國立交通大學

多媒體工程研究所

碩 士 論 文

CWT 中 OpenGL 實作之研究

The Study of OpenGL Implementation of CWT

研 究 生：鄭欽議

指導教授：吳毅成　教授

中 華 民 國　九 十 六　年 六 月

CWT 中 OpenGL 實作之研究

The Study of OpenGL Implementation of CWT

研 究 生：鄭欽議　　Student：Chin-Yi Cheng

指導教授：吳毅成　　Advisor：Yi-Cheng Wu

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

CWT 中 OpenGL 實作之研究

研究生：鄭欽議　　指導教授：吳毅成

國立交通大學 多媒體工程研究所

摘要

在 Java 中，用來開發使用者圖形介面(GUI)的函式庫，如 Abstract Window Toolkit(簡稱 AWT)或是 Swing，為影響以 Java 開發之應用程式繪圖效能最關鍵的一部份。然而 Java 在繪圖效能上的表現卻不夠理想，雖然 Java 的版本一再更新，也針對 Java 本身的繪圖效能作了很多的改善，但還有一部分不算少數的使用者還在使用較舊版本的 Java，因此這些人無法得到新版本所提供的各項效能改進。而且很多軟體當初是在舊版本上開發的，這些軟體也無法直接獲得新版本所提供的效能改善，除非修改原始碼，使用新版本中提出的 API。此外，Java 在不同作業系統上的效能並非一致，這也會造成 Java 程式開發者在效能評估上的困難。

因此，為了要提升 Java 的繪圖效能以及讓現有的 Java 程式可以直接受惠，本論文基於一套開放式的架構 CYC Window Toolkit(簡稱 CWT)[32]之上，實作出 OpenGL 的部份，利用 OpenGL 繪圖函式庫，完成各項的繪圖操作，並針對效能的部份做優化，使其可以在不同的作業系統間達成：提升繪圖效能、效能可攜性。

The Study of OpenGL Implementation of CWT

Student: Chin-Yi Cheng Advisor: Yi-Cheng Wu

Institute of Multimedia Engineering
National Chiao Tung University

Abstract

The libraries such as Abstract Window Toolkit (AWT) and Swing used to develop Graphical User Interface (GUI) are the key parts of rendering performance for applications designed in Java. However, the rendering performance in Java is not very efficient. While new Java versions are released to improve rendering performance, some users still keep using the old ones. They cannot have the benefits from the new Java releases. In addition, the applications, which were developed in the old Java releases, cannot get the benefits of improving rendering performance from the new ones unless modifying the source codes for new APIs. Moreover, the performance of Java on different operating systems is inconsistent. It is hard to estimate the performance of Java applications for developers.

To improve rendering performance in Java and let the existed Java applications benefit from this, this thesis implements the OpenGL part based on CYC Window Toolkit (CWT)[32]. We do all rendering operations with OpenGL and optimize the rendering performance to achieve: performance improvement and portable performance on different operating systems.

誌謝

首先，要感謝的人是我的指導教授，吳毅成教授，在我寫這篇論文的期間，沒有他的辛勤指導與嚴格要求，我無法完成這篇碩士論文。也要感謝我的碩士論文口試委員－許舜欽教授、莊榮宏教授以及顏士淨教授，感謝他們細心審查我的論文，並給予我寶貴的意見。

另外要特別感謝博士班的汪益賢學長，這篇論文的許多原始想法與作法都是從學長來的，在其協助指導下不斷的討論、修正而完成。沒有汪學長的指導，本篇論文一定無法完成。每當遇到問題或瓶頸時，學長總是很有耐心的了解並給予關鍵的意見。非常感謝汪益賢學長辛勤的指導與鼓勵。

同時，也要感謝已經畢業的姜智耀學長，認真且鉅細靡遺的解說當初他的論文內容及遇到的問題，還有黃德彥學長即使有龐大的工作壓力還是透過網路給予我鼓勵跟意見。感謝劉育嘉學長，你當初在實驗室採買的獨立筒單人床，讓我在最後關鍵的衝刺時刻可以有一個舒適的休息環境。最後是蔣承翰學長，給予我很多的意見與討論。

感謝實驗室的同學，君鴻、炯珽、志宏、宜融，在最後的時刻，大家相互鼓勵與打氣的畫面我會永遠記得。還有所有的學弟們，因為有他們的陪伴，讓我的生活充滿了歡樂。

感謝我在交大的好友士暐，在最後的幾十個夜晚的宵夜時間彼此加油，陪我聊天舒緩口試前夕的壓力。還有遠在台灣大學奮鬥的研究生們，嘉偉、柏銓、許平的支持與關心。還有好友立志等。

最後，要感謝的是我的父母、正在捍衛國土的軍人弟弟、女友思嘉，在我求學生涯上所給我的鼓勵與支持，總是默默的支持我，給我最大的信心、動力。謹以此論文，獻給我最摯愛的你們。

目錄

表目錄

圖目錄

第一章、緒論

本章首先會在 1.1 節簡介本論文相關的背景知識，討論 Java 在下一代遊戲軟體製作時，成為開發時的主流程式語言之可行性。1.2 節則討論 Java 在一般遊戲軟體開發中不被大量使用的可能原因。1.3 節中介紹一些可以在 Java 中，用來提升繪圖效能(rendering performance)的繪圖函式庫。在 1.4 節，將提出本論文希望解決的問題以及期望達成的目標。最後在 1.5 節簡介本論文的內容大綱。

1.1 背景介紹

電腦遊戲產業因為國內的特殊背景，使其在國內的遊戲產業中扮演相當重要的角色，尤其是目前最熱門的線上遊戲，幾乎都只侷限在電腦遊戲中。以下簡單介紹電腦遊戲的現況。

1.1.1 電腦遊戲

電腦遊戲若以網路功能來分類，一般可以分作「單機遊戲」以及「線上遊戲」兩種。單機遊戲為不需要連上網路，只要在個人電腦上安裝遊戲軟體後便可以使用且完整執行。線上遊戲則必須連上網路才能玩，一般線上遊戲又可再細分為兩種：[38]

1. 撮合線上遊戲(Match Making Game)
   遊戲伺服器主要負責撮合玩家們之間的對戰，提供數人或是數十人間在伺服器上的互動。像是「明星三缺一 online」[37]、「戲谷麻將館」[35]、「CYC 遊戲大聯盟」[39]等。這類型的遊戲多半是輕鬆休閒的遊戲，所以通常又被稱作「休閒遊戲」(Casual Game)。

2. 多人線上遊戲(Massive Multi-player Online Game，MMOG)
   遊戲伺服器可以服務成千上萬的遊戲玩家，在虛擬的遊戲世界中同時作互動，通常遊戲都有一個設計好的故事背景與世界觀，可以讓玩家們進行角色扮演。例如「天堂」[40]、「魔獸世界」[36]。

因為在台灣拼裝電腦的盛行，使得個人電腦在國內相當普及，更因為網路的發達，讓電腦線上遊戲因此而蓬勃發展。對於現今國內的遊戲開發公司來說，研發線上遊戲幾乎是不可或缺的選擇。

1.1.2 Java 與遊戲軟體開發

目前的遊戲開發大多還是使用 C/C++作為開發用的程式語言，Java 很少被選擇用來研發遊戲，雖然說 Java 有很多優點，例如物件導向設計的優勢，可以大幅縮短程式開發的時程，降低維護成本，從而提高研發人員的生產力。但是因為 Java 在繪圖效能上的表現還是比 C/C++差，所以使得 Java 通常只被拿來開發簡單的休閒遊戲(Casual Game)。

遊戲軟體的開發，早期一般都是使用組合語言，因為效能的考量，所以遊戲的研發都還是以組合語言為主流。C 語言在六零年代被提出，到了八零年代的初期，已經有很多的軟體以 C 語言作為開發用的程式語言了，但是遊戲軟體卻還沒有。一直到了 1993 年，id Software 發行了毀滅戰士(Doom)這款幾乎完全以 C 語言開發的遊戲後，扭轉了所有當時的遊戲設計師對 C 的看法。到現在，已經甚少有完全使用組合語言來開發的遊戲軟體。[7]

遊戲開發中，效能(Performance)與產能(Productivity)均相當重要[34]。如同圖 1-1 所示，組合語言到 C/C++，效能上是可以被接受的，此外，C/C++高階語言的特性，使得軟體研發的產能可以大幅的提升。因此，與組合語言相比，C/C++能夠在更短的時間內開發出更

複雜的遊戲。

不過與 C/C++相比，Java 擁有更多高階語言的優勢，所以本論文接下來將進一步討論 Java 用來開發遊戲軟體的可行性。

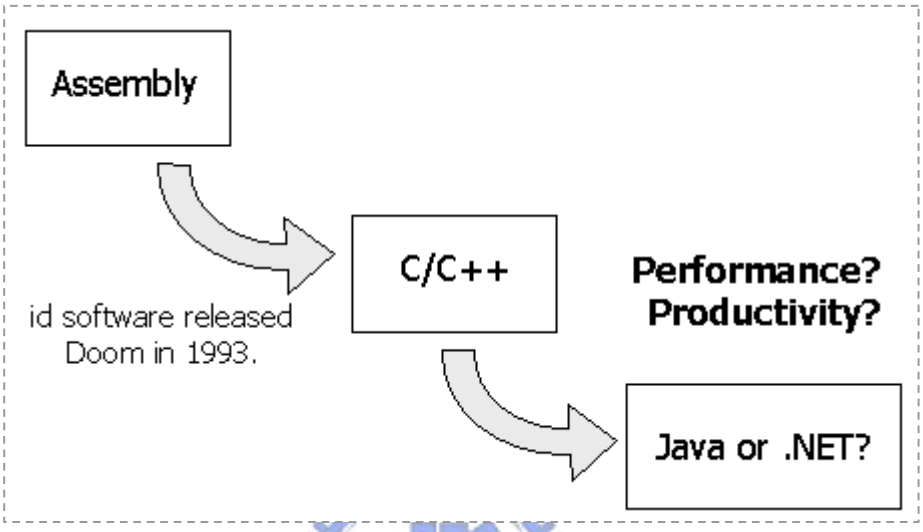

圖 1-1：Java 或 .NET 在遊戲軟體開發上未來的機會

1.2 Java

Java 在 1995 年由 Sun Microsystems(以下簡稱 SUN)正式應用在全球資訊網(World Wide Web)上。Java 主要的設計考量，是希望可以在所有的平臺上適用，達到跨平臺的目標。物件導向程式語言也一直是 Java 在軟體開發上非常重要的一項特性，使其成為熱門的程式語言之一。接下來就 Java 在遊戲開發中所注重的產能與效能上作介紹，並探討 Java 未能成為遊戲開法主流程式語言的可能原因。

1.2.1 產能(Productivity)

有效的縮短開發時程，並讓接下來的維護工作能夠更簡單，提高研發人員的產能，一直都是軟體開發的一項重要課題。Java 擁有很多特點，像是純物件導向的程式語言、豐富的函式庫、內建的記憶體

管理系統(Garbage collection)，以及 Java 最重要的核心價值-跨平臺，使 Java 擁有較好的產能。根據[5]，Java 與 C/C++的產能比較，當一項專案完全以 Java 來開發時，的確可以減短開發的時程，而且在未來的維護方面也可以節省較多的成本，並增加軟體開發的產能。表 1-1 為該報告所提出的資料。

	時間/成本 節省	生產力提升
開發階段	40%	67%
維護階段	30%	42%
整體	25%	

表 1-1: 與 C/C++相比，完全採用 Java 來開發的軟體專案

資料來源：[5]

1.2.2 效能(Performance)

