國立交通大學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

在 CWT OpenGL 版本中混和元件與 3D 圖形的設計

The Design of Mixing Widgets with 3D Graphics for the OpenGL Version of CWT

研 究 生：吳秉儒

指導教授：吳毅成　教授

中 華 民 國 九 十 七 年 八 月

在 CWT OpenGL 版本中混和元件與 3D 圖形的設計

The Design of Mixing Widgets with 3D Graphics for the OpenGL Version of CWT

研 究 生：吳秉儒　　Student：Bin-Ju Wu

指導教授：吳毅成　　Advisor：Dr. I-Chen Wu

國 立 交 通 大 學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

In

Industrial Technology R & D Master Program on

Computer Science and Engineering

Aug 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年八月

在 CWT OpenGL 版本中混和元件與 3D 圖形的設計

研究生：吳秉儒　　指導教授：吳毅成

國立交通大學資訊學院產業研發碩士班

摘要

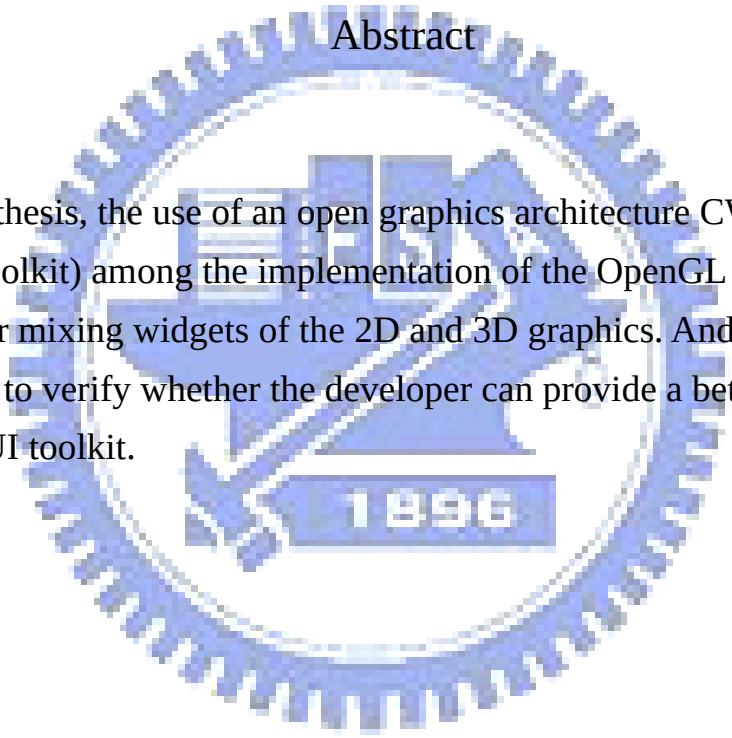
3D 電腦遊戲是目前的趨勢，因為畫面更為真實而吸引了大多數的玩家。而 3D 電腦遊戲設計中，最重要的是遊戲畫面的繪製，遊戲畫面可以分成 3D 與 2D，3D 部分如場景、人物等，2D 部分則是如 UI (User Interface)。為了要讓 3D 遊戲中的 UI 設計跟製作得到高效率與高產能。

本論文使用一套開放的繪圖架構 CWT(CYC Window Toolkit)[7]之中的 OpenGL 實作部分，設計一套混和 CWT 的 2D 元件與 3D 繪圖的架構，並經由實驗驗證是否能提供給開發者較好的 UI 開發選擇。

The Design of Mixing Widgets with 3D Graphics for the OpenGL Version of CWT

Student: Bin-Ju Wu Advisor: Dr.I-Chen Wu

Institute of Industrial Technology R & D Master on
Computer Science College
National Chiao Tung University



Abstract

In this thesis, the use of an open graphics architecture CWT(CYC Window Toolkit) among the implementation of the OpenGL version. Design a system for mixing widgets of the 2D and 3D graphics. And by the experiment to verify whether the developer can provide a better choice of 3D Game UI toolkit.

誌謝

首先要感謝指導教授吳毅成教授的指導，不僅是在學業上還有生活上的關心，我在交大的這兩年，很幸運能受到教授的指導，接觸了許多方面的研究，使我獲益良多。

特別感謝博士班汪益賢學長的指導，即使是在美國，仍然不辭辛勞指導我完成這篇論文的研究，如果沒有他，就沒有這篇論文的產生，學長耐心及細心的指導，使我從完全沒有概念，到能完成這篇論文的研究，預祝學長畢業服役後，在美國的工作順利愉快。

對於實驗室的同學們，感謝你們學業上的幫助，以及平日的聊天紓解研究的壓力，同一個產業專班的哲毅、典餘，以後相處時間還很多，偉傑、汶傑、鼎量、榮欽（rc）、威霖（try）一起渡過在交大的生活，還有學弟妹們在我忙於論文時的幫助。

最後要感謝的是我的家人，我的父母讓我生活無後顧之憂，花了許多時間，等待我結束學生的身分。還有我最親愛的老婆，君蓉，在研一時我們結婚，一年後生下可愛的女兒淨涵，妳是最辛苦的人，在我繁忙時，一個人獨立照顧女兒成長，這篇論文獻給我最愛的妳及寶貝女兒。

目錄

表目錄

圖目錄

第一章、 緒論

本章一開始會在 1.1 節簡介本論文的背景相關知識，包含遊戲依照畫面維度的分類，還有目前熱門的是哪種類型遊戲。1.2 節會討論遊戲程式所使用的語言，由效能及產能兩個面相討論。1.3 節則介紹並討論 Java 中的繪圖函式庫。1.4 節中會提出 2D 在 3D 繪圖中的應用，及本論文希望解決集大成的目標。1.5 節簡介本論文的內容及大綱。

1.1 背景介紹

電腦遊戲隨著近幾年科技的進步，遊戲公司也開發出越來越多聲光娛樂效果俱佳的遊戲，吸引了大批的玩家，在全世界都形成一股風潮，以下會簡介電腦遊戲的分類演進及現況。

如果以遊戲畫面來分類，可以分成 2D、2.5D、3D 的電腦遊戲，以下介紹各分類的遊戲。

A. 2D 遊戲

早期的遊戲，因為軟硬體技術的限制，畫面只有 x 軸、y 軸兩種座標，沒有 z 軸這種深度的關係，畫面叫為單調，例如俄羅斯方塊。

B. 2.5D 遊戲

因為畫面想要 3D 也就是加上對 z 軸深度的需求，但是 3D 需要大量的圖形運算，軟體及硬體技術無法達到，所以使用 2D 畫面去模擬出 3D 的效果，這種遊戲畫面使用許多 2D 繪製的偽 3D 畫面，讓遊

戲玩家有立體的感覺，例如 Doom 及初期的線上遊戲。

C. 3D 遊戲

隨著顯示卡的進步，繪圖能力越來越強，現在要做 3D 的繪圖已經不是困難的事情，立體空間的概念使的遊戲的畫面更為真實，而對空間操作的自由度也更高，如 3D 線上遊戲魔獸世界[14]、天堂 2[15]。

3D 電腦遊戲因為其畫面真實性及操控的自由，吸引了眾多的玩家，如魔獸世界目前以有超過 1000 萬的玩家且持續增加中，也因此現今的電腦遊戲公司也都以開發 3D 電腦遊戲為主，3D 遊戲已經是目前電腦遊戲的趨勢。

1.1.1 開發語言的選擇

對電腦遊戲開發商來說，以技術面的觀點有兩個，一個是效能（Performance），3D 遊戲中最重要的就是繪圖的效能。另一個是產能[16]，也就是遊戲開發的速度與獲利。

而對電腦遊戲程式設計者來說，最重要的是選擇所使用的開發程式語言，目前的 3D 遊戲程式設計師主要所使用得程式語言是 C/C++，在更之前所使用的主流程式語言是 Assembly[3]，因為在當時，覺得 C/C++ 的效能並沒有辦法跟 Assembly 相比，但後來自從 DOOM 這款以 C 語言所

撰寫的第一人稱射擊遊戲問世後，流暢的畫面與設計，讓程式設計師相信 C 也能有不錯的效能，而 C 因為比 Assembly 更好撰寫，所以也獲得更佳的產能。現在也有許多新興的程式語言，經過近年不斷的改進，也有機會取代 C/C++成為下一代遊戲設計的主流語言，其中 Java 就是最有可能程式語言之一。

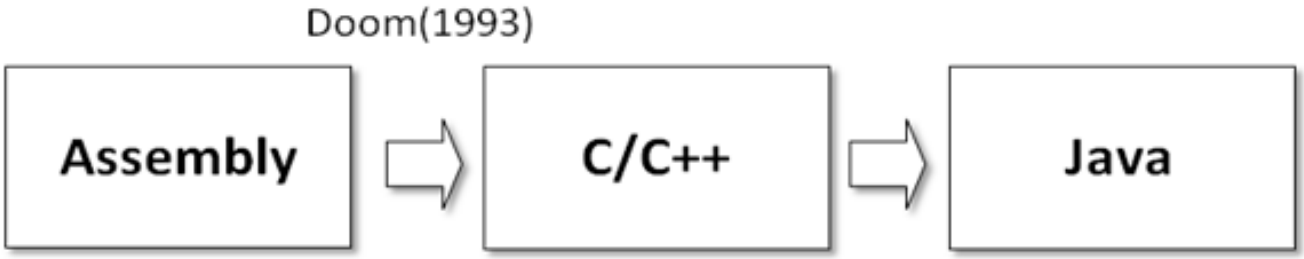


圖 1-1：遊戲程式語言的演進及展望

1.2 Java

Java 是由昇陽（Sun Microsystems）公司於 1995 年開發的一種跨平台的程式語言，一開始是以網路為主要的應用平台，後來隨著網路的擴張與迅速發展，Java 的地位也越來越重要。除了跨平台之外，物件導向方式的程式設計，也讓開發者能更好使用這套語言。在下面會以產能與效能這兩個觀點去討論 Java 是否有足夠的優勢成為下一代的 3D 遊戲主流語言。

1.2.1 產能

產能可以分成兩個面象來談，一個是開發過程，讓開發者能更好使

用，另一個則是維護過程。Java 本身是純物件導向的程式語言，而且有豐富的函式庫，還有記憶體管理系統，使其開發過程更為容易，且 Java 最重要的特點就是可以跨平台，程式碼不用因為在不同平台上使用而重新撰寫，更是大大增加了其使用性。下表顯示，當一個產品完全以 Java 開發時，比使用 C/C++能獲得更好的產能[2]。

	生產力增加	時間/成本 減少
開發過程	67%	40%
維護過程	42%	30%
整體		25%

表 1-1：Java 與 C/C++相比的產能比較
資料來源[2]

