be able to run on multiple platforms. The manufactures of the 3D engine (SWERVE) report that it is also scalable. Since each frame is rendered on the handset, a programmer has the freedom to specify the properties to suit the processor. A key advantage of SWERVE is the ability to load content to a handset as small packets of data rather than a singe large file. This can allow complex 3D scenes to be created dynamically, with data being able to be downloaded in the background to save time and money. In addition, Superscape also reports that the SWERVE 3D engine is capable of producing 15 frames per second (fps) on an ARM 7 processor and 30-40 fps on a faster ARM 9 processor. However, there are no specific statistics provided in relation to cross platform testing. Superscape also suggests that they will be able to ship SWERVE by the end of the first half of 2003. The 3D engine is commercially available through Superscape (Beardow, 2002).

Custom build end-products using SWERVE are currently available commercially, however, the source code or development documentation for the mobile 3D graphics implementations has not been made available by developers, due to copyright laws. These end-products are purpose built and can not be customized to include comparative studies.

2.2 Literature on Previous Findings

The wireless content market is rapidly evolving with new technology being added through industry groups, such as the Java Community Process (JCP) and individual enthusiasts alike. The relevant findings for this study are discussed subsequently.

2.2.1 3D Graphics for Mobile devices

The Japanese mobile industry has been able to successfully implement 3D graphics on mobile devices based on the HI corp. 3D engine. Graphics content is available on the i-mode mobile platform where subscribers can download 3D graphic content through i-modes fast wireless application network i-Appli, subscribers can download 3D content which includes 3D games, Interactive Menus and 3D Screensavers (i-appli: i-mode with Java, 2003).

Furthermore, Henry Minsky has implemented 3D algorithms based on the MathFP class which uses fixed-point calculations for the i-mode and i-appli platforms. The 3D content developed by Minsky is designed for NTT DoCoMo compatible phones, such as the NTT DoCoMo 503i Java-enabled handsets (Minsky 2002).

While a limited number of studies have outlined how 3D algorithms have been implemented on mobile devices, the findings from these studies are not publicly available. The author was unable to gain access to any project documentation or findings on these projects. Only limited information is made available due to copyright laws. The author could not obtain any information on testing procedures or test data collected on these implementations.

2.2.2 Fixed-Point Calculations

Fixed-point calculations are similar to those of floating point, where the fractional part of a float is represented by a whole number in fixed-point. Fixed-point calculations require bit shifts, which are the fastest machine operations. For example converting an integer to a fixed-point number involves shifting the integer the number of precision bits to the left (x<<16).

A study by Giguere (2002) examined the use of fixed-point functions on the CLDC 1.0.3 configuration. The study underlines the use of a J2ME class library (MathFP) developed by Onno Hommes to perform extensive fixed-point calculations, such as trigonometric and logarithmic functions (Hommes, 2000). The MathFP class library is available free of charge for non commercial purposes. Appendix 1 describes the Mathematical functions included in MathFP class library.

There is some evidence of the use of 3D graphics algorithms based on fixed-point. In a research project carried out by Henry Minsky, the use of a class called MathFP to provide software floating point functionality on the i-mode platform was described. This study demonstrated the possibility of performing floating point calculations on the i-mode platform by using NTT DoCoMo class libraries and the MathFP class library (Minsky, 2002).

The findings of the Minsky project are have been made, in part, available on his research website (Minsky, 2002). However, this has not been documented in any detail, with the website claiming that the research has been carried out for personal interest. In addition, the sparse documentation does not contain any testing strategies for the algorithms and collected test data. Furthermore, there is no reference to the start or completion date of Minsky's research.

2.2.3 Emulator Studies

Emulators are designed to accommodate the needs of developers wishing to test their code on a simulation of a mobile phone handset or other portable device. The specific emulator simulates the functionality of the actual mobile phone and it is based on the actual phone software (Nokia Corporation, 2002). There is evidence that research studies have used emulators to assess the functionality of mobile devices. In a study by Lee Butts and Andy Cockburn (2001) on mobile phone text input methods, experimentation methodology included the use of mobile phone emulators to determine the efficiency of the text input methods. The Graphical User Interface of the emulator consists of a phone image, where the keys map to functions in the actual phone software. These are identical to the functions available in the actual phone handset, thus allowing interaction with the mobile emulator possible. The above study has used this function to experiment text input methods to find an optimal solution (Butts, Cockburn, 2001).

Furthermore, emulators are being used to ensure stability of the Java based program content developed for an evolving environment. For example, CYBIRD, a leading Japanese mobile software content developer, has integrated emulator testing as a part of their development cycle to ensure compatibility and stability of the wireless content deployed onto different Japanese mobile platforms such as NTT DoCoMo, J-Phone, KDDI, and DDI POCKET (Nokia Corporaton, 2003). CYBIRD has being developing wireless content for the Japanese consumers since 1998. "As a first-mover in the mobile content space, CYBIRD has benefited," concluded Sadatomo Yoshikawa, senior vice president of CYBIRD. Yoshikawa further states: "The company adopted a strong user orientation early in its business development. Its market position and brand recognition have enabled it to capitalize on relationships with network operators, handset makers like Nokia, technology providers, content providers, and users themselves. Through these relationships CYBIRD continually gathers and leverages a vast collection of information about the wireless market, consumer trends, and technology".

Nokia Corporation has introduced its "build, test and sell" development and marketing strategy for mobile content developers, where the end product can be sold

through global and regional channels. Figure 2.7 outlines the steps involved in the build, test and sell strategy. The Nokia Corporation suggests that its strategy is put in place to assist developers to create quality and compatible software for global Nokia customers (Nokia Coparation, 2003). Through Nokias subscription based forum (Forum Nokia) the latest SDKs, emulators, information and development strategies can be obtained. Nokia encourages developers to test their products on the emulators prior to release to ensure compatibility and stability. A summary of the build, test sell strategy used by Nokia Corporation is outlined bellow,.

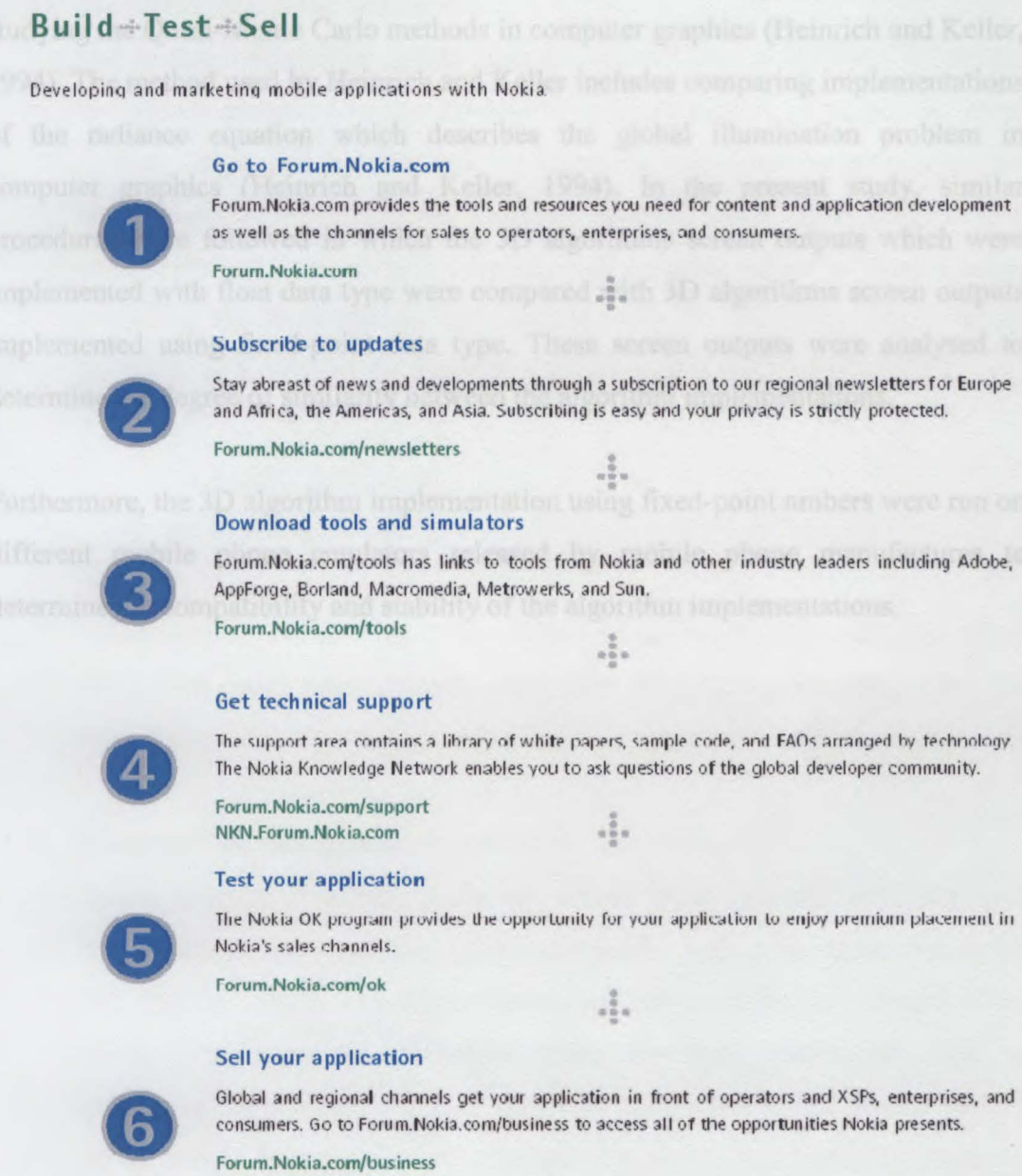


Figure 2.7 Development and Marketing mobile Applications with Nokia (Nokia Corporation, 2003).