關於效能的部份，可以簡單分成執行效能與繪圖效能這兩個部份來討論。一般來說，大家對於 Java 執行效能的印象都是比較不好。但是隨著 SUN 一直不斷推出新的 Java 版本，在執行效能上，Java 也有了相當程度的改進，甚至對部分程式原始碼做某種程度的最佳化後，改善的效果更可以與 C/C++匹敵。從表 1-2 的資料[7]中，可以看出 Java 1.0 到 1.4 版與 C/C++的執行效能比較。

JDK/JRE 版本	比 C++ 慢的程度係數 (Factors slower than C++)
1.0	20 – 40
1.1	10 – 20
1.2	1 – 15
1.3	0.7 – 4
1.4	0.5 – 3

表 1-2：Java 與 C/C++執行效能的比較

資料來源：[7]

Java 效能的提升，主要來自即時編譯技術(Just-In-Time Compilation，簡稱 JIT)[21]，以及後來 SUN 提出的熱點編譯技術(HotSpot Compilation，簡稱 HotSpot)[23]。執行時期的動態編譯技術大大提升了 Java 的執行效能，在一些特殊情況下，甚至能表現得比 C++程式更好[7]。

繪圖效能的部份是本論文的重要討論項目，所以特別獨立在 1.3 節中進行深入的介紹與討論。

1.3 可提升 Java 繪圖效率的繪圖函式庫

電腦硬體技術的演進非常快速，而繪圖加速卡更是一再的推陳出新。也因為繪圖加速卡的進步，使得遊戲中能夠表現的特效以及虛擬場景中的物體都能夠以逼真的方式顯示，讓遊戲的內容能夠更豐富，帶來的視覺效果更好。因此，與繪圖加速卡效能息息相關的繪圖函式庫，在遊戲軟體開發中就佔了相當重要的地位。

目前遊戲開發用的主流繪圖函式庫有兩個。DirectX[12]是由微軟開發，雖然只能夠在微軟視窗作業系統(Microsoft Windows)上使用，但因為微軟視窗作業系統的市場佔有率相當高，使得 DirectX 遊戲有廣大的市場作為後盾。另外一個主流的繪圖函式庫是 OpenGL[31]，OpenGL 是業界的標準，其跨平臺的特性，讓 OpenGL 可以不只在微軟視窗作業系統上使用。

Java 本身的繪圖效能不是很好，主要是因為最早期的繪圖函式庫並沒有使用硬體加速。因此，想要提升 Java 的繪圖效能，需要考量使用具有硬體加速特性的繪圖函式庫。

因此，接下來的幾個段落，將會進一步的介紹與討論 Java 中可以使用的繪圖函式庫。首先，1.3.1 節介紹 Java 最基本的繪圖函式庫，以及在不同 Java 版本中的繪圖效能改進。1.3.2 節接著簡介除了

Java 中基本的繪圖函式庫外，還有哪些其他可能的套件可以使用，這些套件均使用本節一開始提到的 OpenGL 或者是 DirectX，因此具備較好的繪圖效能。

1.3.1 Java 基本繪圖函式庫

Java 中最基本的繪圖函式庫就是抽象視窗工具組(Abstract Window Toolkit，以下簡稱 AWT)[26]，這是一套 Java 提供的繪圖函式庫，其中除了有很多繪圖的物件跟函式可以使用外，還包含了很多視窗的元件可以供開發者使用，對於使用者圖形介面的開發有相當大的幫助。AWT 基本上分成兩大部分，一部分稱作使用者介面工具組(UI Toolkit)，另一部分為繪圖工具組(Graphics Toolkit)。[34]

使用者介面工具組，包含了各式各樣視窗元件的組成架構外，還有事件的處理機制。AWT 為了做到在不同平臺上能夠與該平臺其他的視窗程式有一致性的外觀呈現，利用重型元件(Heavy-weight Component)，在視窗系統底層生成，並由視窗系統來處理。後來推出的 Swing[8]，更是為了讓視窗元件能夠更多樣化，將元件改為軟體繪製的輕型元件(Light-weight Component)。但是因為 Swing 是基於 AWT 的架構所開發出來的，又變成軟體負責處理元件的繪製，所以其繪圖效能十分仰賴 AWT。

繪圖工具組，包含許多繪圖運算以及影像處理功能，是影響 Java 繪圖效能最重要的部份。JDK 1.2 中首次提出的 Java 2D[10]，負責所有繪圖相關的功能，但是為了提供更多新的功能，並降低各平臺差異對畫面一致性的影響，使得 Java 2D 必須以軟體實作的方式來達成，放棄了硬體協助的部份，因此使得效能受到影響。

為此，JDK 1.4 提出了改善方式，在 Win32 平臺利用 DirectX 來獲得硬體加速[17][23]。雖然與 JDK 1.2 比起來，有不錯的改善，但是除了必須改寫原始碼外(使用 VolatileImage 類別[28])，某些繪圖效

能也沒有獲得較好的改善，例如文字的繪製[32]。JDK 1.5 雖然更進一步的提供了 OpenGL 來改善繪圖效能[19]，但是卻有啟用了也沒什麼效果的情況產生。甚至 JDK 1.6 有無法正常顯示繪製結果的狀況出現。

由此可知，Java 的基本函式庫目前仍存在許多問題，在繪圖效能的表現上不盡理想。下一節將會介紹其他 Java 可使用的非基本函式庫。

1.3.2 Java 其他的繪圖函式庫

除了 Java 基本的繪圖函式庫外，透過 Java 中的原生介面(Java Native Interface，以下簡稱 JNI)[18]，也能直接使用以其他語言寫成的原生程式碼(Native code)，例如作業系統所提供的函式庫。因此，也可以透過 JNI 使用一般的繪圖函式庫，如 OpenGL 或 DirectX。但若要透過 JNI 使用這些繪圖函式庫，必須自行產生與原生函式一對一對應的實作。而且若使用 JNI，可能必須放棄 Java 跨平臺的這項優勢，例如 DirectX 只能在微軟視窗作業系統上使用。

要自行產生一對一對應的原生實作相當麻煩，不過目前已有很多套件提供這樣的功能，讓使用者能直接使用到 DirectX 或 OpenGL 這些繪圖函式庫，表 1-3 列出一些可以使用的套件。

簡稱	函式庫名稱	原生支援	提供者
JOGL	Java bindings for OpenGL API [20]	OpenGL	SUN
LWJGL	Light-Weight Java Gaming Library [3]	OpenGL	lwjgl.org[3]
	Microsoft Language Extension [14]	DirectX	Microsoft

表 1-3：Java 上可使用的其他繪圖函式庫[34]

其中，微軟的 Java 虛擬機器(Microsoft Java Virtual Machine，以下簡稱 MSVM)[13]中的微軟語言擴充(Microsoft Language

Extension)提供可以直接使用 DirectX 的 API。MSVM 是內建於微軟的網頁瀏覽器 Internet Explorer(IE)之中，因微軟視窗作業系統的市場佔有率相當高，使得 MSVM 成為一般使用者最容易接觸到的 Java 虛擬機器。

1.4 問題與目標

要提升 Java 的繪圖效能，可安裝新版本的 Java，或使用上述具有硬體加速特性的繪圖函式庫。但使用最新版本 Java 的使用者並非多數，而因 MSVM 只支援到 JDK 1.1，更讓只使用 IE 瀏覽器，不熟悉如何更新 Java 的使用者無法得到 Java 在新版本改良後的繪圖效能。表 1-4 列出目前 Java 各版本使用者的比例，從資料中可以發現 MSVM 的使用者約佔了 16%，而最新版的 Java 6 則只有將近 14% 的使用者。所以仍有一定比例的使用者尚未更新到最新版本。

而若是使用具有硬體加速特性的繪圖函式庫，要面臨的問題是，開發人員必須自行實作所有的視窗元件，因為這些函式庫並沒有 AWT 提供的視窗元件，供開發者設計軟體的使用者圖形介面。即使配合 AWT 一起使用，雖然能因 AWT 的使用者介面工具組而受益，但繪圖效能也會因 AWT 而受到限制。

Java Version	Supported OS	Released Time	Percentage
			2007/6/23
MSVM	Windows	1997/02	16.12%
Sun J2SE 1.3	Windows	2000/05	0.95%
Sun J2SE 1.4	MacOS X	2002/02	10.08%
Sun J2SE 1.5	Linux	2004/09	59.13%
Sun J2SE 1.6	Solaris	2005/11	13.71%

表 1-4：Java 各版本使用者的比例
資料來源：GC Usage Statistics [1][32]

因此在[32]中，提出 CYC Window Toolkit (簡稱 CWT)此架構。CWT 提供與 AWT 相容的應用程式介面，並針對繪圖工具組的部份，設計出可採用不同繪圖函式庫來實作的架構。CWT 擁有幾項特性：提升繪圖效能、與 AWT 相容的介面、可攜性，並開放讓開發人員能直接操作 DirectX 或 OpenGL 物件，使其可以更自由開發所需的功能。

在[32][34]中，已經使用 MSVM 中微軟語言擴充實作出 CWT-DX3 此部份，但只有 MSVM 的使用者能因此獲得硬體加速而改善繪圖效能，使用其他版本的 Java VM 或其他作業系統的使用者，則無法獲得 CWT 這一部分的硬體加速。

因此，本篇論文的系統建立於 CWT 架構上，利用 JOGL 套件，實作 CWT 中 CWT-GL 的部份，期望除 MSVM 的使用者外，其他 Java VM 與其他作業系統的使用者，都能因 CWT 獲得更好的繪圖效能。更令 CWT 在提升繪圖效能這部份，不只侷限在微軟視窗作業系統上，也能與 Java 的跨平臺特性一樣，在其他不同的平臺上使用，並擁有高效且穩定的繪圖效能。[33]

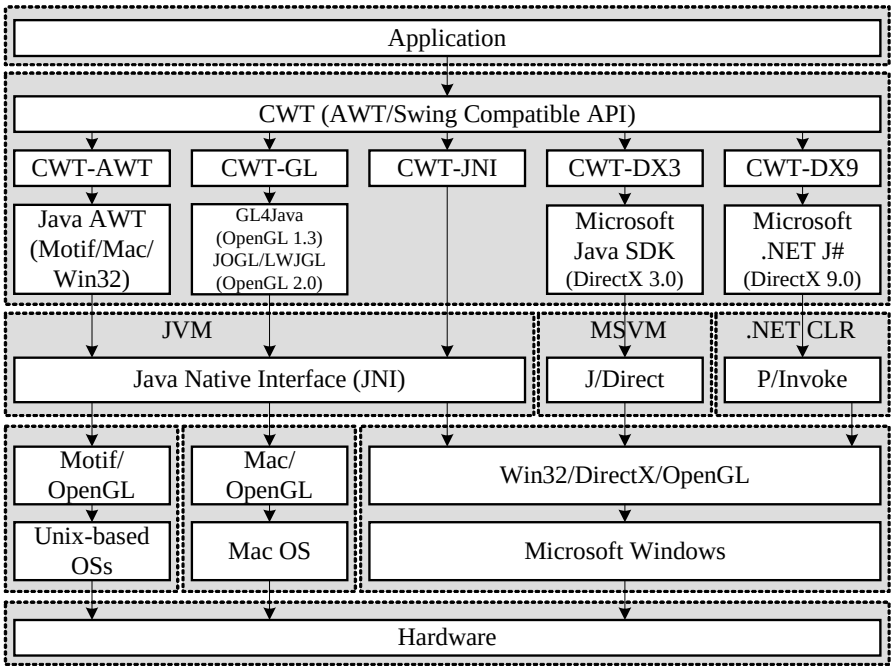

圖 1-2：CWT 架構圖[32]

1.5 本文大綱

本論文第一章介紹 Java 在改善繪圖效能上的相關背景知識，並且提出解決方法以及期望達成的目標。第二章將針對系統架構做詳細的說明。第三章則深入介紹本論文如何利用 OpenGL 完成所有繪圖指令。第四章對本論文所作的實驗結果做分析與討論。最後第五章針對本論文的結果作總結，並進一步討論未來的發展方向。