1.2.2 效能

以執行效能來說，近年來由於 Java 的改版，加入即時編譯技術（Just-In-Time Compilation）[11]，以及熱點動態編譯技術（HotSpot Compilation）[12]，使得 Java 的執行效能大為提昇。下表顯示，Java 與 C++比較，在使用 Java 版本 1.3、1.4 時，有些情況甚至效能會比 C++還好[3]。

JDK/JRE 版本	C++/Java 效能比
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++相比的效能比較
資料來源[3]

1.3 Java 繪圖函式庫

在 3D 遊戲中，繪圖效能對整體執行效能的影響最大。現今的繪圖函式庫，主要分為由微軟開發的 DirectX[13]與另一套公開的 OpenGL[8]，這兩套函式庫都可以藉由繪圖加速卡的加速功能，提昇繪圖的效能，還有繪圖的細膩跟真實度。DirectX 只能使用在是微軟的視窗作業系統底下，而 OpenGL 則是業界的標準，可以跨平台使用，並不限於微軟的視窗系統。下面將會先介紹 Java 的基本繪圖函式庫，有那些基本的功能，之後會在介紹使 Java 的第三方繪圖函式庫，讓 Java 可以使用 OpenGL 及 DirectX，增強其繪圖的能力。

1.3.1 基本繪圖函式庫

Java 的基本繪圖函式庫主要有兩個，一個是最基本的抽象視窗工具組(Abstract Window Toolkit，以下簡稱 AWT)[10]，位達成在個平台上都能繪製一樣的目的，採用重型元件（Heavy-Weight Component），由

作業系統來產生處理。另一套則是 Swing[4]，讓增強了視窗元件的繪製樣式，採用軟體繪製的輕型元件（Light-weight Component），因為是建立在重型元件上的軟體繪製，所以繪圖效能會受限於 AWT。

1.3.2 第三方繪圖函式庫

Java 有眾多的函式庫可以使用，包跨各種的繪圖函式庫，而利用這些第三方提供的繪圖函式庫，Java 就可以去使用 DirectX 及 OpenGL 眾多的繪圖功能，使 Java 的繪圖效能得到很大的提升，下表則是目前可以使用的第三方繪圖函式庫。

函式庫名稱	簡稱	支援	提供者
Light-Weight Java Gaming Library[9]	LWJGL	OpenGL	Lwjgl.org
Java bindings for OpenGL API[1]	JOGL	OpenGL	昇陽
DirectX Graphics[6]	Direct3D	DirectX	微軟

表 1-3：Java 第三方繪圖函式庫[9]

微軟視窗所提供的 Java 虛擬機器(Microsoft Java Virtual Machine)中直接內建了使用 DirectX 的 API，但在其他平台的 Java 虛擬機器並沒有，也使得使用 DirectX 的 Java 繪圖程式，只能在視窗系統上運作。

1.5 問題與目標

在 3D 遊戲中，Java 可以使用第三方繪圖函式庫，如 JOGL，提昇 3D 繪圖的效能跟細膩度，但是在 3D 遊戲中的繪圖不只是只有 3D 而已，還有 2D 的部份，例如像玩家資訊、系統設定、遊戲選單、文字訊息這些圖形化玩家介面（Graphic User Interface，以下簡稱 GUI），而這些介面的製作與使用會對遊戲產生影響。一般開發者要製作這些 GUI 有以下兩種方式

1.5.1 使用 Java 的基本繪圖函式庫 AWT/Swing

因為 AWT/Swing 本身就是 2D 的樣式，而且有現成的元件可以使用，而開發者對這套函式庫也很熟悉，所以在使用上較方便。但缺點是，因為 AWT/Swing 並沒有使用繪圖加速卡的加速功能，所以繪圖效能不好，連帶會影響 3D 遊戲整體的繪圖效能，而且由於 AWT 是重型元件，所以不能做不規則形狀或是半透明的元件效果，對於講求遊戲畫面的 3D 遊戲來說，繪圖樣式不夠多。

1.5.2 使用第三方繪圖函式庫

Java 可透過 JOGL 來使用 OpenGL 畫製 2D 的 GUI，而且可以獲得繪圖加速卡的加速功能，使繪圖效能增強，並且可以做到 AWT 所做不到的不規則形狀及半透明等效果，但相對的，使用第三方繪圖函式庫並沒有提

供現成的元件可以使用，開發者必需花費許多時間自己製作一套元件來使用，這一套新的元件還要去學習如何使用。

	AWT/Swing	OpenGL
效能	X	O
產能	O	X

表 1-4：AWT/Swing 與 OpenGL 製作 GUI 的產能與效能比較

表 1-4 顯示 AWT/Swing 在效能上面不佳，而 OpenGL 在產能也有問題，因此，本論文使用 CYC Window Toolkit(簡稱 CWT)[7]， CWT 藉由第三方繪圖函式庫提出相容於 AWT 的元件實作，來達到效能及產能兼顧，如表 1-5 所示。

	AWT/Swing	OpenGL	CWT
Performance	X	O	O
Productivity	O	X	O

表 1-5：AWT/Swing、OpenGL 與 CWT 製作 GUI 的產能與效能比較

但是，CWT 原先的設計是只用於 2D 的繪圖，並不能使用在 3D 的畫面上，

會造成 3D 遊戲畫面錯誤，本論文的目標則是要讓 CWT 能與 3D 遊戲畫面相容，並且達到比 AWT/Swing 更好的效能，與比第三方繪圖函式庫更好產能。

1.6 本文大綱

本篇論文在第一張會先介紹 Java 在 3D 遊戲繪圖上的背景知識及 GUI 所產生的問題及希望解決的目標。第二張會介紹 CWT 的架構與成果。第三章會詳細介紹本論文如何解決 CWT 在 3D 遊戲 GUI 繪製的問題。第四章是本論文所做的實驗數據及分析。最後第五章會對本論文的研究做一個總結，並討論更近一步的發展方向及內容。

第二章、 系統架構

本論文所使用的系統CWT-GL是建構在CWT之上，所以在2.1節當中，會簡介什麼是CWT，還有它的架構。2.2節中則會介紹CWT-GL，與它使用的第三方繪圖函式庫JOGL。在2.3節中會先介紹OpenGL在做3D繪圖的流程。2.4節會介紹CWT-GL的繪圖流程及直接將CWT-GL與3D繪圖使用在一起時的問題。最後在2.5節中提出可能使用的方法及優缺點。

2.1 CWT 系統架構介紹

CWT是一套使用不同繪圖函式庫實作出相容AWT的API，借此改善原本AWT的繪圖效能[7]。

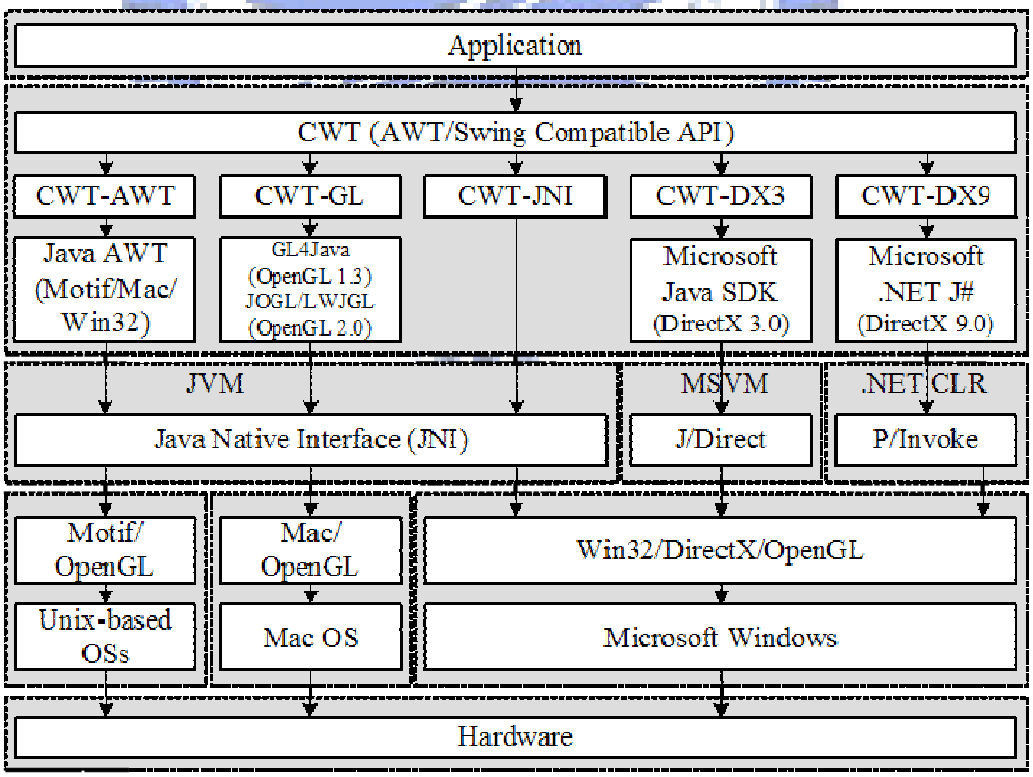


圖 2-1：CWT 架構圖[7]

AWT 的內容主要有元件模式(Component Model)、事件模式(Event Model)及繪圖模式（Painting Model）[7][16]，若應用在 3D 遊戲 GUI 設計上，元件模式提供了相容於 AWT 的各種元件，如下圖所示。