3.0 RESEARCH METHODS

This chapter outlines the method of investigation and the factors that need to be considered to determine the implementation of 3D image manipulations on mobile devices.

3.1 Methodology

The study follows a comparison method of investigation. This research method has been utilized by other researchers such as Stefan Heinrich and Alexander Keller studying the Quasi-Monte Carlo methods in computer graphics (Heinrich and Keller, 1994). The method used by Heinrich and Keller includes comparing implementations of the radiance equation which describes the global illumination problem in computer graphics (Heinrich and Keller, 1994). In the present study, similar procedures were followed in which the 3D algorithms screen outputs which were implemented with float data type were compared with 3D algorithms screen outputs implemented using fixed-point data type. These screen outputs were analysed to determine the degree of similarity between the algorithm implementations.

Furthermore, the 3D algorithm implementation using fixed-point nmbers were run on different mobile phone emulators released by mobile phone manufactures to determine the compatibility and stability of the algorithm implementations.

3.2 Identification of Variables relevant to the study

This section discusses the variables that need to be considered when determining whether the implementation of 3D image manipulations on mobile devices is achievable and thus the elucidation of the research question. There are two main variables identified that have an influence over the study including:

a) The screen size of the emulators.

Due to the difference in screen sizes among different mobile phone brands, the outputs generated by the 3D algorithm implementations cannot be based on static height and width values. To ensure the test experiments are dynamic the different screen height and width are obtained using J2ME methods that retrieve the values at runtime, thus allowing dynamic values to be used when generating the 3D graphics. For example using the following J2ME coding the height and width of the screen can be obtained at runtime:

```
int height =getHeight();
int width =getWidth();
```

These methods retrieve the device values of the height and width instead of using hard coded values which could alter the output generated by the 3D algorithms

b) Brand specific functions on mobile phone emulators.

Brand specific functions such as screen light, sound, and key pad configurations are controlled by brand specific APIs such as the Nokia UI API (Nokia, 2002). The above mentioned functions are not controlled by standardized functions, therefore, these functions cannot be used to implement common experiments which are compatible with different mobile phone brands. In order to avoid compatibility issues, brand specific functions are not used when implementing the 3D algorithms.

4.0 EXPERIMENTAL DESIGN

The experimental procedures and design used within this research study are outlined in this chapter.

4.1 Introduction

In order to determine whether 3D image manipulations are achievable on currently available mobile devices, a test bed was established in which a number of sample test programs were written and used as test cases in the experiments. These test programs used a number of established techniques in order to develop 3D graphics manipulations. Three different algorithmic implementations were used including Rotating, Scaling and Translation techniques.

All test programs were written using J2ME for the CLDC 1.03 and MIDP 1.0 platform. As discussed previously (Section 1.2), this development environment does not support the floating point data type support. The test program implementations were created using JBuilder 8 with MobileSet 3.01, where the Midlets are packaged using the JBuilder Archive building tool. The program implementations were run on emulators using the Nokia Developers Suite, Motorola Launchpad and Siemens Mobility Toolkit.

4.2 Experiment 1:

3D Image Manipulations on J2ME CLDC 1.03 and MIDP 1.0 platform compatible phones, using fixed point algorithms.

This experiment was designed to answer the first sub question of the research question:

> What techniques are the most appropriate for the development of 3D graphics animation using the current J2ME configurations?

Furthermore, this experiment will determine whether the precision calculations required for 3D graphics animations, can be achieved with the use of fixed-point methods?

An investigation was carried out to determine, firstly whether 3D image manipulations could be successfully implemented on the designated J2ME CLDC 1.03 and MIDP 1.0 platform compatible phones which do not support floating point data type. This investigation was carried out as a simulation study in which three test bed programs were tested on three different emulators, representing various mobile device manufacturer devices. The test programs were designed using recognized algorithms, which have been implemented, tested and documented for the J2SE platform by Roger Stevens (Stevens, 1997).

The 3D graphics algorithms (Rotation, Scaling and Translation) were implemented by the author for the J2ME platform, using only fixed-point methods. Although these algorithms have been implemented for the J2SE platform, where float data type methods are available, the CLDC 1.0.3 configuration layer does not support float or double data types, therefore, fixed point methods had to be used for implementing these algorithms for the J2ME platform. Fixed-point methods are introduced using MathFP class developed by Onno Hommes (Hommes, 2000).

In order to make an assessment as to whether the findings from the simulation study can be extrapolated to actual mobile devices, testing of the algorithmic implementations were also carried out on a Java enabled mobile phone (Nokia 6610). The Nokia 6610 mobile phone has the same hardware specifications as the Nokia 7210 mobile phone. The specifications are listed in Appendix 2 and Appendix 3.

4.3 Experiment 2:

A determination of the compatibility and stability of 3D animations on various mobile devices using the current J2ME configuration.

This experiment was designed to answer the second subquestion of the research question:

> Are there differences in the compatibility and stability of 3D animations run on various mobile phone models, using the current J2ME configuration?

This experiment was carried out as a simulation study, in which the 3D algorithm implementations for the J2ME platform used in Experiment 1, were tested, on different mobile manufacture platforms for compatibility and stability. These algorithms were tested on a number of official emulators including Nokia 7210, Motorola A830, and Siemens S55 emulators, which have been released by three major mobile phone manufactures (Nokia, Motorola and Siemens). The algorithm implementations were executed five times on each emulator, in order to establish the compatibility and stability of the algorithm implementations on different mobile brands.

4.4 Test Program Design and Implementation

This section outlines the design and implementations of the 3D, Mandelbrot fractal, and curve generating algorithms, using fixed-point numbers methods. These algorithms were be used as the basis for the test bed programs used for Experiment 1 and 2 of this study. The full code implementation of these test bed programs is provided in Appendix 4, 5 and 6.

4.4.1 3D Rotation Algorithm

The matrix calculation for rotating a 3D coordinate (x,y,z) is shown in the Figure 4.1. This matrix represents rotating a 3D coordinate (x,y,z) counter clockwise, about the z axis. The (x',y',z') coordinate represents the new position of the (x,y,z) point. Theta (Ø) is the angle of rotation in radians. The angle is entered in degrees and has to be converted to radians before calculating the sine and cosine values of Ø.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos ø & -\sin ø & 0 & 0 \\ \sin ø & \cos ø & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 4.1 Matrix Calculation for a 3D Rotation Around the z axis (St-Louis, 1998).

The implementation of this algorithm, firstly converts the angle from degrees to radians, then calculates the cosine and sine values of Ø, and finally multiples the two matrices using the MathFP *Mul( )* method. The implementation code of the rotation algorithm using fixed-point numbers is provided in the Figure 4.2.

```
// Converting angle from degrees to radians
theta = MathFP.Mul(MathFP.toFP(theta), MathFP.PI/180);

//Calculating the cosine and sine values of Ø
int ct = MathFP.Cos(theta);
int st = MathFP.Sin(theta);

//Matrix calculation
int Nyx = (int) (MathFP.Mul(yx, ct) + MathFP.Mul(xx, st));
int Nyy = (int) (MathFP.Mul(yy, ct) + MathFP.Mul(xy, st));
int Nyz = (int) (MathFP.Mul(yz, ct) + MathFP.Mul(xz, st));
int Nyo = (int) (MathFP.Mul(yo, ct) + MathFP.Mul(xo, st));
int Nxx = (int) (MathFP.Mul(xx, ct) - MathFP.Mul(yx, st));
int Nxy = (int) (MathFP.Mul(xy, ct) - MathFP.Mul(yy, st));
int Nxz = (int) (MathFP.Mul(xz, ct) - MathFP.Mul(yz, st));
int Nxo = (int) (MathFP.Mul(xo, ct) - MathFP.Mul(yo, st));
```

Figure 4.2 Rotation Algorithm Implementation using Fixed-Point numbers.

4.4.2 3D Scaling Algorithm

Scaling of a 3D object can be performed by multiplying the 3D coordinates (x,y,z) with a 4 x 4 matrix. The matrix calculation is shown in Figure 4.3. The coordinates, (x',y',z'), represent the new position of the (x,y,z) point after scaling by a constant *f* in all dimensions.

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Figure 4.3 Matrix Calculation for Scaling (St-Louis, 1998).

The implementation of this algorithm requires the MathFP *Mul( )* method to multiple the original matrix with a scaling constant, *f*. The implementation code is displayed in Figure 4.4.

```
xx = MathFP.Mul(xx, f);
xy = MathFP.Mul(xy, f);
xz = MathFP.Mul(xz, f);
xo = MathFP.Mul(xo, f);
yx = MathFP.Mul(yx, f);
yy = MathFP.Mul(yy, f);
yz = MathFP.Mul(yz, f);
yo = MathFP.Mul(yo, f);
zx = MathFP.Mul(zx, f);
zy = MathFP.Mul(zy, f);
zz = MathFP.Mul(zz, f);
zo = MathFP.Mul(zo, f);
```

Figure 4.4 Scaling Algorithm Implementation using Fixed-point numbers.

4.4.3 3D Translation Algorithm

A 3D object can be moved to a new position (Translation) by multiplying the 3D coordinates (x,y,z) with a 4 x 4 matrix. The matrix calculation is shown in Figure 4.5. The coordinates, (x',y',z'), represent the new position of the (x,y,z) point after moving the point by a constant *n* points in all dimensions.

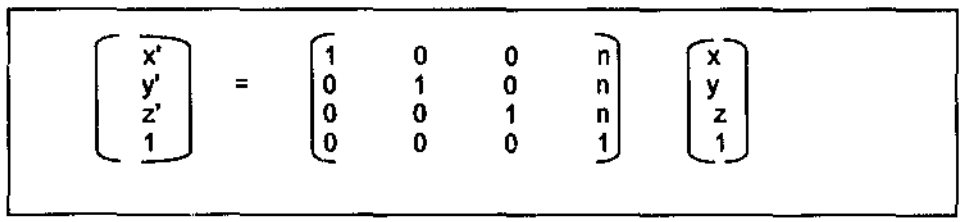


Figure 4.5 Matrix Calculation for Translation (St-Louis, 1998).