第二章、系統架構

由於本論文的系統建構於 CWT 上，因此本章一開始將於 2.1 節簡介 CWT 的設計架構，並詳細說明本論文實作的 CWT-GL 部分，在 2.2 節介紹如何利用 JOGL 此套件來實作。

2.1 CWT 系統架構

本節首先在 2.1.1 節簡介 CWT 的設計原理及架構。接著在 2.1.2 節說明本論文如何實作 CWT 架構中 CWT-GL 的部分。

2.1.1 CWT 的設計原理及架構

CWT 設計的目標是提供與 AWT 相容的 API，並改善繪圖工具組，使其能以不同的繪圖函式庫來實作，達成改善繪圖效能之目的。CWT 架構主要與 AWT 類似，可分為三種模式，以下介紹三種模式的主要功能。[32][34]

1. 元件模式(Component Model)：
   由原生容器(Native Containers)與輕型元件(Light-weight Components)所組成。此處的設計與 Swing 類似，所有輕型元件的繪製都是由軟體負責。原生容器主要包含 Applet、Dialog、Frame 與 Window，除了作為輕型元件的畫布，將所有輕型元件都繪製在其之上外，也負責傳送系統底層所生成的事件。

2. 事件模式(Event Model)：
   事件處理機制基本上與 AWT 相同，但因 CWT 中大部分元件都屬於輕型元件，故必須仰賴原生容器產生的各種事件。為

了將這些事件轉交給 CWT 中的輕型元件，利用 EventManager 負責把原生容器所產生的事件存放在 CWT 的事件佇列(Event Queue)中，等待 CWT 內部的事件分派執行緒(Event Dispatch Thread)將事件分配給對應的輕型元件去處理。

3. 繪圖模式(Painting Model)：
   繪圖模式基本上類似於 AWT 與 Swing 本來的架構，其中最重要的部份如圖 2-1，為了讓繪圖模式能夠支援各式各樣不同的繪圖函式庫，將其抽象化分成數個類別，例如 Graphics、Image、Toolkit 與 WindowContext，因此各種的繪圖函式庫都能藉由繼承這些類別，完成不同的實作。本論文也是透過同樣的方式，完成 CWT-GL 實作。

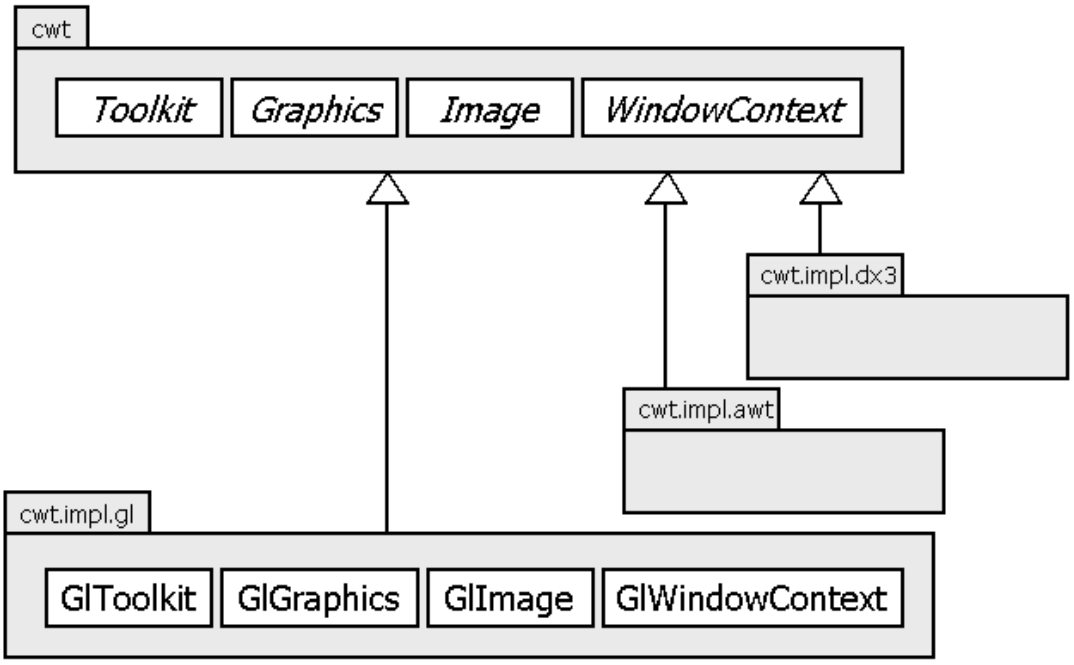

圖 2-1：CWT-GL 與繪圖模式的繼承關係

2.1.2 實作 CWT-GL

本節將詳細介紹 CWT-GL 實作 CWT 中的哪些類別，並解釋各

個類別所負責的工作。

首先簡介 AWT 中元件系統之設計。AWT 中元件的設計是橋接模式(Bridge pattern)。如圖 2-2，分為抽象化之平臺獨立 API (Platform-independent API)部分，以及平台特定實作 (Platform-specific Implementation)部份。圖中以一個視窗物件 (Window)為例，每個 Window 都會有一對應之 WindowPeer 物件。為達成使用 AWT 設計之視窗程式能擁有與同平臺相同之視窗介面，WindowPeer 物件將會有各種不同平臺之實作，如圖中所示之 Windows WindowPeer、Motif WindowPeer 或 MacOS WindowPeer 等。於 AWT 中，Peer 是當該元件即將被顯示在視窗上時，才由 Toolkit 物件產生。

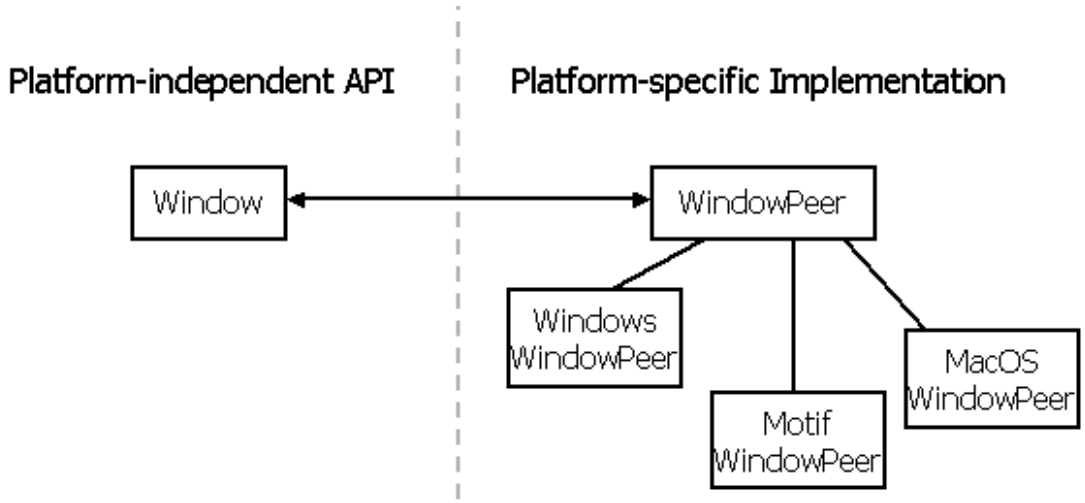

圖 2-2：AWT 中元件之橋接模式設計

1. GlToolkit：

   此類別負責各種原生容器的生成，並在原生容器產生之後，將對應的 GlWindowContext 類別也一併生成。除了原生容器外，也與 AWT 之 Toolkit 類別類似，負責 Image 的各種生成。至於其他輕型元件的生成函式，則被抽象化到其父類別 CwtToolkit 中，與其他繪圖函式庫之實作共用相同的類別函式。

2. GlWindowContext：

由 GlToolkit 產生，存放有對應的原生容器，並且負責生成 GlGraphics 物件，回傳應用程式要求的 Graphics 物件，讓應用程式能透過此物件進行繪製。除 GlGraphics 物件外，也負責處理由抽象元件傳遞來的 Image 生成指令，產生 GlImage 物件。圖 2-2 為 GlWindowContext 生成過程之流程。

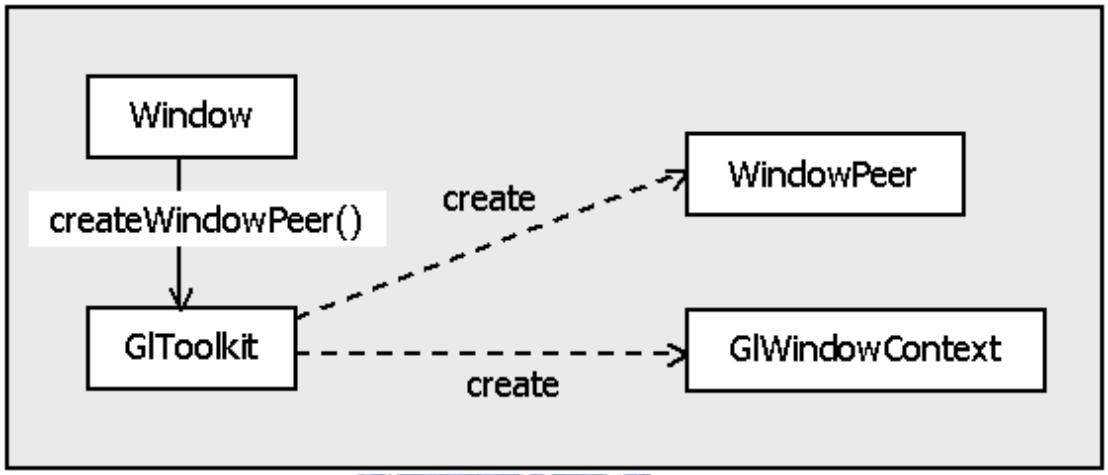

圖 2-3：GlWindowContext 的生成過程

此處以 Window 為例，當視窗元件實際要於螢幕上顯示時，會透過 GlToolkit 要求其產生對應之 Peer 物件，例 WindowPeer。同時此時也會生成一與 WindowPeer 物件對應之 GlWindowContext 物件，GlwindowContext 為負責作為所有輕型元件畫布之重型元件。

3. GlImage：

負責存放 Image 的各種屬性，並且生成對應之原生 Image。GlImage 的生成方式有兩種，其一是由 Toolkit 直接生成，可將存在的檔案或是資料直接生成一個 Image 物件，在之後繪製時使用，也能產生一空白的 Image 作為背景繪圖用。另一種則是透過抽象元件，傳遞生成 Image 指令至 GlWindowContext，最後產生 GlImage 物件。

4. GlGraphics：

此為繪圖模式中最重要的類別，負責所有繪圖指令，一切的繪圖動作最後都由此類別負責運算並繪製出畫面來。圖 2-3 為 GlGraphics 的生成過程，其一是由抽象元件取得繪圖物件，透過 GlWindowContext 產生一對應的 GlGraphics 物件回傳。其二是由 GlImage 取得繪圖物件。透過兩種不同的方式，使其能個別在一般元件或者是圖片上進行繪製。