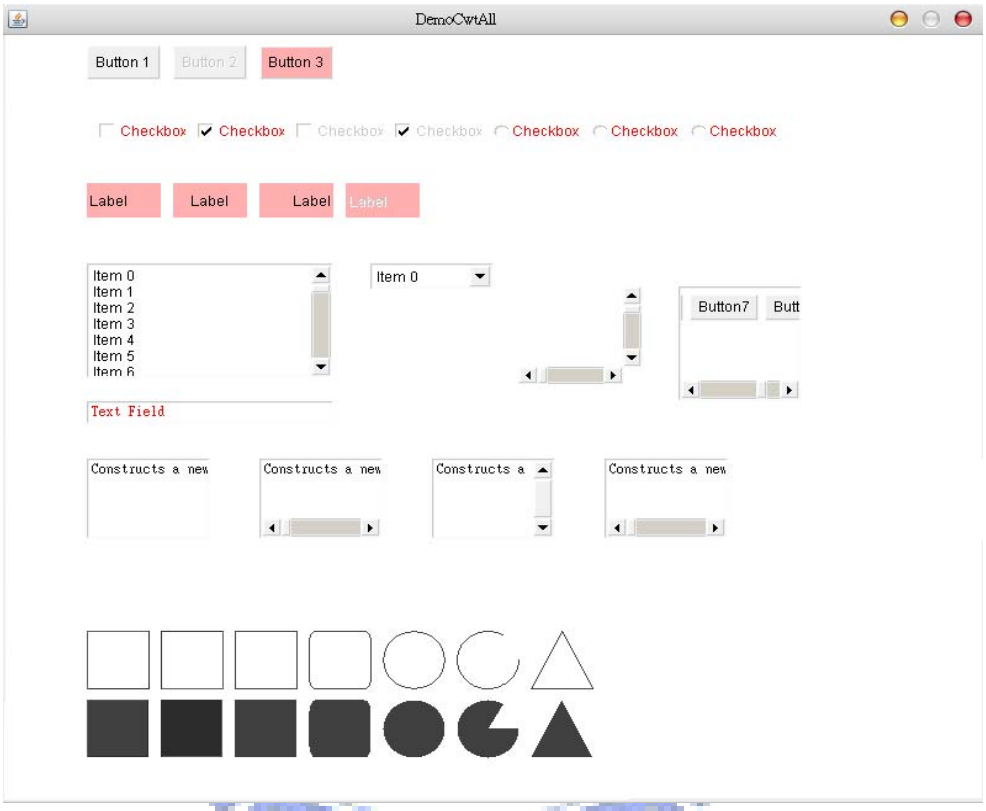


圖 2-2：CWT 元件圖

包含了 Button、Checkbox、Label、List、Choice、Scrollbar 、TextField 及 Canvas 等八大元件。

而對於兩大加速繪圖函式庫 OpenGL 及 DirectX，CWT 都有個別實作出來，OpenGL 的部份叫做 CWT-GL[17]，而 DirectX 的部份叫做 CWT-DX[16]。本論文考慮到 Java 跨平台的特性，因此在選擇使用的函式庫上，使用

一樣是跨平台函式庫 OpenGL 實作的 CWT-GL 作為 3D 遊戲的 GUI 繪圖函式庫。

2.2 CWT-GL

本節將會介紹 CWT-GL 所使用的 JOGL[9]套件，並說明為何使用它。

OpenGL 的時做套件主要的有三種，GL4Java、JLWGL 及 JOGL，GL4Java 的缺點是自從 2003 年起就沒有在更新，所以沒有支援新的 OpenGL 版本，而 JLWGL 與 JOGL 相比，JOGL 是由昇陽公司所開發支援，對於問題的回復速度快，很用心在維護，目前已經納入 Java 的參考標準 JSR-231(Java Specification Request Java Binding for the OpenGL)。

而 OpenGL 是業界 3D 繪圖普遍使用的標準，他跨平台的優勢，及背後支持的眾多廠商，讓繪圖加速卡不斷加入其功能，再未來隨著 OpenGL 新標準的制定，也會有眾多硬體持續支援。

CWT-GL 就是使用 OpenGL 來時做出 AWT/Swing 相容的繪圖函式庫，使用其硬體加速功能，並作了一些最佳化，例如避免不必要的測試計算、單一執行序繪圖、減少狀態改變等[17]。得到一個比 AWT/Swing 效能更好的繪圖函式庫。

2.3 OpenGL 繪圖流程

在 3D 世界的繪圖，從物體座標定位到最後在視窗上顯示，是一連串

的座標轉換，以下就會先說明這些座標轉換的過程。

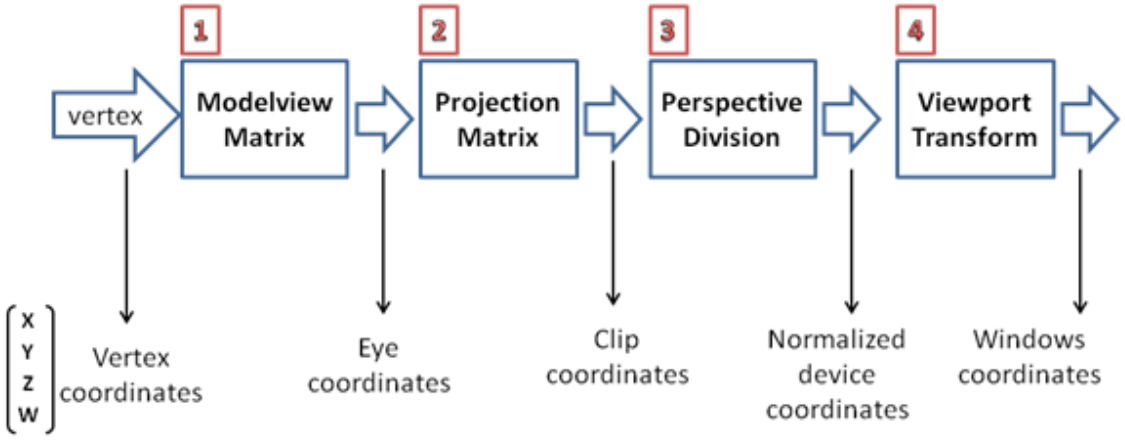


圖 2-3：OpenGL 座標點轉換程序（OpenGL Vertex transformation pipeline）

如圖 2-3 所示，一開始要成像的物體原始座標換轉換成一個一乘四的矩陣，前三個值是 x、y、z 座標，最後一個 z 座標則是縮係數(scaling factor)，此值可用在四個三樹的座標系統中，通常預設值是 1.0。接著座標點會與 Modelview 矩陣，也就是觀察者與場景中物體的轉換關係矩陣運算，產生相對與觀察者位置的座標值。接著座邊值會乘上一個投射矩陣，產生對應的裁減座標（clip coordinates），目的是去掉觀察者範圍外的資料。裁減座標會在除以縮放係數 w，得到標準化的裝置座標（Normalized device coordinates）。而最後 3D 的座標經過視埠轉換（Viewport Transformation）的矩陣運算映射到 2D 的視窗平面上，產生出最後螢幕上所看到的畫面。

圖 2-4 是以 OpenGL 繪製一個立體的方塊為例，經過 OpenGL 座標轉換程序，最後會產生出一個畫立體方塊的視窗。

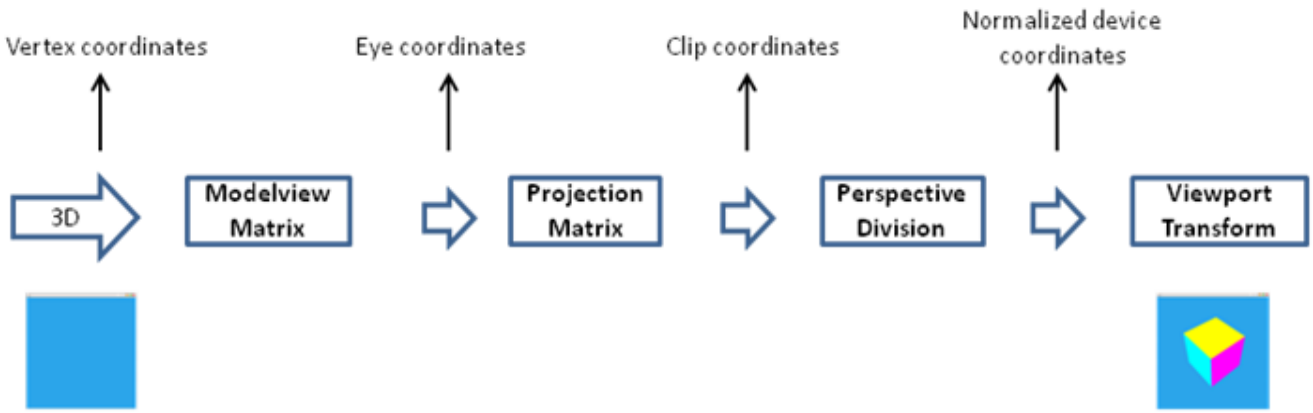


圖 2-4：視窗經過 OpenGL 座標點轉換程序繪出 3D 方塊

圖 2-5 則用立體方塊這個例子，圖示說明成像的過程。

1. Modelview 矩陣，描述觀察者跟立體方塊間的座標
2. 投射(Projection)，在大多數的 3D 遊戲成像中，是使用透視投射（Perspective projection），也就是投射的物體與觀察者越近的部分越大，越遠的部份越小，就像是在真實世界中看物體一樣。
3. 裁減座標，去除方塊範圍以外不要顯示在視窗上畫面。
4. 視埠轉換，將立體方塊最後的畫面映射到視窗（Window）上面。

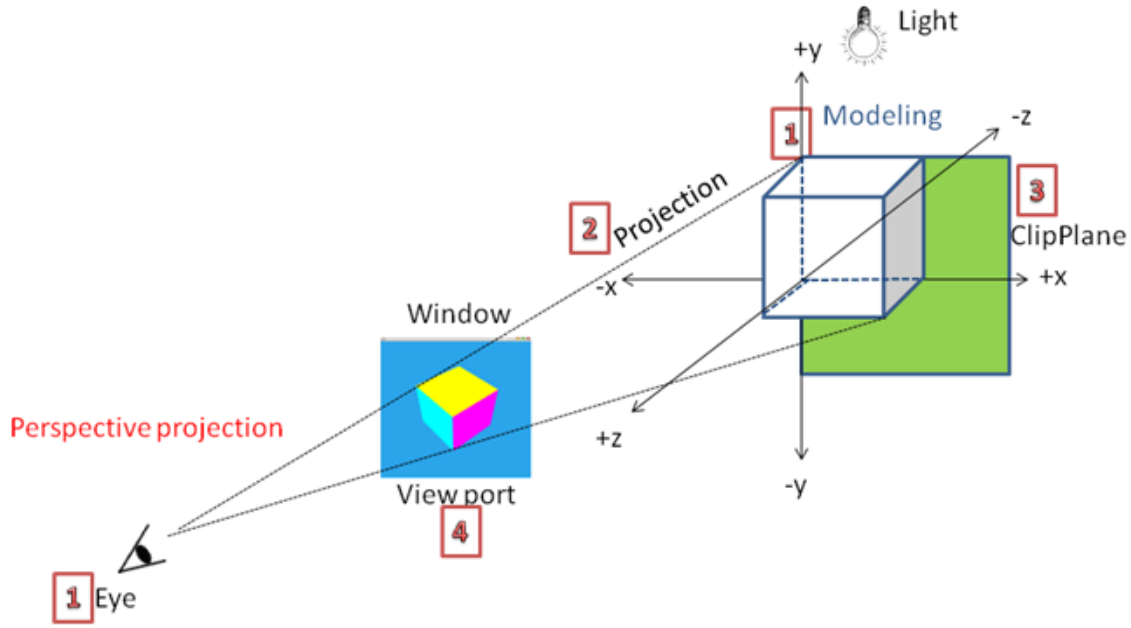


圖 2-5：3D 方塊成像圖解

而這些成像的過程，我們可以看做一個黑盒子，當我們把相關要設定的參數輸入進去時，就會產生相對應的成像，如圖 2-6。開發者可以藉由設定不同的成像參數繪製出不同的畫面。