The implementation of this algorithm requires the MathFP *Mul ( )* method to multiply two fixed-point numbers. The array *v[ ]* holds the starting point coordinates (x,y,z) and array *tv[ ]* holds the new positions coordinates (x',y',z'), after moving the point by a constant value, *nvert,* in all dimensions. The implementation code of this algorithm is displayed in Figure 4.6.

```
for (int i = nvert * 3; (i -= 3) >= 0; ) {
 int x = v[i];
 int y = v[i + 1];
 int z = v[i + 2];
 tv[i] = (int) (MathFP.Mul(x, lxx) +
       MathFP.Mul(y, lxy) + MathFP.Mul(z, lxz) + lxo);
 tv[i + 1] = (int) (MathFP.Mul(x, lyx) +
       MathFP.Mul(y, lyy) + MathFP.Mul(z, lyz) + lyo);
 tv[i + 2] = (int) (MathFP.Mul(x, lzx) +
       MathFP.Mul(y, lzy) + MathFP.Mul(z, lzz) + lzo);
}
```

Figure 4.6 Translation Algorithm Implimentaion using Fixed-Point numbers.

4.4.4 Mandelbrot Fractal Curve Generator Algorithm

The Mandelbrot set is a map of the complex plane over some specific area. In order to generate this fractal curve, a determination of the colour of the pixel on the screen needs to be made. The equation required to achieve this process is shown in Figure 4.7 (Stevens, 1997, p.134). The equation details two complex numbers *Z* and *C* to determine the colour on the image.

$$Z_{n+1} = Z_n + C$$

Figure 4.7 Iterative Equation to Determine the Colour of the Pixel (Stevens, 1997, p.135).

Firstly the implementation of this algorithm requires the conversion of integers into fixed point integers, and then multiplies the two fixed point numbers. The implementation code comprises of iterative loops, which calculate the pixel colour for each row and column coordinate pair on the image. The implementation code of this algorithm is displayed in Figure 4.8.

```
//Iterative calculation

for (col = 0; col < xres; col++) {

    P = (left + (col >>> 32)) * (deltaP);

    for (row = 0; row < yres; row++) {

     Q = (top - (row >>> 32)) * (deltaQ);
     x = y = ZERO;

     for (index = 0; index < 64; index++) {
      xsq = (x * x);
      ysq = (y * y);

      if (xsq + ysq > 4) {
       break;
      }
      y = 2 * x * y + Q;
      x = xsq - ysq + P; }
      }
    }
```

Figure 4.8 Iterative Mandelbrot Fractal Curve Implimentaion using Fixed-point numbers.

4.4.5 Cartesian Curve Generator Algorithm

The Cartesian curve can be generated by obtaining the y coordinate for each x coordinate through a sine conversion based equation (Stevens, 1997, p.100). This equation is shown in Figure 4.9.

$$y = \sin(x) / x$$

Figure 4.9 Equation to generate a Cartesian curve (Stevens, 1997, p.100).

The implementation of this algorithm requires the calculation y values for for each x value when x is not equal to 0 (to avoid a divide by zero error). The y value is generated using the MathFP *Div( )* and MathFP *Sin( )* methods. The curve is generated using short line segments to create a smooth curve. The implementation code is outlined in Figure 4.10.

```
//Draw the Cartesian x=0 and y=0 curves

g.drawLine(0, getHeight() / 2, getWidth(), getHeight() / 2);
g.drawLine(getWidth() / 2, 0, getWidth() / 2, getHeight());


//Draw the Cartesian y=Sin(x)/x curve

  for (col = 0; col < getWidth(); col++) {

     x = MathFP.Div( (col- (getWidth() / 2)),12);

    if (x!=0){
      y = MathFP.Div(MathFP.Sin(x),x);

    }

    row =getHeight()/2- (MathFP.Mul(y, getHeight() / 2)) ;
    g.drawLine(old_col, old_row, col, row);

    old_col = col;
    old_row = row;
```

Figure 4.10 Cartesian Curve Implementaion using Fixed-point numbers.

4.5 Experimental Design Requirements

A number of design considerations have been determined to be important requirements for this study. These include:

1. Float and double data types can not be used to implement the rotation, transformation, scaling. Mandelbrot fractal curve and cartesian curve generating algorithms for the current J2ME configuration (CLDC 1.0.3 and MIDP 1.0) as these data types are not supported in the CLDC 1.0.3 configuration layer.

2. The algorithm implementations can not use manufacturer specific APIs (e.g. Nokia UI API released by Nokia) as the algorithm implementations used in this study have been tested on different mobile phone manufacturer platforms.

3. The MathFP class is redesigned to include only the methods needed for the study. The methods included in the MathFP class is outlined in Table 4.

Table 4 Methods included in the redesigned MathFP class

Operation	Description
toInt (int n)	Converts a fixed-point number to a normal integer
toFP (int n)	Converts a normal integer to a fixed-point number
Mul (int n, int m)	Multiplies two fixed-point number
Div (int n, int m)	Divides two fixed-point number
Sqrt (int n)	Returns the square root of the fixed-point number
Sin (int r)	Returns the sine value of the radian fixed-point number
Cos (int r)	Returns the cosine value of the radian fixed-point number
Tan (int r)	Returns the tangent value of the radian fixed-point number

4. The algorithm implementations are executed using the official emulators released by Nokia, Motorola and Siemens.

5. All three Emulators emulate colour screen compatible mobile phones. Note: Appendix 2 contains a detailed emulator specification for each emulator used in this study.

5. The algorithm implementations are loaded via a Nokia PC-Phone cable link onto a Nokia 6610 mobile phone and executed.

5.0 Results and Conclusions

The experimental outcomes and the conclusions derived from the observations are examined in this section.

5.1 Results

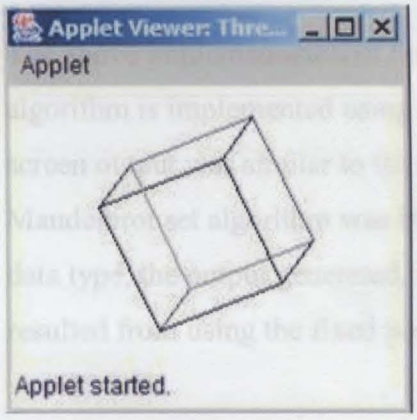
5.1.1 Experiment 1:

3D Image Manipulations on J2ME CLDC 1.03 and MIDP 1.0 platform compatible phones, using fixed point algorithms.

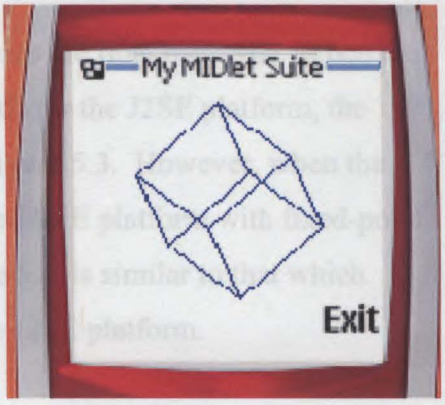
a) 3D Algorithms

The experiment was completed successfully. The test bed programs compiled without any errors and a Java archive was created. This archive was used to execute the test algorithms on the Nokia, Motorola, and Siemnes emulators. This was achieved successfully without any compile and runtime errors. Section 5.2.1.1 shows the screen shots captured with a print screen command, while the program was being executed on the various emulators. The experiment was replicated five times with the same expected outcome achieved for each replication.

In order to validate these results on an actual mobile phone, the Java archive was loaded on to the Nokia 6610 mobile phone, via the DUK-5 cable. The test program was executed without any runtime errors. The emulator output of the rotating 3D cube wireframe was found to be identical to the actual output on the Nokia 6610 screen. The screen outputs of the two test program implementations are shown in Figure 5.1. The J2SE implementation has been executed as an applet and the J2ME implementation has been executed on a Nokia mobile phone emulator. The comparison of screen outputs indicated that the implementations produce the same graphical outputs.



a) 3D wireframe generated using float data type on the J2SE platform



b) 3D wireframe generated using fixed-point data type on the J2ME platform

Figure 5.1 3D Cube Wireframe Screen outputs

b) Mandelbrot Fractal Curve Algorithm

The iterative algorithm to generate a Mandelbrot set compiled without errors and a Java archive was created successfully. This archive was used to execute the program on the three emulators. The emulator screen outputs were captured using a print screen command. The J2SE implementation of the Mandelbrot set produces a image such as the the image shown in Figure 5.2.

Figure 5.2 Mandelbrot set algorithm implementation for the J2SE platform with Float data type

The above implementation of the Mandelbrot set uses the float data type. When the algorithm is implemented using fixed-point methods for the J2SE platform, the screen output was similar to the image shown in Figure 5.3. However, when the Mandelbrot set algorithm was implemented for the J2ME platform with fixed-point data type, the output generated, as shown in Figure 5.4, is similar to that which resulted from using the fixed point data type of the J2SE platform.