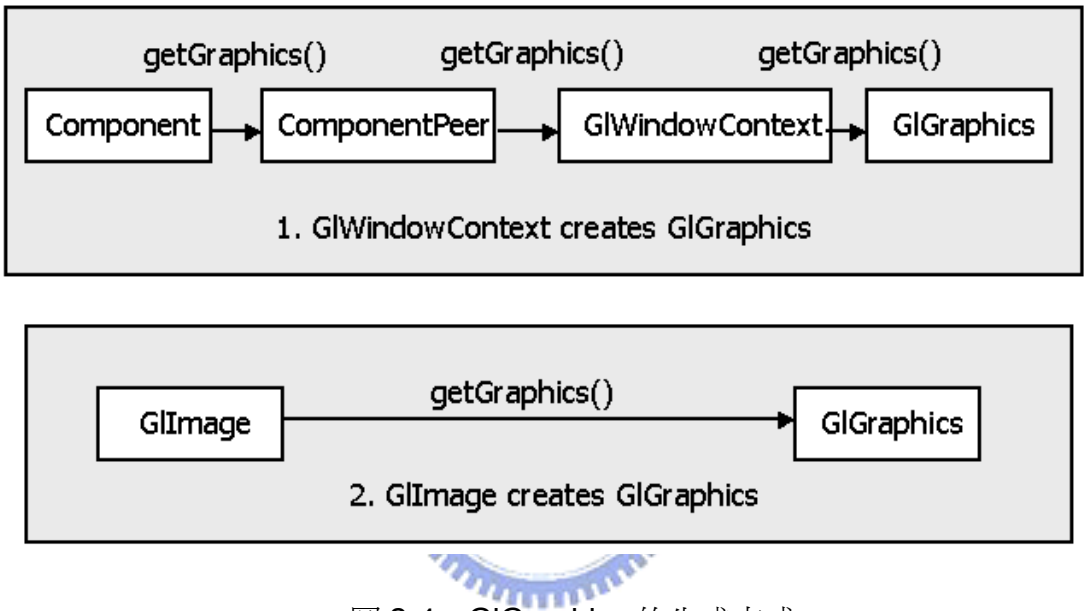

圖 2-4：GlGraphics 的生成方式

2.2 利用 JOGL 實作 CWT-GL

本節介紹本論文所選用的 JOGL[20]套件，並說明如何將此套件套用於 CWT 架構之中。

JOGL 為一套件，目的讓使用 Java 開發的應用程式，能透過它完整使用到 OpenGL 2.0 的所有 API，甚至是各種不同廠商所提供之擴充功能。JOGL 目前支援 Java 1.4 與之後的版本。本論文選用 JOGL 作為使用 OpenGL 此繪圖函式庫的原生介面套件。

JOGL 中，提供兩種基本的 AWT/Swing 相容之視窗元件：

GLCanvas 與 GLJPanel，以及一事件偵測程式(Event Listener)，稱作 GLEventListener，透過這兩種元件，讓使用者能簡單的將 OpenGL 套入一視窗應用程式中，利用 GLEventListener 內定義之特殊的回呼函式(Callback Function)，開發者便能在不同事件發生時，依不同的函式來更新目前視窗裡的畫面。

CWT 的設計是以原生容器當作最底層的畫布，在其上繪製輕型元件，而在 CWT-GL 的實作中，將改成以 GLCanvas 作為最底層之畫布，所有的輕型元件，透過 GLCanvas 所提供之 OpenGL 物件進行繪製。因此 CWT-GL 實作步驟為：先生成一 GLCanvas 物件，將此物件加入原本的原生容器中，故在 CWT-GL 中的 GlWindowContext 除負責存放對應的原生容器外，也負責生成 GLCanvas 物件，並將此物件的大小調整成與原生容器一樣，完全覆蓋在原生容器之上。

第三章、系統實作

本章將會詳細說明，如何利用 JOGL 套件直接使用 OpenGL 繪圖函式庫的功能，完成 CWT-GL 中各項繪圖功能。首先 3.1 節說明如何利用 OpenGL 一些基本原理來完成 CWT-GL 裡面的所有繪圖操作。接著 3.2 節將介紹 CWT 提供的背景繪圖(Offscreen Rendering)。最後 3.3 節會討論各種改善系統效能的方式。

3.1 繪圖功能實作方式

OpenGL 繪圖函式庫在 1992 年由 Silicon Graphics Inc. (SGI)所提出，是一個跨平臺並被業界視為標準的三維繪圖函式庫[31]。許多著名的遊戲都有使用 OpenGL 函式庫，例如 id Software 開發的毀滅戰士三以及雷神之鎚四。[6]

因為 OpenGL 本為三維繪圖函式庫，故很多函式及功能都是以三維的觀點來設計，而 CWT 因參考 AWT 之架構，所以主要還是以二維的繪圖為主。因此，要利用三維運算來模擬二維，有一些要點要注意，例如投影方式的選擇。

如圖 3-1 所示，OpenGL 繪圖函式庫中，提供兩種投影的方式，其一稱作透視投影(Perspective Projection)，另一種則為垂直投影(Orthographic Projection)。

1. 透視投影

   會使距離觀察者較遠之物體變得比原本尺寸小，若是相同大小之兩物體被放在離觀察者不同遠近的兩個位置時，經過透視投影後，較遠的物體看起來會比近的小。透視投影比較有立體感，也較貼近真實，因為這與攝影的鏡頭以及人眼的作用類似。

2. 垂直投影

   與透視投影不同，物體距離觀察者的遠近，並不會影響到投影

後的尺寸。這種投影方式多用於工業設計軟體上，因為要求精確地將物體各種重要性質真實呈現，不會因為投影使結果有變形或不符實際的狀況。

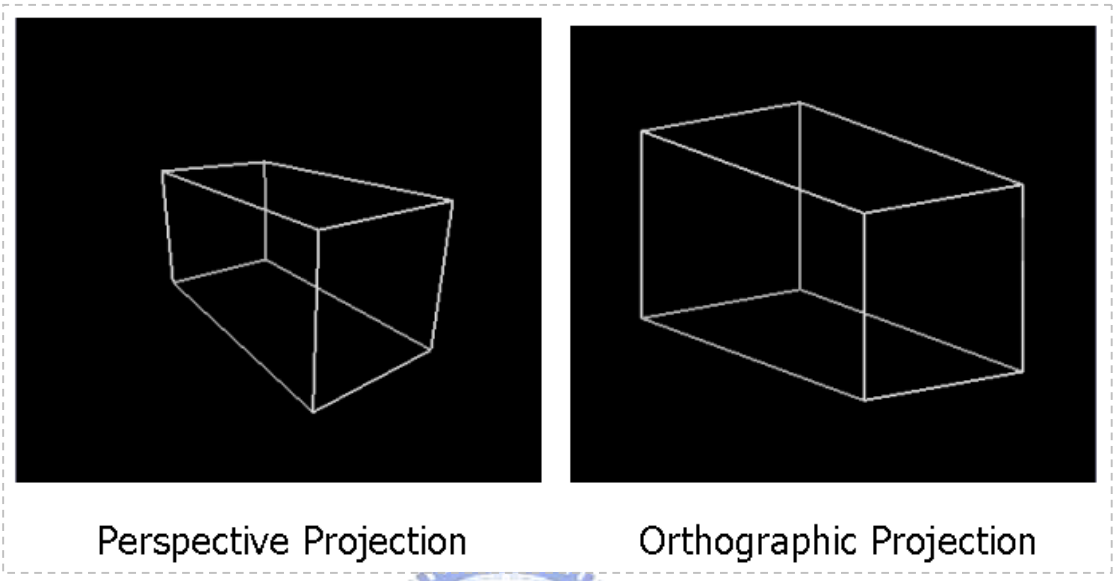

圖 3-1：透視投影與垂直投影

繪圖指令可分成三種，幾何圖形的繪製、圖片的繪製、字串的繪製。接著將分別介紹如何以 OpenGL 繪圖函式庫實作各種繪圖功能。

1. 幾何圖形的繪製

   圖 3-2 為 AWT 中所有的幾何圖形繪製指令，OpenGL 最基本的繪圖元素為點、直線以及三角形，因此若要以 OpenGL 繪製出這些基本圖形，得以一連串之直線或三角形組合出各種不同的圖形。其中無法直接透過線段或三角形組合而成的，就以逼近的方式來實作，例如圓弧、橢圓、扇形等等。

圖 3-2：AWT 中基本的幾何圖形[25]

圖 3-3 為以連續的線段模擬圓形，利用角度與圓的半徑，可求出圓周上任意位置的點座標。而應該以多少角度的間距逼近，從圖中可以發現，當以十五度為間距逼近時，人眼已經感覺不出其為一有稜有角的多邊形，而會將它當作是一個平滑的圓形。對填滿的圓則利用類似的方式，只是基本元素由線段改為填滿的三角形。

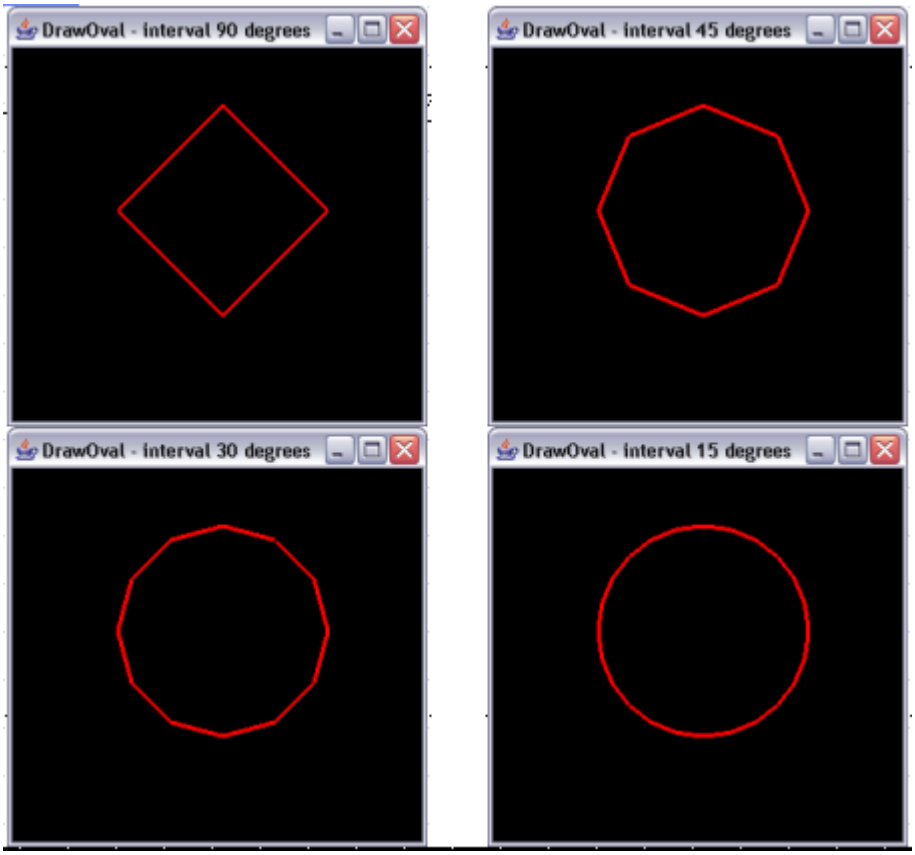

圖 3-3：用不同數量的線段模擬圓

2. 圖片的繪製

為了能受惠於硬體加速，繪製圖片的指令將以材質貼圖(Texturing)的方式實作，將一張張需要繪製的圖片，先載入至顯示卡的記憶體中，產生成一張張的材質，當要繪製該圖片時，如圖 3-4 所示，將對應的材質(Texture)貼到與圖片大小相同的矩形(Rectangle)上，達到圖片繪製的效果。

