國立交通大學

電機與控制工程學系

碩士論文

以數張影像迅速建立一物體之 3D 模型

Rapidly Constructing the 3D Model from Multiple Images of an Object

研 究 生：林桓宇

指導教授：李祖添 博士

杜國洋 博士

中華民國九十四年七月

以數張影像迅速建立一物體之 3D 模型

研究生：林桓宇　　指導教授：李祖添 博士
杜國洋 博士

國立交通大學電機與控制工程系 碩士班

摘要

在本論文中，我們提出了一個既簡便又快速的方法來重建物體的 3D 模型。由於我們使用了目前最爲流行的數位相機來拍攝物體的六視圖影像，因此整體的實驗設備相當便宜且實驗環境也相當的簡單而容易設立。我們提出了一些演算法來處理這些影像，包括背景濾除和各視圖對應等，並進一步地將所有部分串連起來成爲一個完整的流程來進行整個物體之 3D 模型的重建工作。經由四個物體實際重建後的結果顯示，我們所提出的方法確實是相當地迅速，且重建後所展示的模型也十分地逼真，結果顯示所提方法極具實用性。

Rapidly Constructing the 3D Model from Multiple Images of an Object

Student：Huan-Yu Lin　　Advisor：Tsu-Tian Lee

Kuo-Yang Tu

Department of Electrical and Control Engineering

National Chiao Tung University

Abstract

In this thesis, we propose a method to quickly reconstruct the 3D model of an object in a simple and easy way. Because we use the most popular digital camera at present to capture the six-view images of the object, all experimental equipments are quite inexpensive and the experimental environment can be easily set up. We propose some algorithms, such as background-filtering and view-corresponding, to deal with these images, and then link all algorithms to become a complete procedure to reconstruct the 3D model of the object. After reconstructing four objects, results show that the proposed method is indeed quick and the 3D model that has been reconstructed is almost identical to the real object itself. Experimental results show the applicability of the proposed method.

誌　謝

在此由衷的感謝恩師李祖添教授以及杜國洋副教授在研究上所給予的悉心指導和鼓勵，從兩位老師身上學得不僅是專業領域的知識，更學習到做學問與研究的方法，使我受益良多。此外，也感謝所有實驗室的同學們不吝於學業上的切磋和生活上的照顧，使我能充實地度過這兩年的研究生活。當然更要感謝我的父母，因爲有他們的全力支持，我才能無後顧之憂地完成本論文，進而取得碩士學位。

最後僅以本論文獻給我的父母、家人及所有關心我的師長與朋友們，並和他們一起共享這份成果與榮耀。

目錄

圖像目錄

第一章
序論

1.1 前言

隨著科技的大幅進展，人們對於生活水準的要求也越來越高，進而造成視聽感官方面的相關產品與應用有了大幅度地成長。就視覺方面而言，進步最多的不外乎就是 3D 相關技術與產品，其儼然已經成爲了生活中不可或缺的主流元素之一。以電影而言，其實就已經有不少部片子應用了先進的 3D 特效技術來呈現更爲逼真且氣勢磅礴的場景，例如像是知名的魔戒與侏羅紀公園等系列電影。而日本也在很早之前便引入了 3D 技術來製作了不少知名的動畫，後來甚至造成好萊塢的仿效與跟進並製作出了一些知名的 3D 動畫電影席捲全球票房。不過 3D 技術用於動畫上的理由卻與電影不太相同。由於動畫本身獨特的性質，例如像是違反物理原則或是將幻想具像化等等，使它可以辦到電影中難以辦到的事。不過相對的它也有電影中所沒有的缺點，那就是嚴重地喪失空間感且擬真度太低。因此基於上述的種種原因便大量地引入了 3D 技術來加強其優點並改善其缺點，以滿足人們對於幻想與現實間相互調合的期待，例如像是迪士尼的恐龍以及史克威爾的太空戰士等動畫。