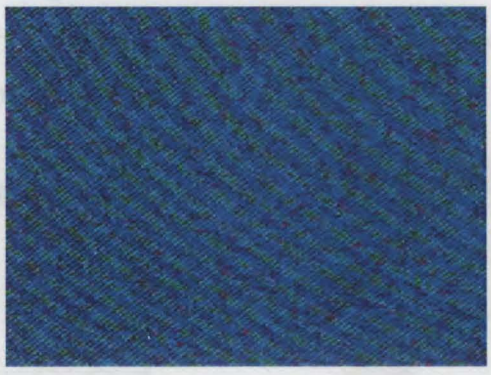


Figure 5.3 Mandelbrot set algorithm implementation for the J2SE platform with Fixed-point data type

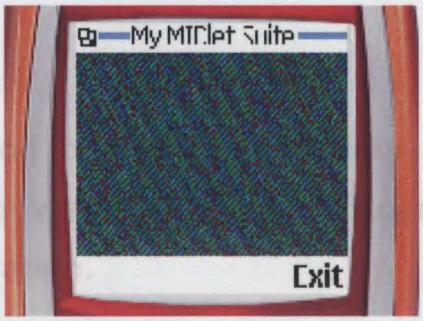


Figure 5.4 Mandelbrot set algorithm implementation for the J2ME platform with Fixed-point data type

As the screen outputs indicate, the implementations produce the same outputs when using fixed-point data type to implement the algorithm on both platforms. However, the screen outputs do not match when using different data type based implementation, For example, using a float data type on the J2SE platform and fixed point data type of the J2ME platform.

c) Cartesian Curve Algorithm

A Cartesian Curve was generated on the J2ME configuration with fixed-point support. Although the author was able to generate a portion of the Cartesian curve the author was unable to generate the full Cartesian curve using the $y = Sin(x)/x$ equation. The full curve implementation and the partial curve obtained by the calculation are shown in Figure 5.5 and 5.6 respectively.

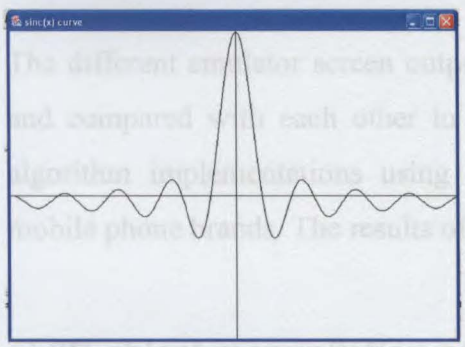


Figure 5.5 y= Sin(x)/x Curve representation on the J2SE platform

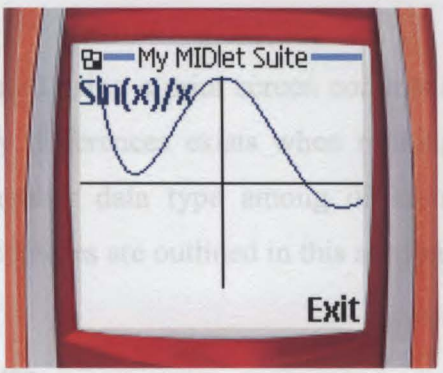


Figure 5.6 y=Sin(x)/x Curve representation on the J2ME platform

5.1.2 Experiment 2:

A determination of the compatibility and stability of 3D animations on various mobile devices using the current J2ME configuration.

The stability and compatibility of the 3D implementations used in Experiment 1 were tested on three emulators released by major mobile phone manufacturors (Nokia, Motorola and Siemens). There was no occurance of run time errors during the execution of these implementations on the emulators. The implementations were loaded on to a Java enabled mobile phone; the Nokia 6610. The Nokia Developer suite was used to transfer the midlets on to the mobile phone via a Nokia PC-Phone link. Similarly, no problems were encountered while loading or executing the midlets on the mobile phone and the outputs matched the emulator outputs.

The execution of all test programs on the emulators was replicated ten times per emulator. For each replication, the screen output was captured and compared with the other screen captures, in order to verify the compatibility and stability of the algorithm implementations. The screen captures of the above implementations run on the three emulators are shown in Section 5.1.2.1.

5.1.2.1 Screen Outputs

The different emulator screen outputs were captured using a print screen command and compared with each other to determine any differences exists when running algorithm implementations using fixed-point number data type among different mobile phone brands. The results of these screen captures are outlined in this section.

a) 3D object manupilation screen outputs

The screen outputs captured while running the 3D graphics algorithms are shown in Figure 5.7. The output generated by the program produces a rotating cube wireframe, which rotates along its axis, until exit is pressed. Although the screen sizes are

different among mobile phone brands, using coding to access the screen size can overcome compatibility issues.

Nokia 7210 Emulator | Motorola A830 Emulator | Siemens S55 Emulator

Figure 5.7 Emulator screen outputs running 3D graphics implimentation

b) Mandelbrot Fractal Curve Generator screen outputs

It was found that the Mandelbrot fractal curve generator did not produce accurate results as discussed previously in Section 5.1.1. However, the algorithm was found to produce similar images on all three different emulators.

Nokia 7210 Emulator | Motorola A830 Emulator | Siemens S55 Emulator

Figure 5.8 Emulator screen outputs running fractal curve implimentation

c) Cartesian Curve Generator Screen Outputs

The generated Cartesian curve was found to be similar on all three emulators. However, there was some difference due to differences in the height and width dimensions of the various emulator screens. As a result, the generated curve was stretched either vertically or horizontally, as shown in the screen outputs displayed in Figure 5.9. In addition, although CLDC 1.0.3 does not support point type, the Cartesian curve generator produces a smooth curve using the *drawline ( )* method.

Nokia 7210 Emulator Motorola A830 Emulator Siemens S55 Emulator

Figure 5.9 Emulator screen outputs running Cartesian curve implimentation

5.2 Conclusions

The observations from Experiment 1 and 2 indicate that 3D graphics implementations are possible on the current widely available J2ME configuration (CLDC 1.0.3 and MIDP 1.0). Although the CLDC 1.0.3 layer does not support float or double data types, these data types can be represented by using fixed-point numbers. The 3D algorithms, requiring floating point accuracy, can be implemented by using fixed-point numbers and fixed-point methods on the CLDC 1.0.3 layer as this configuration layer supports integer and long data types.

Further more, the study suggests the use of fixed-point number data types to perform iterative calculations, such as fractal curve generating using Mandelbrot set algorithm and Cartesian curve generating using an iterative algorithm, does not produce full implementations of the fractal curves. Although the iterative algorithms did not produce accurate results, the algorithms did produce identical results on all three test emulators and the J2ME compatible Nokia 6610 mobile phone. The Simulation study demonstrates that the 3D graphics implementations and the iterative Fractal curve and Cartesian curve implementations are compatible and can be executed on different manufacturer platforms.

Furthermore, the compatibility of these implementations validates the stability of the redesigned MathFP class and fixed-point functions used on the CLDC 1.0.3 layer. The MathFP class provides fixed-point number data type as a substitute for float data type as well as math functions which are not available on the CLDC 1.0.3 layer.

5.3 Strength, Weaknesses and Limitations

5.3.1 Strengths

The author was able to successfully implement basic 3D graphics algorithms (Rotation, Scaling and Translation), iterative Fractal Curve and Cartesian curve implementations for the widely used CLDC 1.0.3 configuration and MIDP 1.0 profile layers.

Furthermore, the study successfully executed the implementations on three emulators released by major mobile phone manufacturers (Nokia, Motorola, Siemens) and the implementations were successfully loaded on to a Java enabled mobile phone (Nokia 6610) and executed without errors.

5.3.2 Weaknesses

The iterative algorithms, when implemented using fixed-point data type did not produce accurate results. For example, the Cartesian curve generated does not mach the correct representation of the equation ($y = sin(x)/x$).

Although the intention of the author to generate a smooth curve was achieved, only a partial curve was achieved and further studies are required to achieve the the full curve representation on the CLDC 1.0.3 layer.

The author did not collect any test data to validate the performance of the algorithm implementations using fixed-point numbers on different mobile phone manufacture emulators. Because the emulators do not emulate the actual performance of the mobile phone. Therefore in order to measure the differences in performance actual Java compatible phones are required.

The differences among various mobile phone brands, such as processor power and on board memory, may have affected the performance of the implementations. A further measurement is needed to be carried out to validate the performance issues of the algorithms.

5.3.3 Limitations

The study did not implement complex 3D algorithms, such as lighting and texture mapping due to time constraint. In addition, the experiments were carried out only on three emulators, provided by three mobile phone manufactures, Nokia, Motorola and Siemens. Furthermore, the experiment tested the algorithms only on one mobile phone, a Nokia 6610 model.

5.4 Recommendations for future studies

Although the industry has taken steps towards standardizing the mobile 3D industry, these steps are yet to be finalized. The study introduces an alternative way of performing 3D graphics on the current CLDC 1.0.3 configuration layer. Further investigations could be made to assess whether other complex algorithms such as lighting and texture mapping can be implemented on CLDC 1.0 compatible mobile phones.

In addition, the study needs to be extended to investigate the performance issues related to 3D algorithm implementations using fixed-point numbers.

Furthermore the study needs to be extended to investigate the differences between float data type iterative calculations and fixed-point iterative calculations, in order to determine if there are any differences between the results produced when performing iterative calculations using float data type and fixed-point data type.

6.0 REFERENCES

1Q 2003 Phone sales figures. (2003). [On line]. Available WWW:http://www.mobileoffice.co.za/news_2003/1-2003_phonesales_figures.htm

3D Graphics on Mobile Phones. (2003). [On line]. Available WWW:http://www.nokia.com/nokia/0,5184,5400,00.html

Beardow. P, (2002). *Swerve white paper- Enabling wireless interactive 3D.* [On line]. Available WWW: http://www.superscape.com/resources/index.asp

Byous. J, (2002). *A Look at the New JCP — the Evolution Continues.* [On line]. Available WWW: http://Java.sun.com/features/2002/10/new_jcp.html

Butts. L, Cockburn. A, (2001). An Evaluation of Mobile Phone Text Input Methods. *Conferences in Research and Practice in Information Technology* (Vol. 7)

Deitel. H, M, Deitel. P, J, (1999). *Java how to program.* (3rd Ed.) New Jersey: Prentice-Hall, Inc.