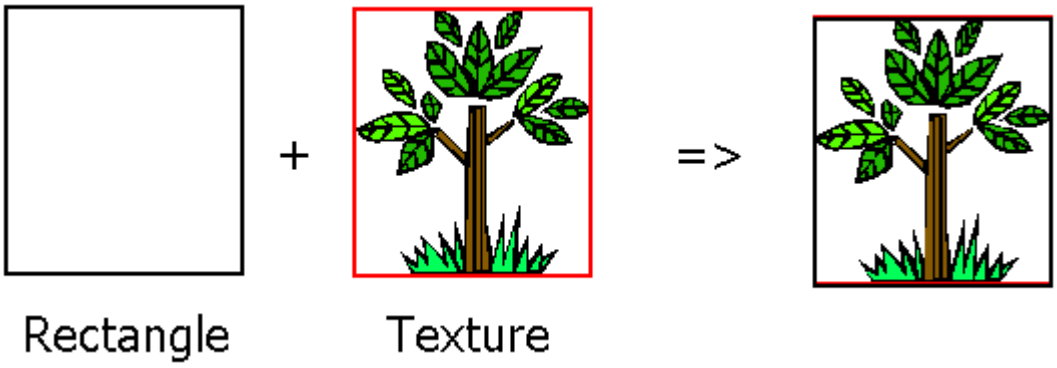

圖 3-4：用材質貼圖來達成圖片繪製

但是 OpenGL 繪圖函式庫在產生材質時有些限制，材質的長與寬必須為二的冪次，若是原始圖片(Source image)之長寬並非二的冪次，就必須如圖 3-5 所示，要事先將圖片的長與寬放大至二的冪次，此時若還是以一比一的比例進行貼圖，因材質(Texture)大小已較原始圖片大，故圖片繪製結果會縮小。因此在貼圖時，長與寬並非二的冪次之情況必須計算對應的材質區塊，避免將整張尺寸已變大之材質以一比一之比例貼上。不過 OpenGL 的擴充功能可以支援長寬並非二的冪次之材質，但此功能在較新的顯示卡上才有支援。

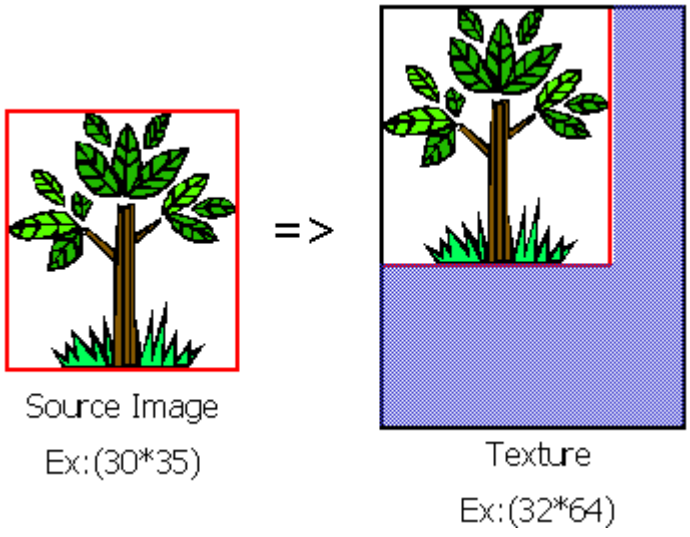

圖 3-5：材質的長與寬必須是二的冪次

3. 字串的繪製

OpenGL 本身的 API 並沒有提供直接繪製字串之功能，因此字串的繪製方式有三種：位元映射圖(Bitmap)、材質貼圖(Texture)以及輪廓繪製(Outline)。圖 3-6 為三種不同繪製方式所展現出來的不同效果。

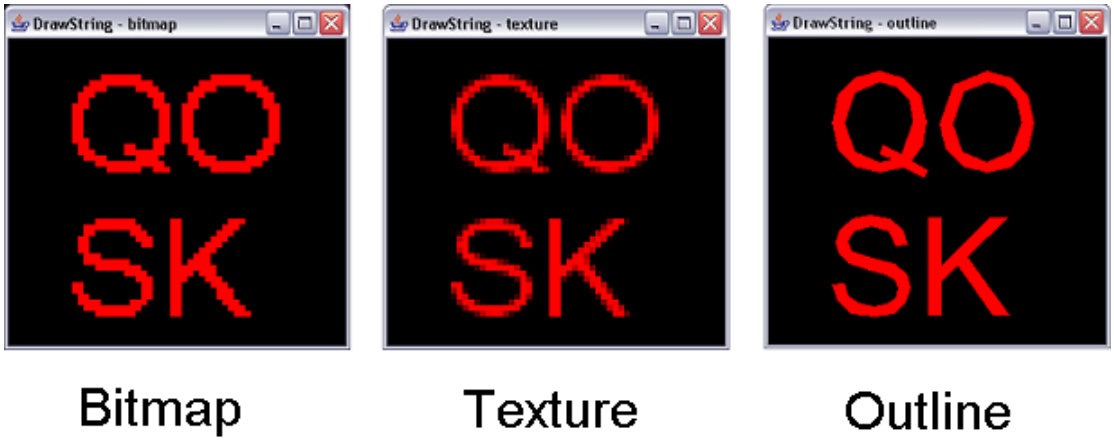

圖 3-6：字串繪製的三種方式

位元映射圖儲存了文字的像素(pixel)資料，每個像素以一位元儲存。材質貼圖為將每個字都先以圖片的方式繪製出來，最後在繪製字串時，將每個字都以材質的方式繪製到對應的位置。這種方式能夠擁有較好的效果，因為可以在繪製成圖片時，選用較好的解析度甚至先作反鋸齒效果後，再以材質貼圖的方式繪製。然而這兩種方式，在直接放大來繪製字型點數大的字串時，會使得繪製的品質降低，如圖 3-6 Bitmap 與 Texture 兩張圖所示，在邊緣的地方將會看到鋸齒狀出現。而材質貼圖比位元遮罩好的原因在於其受惠於硬體加速，因此繪圖效能比較好。

輪廓繪製是將每個字的輪廓以線段的方式組合成一個或數個多邊形，如圖 3-6 Outline 此圖所示，因這種方式將字以幾何圖形呈現，所以在繪製尺寸較大的字時不會有品質變差的情況產生，還能夠應用於三維立體的字，或是其他幾何圖形能擁有的效果，例如光影。但當繪製筆畫較複雜的文字時，例如中文字，

因為要處理的多邊形變多，使得繪製的效能會因此下降。

本論文考慮以上各種方式的優缺點後，採用材質貼圖與輪廓繪製兩種方式混合使用。當字型大小大於 16 點時，因圖片耗用記憶體的考量，我們採用輪廓繪製的方式。而字型大小在 16 點以下時，則選用材質貼圖的方式，但為改善材質貼圖繪字時的缺點，將動態產生不同大小之材質，用作不同大小字體的繪製。為減少重覆產生材質所造成的影響，我們快取(Cache)了已經產生過的材質，未來若要繪製同樣的字時，就不必再重新產生同樣的材質，對於效能上將會有較好的提升。

3.2 背景繪圖

繪圖指令通常直接修改螢幕顯示記憶體(Frame buffer)，也就是所有修改會立即顯示於螢幕上，稱作前景繪圖(Onscreen Rendering)。如果將繪製的目標由螢幕顯示記憶體換為另一個不會馬上顯示出來的地方，例如自己產生的暫存區(Back buffer)，稱做背景繪圖(Offscreen Rendering)。圖 3-7 以簡單的圖示說明兩種繪圖方法。背景繪圖通常用來生成中介的材質，或是繪製電腦動畫所需的一連串圖片，及作為背景暫存區以避免畫面更新時造成的閃爍。

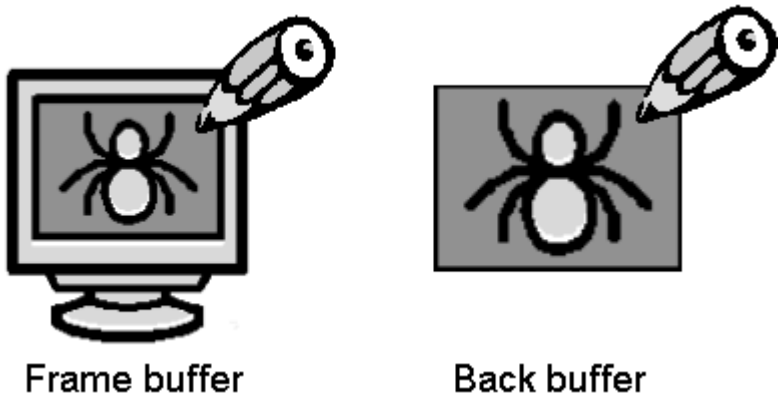

圖 3-7：前景繪圖與背景繪圖的簡單圖示

OpenGL 繪圖函式庫中，有三種方式可以完成背景繪圖。Pixel

Storage、Pixel Buffer(簡稱 Pbuffer)[30]與 Framebuffer Object(簡稱 FBO)[29]。雖然大部分的顯示卡都有支援 Pixel Storage，但由於它是比較早期的方法，實作為透過 API 對背景的 Frame buffer 進行存取，這樣會使得資料必須從顯示卡的記憶體搬移至主記憶體中，因此效能表現上並不佳。Pbuffer 則為之後提出的，與原本繪圖系統的前景緩衝區架構類似，因此會擁有許多相同的資料結構，所以實作上會浪費不少空間，並且在使用上也較不方便。最後，OpenGL 在 2003 年提出新的技術 FBO，不管在實作或是使用上都作了很大的改進。故本論文在 CWT 架構中內建的背景繪圖乃是利用 FBO 來實作，提供一效能不錯的背景繪圖。當無法使用 FBO 時，則改用 Pbuffer 來處理背景繪圖，若是連 Pbuffer 都無法使用，則直接使用原本的 Java AWT 來處理，最後再以材質貼圖的方式來繪製。

3.3 改善效能的方式

本節介紹各種在本論文中改善效能的方式，3.3.1 節討論有關 OpenGL 中各項狀態設定與效能的關係，要達到高效能，就必須避免各種不必要的測試以及選用計算量較少的狀態設定。3.3.2 節介紹單一執行緒繪圖(Single-Threaded Rendering，簡稱 STR)[24][33]，以及在 CWT 中如何實作此一部分。3.3.3 節提出應該減少 OpenGL 狀態的改變，例如目前使用的材質，顏色，投影矩陣等等[33]。

3.3.1 避免不必要的測試與計算

OpenGL 繪圖函式庫本身在底層存放有各式各樣不同的狀態，舉凡繪圖的顏色、背景的底色、投影矩陣、目前啟用的材質或者是混色功能、深度測試等等的啟用與否，充滿各式各樣的狀態參數可以設定，但並非每一種狀態，每一項運算在應用程式中都是必須的。適當關閉各項不必要的運算，或選用運算較少的參數，都能提升執行時的繪圖效能[2]。例如：在繪製不透明圖時，可以關閉透明值測試以及混

色功能，或連續繪製圖片時，避免重複開啟與關閉貼圖功能等。除了關閉不必要的功能，也能透過選用運算較少的操作，例如：材質貼圖時，可選用利用內插的方式求出並非有一對一對應的像素之顏色，或直接以最近的對應點顏色作為該像素之色彩值，後者的運算自然較少。這些方式都能改善執行時的效能。

3.3.2 單一執行緒繪圖

OpenGL 原是設計以單一執行緒(Thread)執行繪圖指令的函式庫，若要使用多執行緒，必須非常注意執行緒切換的同步問題，否則可能會出現繪製不正常或甚至當機情況。而 OpenGL 驅動程式，通常也會為單一執行緒特別做最佳化。Java 在 J2SE 1.5 提出 OpenGL 的繪圖管線，未特別處理此問題，然而 Java AWT 本身允許在多執行緒環境下執行，導致開啟 OpenGL 繪圖管線後，產生許多問題。為改善許多執行時會發生的錯誤，在 J2SE 1.6 的版本實作單一執行緒繪圖(Single-Threaded Rendering)的機制[24]。因 CWT-GL 底層也使用 OpenGL 繪圖函式庫，故本論文也在 CWT 的架構中，加入了單一執行緒繪圖的機制，除了能避免 OpenGL 可能會發生的錯誤外，也因單一執行緒繪圖此架構，可以提昇效能[33]。