而在遊戲方面，更是早已將 3D 技術利用到極致化，不但視其爲

主流標準而廣泛地應用，且更具有越來越複雜的趨勢。不過推究其原因，大部分都不外乎是爲了呈現更加逼真的遊戲場景以達到虛擬實境的效果，特別是像各種的模擬類型遊戲或是第一人稱射擊類型遊戲等，幾乎已經是全數走向非常近乎真實的 3D 互動系統，例如知名的 Counter Strike 和潛龍諜影等遊戲。此外，若我們觀察硬體也可以發現，名列全世界企業營收排行前五名內之兩家生產 3D 繪圖晶片的公司—NVIDIA 和 ATI，所出產的繪圖晶片之核心時脈也以極爲驚人的速度成長。以 NVIDIA 公司的繪圖晶片產品爲例，由早期的 GeForce 2 MX 之 fill rate 爲 700 M texels/s 而記憶體頻寬爲 2.7 GB/s，一直進步到現今的 GeForce 6800 Ultra 之 fill rate 爲 6400 M texels/s 而記憶體頻寬爲 33.6 GB/s，實不難看出彼此之間的差距有多巨大。再以家用遊戲機爲例，最早期任天堂所出產的紅白機只能繪製色彩數不多且簡單幾何圖形的 2D 畫面而已，直到 SONY 出產的 PlayStation 主機後，才開始具備了不錯的 3D 繪圖能力。而近期的家用主機，如微軟所出產的 XBOX 和 SONY 所出產的 PlayStation 2 等，其 3D 繪圖能力更是大幅度地加強，甚至已能將有如 3D 動畫般的場景進行即時地運算處理並繪製於螢幕上，使玩家感受到身歷其境的效果，而這也是長久以來不少人所期待與渴望的目標。

以上所提出的這些例子，在在都證明了 3D 技術已經是現今的主

流技術之一。因爲我們是身處在 4D 的空間中，即 3D 座標空間加上一個時間軸，因此我們理所當然地會想用 3D 技術來呈現一些事物，以便能更簡單且更真實地展示我們所想要表達的內容。而這便是許多人會投入到 3D 技術與虛擬實境等相關研究的原因之一，因此我們的研究當然也不例外。唯有不斷地提高 3D 技術應用，我們的視覺感官才能享受到更爲真實的場景以及各種全新的虛擬體驗，進而提升我們整體的生活水準與樂趣，而這不儘是大多數人的希望，更是人類對於感官享受最爲自然且原始之慾望的追求。

1.2 研究動機

在進一步提出本論文自創的方法前，我們事先閱讀了不少論文以瞭解這方面的技術與進展程度。有許多論文提出了不少不錯的方法，雖然這些方法大部分都不需使用到很複雜且昂貴的實驗儀器，但通常都具有非常複雜的技術背景，例如像是複雜的演算法或是需要具有很高深的理論基礎背景等，因此儘管他們普遍都具有很不錯的 3D 模型重建效果，但卻因複雜度過高而不容易被大量地使用，亦或是因實用性不高而難以廣泛地應用於各方面上，例如可能有些條件過度嚴苛不貼近現實狀況或是處理時間過長而令人無法接受等缺點。然而也有另一派的方法剛好相反，方法極其簡單且容易學習，處理時間也算迅速且使用條件也非常寬鬆，但其致命的缺點就是儀器問題。以較爲多數

所使用的 3D 掃描器而言，其造價十分昂貴，實非一般人可以負擔的起，也因此難以成爲現今所採用的主流重建技術。

基於上述各方法的優缺點，我們開始思索要如何做才能融合兩者的優點並拋棄掉兩者的缺點，換言之，就是使用很便宜的儀器且非常簡單的方法就能迅速地重建一個物體的 3D 模型。這樣子的方法不但不需花費太多經費在儀器上，且處理上也更爲簡單和快速，比較適合在各方面來大量地推廣和應用。對於一般有興趣的人而言，只要參考本論文方法就能在家自己 DIY 且能立即展示出 3D 化的作品成果，甚至若再進一步略加處理的話，要製作出一部簡易的 3D 動畫電影也非難事，而這就是本論文的最大的貢獻之處：讓 3D 技術進入平民化的時代。本論文經過長久的研究之後，也終於不負眾望地達到預期的目標，且重建效果也非常理想，的確是一個既簡易又有效的好方法。