Donelan. J, (2003). *Graphics to go.* [On line]. Available WWW:http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=Articles&Subsection=Display&ARTICLE_ID=171451

Ericsson. (2003). *Sony Ericsson J2ME SDK.* [On line]. Available WWW:http://www.ericsson.com/mobilityworld/sub/open/technologies/Java/tools/j2me_sdk

Giguere. E, (2002). *Using Fixed Point Arithmetic in the Connected Limited Device Configuration.* [On line]. Available WWW: http://wireless.Java.sun.com /midp/ttips/fixpoint/

Heinrich. S, Keller. A (1994). *Quasi-Monte Carlo Methods in Computer Graphics, Part II: The Radiance Equation.* [On line]. Available WWW: http://graphics.uni-ulm.de/REQ.pdf

Hommes. O, (2000). *MathFP-Fixed point integer math.* [On line]. Available WWW: http://home.rochester.rr.com/ohommes/MathFP/

i-appli: i-mode with Java. (2003). [On line]. Available WWW: http://64.56.185.29/i-mode/evolution/iappli.html

Introduction to Mobile Entertainment Solutions. (2003). [On line]. Available WWW: http://www.forum.nokia.com/main/1,35452,1_75,00.html

Java Community Progress background. (2003). [On line]. Available WWW: http://www.jcp.org/aboutJava/communityprocess/background.html

JCP Procedures How the Process works. (2003). [On line]. Available WWW: http://www.jcp.org/en/procedures/overview

JSR 184 Mobile 3D Graphics API for J2ME™. (2003). [On line]. Available WWW:http://www.jcp.org/en/jsr/detail?id=184

JSR-000184 Mobile 3D Graphics API for J2ME ™ Specification 0.12 Public Review Draft. (2003). [On line]. Available WWW: http://jcp.org/aboutJava /communityprocess/review/jsr184/index.html

Kewney. G, (2002). *Mobile devices to get 3D graphics over wireless links.* [On line]. Available WWW: http://www.news-wireless.netarticles/020415-armtrio.html

Mahmoud. Q.H, (2003). *Future Java technology for the wireless service industry.* [On line]. Available WWW:http://wireless.Java.sun.com/midp/articles /j2mefuture

Minsky. H. *iMode Mobile Phone and Wireless Dev Notes.* [On line]. Available WWW: http://www.ai.mit.edu/people/hqm/imode/

Motorola. *J2ME™ Technology.* [On line]. Available WWW: https://idenonline.motorola.com/ideveloper/system/j2me.cfm

Muchow. J. W, (2002). *Core J2ME technology and MIDP.* USA: Prentice Hall

Musser, D.R. (1996). *C++ programming with the standard template library.* USA: Addison Wesley.

Nokia Corporation (2003). [On line]. Available WWW: http://www.forum.nokia.com/main.html

OpenGL ES overview. (2003). [On line]. Available WWW: http://www.khronos.org/embeddedapi/index.html

Ortiz. C.E, (2002). *A survey of J2ME today.* [On line]. Available WWW: http://wireless.Java.sun.com/getstart/articles/survey/

Pohl, I. (1997). *Object-oriented programming using C++.* (2nd ed.). USA: Addison Wesley.

Riggs. R, Taivalsaari. A, VandenBrink. M. (2001). *Programming wireless devices with the Java 2 platform, miro edition.* USA: Addison-Wesley

Riggs. R, Taivalsaari. A, VandenBrink. M. (2003). *Programming wireless devices with the Java 2 platform, miro edition.* (2nd ed.). USA: Addison-Wesley

Siemens. (2002). *Series 60 Platform.* [On line]. Available WWW:http://www2.siemens.fi/developers.jsp

Stevens. R, T. (1997). *Graphics programming with Java.* USA: Charles River Media, Inc

St-Louis. J. (1998). *Mathematics of 3D Graphics.* [On line]. Available WWW:http://pages.infinit.net/jstlouis/3dbhole/mathematics_of_3d_graphics.html#Intro

Superscape. *Resource Library.* [On line]. Available WWW:http://www.superscape.com/resources/index.asp

Visuals attract people to games. (2003). [On line]. Available WWW: http://www.nokia.com/nokia/0,5184,5400,00.html

What s i-mode? (2003). [On line]. Available WWW: http://64.56.185.29/i-mode/introducing/index.html

7.0 APPENDICES

Appendix 1: Mathematical Functions Included In MathFP version 2.0.6

Operation	Description	Operation	Description
abs(n)	absolute number of n	acos(n)	arc cosine of n
add(n,m) or n + m	add n and m	asin(n)	arc sine of n
sub(n,m) or n – m	subtract m from n	sin(n)	sine of n
mul(n,m)	multiply n with m	cos(n)	cosine of n
div(n,m)	divide n by m	tan(n)	tangent f n
sqrt(n)	extract root of n	cot(n)	cotangent of n
max(n,m)	biggest number of n,m	rount(n,d)	round n to d digits
min(n,m)	smallest number of n,m	log(n)	logarithm of n
exp(n)	e raised to n	pow(b,e)	b raised to e
atan(n)	arctangent of n	atan2(y,x)	principal atan of y/x

(Hommes, 2000)

Appendix 2: Emulator Specifications

Nokia 7210 Emulator Specifications	Motorola A830 Emulator Specifications	Siemens S55 Emulator Specifications
Screen size: 128x128	**Screen size:** 135x220	**Screen size:** 101x80 pixels,
Screen area reserved for content: 128x96	**Colour Depth:** 12	**Colours:** 256
Colors: 4096	**Maximum Jar Size:** 128KB	**Memory:** 1MB
Available Memory: Heap memory size 200 KB	**Java support:** CLDC 1.0.3 MIDP 1.0	**Java support:** CLDC 1.0.3 MIDP 1.0
Java support: CLDC 1.0.3 MIDP 1.0 Nokia UI API		
Memory for Midlets: 600KB		

Appendix 3: Nokia 6610 Specifications

Screen Size:	128x 128
Colours:	4096
Heap Memory:	200KB
Java support:	CLDC 1.0.3, MIDP 1.0, Nokia UI API
Memory for Midlets:	600KB

(Nokia, 2003)

Appendix 4: 3D Algorithm Implementation Code

CubeDisplayable.java

```
/*
 * @(#)cubeDisplayable.java 1.0 03/08/10
 * @(author)Nuwan Hettiarachchi
 */

/** cubeDisplayable extends from the J2ME Canvas class */

package cube;

import javax.microedition.lcdui.*;
import net.jscience.math.*;

public class cubeDisplayable
   extends Canvas
   implements CommandListener {
  public int count = 0;
  public int state = TEST1;
  public static final int TEST1 = 1;
  Model3D md;
  CubeMIDlet mid;
  Matrix3D amat, tmat;
  int xfac;
  int POINT_SEVEN = 45875; //0.7
  int scalefudge = POINT_SEVEN;
  int q = 0;
  int weidth = getWidth();
  int height = getHeight();
  public cubeDisplayable(CubeMIDlet mid) {
    Model3D m = new Model3D();
    md = m;
    m.findBB();

    int xw = md.xmax - md.xmin;
    int yw = md.ymax - md.ymin;
    int zw = md.zmax - md.zmin;
    if (yw > xw) {
      xw = yw;
    }
    if (zw > xw) {
      xw = zw;
    }
    int f1 = MathFP.Div(MathFP.toFP(weidth), xw);
    int f2 = MathFP.Div(MathFP.toFP(height), xw);
    xfac = MathFP.Mul(POINT_SEVEN, MathFP.Mul( (f1 < f2 ? f1 : f2),
scalefudge));
```

```
  amat = new Matrix3D();
  tmat = new Matrix3D();

  try {
   jbInit();
  }
  catch (Exception e) {
   e.printStackTrace();
  }
}

/**Component initialization*/
private void jbInit() throws Exception {
  setCommandListener(this);
  addCommand(new Command("Exit", Command.EXIT, 1));
}

/**Handle command events*/
public void commandAction(Command command, Displayable displayable) {

  if (command.getCommandType() == Command.EXIT) {

    CubeMIDlet.quitApp();
  }
}

int alpha = 1; //<<16
int beta = 1; //<<16;

public void spin() {
  tmat.unit();
  tmat.xrot(alpha % 360); //Rotate the object on the x axis
  tmat.zrot(beta % 360); // Rotate the object on the y axis
  amat.Mult(tmat);
}

public void paint(Graphics g) {

  g.setColor(255,255,255);
  g.fillRect(0, 0, weidth, height);
  int cx = weidth / 2;
  int cy = height / 2;
  spin();

  md.mat.translate( - (md.xmin + md.xmax) / 2,
              - (md.ymin + md.ymax) / 2,
              - (md.zmin + md.zmax) / 2);
  md.mat.Mult(amat);
  md.mat.scale(xfac, -xfac, 16 * (MathFP.Div(xfac, MathFP.toFP(weidth))));
```

```
    md.mat.translate(MathFP.toFP(weidth / 2), MathFP.toFP(height / 2),
                8 << 16);
    md.transformed = false;
    md.paintWireFrame(g);

    md.mat.unit();

  }

}
```

CubeMIDlet.java

```
/*
 * @(#)cubeDisplayable.java 1.0 03/08/10
 * @(author)Nuwan Hettiarachchi
 */

/** cubeMIDlet extends from the J2ME MIDlet class */

package cube;

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

public class CubeMIDlet
   extends MIDlet
   implements Runnable {
  private static CubeMIDlet instance;
  CubeMIDlet m;
  private cubeDisplayable displayable = new cubeDisplayable(m);

  /** Constructor */
  public CubeMIDlet() {
   instance = this;
  }

  /** Main method */
  cubeDisplayable c;
  public void startApp() {
   c = new cubeDisplayable(this);

   Display.getDisplay(this).setCurrent(displayable);
   Thread runner = new Thread(this);
   runner.start();

  }

  public void run() {
   c.count = 0;
   c.state = c.TEST1;
   for (; ; ) {
    try {
     Thread.sleep(0);
    }
    catch (Exception e) {}

    displayable.repaint();

   }
```

```
  }

  /** Handle pausing the MIDlet */
  public void pauseApp() {
  }

  /** Handle destroying the MIDlet */
  public void destroyApp(boolean unconditional) {
  }

  /** Quit the MIDlet */
  public static void quitApp() {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }
}
```