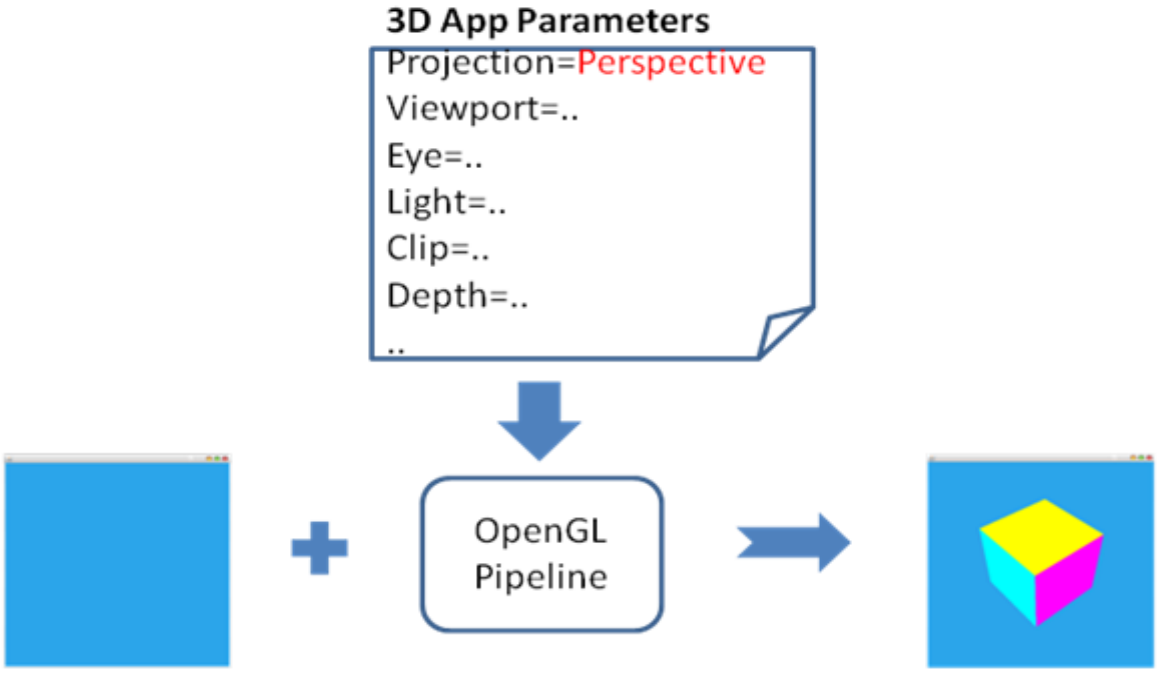


圖 2-6：3D 方塊繪製參數設定 OpenGL 繪圖流程示意圖

2.4 CWT-GL 的繪圖流程

CWT-GL 也是一樣使用 OpenGL 的座標轉換流程成像，圖 2-7 顯示在一個視窗上會出一個按鈕（Button）元件。

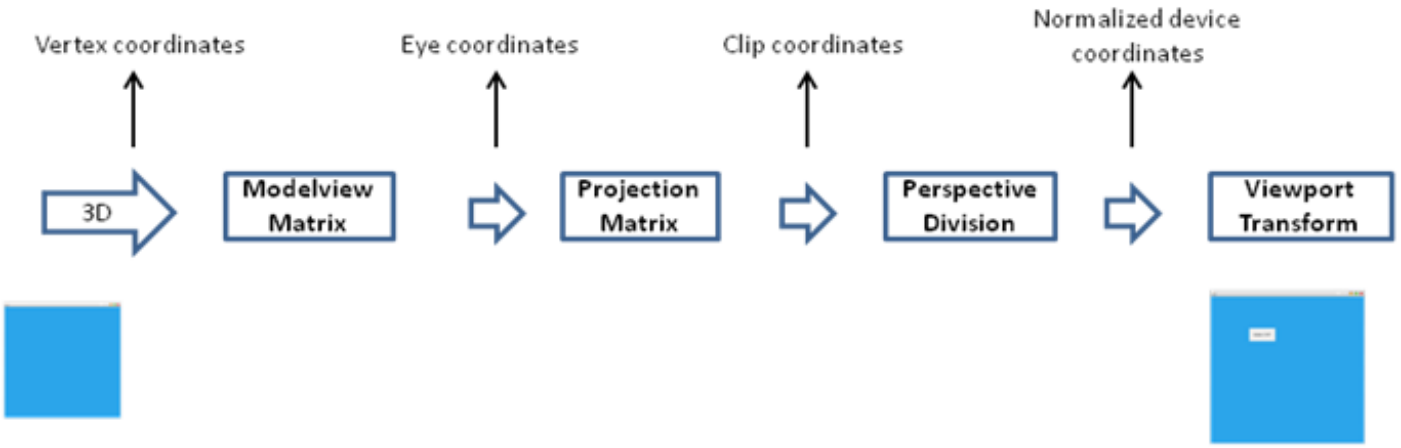


圖 2-7：視窗經過 OpenGL 座標點轉換程序繪出 CWT 元件

圖 2-8 則用 CWT 元件 Button 這個例子，圖示說明成像的過程。

1. Modelview 矩陣，描述觀察者跟 Button 間的座標。
2. 投射(Projection)，在 CWT-GL 中所使用的投射方式是正交投射（Orthographic projection），不管觀察者與物體之間的遠近，所看到的物體大小是一樣的，元件的大小是交由開發者自己去設定。
3. 裁減座標，去除 Button 範圍以外不要顯示在視窗上畫面。
4. 視埠轉換，將 2D 的 Button 畫面映射到視窗（Window）上面。