圖 3-8 所示描述 CWT 使用的單一執行緒繪圖機制[33]。各種不同的執行緒發出繪圖指令(Rendering Operation)，全部都先存放於一個佇列(Rendering Operation Queue)中，再由另一支獨立的執行緒負責處理這些繪圖指令，將其送到 CWT-GL 實作中的 GlWindowContext 物件裡的 GLCanvas 物件，即圖中的原生繪圖函式庫(Native Graphics Libraries)，由 GLCanvas 做最終繪製指令的處理。透過這樣的方式，除避免掉每送出一次繪圖指令就得直接存取到原生介面外，對於在佇列中的各式各樣不同的繪圖指令，還能做批次(Batch)處理，將相同類型的指令盡量集合在一起處理，例如在 3.3.3 小節中介紹的最佳化方式。因為 OpenGL 各種文件[11]都會建議將相同的指令一次處理完畢，對於效能上會有比較好的表現。

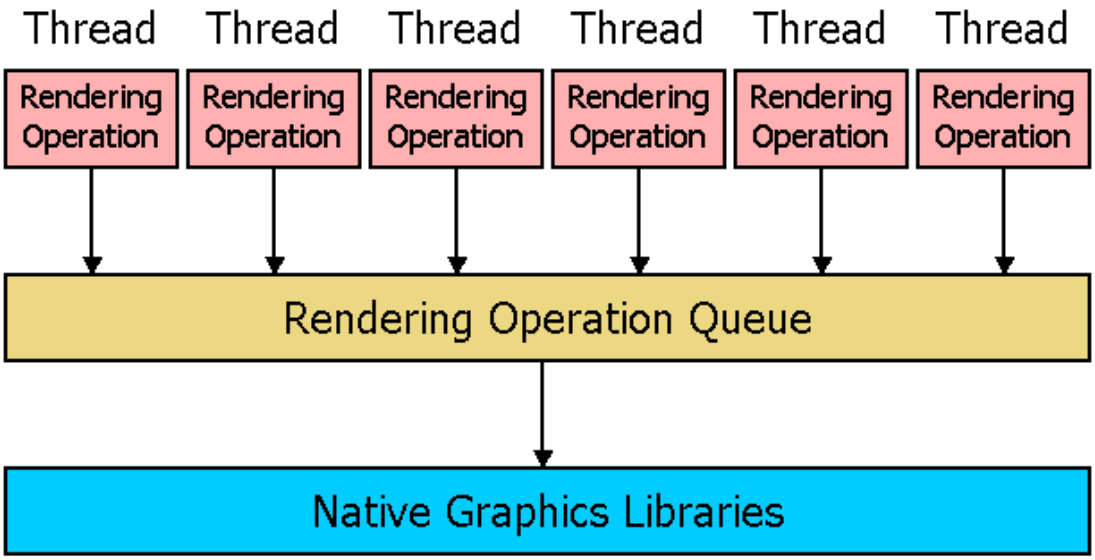

圖 3-8：單一執行緒繪圖架構

3.3.3 減少狀態改變

處理某些繪圖指令時，通常需要設定許多相關的狀態，例如目前使用的材質、顏色、投影矩陣、裁切平面等等。不過，當處理一連串的繪圖指令時，有些狀態是不必重複設定的，減少這些多餘的狀態設定，對效能會有相當程度的改進。接著以本論文實作中對效能提升最有幫助的兩項做說明[33]。

1. 減少重複綁定(Bind)相同材質：

   遊戲軟體為呈現獨特的使用者介面或畫面都會使用自行繪製的圖片，所以在繪製遊戲畫面時，必須大量不斷地執行繪製圖片的指令，因此，底層必須不斷切換目前使用的材質。而切換材質是相當費時的運算，若是繪製一連串的圖片時，能減少材質間的切換，就能讓效能有顯著的提升。

   以圖 3-9 為例，若要連續繪製三張麻將牌，每繪製一張就設定一次目前使用的材質，將需要三次的材質綁定。但因為是連續繪製同樣的圖片三次，只要設定一次的材質綁定，接下來的圖片繪製

就能直接省略綁定的步驟。

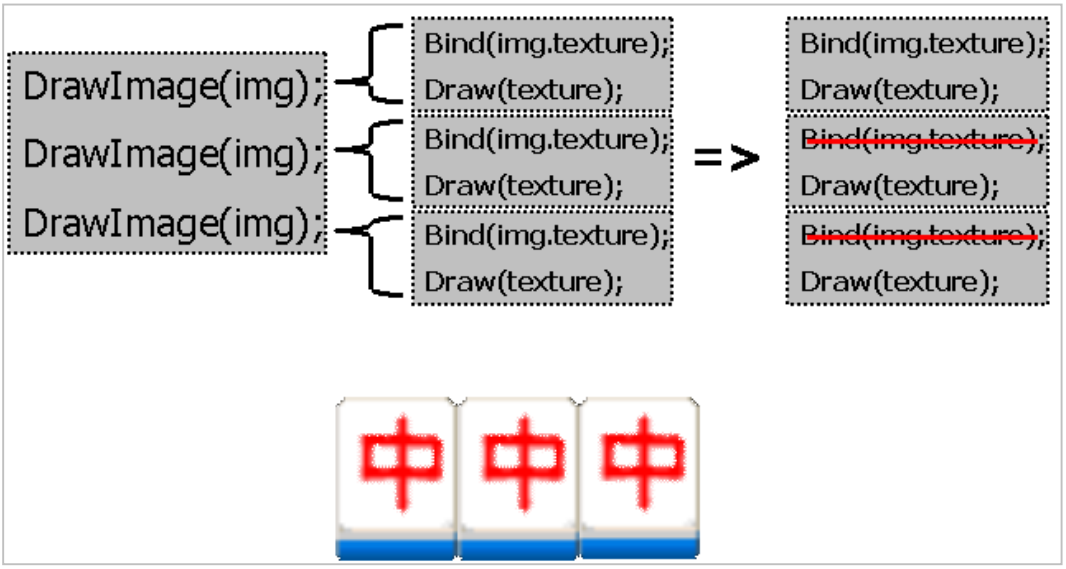

圖 3-9：繪製圖片時，減少材質間的切換次數

2. 減少 glBegin/glEnd 的數量：

在 OpenGL 中，glBegin 與 glEnd 兩個函式的組合負責將場景中的點座標位置送至底層顯示卡，讓硬體利用這些點座標繪製出對應的畫面。以圖 3-10 為例，繪製一矩形時，必須要有一對的 glBegin 與 glEnd 函式來將四個點的座標送入顯示卡。

圖 3-10：繪製矩形，使用一組 glBegin 與 glEnd 函式

而圖 3-11 表示連續繪製三個矩形時，可以利用連續三組的 glBegin 與 glEnd 之函式呼叫傳送三個矩形個別的頂點座標，或只使用一組的函式，連續傳送 16 個點座標至顯示卡。

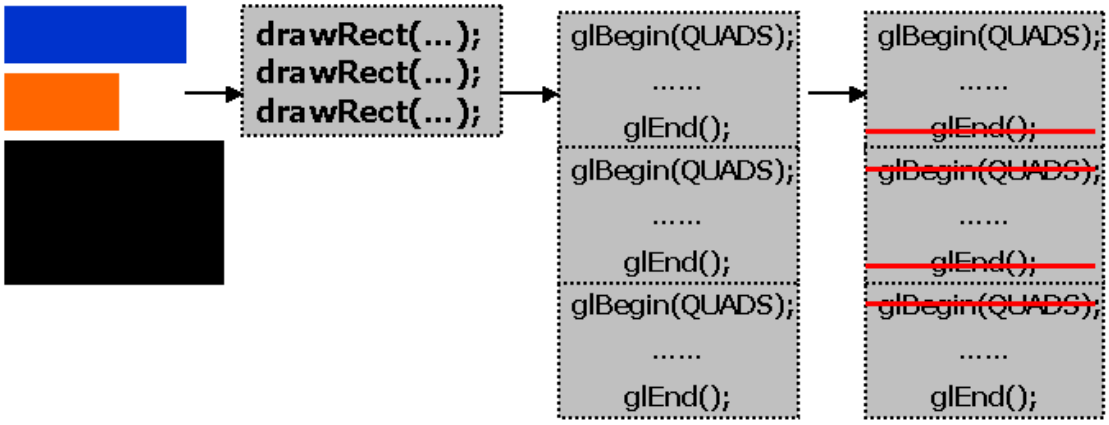

圖 3-11：繪製三個矩形時的兩種方式

只使用一組函式呼叫，效能上的表現會比連續三組的方式好，因 glBegin 會引發 OpenGL 內部檢驗狀態的動作，若要繪製的圖形為連續且只需要一組 glBegin 與 glEnd 函式就能完成，此時應該避免採用連續多組的函式呼叫來實作。

第四章、實驗數據分析與討論

本章將於 4.1 節說明本論文中使用的各種實驗環境，及各項實驗項目，並在 4.2 節分析數據與討論。

4.1 實驗環境

表 4-1 與表 4-2 為本論文實驗用之兩台機器的硬體規格，一台為個人桌上型電腦，另一台則是蘋果電腦生產的 iMac，為了使實驗結果一致，此兩台機器之規格必須盡量雷同。本論文將在兩台機器上建立四種不同的執行平臺，如表 4-3，將在這四種平臺上個別實驗。

CPU	AMD Opteron 144 (1.8GHz)
RAM	1 GB DDR400
Graphics Card	ATI Radeon x1650 256 MB VRAM • DirectX 9.0 and OpenGL 2.0 supports • Driver version: • Windows: Catalyst™ 7.5 Display Driver • ATI Proprietary Linux x86 Display Driver 8.37.6 • Resolution: 1280x1024@60 32bit

表 4-1：個人桌上型電腦硬體配備

CPU	Intel Core 2 Duo (2.0GHz)
RAM	1 GB DDR2 667
Graphics Card	ATI Mobility Radeon x1600 128 MB VRAM • DirectX 9.0 and OpenGL 2.0 supports • Resolution: 1440x900@60 32bit

表 4-2：蘋果電腦 iMac 硬體配備

WinXP	1. Windows XP Professional(SP2) on PC
WinVista	2. Windows Vista Business on PC
Fedora	3. Fedora Core 6 on PC
MacOS	4. MacOS 10.4.8 on iMac

表 4-3：四個實驗平臺

本論文使用之兩個測試程式，與[32][33][34]相同。第一個測試程式為執行一動畫程式，量測繪製基本圖形的效能。第二個測試為真正遊戲之繪圖效能（四川麻將，於 4.2.5 節描述）。藉此比較各平臺間繪圖效能的差異。此動畫程式會選擇一種繪製內容，總共繪製 20000 次，視窗的大小為 600*300，單位為像素(pixel)。實驗之繪製內容分為以下幾種：

1. 圖片：透明圖、不透明圖與在執行時期做鏡像反射後的圖
   圖片的解析度均為 110*110，單位為像素。

2. 文字：固定文字與一段文章
   文字大小均為 12 點，固定文字為 Running，而文章中均為中文字。

3. 幾何圖形的填滿：矩形與圓形
   矩形的大小為 110*100，圓的直徑為 110，單位均為像素。

4.2 實驗數據分析

本節將個別分析與討論 4.1 節所提之四個實驗平臺之數據，先於 4.2.1 節以 Windows XP 上之數據分析 AWT 與 CWT-GL 效能的優缺點，分為三個部份比較：圖片、文字以及幾何圖形。接著在 4.2.2、4.2.3 及 4.2.4 節分別比較其他平臺上 AWT 與 CWT-GL 的效能。最後於 4.2.5 節討論將 CWT-GL 應用於 CYC 遊戲大聯盟開發之四川麻將後，不同平臺及不同 Java 版本下之繪圖效能。

本節表格中以 JVM 簡稱 Java 虛擬機器(Java Virtual Machine)，並在其之後註明所使用的版本，例如 JVM 1.4.2_14 表示為 J2SE 1.4.2_14 此版本。Managed Image 為 J2SE 1.4 中改善繪圖效能的一項機制[4]。括號中如 sun.java2d.translaccel=true 或 sun.java2d.opengl=true 則表示開啟半透明度之加速與啟用 OpenGL 繪圖管線之實驗。