Matrx3D.java

```
/*
 * @(#)Matrix3D.java 1.8 03/09/16
 */

/**A fairly conventional 3D matrix object that can transform sets of
 * 3D points and perform a variety of manipulations on the transform*/

/** The Matrix3D class has been changed by replacing the float data type with
 *  integer data type and by using MathFP class to include fixed-point methods
 */

package cube;

import net.jscience.math.*;

/**
 * <p>Title: 3D Graphics using Fixed-Point </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: ECU</p>
 * @author not attributable
 * @version 7.0
 */

public class Matrix3D {
  public static final int ONE = 1 << 16;
  int xx, xy, xz, xo;
  int yx, yy, yz, yo;
  int zx, zy, zz, zo;

  /** Create a new unit matrix */
  public Matrix3D() {
    xx = ONE;
    yy = ONE;
    zz = ONE;
  }

  /** Scale by f in all dimensions */
  public void scale(int f) {
    xx = MathFP.Mul(xx, f);
    xy = MathFP.Mul(xy, f);
    xz = MathFP.Mul(xz, f);
    xo = MathFP.Mul(xo, f);
    yx = MathFP.Mul(yx, f);
    yy = MathFP.Mul(yy, f);
    yz = MathFP.Mul(yz, f);
    yo = MathFP.Mul(yo, f);
```

```
  zx = MathFP.Mul(zx, f);
  zy = MathFP.Mul(zy, f);
  zz = MathFP.Mul(zz, f);
  zo = MathFP.Mul(zo, f);
 }

 /** Scale along each axis independently */
 public void scale(int xf, int yf, int zf) {
  xx = MathFP.Mul(xx, xf);
  xy = MathFP.Mul(xy, xf);
  xz = MathFP.Mul(xz, xf);
  xo = MathFP.Mul(xo, xf);
  yx = MathFP.Mul(yx, yf);
  yy = MathFP.Mul(yy, yf);
  yz = MathFP.Mul(yz, yf);
  yo = MathFP.Mul(yo, yf);
  zx = MathFP.Mul(zx, zf);
  zy = MathFP.Mul(zy, zf);
  zz = MathFP.Mul(zz, zf);
  zo = MathFP.Mul(zo, zf);
 }

 /** Translate the origin */
 public void translate(int x, int y, int z) {
  xo += x;
  yo += y;
  zo += z;
 }

 /** rotate theta degrees about the y axis */
 public void yrot(int theta) {
  theta = MathFP.Mul(MathFP.toFP(theta), MathFP.PI/180);
  int ct = MathFP.Cos(theta);
  int st = MathFP.Sin(theta);

  int Nxx = (MathFP.Mul(xx, ct) + MathFP.Mul(zx, st));
  int Nxy = (MathFP.Mul(xy, ct) + MathFP.Mul(zy, st));
  int Nxz = (MathFP.Mul(xz, ct) + MathFP.Mul(zz, st));
  int Nxo = (MathFP.Mul(xo, ct) + MathFP.Mul(zo, st));

  int Nzx = (MathFP.Mul(zx, ct) - MathFP.Mul(xx, st));
  int Nzy = (MathFP.Mul(zy, ct) - MathFP.Mul(xy, st));
  int Nzz = (MathFP.Mul(zz, ct) - MathFP.Mul(xz, st));
  int Nzo = (MathFP.Mul(zo, ct) - MathFP.Mul(xo, st));

  xo = Nxo;
  xx = Nxx;
  xy = Nxy;
  xz = Nxz;
  zo = Nzo;
```

```
  zx = Nzx;
  zy = Nzy;
  zz = Nzz;
}

/** rotate theta degrees about the x axis */
public void xrot(int theta) {
  theta = MathFP.Mul(MathFP.toFP(theta), MathFP.PI/180);
  int ct = MathFP.Cos(theta);
  int st = MathFP.Sin(theta);

  int Nyx = (int) (MathFP.Mul(yx, ct) + MathFP.Mul(zx, st));
  int Nyy = (int) (MathFP.Mul(yy, ct) + MathFP.Mul(zy, st));
  int Nyz = (int) (MathFP.Mul(yz, ct) + MathFP.Mul(zz, st));
  int Nyo = (int) (MathFP.Mul(yo, ct) + MathFP.Mul(zo, st));

  int Nzx = (int) (MathFP.Mul(zx, ct) - MathFP.Mul(yx, st));
  int Nzy = (int) (MathFP.Mul(zy, ct) - MathFP.Mul(yy, st));
  int Nzz = (int) (MathFP.Mul(zz, ct) - MathFP.Mul(yz, st));
  int Nzo = (int) (MathFP.Mul(zo, ct) - MathFP.Mul(yo, st));

  yo = Nyo;
  yx = Nyx;
  yy = Nyy;
  yz = Nyz;
  zo = Nzo;
  zx = Nzx;
  zy = Nzy;
  zz = Nzz;
}

/** rotate theta degrees about the z axis */
public void zrot(int theta) {
  theta = MathFP.Mul(MathFP.toFP(theta), MathFP.PI/180);
  int ct = MathFP.Cos(theta);
  int st = MathFP.Sin(theta);

  int Nyx = (int) (MathFP.Mul(yx, ct) + MathFP.Mul(xx, st));
  int Nyy = (int) (MathFP.Mul(yy, ct) + MathFP.Mul(xy, st));
  int Nyz = (int) (MathFP.Mul(yz, ct) + MathFP.Mul(xz, st));
  int Nyo = (int) (MathFP.Mul(yo, ct) + MathFP.Mul(xo, st));

  int Nxx = (int) (MathFP.Mul(xx, ct) - MathFP.Mul(yx, st));
  int Nxy = (int) (MathFP.Mul(xy, ct) - MathFP.Mul(yy, st));
  int Nxz = (int) (MathFP.Mul(xz, ct) - MathFP.Mul(yz, st));
  int Nxo = (int) (MathFP.Mul(xo, ct) - MathFP.Mul(yo, st));

  yo = Nyo;
  yx = Nyx;
  yy = Nyy;
```

```
  yz = Nyz;
  xo = Nxo;
  xx = Nxx;
  xy = Nxy;
  xz = Nxz;
}

/** Multiply this matrix by a second: M = M*R */
public void Mult(Matrix3D rhs) {
  int lxx = MathFP.Mul(xx, rhs.xx) + MathFP.Mul(yx, rhs.xy) +
     MathFP.Mul(zx, rhs.xz);
  int lxy = MathFP.Mul(xy, rhs.xx) + MathFP.Mul(yy, rhs.xy) +
     MathFP.Mul(zy, rhs.xz);
  int lxz = MathFP.Mul(xz, rhs.xx) + MathFP.Mul(yz, rhs.xy) +
     MathFP.Mul(zz, rhs.xz);
  int lxo = MathFP.Mul(xo, rhs.xx) + MathFP.Mul(yo, rhs.xy) +
     MathFP.Mul(zo, rhs.xz) +
     rhs.xo;

  int lyx = MathFP.Mul(xx, rhs.yx) + MathFP.Mul(yx, rhs.yy) +
     MathFP.Mul(zx, rhs.yz);
  int lyy = MathFP.Mul(xy, rhs.yx) + MathFP.Mul(yy, rhs.yy) +
     MathFP.Mul(zy, rhs.yz);
  int lyz = MathFP.Mul(xz, rhs.yx) + MathFP.Mul(yz, rhs.yy) +
     MathFP.Mul(zz, rhs.yz);
  int lyo = MathFP.Mul(xo, rhs.yx) + MathFP.Mul(yo, rhs.yy) +
     MathFP.Mul(zo, rhs.yz) +
     rhs.yo;

  int lzx = MathFP.Mul(xx, rhs.zx) + MathFP.Mul(yx, rhs.zy) +
     MathFP.Mul(zx, rhs.zz);
  int lzy = MathFP.Mul(xy, rhs.zx) + MathFP.Mul(yy, rhs.zy) +
     MathFP.Mul(zy, rhs.zz);
  int lzz = MathFP.Mul(xz, rhs.zx) + MathFP.Mul(yz, rhs.zy) +
     MathFP.Mul(zz, rhs.zz);
  int lzo = MathFP.Mul(xo, rhs.zx) + MathFP.Mul(yo, rhs.zy) +
     MathFP.Mul(zo, rhs.zz) +
     rhs.zo;

  xx = lxx;
  xy = lxy;
  xz = lxz;
  xo = lxo;

  yx = lyx;
  yy = lyy;
  yz = lyz;
  yo = lyo;

  zx = lzx;
```

```
  zy = lzy;
  zz = lzz;
  zo = lzo;
 }

 /** Reinitialize to the unit matrix */
 public void unit() {

  xo = 0;
  xx = ONE;
  xy = 0;
  xz = 0;
  yo = 0;
  yx = 0;
  yy = ONE;
  yz = 0;
  zo = 0;
  zx = 0;
  zy = 0;
  zz = ONE;
 }
 /** Transform nvert points from v into tv.  v contains the input
    coordinates in starting point.  Three successive entries in
    the array constitute a point.  tv ends up holding the transformed
    points as integers; three successive entries per point */

 public void transform(int v[], int tv[], int nvert) {
  int lxx = xx, lxy = xy, lxz = xz, lxo = xo;
  int lyx = yx, lyy = yy, lyz = yz, lyo = yo;
  int lzx = zx, lzy = zy, lzz = zz, lzo = zo;
  for (int i = nvert * 3; (i -= 3) >= 0; ) {
   int x = v[i];
   int y = v[i + 1];
   int z = v[i + 2];
   tv[i] = (int) (MathFP.Mul(x, lxx) + MathFP.Mul(y, lxy) +
            MathFP.Mul(z, lxz) + lxo);
   tv[i +
      1] = (int) (MathFP.Mul(x, lyx) + MathFP.Mul(y, lyy) +
            MathFP.Mul(z, lyz) + lyo);
   tv[i +
      2] = (int) (MathFP.Mul(x, lzx) + MathFP.Mul(y, lzy) +
            MathFP.Mul(z, lzz) + lzo);
  }
 }
 public String toString() {
  return ("[" + xo + "," + xx + "," + xy + "," + xz + ";"
       + yo + "," + yx + "," + yy + "," + yz + ";"
       + zo + "," + zx + "," + zy + "," + zz + "]");
 }
}
```