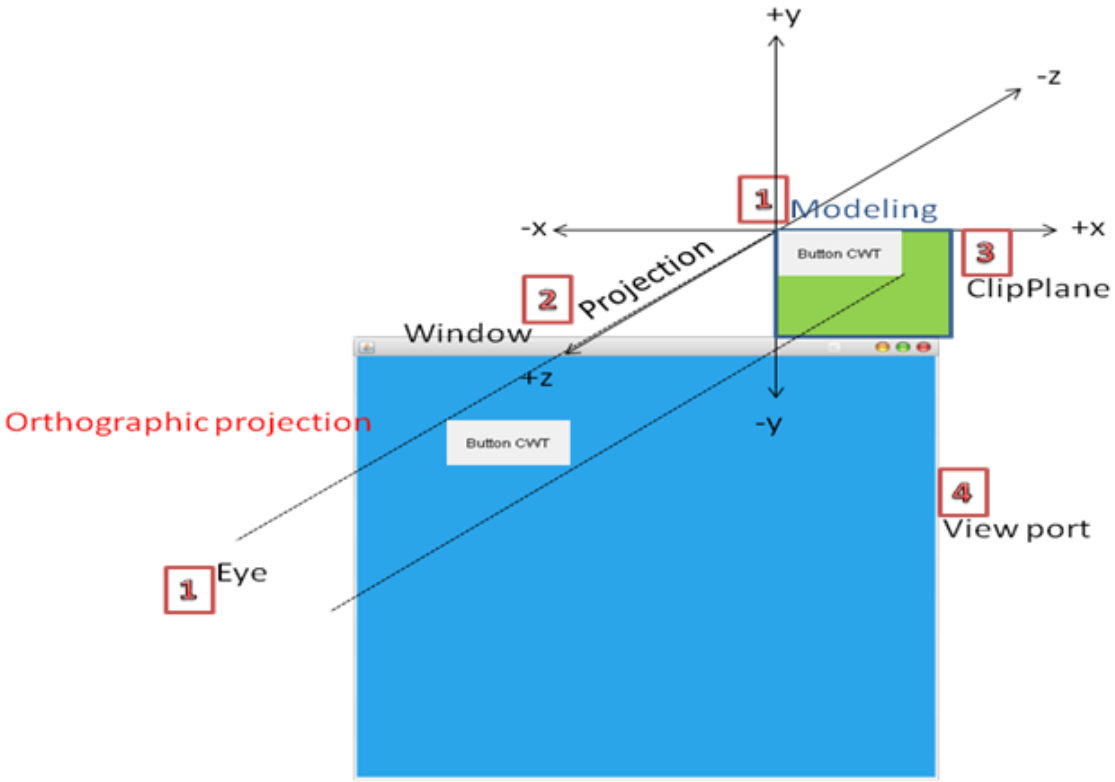


圖 2-8：CWT 元件成像圖解

而這些流程我們也可以視作把繪圖參數設定進 OpenGL 流程後繪製的成像，如圖 2-9 所示。

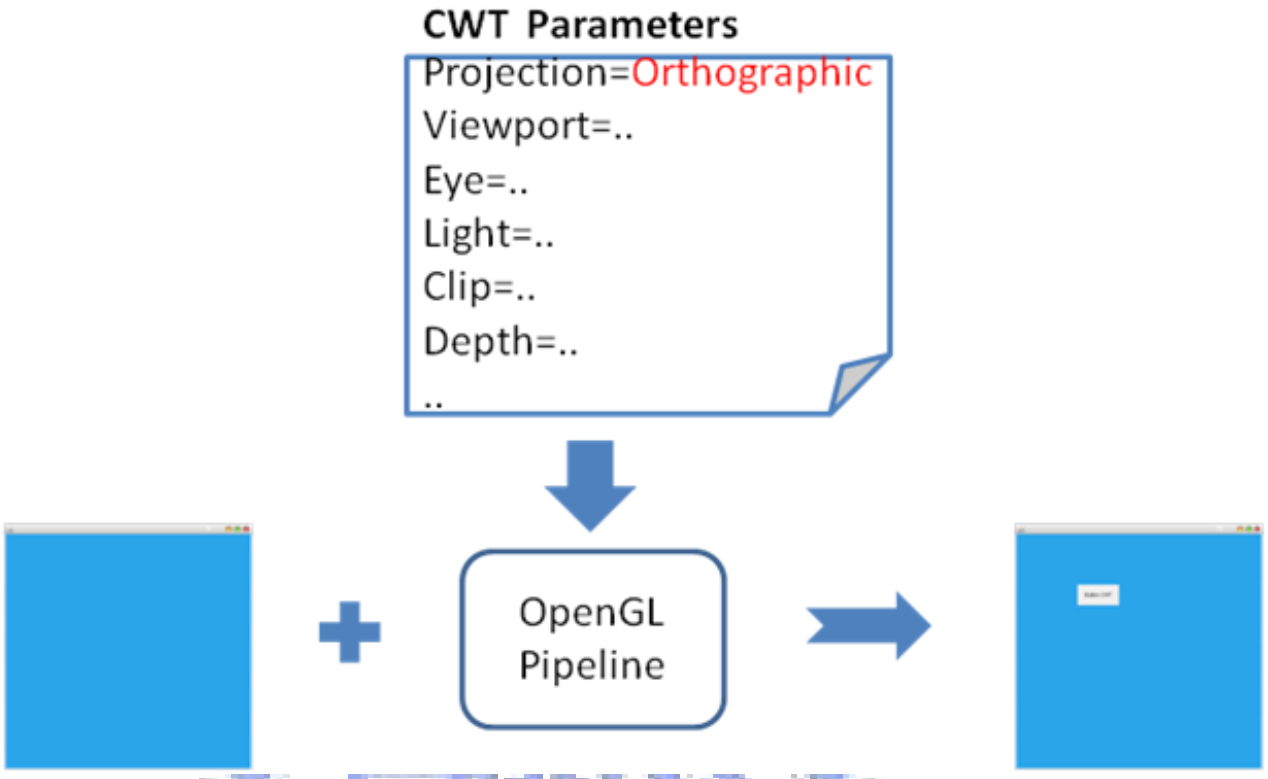


圖 2-9：3D 方塊繪製參數設定 OpenGL 繪圖流程示意圖

如果要將 CWT 元件使用在 3D 畫面上，很直覺的會把兩個畫面疊在一起畫，如圖 2-10，就可以畫出我們想要的畫面。

圖 2-10：3D 方塊混合 2D CWT 元件

以 OpenGL 的繪圖流程來說，就是把參數全部丟進去，但是這時就會產生問題，OpenGL 會同時接到同一個參數的不同設定，例如說像投影如

圖，3D 方塊是使用透視投影，而 CWT 是使用透視投影，這時就會產生衝突，如果是採用透視投影，則 CWT 元件將不會正確的被會出到視窗上，若使用正交投影，則 3D 方塊的顯示就會錯誤，如圖 2-11、2-12。

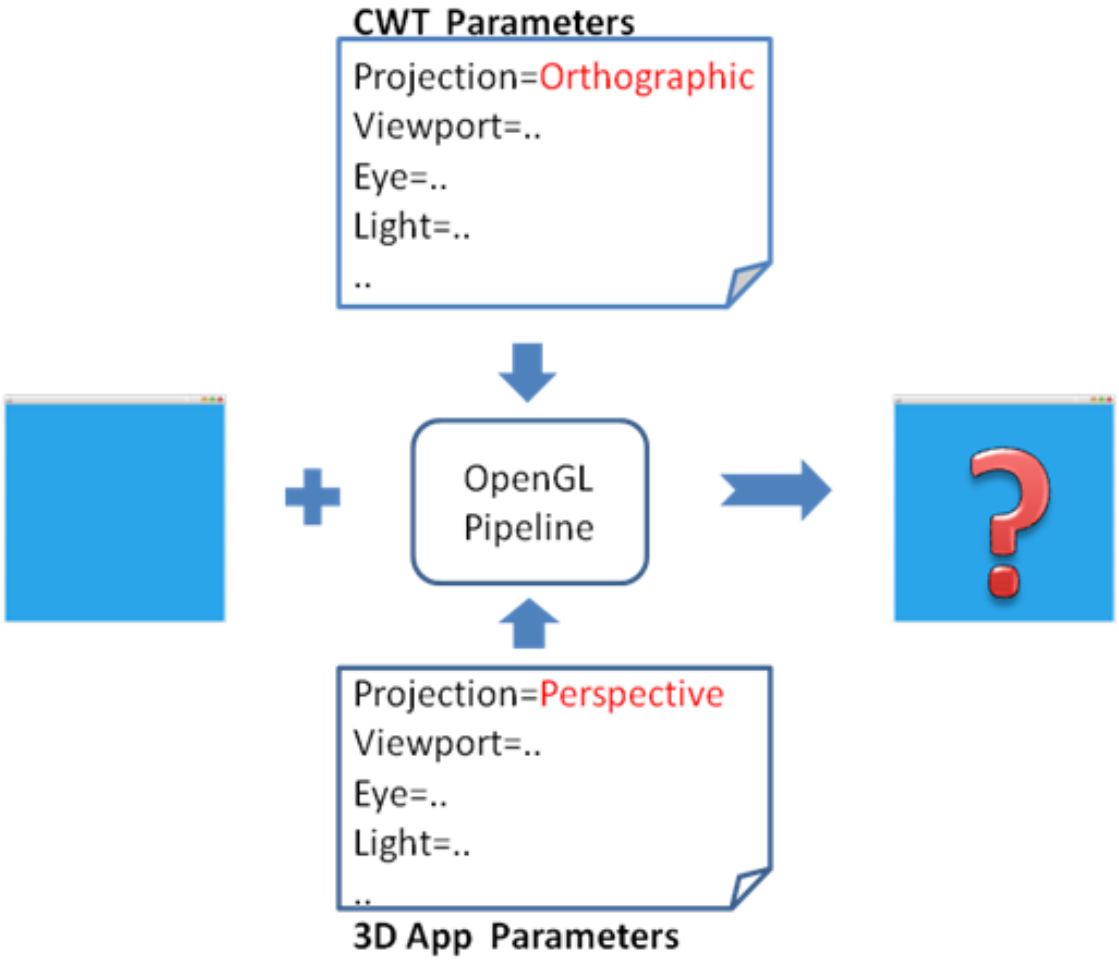


圖 2-11：3D 方塊混合 2D CWT 元件繪製參數設定 OpenGL 繪圖流程示意圖

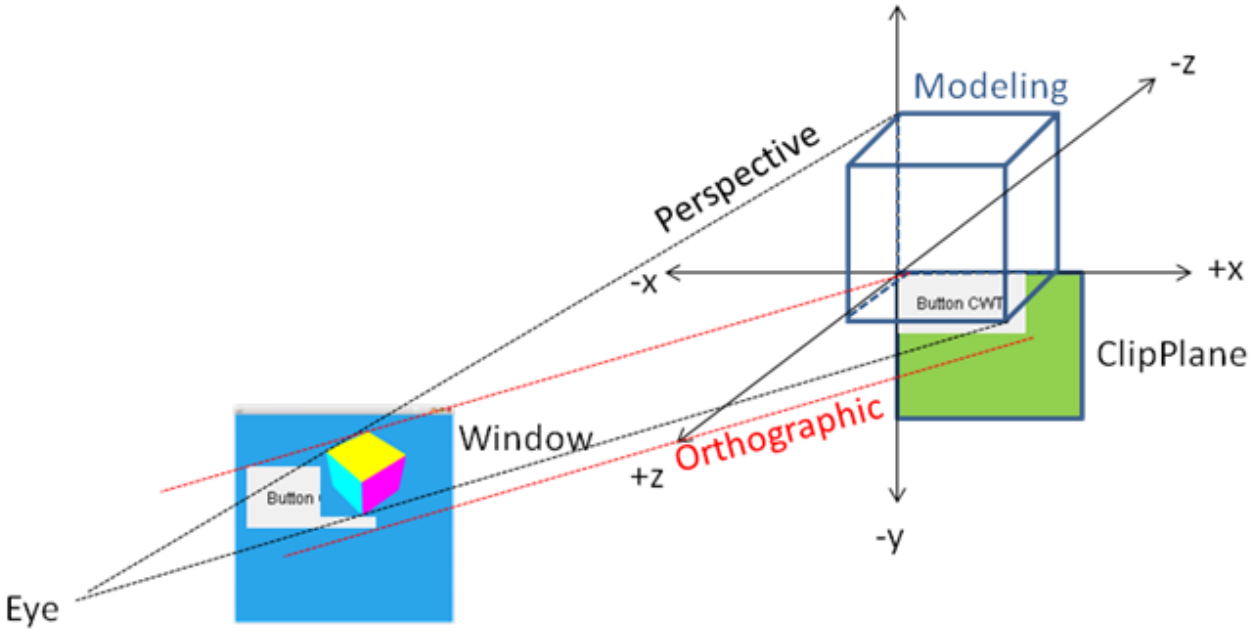


圖 2-12：3D 方塊混合 2D CWT 元件成像圖解

本論文為了解決這個問題，設計了一套繪圖流程，使的 CWT 元件能夠正確的與 3D 繪圖混和使用。

2.5 混合 CWT-GL 與 3D 繪圖的可能方法

在本節中將會提出把 2D 元件與 3D 繪圖一起混合使用的可能方法。在 2.5.1 節中提出一般電玩遊戲中使用的一個技術叫做公告板（Billboard）。2.5.2 節中則提出背景繪圖(Off-screen rendering)的方法。但這兩個方法各有其缺點。

2.5.1 公告板（Billboard）

公告版技術可以想像成有一個人舉著一個公告板要你看到上面的資訊，所以他必須把它所舉的牌子，一直面向你眼睛所看的那個方向，這

樣你不管走到哪，眼睛怎麼移動，都可以看到公告板正面的內容。

在 OpenGL 中必須要自己實作公告板的技術。使用眼睛（Eye）的位置矩陣與投影矩陣，還有視埠的位置矩陣去做數學運算後，得到公告板要放置的位置，並且把要繪製的東西像貼圖（Texture）一樣貼上去。

圖 2-13:使用公告板混合 2D CWT 與 3D 方塊流程示意圖

隨著畫面的進行，這些矩陣也不斷改變，相對的公告板的位置也要不停的運算，因為運算公告板的函式是沒有使用繪圖加速卡的加速功能，所以必須全靠中央處理器去運算，因此遊戲的效能就會受到影響而變差。

2.5.2 背景繪圖(Off-screen rendering)

另一個長使用的方法是背景繪圖，這個方法常用在繪製連續動畫時避免螢幕閃爍，在記憶體先產生一塊空間，將要繪製的內容話到這塊空間上，在一次輸出貼到螢幕上，這樣畫面上看到的是直接畫好的內容，而不會看到繪製的過程。

同樣，可以嘗試使用這種技術將 CWT-GL 的元件先繪製在另一塊記憶

體空間中，而 3D 繪圖則有另一塊繪製空間，做獨立的繪圖，當兩塊空間繪圖完畢時，可以將元件的那塊空間，複製貼到 3D 繪圖的成像上，在一起輸出到螢幕上，如圖 2-14 所示。