4.2.1 Windows XP 上 AWT 與 CWT-GL 之比較

這一節以 Windows XP 上的數據對 AWT 與 CWT-GL 做比較，分為圖片、文字以及幾何圖形。

1. 圖片：透明圖、不透明圖與鏡像反射圖
   實驗結果如表 4-4，AWT 的部份分為是否使用 Managed Image 此新版本提供的機制，以及是否開啟 J2SE 1.5 提出之 OpenGL 繪圖管線。從表格中可以發現，透明圖、不透明圖及鏡像反射圖三項實驗，CWT-GL 的表現為所有環境中最好的，即使最新版本的 Java，CWT-GL 的效能也約比 AWT 快了 27%到 87%。在執行時期所作的鏡像反射圖的部份，舊版本之效能與 CWT-GL 的差距則相當大，大部分的環境下效能都較 CWT-GL 慢約 800%，不過由表中 AWT 的數據可發現，以 JVM1.6.0_01 為例，使用 Managed Image 及開啟半透明加速後，這部份的效能提昇約 344%。

2. 文字：固定文字與文章
   實驗結果如表 4-4，AWT 在沒有使用新 API 時，效能反而較佳，原因為 J2SE 1.4 時提出，用作加速繪圖效能之 VolatileImage 類別並不支援字串的繪製。由數據得知在固定字串或一段文章繪製實驗中，CWT-GL 的表現並不算是最好，如在 JVM 1.5.0_11 下，使用舊 API 之情形下，效能約比 CWT-GL 快 50%甚至是 200%。

3. 幾何圖形：矩形與圓形

實驗結果如表 4-4，矩形的部份，AWT 使用 Managed Image 並開啟正確的設定後，JVM1.6.0_01 之效能比 CWT-GL 約快 13%，但其他版本則約慢 50%甚至超過 300%。圓的繪製因採用線段逼近之方式，故 CWT-GL 的效能為所有環境中最好，即使 JVM1.6.0_01 也較 CWT-GL 慢 100%以上。

Library	Virtual Machines	Trans. Images	Opaque Images	Mirrored Images	Simple Strings	Article	Rec-tangles	Circles
AWT	Image							
	MSVM 1.1.4	7141	2655	7139	46	131	70	2071
	JVM 1.4.2_14	407	392	992	72	686	170	543
	JVM 1.5.0_11	407	392	982	38	56	170	557
	JVM 1.6.0_01	417	448	818	35	55	161	265
	Managed Image							
	JVM 1.4.2_14	131	154	1011	357	986	120	2159
	JVM 1.5.0_11	131	148	992	148	162	396	2141
	JVM 1.6.0_01	131	148	831	149	166	396	2144
	Managed Image (sun.java2d.translaccel=true)							
	JVM 1.4.2_14	131	155	238	375	1002	120	2160
	JVM 1.5.0_11	131	148	1067	155	163	396	2141
	JVM 1.6.0_01	187	187	187	65	145	71	537
	Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	353	353	3010	215	219	134	664
	Managed Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	353	353	353	215	219	134	661
CWT	CWT OpenGL							
	JVM 1.4.2_14	103	144	102	118	136	81	120
	JVM 1.5.0_11	102	144	102	116	85	83	121
	JVM 1.6.0_01	100	140	100	103	76	81	101
	CWT DirectX							
	MSVM 1.1.4 (DX3)	154	155	155	70	172	397	2287

表 4-4：Windows XP 上 AWT 與 CWT-GL 的比較

4.2.2 Windows Vista 上 AWT 與 CWT-GL 之比較

本節將討論 CWT-GL 在 Windows Vista 上之效能。與上節相同，數據將以圖片、文字以及幾何圖形三部份作分析，並比較 CWT-GL 與 AWT 之繪圖效能。表 4-5 為 Windows Vista 上的實驗結果。

Library	Virtual Machines	Trans. Images	Opaque Images	Mirrored Images	Simple Strings	Article	Rec-tangles	Circles
AWT	Image							
	MSVM 1.1.4	2533	658	2536	101	199	164	246
	JVM 1.4.2_14	409	390	990	73	1446	172	561
	JVM 1.5.0_11	415	395	997	39	57	173	583
	JVM 1.6.0_01	426	453	827	34	55	165	274
	Managed Image							
	JVM 1.4.2_14	123	146	1023	613	2031	110	4076
	JVM 1.5.0_11	414	395	998	40	59	173	568
	JVM 1.6.0_01	425	453	826	35	56	164	277
	Managed Image (sun.java2d.translaccel=true)							
	JVM 1.4.2_14	123	146	343	645	2045	110	4089
	JVM 1.5.0_11	414	396	997	40	59	174	565
	JVM 1.6.0_01	427	449	816	34	55	162	266
	Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	570	582	19600	637	792	478	1132
	Managed Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	558	568	558	623	775	470	1126
CWT	CWT OpenGL							
	JVM 1.4.2_14	106	151	105	137	148	100	137
	JVM 1.5.0_11	100	146	100	126	96	97	134
	JVM 1.6.0_01	96	143	96	110	83	94	111
	CWT DirectX							
	MSVM 1.1.4 (DX3)	84	97	84	91	193	340	843

表 4-5：Windows Vista 上 AWT 與 CWT-GL 的比較

1. 圖片：

與 Windows XP 上之數據類似，雖然 JVM1.4.2_14 使用 Managed Image 時，不透明圖的效能與 CWT-GL 幾乎一樣，但整體上，CWT-GL 在圖片的三項實驗中，效能仍是最好，除上述所提之 JVM1.4.2_14 外，CWT-GL 都較其他狀況下 AWT 的效能快 200%以上。

2. 文字：
   AWT 使用舊 API 時效能較好的問題，在 Windows Vista 上較少，除 JVM1.5.0_11 開啟 OpenGL 繪圖管線後，繪製文字的效能大幅降低外，其他版本並無 Windows XP 上之問題。效能上，CWT-GL 仍較 AWT 某些情形慢約 30%至 300%，如 JVM1.5.0_11 無啟用 OpenGL 繪圖管線之環境。

3. 幾何圖形：
   CWT-GL 於矩形與圓形的繪製都為所有環境下最快，即使 AWT 於 JVM1.6.0_01 執行，效能仍慢 60%至 100%。

基本上，CWT-GL 於 Windows Vista 的效能表現與 Windows XP 上類似，但 AWT 各項實驗結果則與 Windows XP 上不同，例如鏡像反射圖之實驗，於 Windows XP 上為 JVM1.6.0_01 使用 Managed Image 之下效能最好，而在 Windows Vista 上，反而於 JVM 1.4.2_14 使用 Managed Image 之下效能較好，如此效能不一致之情形也出現於其他平臺上。

4.2.3 Fedora 上 AWT 與 CWT-GL 之比較

表 4-6 為 Fedora 上之實驗數據，因 MSVM 只能於微軟視窗作業系統上使用，故 Fedora 上無 MSVM 之實驗結果。

1. 圖片：
   此部分仍是 CWT-GL 擁有最好的繪圖效能，透明與不透明圖的效

能約比 AWT 快 30%至 300%。鏡像反射圖的實驗中，AWT 只有 JVM 1.5.0_11 使用 Managed Image 及開啟 OpenGL 繪圖管線這項實驗的效能與 CWT-GL 差距較小，約慢了 90%，其它的狀況下 CWT-GL 均快了 900%以上。而未開啟 OpenGL 繪圖管線的環境，與 Windows XP 及 Windows Vista 上之結果不同，可以發現於 Fedora 上，使用 Managed Image 與否對效能並沒有明顯影響，原因是 Managed Image 為透過 DirectX 改善圖片繪製效能，但 Fedora 上無法使用 DirectX，因此並無受惠硬體加速，故效能沒有影響。

2. 文字：

   Fedora 上無舊版 API 效能較好之問題，且與圖片之實驗結果相同，使用 Managed Image 與否並不影響效能。CWT-GL 仍較 AWT 某些情形慢約 40%至 400%以上，如 JVM1.5.0_11 無啟用 OpenGL 繪圖管線之環境。

3. 幾何圖形：

   除 JVM1.5.0_11 開啟 OpenGL 繪圖管線此環境下之矩形實驗外，CWT-GL 矩形與圓形之繪圖效能都為所有環境下最快，均約快 100%甚至 100%以上。

   CWT-GL 之實驗數據除字串繪製略慢外，其他數據則與其他平臺類似。AWT 整體上以 JVM 1.5.0_11，開啟 OpenGL 繪圖管線且使用 Managed Image 為最快，此點又與 Windows XP 及 Windows Vista 之實驗結果不同。

Library	Virtual Machines	Trans. Images	Opaque Images	Mirrored Images	Simple Strings	Article	Rec-Tangles	Circles
AWT	Image							
	MSVM 1.1.4	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	JVM 1.4.2_14	425	461	2408	107	832	237	629
	JVM 1.5.0_11	394	507	1449	40	57	169	536
	JVM 1.6.0_01	389	501	1435	36	53	166	398
	Managed Image							
	JVM 1.4.2_14	426	461	2480	108	855	237	633
	JVM 1.5.0_11	395	507	1442	39	55	168	533
	JVM 1.6.0_01	390	502	1434	36	53	166	391
	Managed Image (sun.java2d.translaccel=true)							
	JVM 1.4.2_14	424	461	2479	106	825	237	633
	JVM 1.5.0_11	394	507	1443	40	56	169	529
	JVM 1.6.0_01	390	502	1435	36	53	166	390
	Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	193	193	5242	177	310	76	861
	Managed Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	194	194	194	175	308	76	820
CWT	CWT OpenGL							
	JVM 1.4.2_14	107	148	106	163	161	84	158
	JVM 1.5.0_11	106	147	105	145	104	84	162
	JVM 1.6.0_01	105	146	103	124	92	82	129
	CWT DirectX							
	MSVM 1.1.4 (DX3)	N/A	N/A	N/A	N/A	N/A	N/A	N/A

表 4-6：Fedora 上 AWT 與 CWT-GL 的比較

4.2.4 MacOS 上 AWT 與 CWT-GL 之比較

表 4-7 為 MacOS 上之實驗數據。因蘋果電腦尚未於 MacOS 上釋出 J2SE 1.6 正式版本，故表格內無 J2SE 1.6 之數據。MSVM 之數據與 Fedora 上理由相同。

1. 圖片：

此部分仍是 CWT-GL 效能最好。透明圖與不透明圖之實驗，CWT-GL 之效能較 AWT 快約 125%至 200%。鏡像反射圖則快約 250%至 700%以上。AWT 在同樣版本下，除 JVM1.4.2_14 之鏡像反射圖實驗外，彼此的效能差異不受參數設定或 API 使用影響。

Library	Virtual Machines	Trans. Images	Opaque Images	Mirrored Images	Simple Strings	Article	Rec-tangles	Circles
AWT	Image							
	MSVM 1.1.4	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	JVM 1.4.2_14	380	416	2448	196	428	69	608
	JVM 1.5.0_11	283	239	961	46	69	149	364
	JVM 1.6.0_01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Managed Image							
	JVM 1.4.2_14	380	416	440	198	428	69	605
	JVM 1.5.0_11	284	240	980	46	69	149	364
	JVM 1.6.0_01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Managed Image (sun.java2d.translaccel=true)							
	JVM 1.4.2_14	380	416	442	197	430	69	605
	JVM 1.5.0_11	284	240	981	46	69	149	358
	JVM 1.6.0_01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	283	239	961	46	69	149	365
	Managed Image (sun.java2d.opengl=true)							
	JVM 1.5.0_11	284	239	980	46	69	149	359
CWT	CWT OpenGL							
	JVM 1.4.2_14	126	156	126	182	263	113	117
	JVM 1.5.0_11	122	153	122	141	115	109	107
	JVM 1.6.0_01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CWT DirectX							
	MSVM 1.1.4 (DX3)	N/A	N/A	N/A	N/A	N/A	N/A	N/A