Model3D.java

```
/*
 * @(#)Model3D.java 2.0 03/010/20
 */

/* A class that represent a 3dimentional Cube object which can be rotated and
   scaled
 */


package cube;
import javax.microedition.lcdui.*;
import net.jscience.math.*;

/**
 * <p>Title: 3D Graphics using Fixed-Point </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: ECU</p>
 * @author not attributable
 * @version 7.0
 */

class Model3D {

  int vert[];
  int tvert[];
  int nvert, maxvert;
  int con[];
  int ncon, maxcon;
  boolean transformed;
  Matrix3D mat;
  MathFP math;

  int xmin, xmax, ymin, ymax, zmin, zmax;

  Model3D() {

    math =new MathFP();
    mat = new Matrix3D();
    mat.xrot(30);
    mat.yrot(20);

    int ONE = 1 << 16;
    addVert(0, 0, 0);
    addVert(ONE, 0, 0);
    addVert(ONE, ONE, 0);
    addVert(0, ONE, 0);
```

```
  addVert(0, 0, ONE);
  addVert(ONE, 0, ONE);
  addVert(ONE, ONE, ONE);
  addVert(0, ONE, ONE);

  add(0, 1);
  add(1, 2);
  add(2, 3);
  add(3, 0);

  add(4, 5);
  add(5, 6);
  add(6, 7);
  add(7, 4);

  add(0, 4);
  add(1, 5);
  add(2, 6);
  add(3, 7);

 }

 /** Add a vertex to this model */
 int addVert(int x, int y, int z) {
  int i = nvert;
  if (i >= maxvert)
   if (vert == null) {
    maxvert = 100;
    vert = new int[maxvert * 3];
   }
   else {
    maxvert *= 2;
    int nv[] = new int[maxvert * 3];
    System.arraycopy(vert, 0, nv, 0, vert.length);
    vert = nv;
   }
  i *= 3;
  vert[i] = x;
  vert[i + 1] = y;
  vert[i + 2] = z;
  return nvert++;
 }

 /** Add a line from vertex p1 to vertex p2 */
 void add(int p1, int p2) {
  int i = ncon;
  if (p1 >= nvert || p2 >= nvert)
   return;
  if (i >= maxcon)
   if (con == null) {
```

```
      maxcon = 100;
      con = new int[maxcon];
    }
    else {
      maxcon *= 2;
      int nv[] = new int[maxcon];
      System.arraycopy(con, 0, nv, 0, con.length);
      con = nv;
    }
  if (p1 > p2) {
    int t = p1;
    p1 = p2;
    p2 = t;
  }
  con[i] = (p1 << 16) | p2;
  ncon = i + 1;
}

/** Transform all the points in this model */
void transform() {
  if (transformed || nvert <= 0)
    return;
  if (tvert == null || tvert.length < nvert * 3)
    tvert = new int[nvert * 3];
  mat.transform(vert, tvert, nvert);
  transformed = true;
}

/* Quick Sort implementation*/

private void quickSort(int a[], int left, int right) {
  int leftIndex = left;
  int rightIndex = right;
  int partionElement;
  if (right > left) {

    /* Arbitrarily establishing partition element as the midpoint of
     * the array.
     */
    partionElement = a[ (left + right) / 2];

    // loop through the array until indices cross
    while (leftIndex <= rightIndex) {
      /* find the first element that is greater than or equal to
       * the partionElement starting from the leftIndex.
       */
      while ( (leftIndex < right) && (a[leftIndex] < partionElement))
        ++leftIndex;

      /* find an element that is smaller than or equal to
```

```
      * the partionElement starting from the rightIndex.
      */while ( (rightIndex > left) &&
            (a[rightIndex] > partionElement))
       --rightIndex;

       // if the indexes have not crossed, swap
       if( leftIndex <= rightIndex )
       {
         swap(a, leftIndex, rightIndex);
         ++leftIndex;
         --rightIndex;
       }
     }

     /* If the right index has not reached the left side of array
      * must now sort the left partition.
      */
     if( left < rightIndex )
       quickSort( a, left, rightIndex );

     /* If the left index has not reached the right side of array
      * must now sort the right partition.
      */
     if( leftIndex < right )
       quickSort( a, leftIndex, right );

   }
}

private void swap(int a[], int i, int j)
{
  int T;
  T = a[i];
  a[i] = a[j];
  a[j] = T;
}


/** eliminate duplicate lines */
void compress() {
   int limit = ncon;
   int c[] = con;
   quickSort(con, 0, ncon - 1);
   int d = 0;
   int pp1 = -1;
   for (int i = 0; i < limit; i++) {
      int p1 = c[i];
      if (pp1 != p1) {
         c[d] = p1;
         d++;
```

```
        }
        pp1 = p1;
    }
    ncon = d;
}

int gr[] = new int[16];


/** Paint this model to a graphics context.  It uses the matrix associated
    with this model to map from model space to screen space.
    The next version of the browser should have double buffering,
    which will make this *much* nicer */
void paintWireFrame(Graphics g) {



  if (vert == null || nvert <= 0) {
      return;
    }

   transform();
    int lg = 0;
    int lim = ncon;
    int c[] = con;
    int v[] = tvert;
    if (lim <= 0 || nvert <= 0)
      return;
    for (int i = 0; i < lim; i++) {
      int T = c[i];
      int p1 = ((T >> 16) & 0xFFFF) * 3;
      int p2 = (T & 0xFFFF) * 3;
      int grey = v[p1 + 2] + v[p2 + 2];
      if (grey < 0)
        grey = 0;
      if (grey > 15)
        grey = 15;
      if (grey != lg) {
        lg = grey;

        g.setColor(0,0,255);
      }
      int x1 = math.toInt(v[p1]);
      int y1 = math.toInt(v[p1 + 1]);
      int x2 = math.toInt(v[p2]);
      int y2 = math.toInt(v[p2 + 1]);

      g.drawLine( x1 , y1 , x2 , y2 ); // draw the vertices

    }
```

```
}

/** Find the bounding box of this model */
void findBB() {
 if (nvert <= 0)
  return;
 int v[] = vert;
 int xmin = v[0], xmax = xmin;
 int ymin = v[1], ymax = ymin;
 int zmin = v[2], zmax = zmin;
 for (int i = nvert * 3; (i -= 3) > 0; ) {
  int x = v[i];
  if (x < xmin)
   xmin = x;
  if (x > xmax)
   xmax = x;
  int y = v[i + 1];
  if (y < ymin)
   ymin = y;
  if (y > ymax)
   ymax = y;
  int z = v[i + 2];
  if (z < zmin)
   zmin = z;
  if (z > zmax)
   zmax = z;
 }
 this.xmax = xmax;
 this.xmin = xmin;
 this.ymax = ymax;
 this.ymin = ymin;
 this.zmax = zmax;
 this.zmin = zmin;
}

}
```

Appendix 5: Fractal Curve Algorithm Implementation Code

FractalDisplayable.java

```
/*
 * @(#)FractalDisplayable.java      1.1 03/10/15
 * @(author)Nuwan Hettiarachchi
 */

/** FractalDisplayable extends from the J2ME Canvas class */

import javax.microedition.lcdui.*;
import net.jscience.math.*;

public class FractalDisplayable
    extends Canvas
    implements CommandListener {
  /**Construct the displayable*/
  int left = ( -134348 >>> 32) / 4, right = (117964 >>> 32) / 4,
      top = (104857 >>> 32) / 4,
      bottom = ( -104857 >>> 32) / 4, P, Q,
      deltaP = 389 >>> 32, deltaQ = 436 >>> 32; //32 bit fixed-
points
//double left=-2.05, right=1.8, top=1.6, bottom=-1.6,
// P, Q, deltaP=0.0059375, deltaQ=0.00666667;
  int row, col, xres = 320, yres = 240;

  public FractalDisplayable() {
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**Component initialization*/
  private void jbInit() throws Exception {
    // set up this Displayable to listen to command events
    setCommandListener(this);
    // add the Exit command
    addCommand(new Command("Exit", Command.EXIT, 1));
  }

  /**Handle command events*/
  public void commandAction(Command command, Displayable
displayable) {
    /** @todo Add command handling code */
    if (command.getCommandType() == Command.EXIT) {
// stop the MIDlet
      FractalMIDlet.quitApp();
    }
  }
  protected void paint(Graphics g) {
    int ZERO = MathFP.toFP(0);
    int x, y, xsq, ysq;
    x = y = xsq = ysq = ZERO;
    int index;

    for (col = 0; col < xres; col++) {
```

```
P = (left + (col >>> 32)) * (deltaP);

for (row = 0; row < yres; row++) {

  Q = (top - (row >>> 32)) * (deltaQ);
  x = y = ZERO;

  for (index = 0; index < 64; index++) {
    xsq = (x * x);

    ysq = (y * y);

    if (xsq + ysq > 4) {
      break;
    }
    y = 2 * x * y + Q;
    x = xsq - ysq + P;
  }

  switch (index % 16) {
    case 0:
      g.setColor(0, 0, 0);
      break;
    case 1:
      g.setColor(0, 0, 168);
      break;
    case 2:
      g.setColor(0, 168, 0);
      break;
    case 3:
      g.setColor(0, 168, 168);
      break;
    case 4:
      g.setColor(168, 0, 0);
      break;
    case 5:
      g.setColor(168, 0, 168);
      break;
    case 6:
      g.setColor(168, 84, 0);
      break;
    case 7:
      g.setColor(168, 168, 168);
      break;
    case 8:
      g.setColor(84, 84, 84);
      break;
    case 9:
      g.setColor(84, 84, 255);
      break;
    case 10:
      g.setColor(84, 255, 84);
      break;
    case 11:
      g.setColor(84, 255, 255);
      break;
    case 12:
      g.setColor(255, 84, 84);
      break;
    case 13:
```

```
            g.setColor(255, 84, 255);
            break;
          case 14:
            g.setColor(255, 255, 84);
            break;
          case 15:
            g.setColor(255, 255, 255);
            break;
        }
        g.drawLine(col, row, col, row);
      }
    }
  }
}
```