這種方法與公告板比不需要去做位置矩陣的運算，但是要多使用一塊記憶體空間來做背景繪圖，而且最後還要要做一個複製貼上的動作而影響效能，這也是目前 AWT/Swing 所使用的方法。

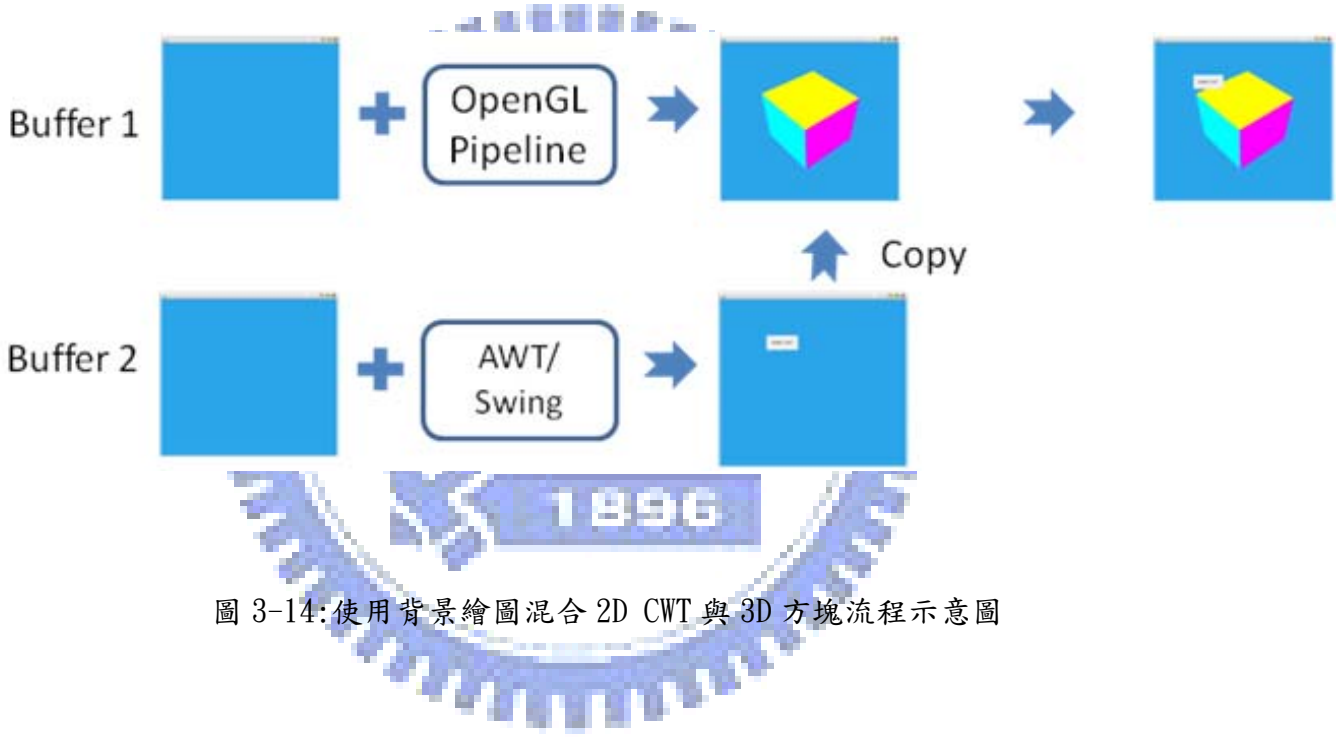


圖 3-14:使用背景繪圖混合 2D CWT 與 3D 方塊流程示意圖

第三章、 系統實作

在第三章中將會詳細說明，如何把 CWT-GL 的元件系統與 3D 繪圖環境做結合，並畫出正確的畫面。首先在 3.1 節中將會說明如何將 OpenGL 的 3D 繪圖與 CWT-GL 結合在一起，並且改進第二章末節提出的解決方法。在 3.2 節中列出在系統設計時需要注意的一些議題。

3.1 混合 CWT-GL 與 3D 繪圖的系統設計

要解決混和在設定的問題本論文設計的方法是使用串連的方式，將 3D 繪圖與 CWT 的元件分開在不同的 OpenGL 繪圖流程個別繪製。這種方式就可以保證一次只有一種參數會設定進去，使 OpenGL 繪圖流程在繪製時不會產生衝突。如圖 3-11，從左邊開始，空白的視窗經過立體方塊設定參數進入 OpenGL 繪圖流程，會出在視窗上的立體方塊後，CWT 再將要繪製 Button 元件的參數傳入 OpenGL 繪圖流程，此時立體方塊已經正確畫完，CWT 就可以使用它預設的成像方式會出正確的 Button 在視窗上面。

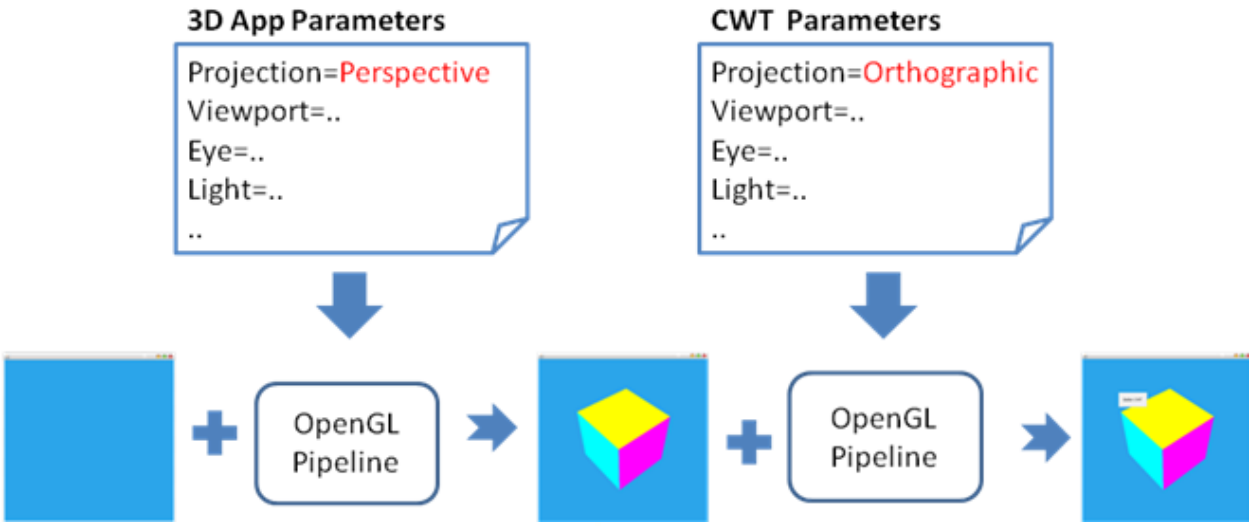


圖 3-13：3D 圖形混合 2D CWT 元件繪圖流程架構圖

3.2 混合 CWT-GL 與 3D 繪圖的實作內容議題

在這一節中，將會討論實作系統時要注意的一些議題。在 3.2.1 中，會說明如何將 2D 元件放置在畫面最頂端。3.2.2 節中，說明要回復的位置矩陣參數。3.2.3 節提出一般 3D 繪圖中打光的問題。3.2.4 節則提出混合兩個繪圖流程要注意的裁切問題。

3.2.1 元件置頂

在 3D 的遊戲畫面中，GUI 不行被遊戲畫面遮住，所以 CWT 的元件必須繪製在畫面的最上層，也就是要 3D 程式繪製在視窗上面後 CWT 才進行繪製，為了達到這點，在註冊進 OpenGL 繪圖程序時，CWT 要在 3D 程式註冊後才註冊，但是最底層的視窗也是 CWT 建置的，若直接在 3D 程式後才註冊 CWT，那會造成連視窗無法繪出的情況發生，所以本論文的

解髮是，CWT 先註冊繪出視窗，接著 3D 程式要註冊時，先移除 CWT 的註冊，等 3D 程式註冊完後在註冊回去，如圖 3-12，這樣就可以解決 CWT 要在 3D 程式之後繪製的問題。

這裡還要注意一點就是，當 CWT 在繪製時要關掉深度測試，深度測試就是指當 OpenGL 在繪製流程時，會依據當時物件所在的 z 軸座標來呈現上下遠近的關係，即使是後畫元件，因為深度測試發現比 3D 程式物件的 z 座標要小，也會繪製在 3D 物件之後而被遮住，所以當 CWT 在繪製時要關掉深度測試，以確保 CWT 元件會繪製在最上方不會被遮蔽。

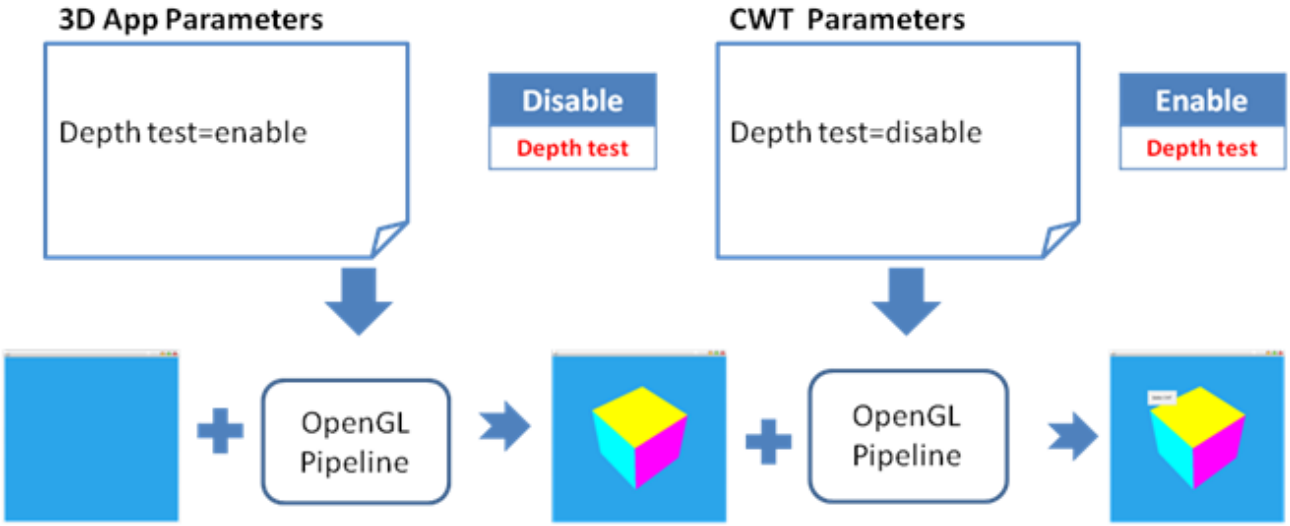


圖 3-14：3D 圖形混合 2D CWT 元件繪圖流程架構圖-關閉深度測試

3.2.2 位置矩陣參數回復

開發者在繪製 3D 畫面時，會依據需要使用許多 OpenGL 繪圖流程的參數設定，而輪到要繪製 CWT 時會改用 CWT 的繪圖設定如投影、視點、

視埠等，但是下次繪製 3D 畫面時，如果開發者沒有再次設定要使用的參數，則 OpenGL 就會用現有最新的也就是使用 CWT 的參數去做繪製，這樣畫面就會出錯，而我們為了能讓開發者能夠像使用 AWT 一樣撰寫程式，以達到增加產能，開發者不需要因為使用 CWT 代替 AWT 而對程式進行修改，例如：每次要繪圖時都重新設定參數，因為 AWT 並不會影響原來的 3D 繪圖參數設定。