表 4-7：MacOS 上 AWT 與 CWT-GL 的比較

2. 文字：

與 Fedora 上之測試數據相似，使用舊版本 API 的實驗數據，並沒

有較新 API 好，此情形與 Windows XP 以及 Windows Vista 不同。CWT-GL 繪製文字的效能仍較 AWT 某些環境慢，如各項關於 JVM1.5.0_11 之實驗，約慢 60%至 280%。

3. 幾何圖形：
   除 JVM1.4.2_14 下之矩形實驗外，CWT-GL 矩形與圓形之繪圖效能都為所有環境下最快，均約快 30%甚至 300%以上。

MacOS 上之數據，CWT-GL 文字繪製之效能仍較 AWT 某些版本慢，但 CWT-GL 的各項繪圖效能仍相似於其他平臺上之數據，反觀 AWT，各版本間各種繪圖效能的差異仍相當明顯，且其他平臺上效能較好之版本，如 Windows XP 上之 JVM1.6.0_01 使用 Managed Image，或 Windows Vista 上之 JVM 1.4.2_14 使用 Managed Image，於 MacOS 上並非為效能最好之版本。

4.2.5 實際應用於已開發之遊戲軟體

圖 4-1 為將 CWT-GL 應用於已開發遊戲上之畫面，此畫面為四川麻將執行畫面，如圖所示每次畫面更新除繪製一些文字及幾何圖形外，最大量的繪圖運算為繪製圖片，整個畫面均由各種不同之圖片所組成，畫面中央更有 144 張麻將牌。

表 4-8 為此遊戲於不同作業系統下之實驗數據。綜觀所有平臺與各種版本的實驗數據，AWT 的繪圖效能，均較 CWT-GL 慢約 20%以上。而只有在 Windows XP 上，使用 Managed Image 並開啟半透明加速之 JVM 1.6.0_01 實驗數據，效能才較 CWT-GL 好約 24%。

圖 4-1：四川麻將的執行畫面

無論是 4.2.1 至 4.2.4 節中的實驗數據，或本節應用至已開發遊戲上之實驗結果，AWT 在不同版本間，或使用不同之 API 與參數設定，於效能有明顯差異，以表 4-8 為例，Windows XP 上以 JVM 1.6 使用 Managed Image 並開啟半透明加速之環境效能最好，但 Fedora 上，此環境之效能並非為最好。反觀 CWT-GL，繪圖效能於各平臺上及各版本間是穩定的，能夠預期不會因使用者安裝之 JVM 不同而有效能上之巨大差異，對開發者來說，效能上的預期是較可控制的。

Virtual Machines	WinXP	WinVista	Fedora	MacOS
Image				
MSVM 1.1.4	178446	131053	N/A	N/A
JVM 1.4.2_14	28037	38428	65798	47756
JVM 1.5.0_11	25194	28158	58079	21888
JVM 1.6.0_01	24622	27301	56767	N/A
Managed Image				
JVM 1.4.2_14	30668	58914	63398	95178
JVM 1.5.0_11	27701	27812	58172	21807
JVM 1.6.0_01	27667	27390	57161	N/A
Managed Image (sun.java2d.translaccel=true)				
JVM 1.4.2_14	33460	40075	63038	94109
JVM 1.5.0_11	23564	27714	57595	21793
JVM 1.6.0_01	9748	27376	56967	N/A
Image (sun.java2d.opengl=true)				
JVM 1.5.0_11	70057	165378	118091	21907
Managed Image (sun.java2d.opengl=true)				
JVM 1.5.0_11	31219	98900	35542	21740
CWT OpenGL				
JVM 1.4.2_14	12088	19635	15242	14258
JVM 1.5.0_11	12123	18976	14188	12242
JVM 1.6.0_01	11631	18256	14147	N/A
CWT DirectX				
MSVM 1.1.4 (DX3)	21122	25565	N/A	N/A

表 4-8：不同作業系統中，四川麻將的執行效能

第五章、結論與未來展望

CWT[32][33][34]的提出， 目標為提升 Java 繪圖效能，並讓現有之 Java 應用程式能直接受惠 CWT 內部實作所使用到之各種繪圖函式庫。而本論文完成實作 CWT 中 CWT-GL 部份，使 CWT 此系統除原本特性外，更加強在各種不同作業系統上，能擁有穩定且高效之繪圖效能。本論文讓 CWT 在各作業系統上有兩項特性：

1. 提升繪圖效能 (Performance improvement)

根據實驗結果，CWT-GL 確實改善了 Java 繪圖效能。因 CWT-GL 的完成，讓 CWT 除原本只能於微軟視窗作業系統使用之 CWT-DX3 實作外，現在也能於其他作業系統上擁有高效之繪圖效能。

雖然新的 Java 版本在某些實驗中，如文字的繪製，效能較好，但因其執行時畫面會閃爍甚至顯示不正確，導致開發者無法預期程式之執行畫面，即使其他版本之繪圖效能有較 CWT-GL 好，但需使用新的 API，使已經開發完成之軟體必須再次修改原始碼，甚至在執行時必須設定特別旗標，才能夠獲得較好的繪圖效能。

2. 效能可攜性 (Portability)

本論文利用 JOGL 完成 CWT 中 CWT-GL 實作部份，並透過實驗證實 CWT 能夠於 Windows XP、Windows Vista、Fedora 與 MacOS 上正確執行，並從實驗數據證實其高效之繪圖效能。更重要的，CWT-GL 於不同作業系統上之效能差異較小，讓程式開發者，能預期開發出之應用程式的繪圖效能，至少於大部分作業系統上，不會因使用者所使用之 Java 版本不同，或不了解參數設定而有巨大的差異。

雖然本論文將 CWT 中 CWT-GL 完成，並利用實驗證實跨平臺的特性及高效的繪圖效能，但 CWT-GL 還有一些不足之處未來必須繼續改進的。

1. 繪製字串的效能提升

   繪製字串之效能，尚有提升的空間，從實驗數據，目前的效能距離 AWT 還有可以改善之部分，期望採用不同之實作方式改進這部份的效能。

2. 實用性

   目前 CWT-GL 的使用上仍會有錯誤產生，要達到可讓開發者們使用尚有需要完成之部分，期望可完成至能釋出讓遊戲開發者們使用。

參考文獻

[1] Andrew Gray, "GC Usage Statistics" http://216.147.18.102/dist/stats.shtml

[2] Bob Kunhne, Tom True, Allan Commike, Dave Shreiner, "Performance OpenGL: Platform Independent Techniques", Proc. ACM SIGGRAPH 2005 Course #9

[3] Caspian Rychlik-Prince, Brian Matzon, Elias Naur, Erik Duijs, Ioannis Tsakpinis, Mark Bernard, "LWJGL, Lightweight Java Game Library", available from http://www.lwjgl.org/.

[4] Chet Haase, "BufferedImage as Good Butter, Part II", http://weblogs.java.net/blog/chet/archive/2003/08/bufferedimage_a_1.html

[5] Evan Quinn and Chris Christiansen, "Java Pays – Positively, " IDC Bulletin #W16212, 1998. http://www.idcresearch.com/

[6] id Software Inc., http://www.idsoftware.com

[7] Jacob Marner, "Evaluating Java for Game Development", Department of Computer Science University of Copenhagen, Denmark, 2002.

[8] James Elliott, Robert Eckstein (Editor), Marc Loy, David Wood, Brian Cole, "Java Swing", Second Edition, O'Reilly, 2002.

[9] John Zukowski, "Java AWT Reference", O'Reilly, 1997.

[10]Jonathan Knudsen, "Java 2D Graphics", O'Reilly 1999.

[11]Mason Woo, Jackie Neider, Tom Davias, "OpenGL Programming Guide", Second Edition, Addsion-Wesley, 1997

[12]Microsoft Corporation, "DirectX 7.0 SDK", 2000, available from http://www.microsoft.com/.

[13]Microsoft Corp., "Microsoft Java Virtual Machine Support," available from http://www.microsoft.com/mscorp/java

[14]Microsoft Corporation, "Microsoft SDK for Java 4.0", 1999, available from http://www.microsoft.com/.

[15]Robert Wells, "Java offers increased productivity." 1999, available from http://www.wellscs.com/robert/java/productivity.htm.

[16]Sun Microsystems, "A Jolt of Efficiency", 1998, available from http://java.sun.com/features/1998/07/efficiency.html.

[17]Sun Microsystems, "Graphics Performance Improvements", available from http://java.sun.com/products/java-media/2D/perf_graphics.html .

[18]Sun Microsystems, "JDK 1.1.8 Documentation", 1998, available from http://java.sun.com/products/archive/jdk/1.1/index.html.

[19]Sun Microsystems, "JDK 5.0 Documentation", 2004, available from http://java.sun.com/j2se/1.5.0/docs/index.html.

[20]Sun Microsystems, "JOGL, Java bindings for OpenGL API", available from http://jogl.dev.java.net.

[21]Sun Microsystems, "Java 2 SDK, Standard Edition Documentation Version 1.2.2_006", 1999, available from http://java.sun.com/products/archive/j2se/1.2.2_017/index.html.

[22]Sun Microsystems, "Java 2 SDK, Standard Edition Documentation Version 1.3.1", 2001, available from http://java.sun.com/j2se/1.3/docs/index.html.

[23]Sun Microsystems, "Java 2 SDK, Standard Edition Documentation Version 1.4.2", 2003, available from http://java.sun.com/j2se/1.4.2/docs/index.html.

[24]Sun Microsystems, "Java SE 6 Features and Enhancements", available from http://java.sun.com/javase/6/webnotes/features.html

[25]Sun Microsystems, "The Java Tutorials", available from http://java.sun.com/docs/books/tutorial/index.html

[26]Sun Microsystems, "Painting in AWT and Swing", available from http://java.sun.com/products/jfc/tsc/articles/painting/index.html.

[27]Sun Microsystems, "The AWT Native Interface", available from http://java.sun.com/j2se/1.5.0/docs/guide/awt/1.3/AWT_Native_Interface.html.

[28] Sun Microsystems, "VolatileImage API User's Guide ", 2001, ftp://ftp.java.sun.com/docs/j2se1.4/VolatileImage.pdf.

[29] The OpenGL Architecture Review Board (ARB), "Framebuffer object extension", available from http://www.opengl.org/registry/specs/EXT/framebuffer_object.txt

[30] The OpenGL Architecture Review Board (ARB), "Pixel buffer extension", available from http://www.opengl.org/registry/specs/ARB/wgl_pbuffer.txt

[31] The OpenGL Architecture Review Board (ARB), "OpenGL", available from http://www.opengl.org/.

[32] Yi-Hsien Wang, I-Chen Wu, and Jyh-Yaw Jiang, "A portable AWT/Swing architecture for Java game development," Software Practice and Experience, Vol. 37, Issue 7, June 2007; 727–745.

[33] Yi-Hsien Wang, and I-Chen Wu, "An AWT/Swing like graphics toolkit for cross-platform Java game development," submitted to Software Practice and Experience, September 2007.

[34] 姜智耀, "CWT — The AWT API over Different Graphics Libraries", 交通大學資訊工程系, 碩士論文, 2005

[35] 第三波資訊, "戲谷麻將館", available from http://www.mjonline.com.tw/.

[36] 智凡迪, "魔獸世界", available from http://www.wowtaiwan.com.tw/.

[37]鈊象電子, "明星三缺一 online", available from http://www3.gametower.com.tw/Games/Star31/index.aspx.

[38]傅鏡暉, "線上遊戲產業之道：數位內容、營運經驗", 上奇科技, 2004.

[39]群想網路科技, "CYC 遊戲大聯盟", available from http://cycgame.com.

[40]遊戲橘子, "天堂", available from http://service.gamania.com/lineage/index.asp.