FractalMIDlet.java

```
/*
 * @(#)FractalMIDlet.java     1.0 03/10/10
 * @(author)Nuwan Hettiarachchi
 */

/** FractalMIDlet extends from the J2ME MIDlet class */


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

public class FractalMIDlet extends MIDlet {
  private static FractalMIDlet instance;
  private FractalDisplayable displayable = new FractalDisplayable();

  /** Constructor */
  public FractalMIDlet() {
    instance = this;
  }

  /** Main method */
  public void startApp() {
    Display.getDisplay(this).setCurrent(displayable);
  }

  /** Handle pausing the MIDlet */
  public void pauseApp() {
  }

  /** Handle destroying the MIDlet */
  public void destroyApp(boolean unconditional) {
  }

  /** Quit the MIDlet */
  public static void quitApp() {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }
}
```

Appendix 6: Cartesian Curve Generating Algorithm Implementation Code

SincDisplayable.java

```
/*
 * @(#)SincDisplayable.java   1.4 03/11/20
 * @(author)Nuwan Hettiarachchi
 */

/** SincDisplayable extends from the J2ME Canvas class */

import javax.microedition.lcdui.*;
import net.jscience.math.*;

public class SincDisplayable
    extends Canvas
    implements CommandListener {

  public SincDisplayable() {
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {

    setCommandListener(this);

    addCommand(new Command("Exit", Command.EXIT, 1));
  }

  public void commandAction(Command command, Displayable
displayable) {

    if (command.getCommandType() == Command.EXIT) {

        SincMIDlet.quitApp();
    }
  }

  public void paint(Graphics g) {

    int x, y=1;
    int col, row, old_col = 0, old_row = 0;
    g.setColor(255, 255, 255);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0, 0, 0);

    g.drawLine(0, getHeight() / 2, getWidth(), getHeight() / 2);
    g.drawLine(getWidth() / 2, 0, getWidth() / 2, getHeight());
    g.setColor(0, 0, 168);
    g.drawString("Sin(x)/x", 0, 0, 0);
    for (col = 0; col < getWidth(); col++) {
```

```
        x = MathFP.Div( (col - getWidth() / 2), 12<<32);

        if (x!=0){


        y = MathFP.Div(MathFP.Sin(x), x);

        }
        row =getHeight()/2- (MathFP.Mul(y, getHeight() / 2)) ;

        g.drawLine(old_col, old_row, col, row);

        old_col = col;
        old_row = row;
      }
    }
}
```

SincMIDlet.java

```
/*
 * @(#)SincMIDlet.java  1.0 03/11/15
 * @(author)Nuwan Hettiarachchi
 */

/** SincMIDlet extends from the J2ME MIDlet class */


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

public class SincMIDlet
    extends MIDlet {
  private static SincMIDlet instance;
  private SincDisplayable displayable = new SincDisplayable();

  public SincMIDlet() {
    instance = this;
  }

  public void startApp() {
    Display.getDisplay(this).setCurrent(displayable);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }

  public static void quitApp() {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }
}
```

Computer Science

Course Code: MACSCI

Course Outline and ACS Cross Reference Table

School of Computer and Information Science

Edith Cowan University

2004

Computer Science (Major Code: MACSCI)

This format allows the applicant to provide a graphical view of the course structure to assist the Accreditation Panel in its analysis of the course content, especially in terms of breadth and depth of the content. If you use this format please use the colour codes provided at the bottom of the form to indicate mandatory and elective ICT units as opposed to non ICT units. It would also assist if prerequisite subject are linked in some way (a red line is suggested). If units are not of equal weighting please indicate with the name of the unit the percentage that units constitutes of the whole course.

YEAR 1

Semester 1	Semester 2
CSG1132 Communicating in an IT Environment	CSG1105 Applied Communications
CSP1150 Programming Principles	CSI1101 Computer Security
ENS1161 Computer Fundamentals	CSG1206 Operating Systems Prereq: ENS1161 Computer Fundamentals
CSI1241 Systems Analysis	CSG1207 Systems and Database Design Prereq: CSI1241 Systems Analysis

YEAR 2

Semester 1	Semester 2
CSP2204 Data Structures Prereq: CSP1150 Programming Principles [Equiv: CSP1250 Data Structures and CSP1243 Data Structures using ADA]	CSG2245 Computer Science Methods Prereq: CSP1250 Data Structures with Java and ENS1161 Computer Fundamentals
CSP2343 Object Oriented Programming with C++ Prereq: CSP2347 UNIX and C	CSG3204 Information Services Management Prereq: CSI1241 Systems Analysis plus at least 60 credit points at second year level [Equiv: IST3330 Information Services Management]

YEAR 3

Semester 1	Semester 2
CSP3241 Internet and Java Programming Prereq: CSP1150 Programming Principles or equivalent, and one other programming language unit.	CSG2341 Intelligent Systems Prereq: CSP1250 - Data Structures with Java
Elective	CSP3341 Programming Languages and Paradigms Prereq: CSP2343 Object-oriented Programming with C++ or CSP3241 Internet and Java Programming

ACS COURSE ACCREDITATION: CROSS REFERENCE TABLE

COURSE: COMPUTER SCIENCE CODE: MACSCI

Old Code	New Code	Title	Concurrent	Pre-Req	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11	3.12	3.13	3.14
	CSG1132	Communicating in an IT Environment									✓	✓						
	CSP1150	Programming Principles																
	ENS1161	Computer Fundamentals			✓				✓	✓								
	CSI1241	Systems Analysis				✓						✓		✓		✓	✓	
CSG2130	CSG1105	Applied Communications					✓	✓										
CSI2201	CSI1101	Computer Security													✓			
CSG2343	CSG1206	Operating Systems			✓						✓	✓						✓
CSI2341	CSG1207	Systems and Database Design				✓	✓		✓									
	CSP2204	Data Structures																
	CSP2343	Object Oriented Programming with C++																
	CSG2245	Computer Science Methods																
	CSG3204	Information Services Management																
	CSP3241	Internet and Java Programming											✓					
	CSG2341	Intelligent Systems										✓						
	CSP3341	Programming Languages and Paradigms				✓			✓				✓					✓

RED = INCLUDED IN CURRENT ACCREDITATION
GREEEN = INCLUDED IN CURRENT ACCREDITATION UNDER OLD CODE

3.7 ETHICS/SOCIAL IMPLICAITONS/PROFESSIONAL PRACTICE

Topic	Where in the Course is this topic addressed? (Subject Name/Number)	Is this topic assessed? If, so what form does the assessment take?	Is the subject containing this topic mandatory?
a) Introduction - what has Ethics got to do with me?			
b) Social Issues - Culture and heritage - Culture and Technology			
c) Global Issues			
d) Organisational Issues -Application of technology in Australian Business - Ethical issues in private and public sectors			
e) Technology			
f) Belief Systems The law and computer crime - Reliability, safety in software systems			
g) Responsibility – personal & community			
h) IT Professional Codes - ACS Codes of Ethics - ACS Codes of Conduct			

3.8 INTERPERSONAL COMMUNICATION

Topic	Where in the Course is this topic addressed? (Subject Name/Number)	Is this topic assessed? If, so what form does the assessment take?	Is the subject containing this topic mandatory?
a) Written Communication - effective expression - logical ordering of ideas - format and content of reports and formal documents - technical writing and documentation proposals and procedures			
b) Verbal Communication - structuring material for oral presentation - oral presentation of information - the use of appropriate supporting technology - effective speaking and audience management			
c) Interpersonal Skills - interview techniques - managing group dynamics • technical reviews • formal and informal meetings - negotiation skills team management and conflict resolution			

3.10 PROJECT MANAGEMENT/QUALITY ASSURANCE

Topic	Where in the Course is this topic addressed? (Subject Name/Number)	Is this topic assessed? If, so what form does the assessment take?	Is the subject containing this topic mandatory?

a) Concepts and Models
- project definition
- project success
- measuring success
- post-implementation reviews
- project size
  • lines of code
  • effort/duration
  • function points
- project life cycle			

b) Project Management Techniques - steering committees - project justification - project planning - project development strategies - methodologies - risk assessment - estimation - scheduling - project tracking and reporting			
c) Introduction to Software Quality - understand and measuring quality - the costs and benefits of quality - roles of people in producing quality software - factors that impact on quality of software			

Topic	Where in the Course is this topic addressed? (Subject Name/Number)	Is this topic assessed? If, so what form does the assessment take?	Is the subject containing this topic mandatory?
d) **Software Quality Planning** - role of planning - software quality requirements - preparing a software quality plan - implementing a software quality plan - preparing a quality manual			
e) **Processes for Assuring the Quality of Software** - risk management - conformance to standards - reviews, audits, walkthroughs and inspections - verification, validation and testing - configuration management			
f) **Product Quality** - software product standards - quality attributes of software - product characteristics of quality software - measuring and evaluating product quality and associated metrics			
g) **Process Quality** - software process standards - process definition - process measurement			
h) **Process Assessment** - process improvement - capability evaluation - procurement of software			

i) **Post Development Software Quality Assurance** - maintenance and evolution of software - re-engineering of software - software product quality improvement			