本論文設計使用 PushMatrix 跟 PopMatrix 將 3D 繪圖的參數在要畫 CWT 之前存起來，等 CWT 畫完之後，再將繪圖參數取出設定回 3D 程式，如圖 3-13。

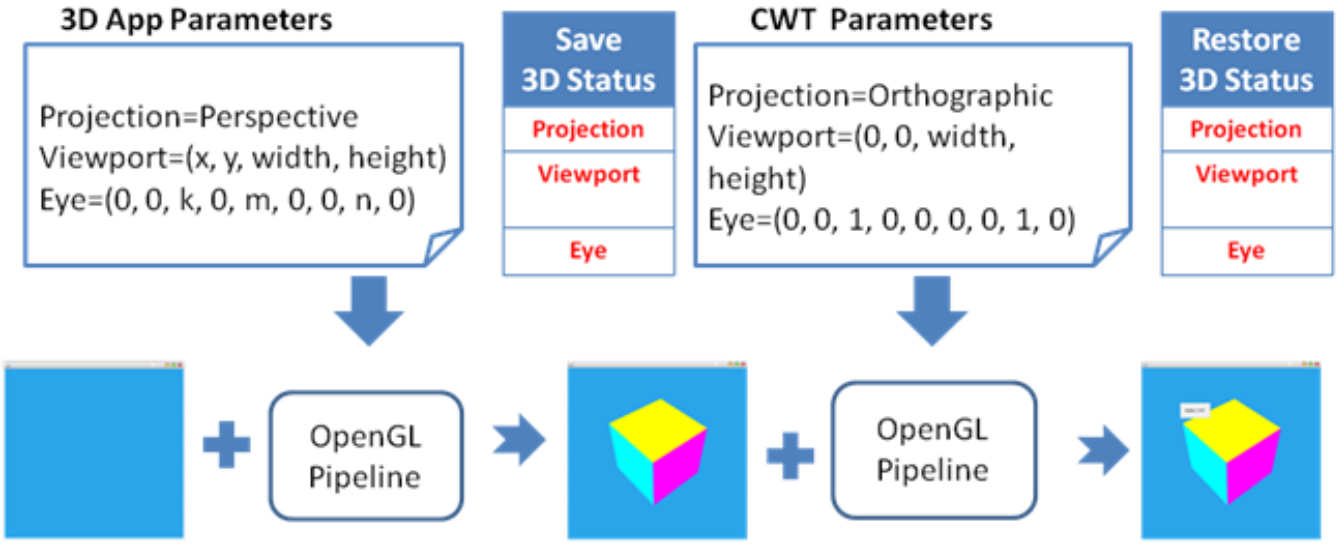


圖 3-15：3D 圖形混合 2D CWT 元件繪圖流程架構圖-回復 3D 繪圖參數

3.2.3 打光問題

在 3D 繪圖中，常會利用打光來讓物體產生陰影及遠近效果，使畫面

更為逼真，但是 GUI 一定都是清楚顯示在畫面上且為 2D，所以並沒有打光的機制，如果加上打光，有可能會因為 CWT 元件的座標在光源後面，因背對光源而造成畫面顯示的不正常，所以在繪製 CWT 前，必須先判斷 3D 應用程式是否有使用打光，如果有，則將打光取消，等 CWT 繪製完畢後再打開，如圖 3-14。

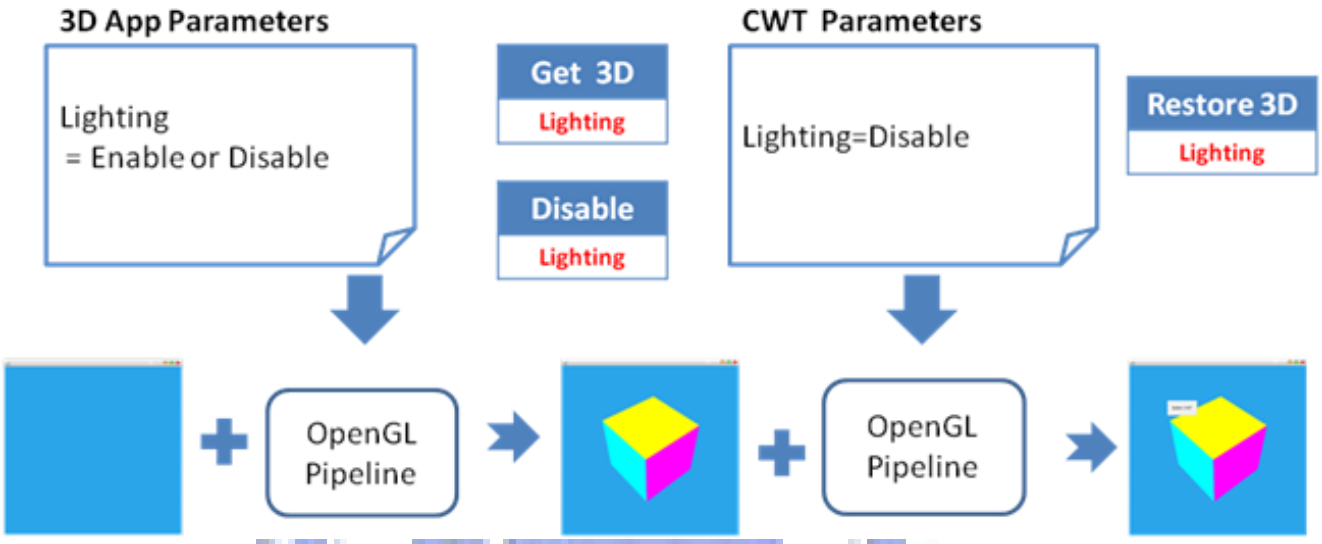


圖 3-16：3D 圖形混合 2D CWT 元件繪圖流程架構圖-關閉光源設定

3.2.4 裁切問題

裁切(Clip)是在大量繪製圖形中取出所需要部分的重要功能，像是在繪製 3D 方塊時，會把 3D 方塊以外的部份裁切掉，如 CWT 的 Button，但是當 CWT 繪製 Button 前要先將原來 3D 程式的裁切設定存起來，然後重新設定裁切的範圍，等繪製完後再恢復原來 3D 程式的裁切設定，如圖 3-15。

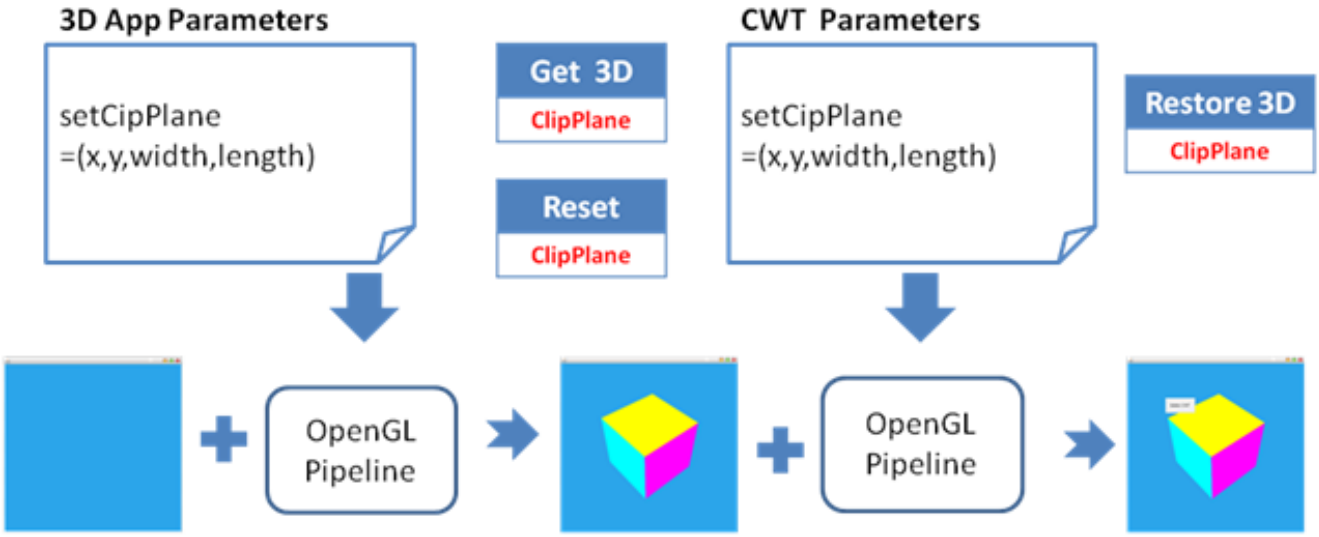


圖 3-17：3D 圖形混合 2D CWT 元件繪圖流程架構圖-重設裁切參數

最後本論文列出所有要改變的3D繪圖參數跟相關的函式，如表3-1，最右邊的CWT繪圖所使用的參數設定，是固定不可以改變的。

狀態	3D 繪圖設定	CWT 繪圖設定
視點	PushMatrix gluLookAt (0,0,10, 0,0,0, 0,1,0)	PopMatrix gluLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0);
投射	PushMatrix gluPerspective(0, width, height, 0)	PopMatrix gluOrtho2D(0,width,-height,0)
視埠	PushMatrix glViewport (x, y, width, height)	PopMatrix glViewport(0,0,width,height);
深度測試	glIsEnabled(GL_Depth_test)	glDisable(GL_Depth_test)
光源	glIsEnabled (GL_LIGHTING)	glDisable(GL_LIGHTING)
裁切	glGetClipPlane	glSetClipPlane

表 3-1：3D 圖形混合 2D CWT 元件繪圖參數設定函式表

第四章、 實驗數據分析與討論

本章的4.1節將會先說明實驗環境,實驗項目與所得到的實驗數據。4.2節將會就得到的實驗數據座分析與討論。

4.1 實驗環境

本實驗目的是要測試當3D繪圖程式使用AWT、Swing及CWT當做UI介面時的繪圖效能。表4-1則是本實驗的軟硬體環境。

中央處理器	Intel Core Duo T2500 2.0GHz
記憶體	2G DDR2 667
繪圖卡	ATI Mobility Radeon x1600 128MB VRAM 支援 OpenGL 2.0 螢幕解析度 1280x800
作業系統	WinXP SP3
Java 版本	1.6.0_07

表 4-1：實驗軟硬體設備表

本次實驗所使用的3D測試程式，是一個不斷在旋轉的立方體，代表

3D 繪圖的內容，並且完全使用表 3-1 所使用到的參數，此方塊六個面的顏色都不一樣。而在 2D CWT-GL 部分，則會分別測試在同一個畫面上，也就是在旋轉的方塊放上 1 個元件，在這邊我們選擇用（Checkbox），與放上 60 個元件如圖 4-1。為何會選擇 60 的原因是因為現今大多數的 3D 遊戲 GUI 介面大多最多 50 左右，所以本實驗測試選擇放置 60 元件到 3D 畫面上來測試是否繪圖效能會受到影響。

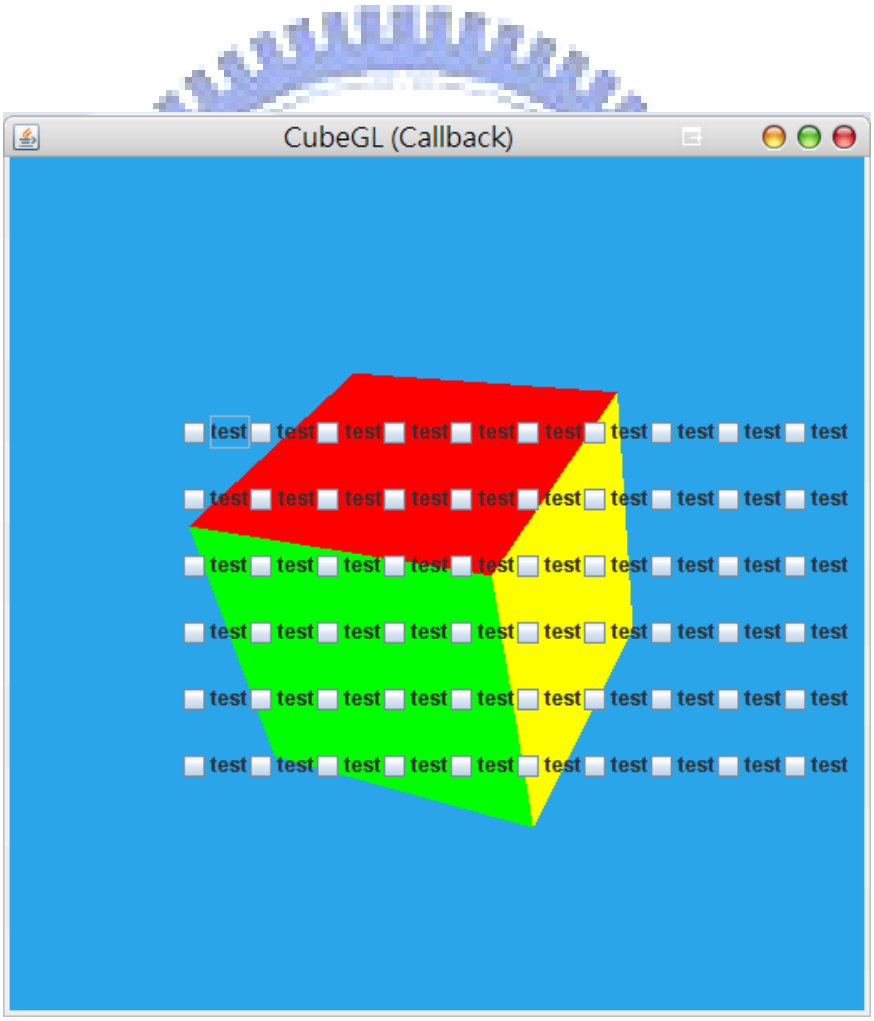


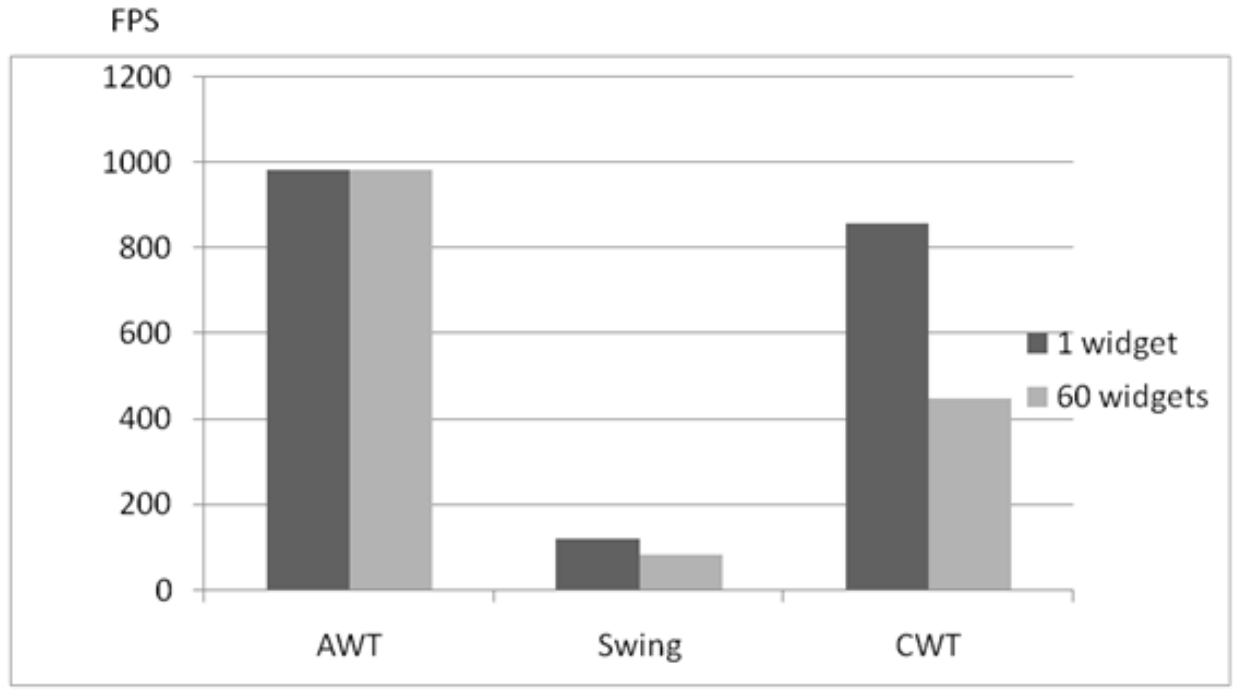
圖 4-1：3D 圖形混合 2D 元件實驗畫面

此外，因為現今 3D 遊戲 GUI 的設計多樣式，並部會固定使用原來就

設計好的 AWT 元件樣式，所以本實驗也會測試元件是否可以做透明及形狀的改變這些效果。

4.2 實驗數據分析

在這次實驗中測得實驗數據表 4-2，y 軸表示的是每秒鐘畫面畫了幾次（Frames per second，以下簡稱 FPS），當 FPS 值越高時代表繪製的效能越好，x 軸則列出三組數據 AWT、Swing、CWT，並個別測試在 1 個元件及 60 個元件時的 FPS 變化，最下面那排，則表示這三種方法是否可以做到透明及不規則形狀的效果。



Irregular & translucent	X	O	O

表 4-2：實驗數據分析長條圖

首先 AWT 的速度最快，不管是繪製 1 個元件或是 60 個元件其 FPS 都達到 900 多，這是受惠於重型元件由作業系統生成，而本程式都是由 OpenGL 所繪製，在測試時所得到的數據相當於是方塊本身繪圖的時間而已，並沒有加上 AWT 元件火繪製花費的時間，但相對的，重型元件的缺點是不能做透明及矩行以外的形狀改變，在現在講求畫面的 3D 遊戲上幾乎沒辦法使用。

而 Swing 因為是輕形元件，所以可以做到透明跟形狀上的改變，但必須使用軟體繪製在別的 Frame buffer 上，再複製到 OpenGL 的 Frame buffer，而且在繪製的過程還要做切換，Java 繪圖函式又無法使用繪圖加速卡的功能，所以 FPS 只有 120，而當元件增加到 60 個時，繪製速度會低於 100 降到 85 附近。

本論文所整合的 CWT 元件系統，因為是使用 OpenGL 實做，所以做到透明或是形狀改變都沒有問題，而且藉由 OpenGL 的繪圖加速卡加速，在單一元件時 FPS 可以達到 860 是 Swing 的 7 倍多，在多元件的情況下，FPS 下降到 460，幾乎是原來的一半但是與 Swing 相比，是 Swing 的 5 倍多，證明即使在多元件使用時，仍然可以得到比 Swing 更好的繪圖校效能。

本實驗證明了 CWT-GL 的元件可以與 3D 繪圖混合使用，並且得到比

Swing 更好的效能，同時因為 CWT-GL 可以做到元件透明或是不規則形狀的效果，所以比 AWT 更為實用。與純 OpenGL 相比，CWT-GL 已實作出了現成的元件庫，讓開發者可以直接使用，在產能方面得到更佳的結果。

第五章、 結論與未來展望

本論文的目標是使 CWT 能與 3D 畫面做一個整合，讓 3D 遊戲開發者能有現成的元件庫可以使用，且提昇 Java 基本繪圖函式庫的效能。經過實驗證明，CWT 的元件系統確實能與 3D 繪圖畫面共同使用，而且可以做到透明及不規則形狀的效果，在效能測試上，更可以提高使用 Swing 作為 GUI 的 3D 繪圖畫面效能到 5 倍甚至是 7 倍之多，達到我們開發遊戲所需要的效能跟產能。

本論文最主要是要證明 CWT 能與 3D 繪圖混用，但要實際使用在現今的 3D 遊戲上面，還要進行更多及更複雜的測試，此外，當元件數量變多時，CWT 的效能會下降許多，在未來這也是可以改進的地方。

期望 CWT 能實際應用於 3D 遊戲繪圖上，也希望藉由本論文在效能上面的提升，能使 3D 遊戲更為進步。

參考文獻

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

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

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

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

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

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

[7] 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

[8] 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.

[9] 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.

[10]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

[11]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.

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

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

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

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

[16]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

[17]鄭欽議," The Study of OpenGL Implementation of CWT" 交通大學資訊工程系, 碩士論文, 2007