1.3 章節內容概述

除了本章內容已經詳述過了之外，剩餘的各章內容我們將在此一一進行概述。在第二章部分我們將對於 3D 相關背景技術進行說明，並且比較和討論其他論文中所提出的相關重建技術，藉以凸顯本論文方法之優點所在。而在第三章中，我們將提出我們所自創的方法，在此章中會詳述相關的實驗環境、演算法以及步驟流程，期能詳盡地描

述本方法之核心精神以及實際上的重建做法。至於在第四章的部分，我們則是針對了四個物體實際地進行了 3D 模型的重建工作，並展示其相關結果與比較，以便驗證本論文所提出的方法確爲有效且可行。最後在第五章的部分則是提出我們的整體結論，並探討未來還有哪些方向能再更深入的進行研究，以使整個 3D 技術能繼續往前邁進一大步。

第二章

相關研究背景與知識

2.1 傳統 3D 模型重建方法概述

一般而言，傳統建立 3D 模型的方法不外乎是經由人力並利用相關 3D 繪圖軟體，例如 3dsmax 或 Maya 等，來慢慢繪製出所想要的物體模型，之後再經過材質貼圖、打光與混疊等處理程序之後，才能形成螢幕上所看到的模型樣貌。因此可想而知，其製作往往耗時良久，不但沒有效率且對繪製師而言也是一大負擔。一旦場景變的更爲巨大而所要繪製的模型更多的話，常常會出現人力嚴重不足或是製作時間極爲冗長等問題。此外，此方法對於繪製師的經驗與技術也相當地講究，因爲這兩個因素將嚴重影響到模型的製作時間以及逼真程度，也因此可以看出其對人力素質的要求頗高，而非人人都是可以勝任這項工作的。雖然以上不少缺點是嚴重的致命傷，但它仍具有更爲出色的優點，那就是可以將模型製作到極爲細緻且逼真，而這也是直到現在此方法仍未被時勢潮流所淘汰的主要原因。

2.2 自動化 3D 模型重建之相關研究方法概述

除了傳統的方法外，大部分論文研究的重心都在於如何利用一些儀器與演算法來達到自動化重建 3D 模型的目的。既然名曰自動化，想當然爾就是以電腦來進行大部分的運算處理工作，因此跟傳統人工

的方式比較起來，它具有製作時間短、人力需求量少以及學習門檻低等優點，而這些優點顯然與傳統人工方式的缺點互補，因此我們可以預見其缺點當然不外乎是製作出來的模型細緻程度不足，進而給人感覺與原物體並不太相像，換言之，即不夠逼真。此外像是儀器昂貴與封閉面等問題，也深深的影響它在實用上的價值。不過隨著技術的成長，這些缺點也一直在改進中，相信會有那麼一天，自動化重建技術會變的更爲普及且成爲大多數人所使用的主流技術之一。

大部分的自動化重建技術可以概略粗分爲兩派，一派爲使用 3D 掃描器或是一些較爲複雜的儀器並外加一些演算法處理來達成重建 3D 模型的目的，但其缺點爲儀器複雜且昂貴，因此只有在足夠的金錢奧援下才有可能進行，而這對一般有興趣從事相關工作的人是一個很高難度的門檻，故其實用價值較低且較少人研究，例如：[1]、[2]、[3]、[4]、[5]、[6]、[7]、[8]、[9]中所使用的相關實驗儀器與設備。至於另一派則是使用較爲簡便的儀器來蒐集物體的影像資料，接著進一步利用影像處理或是一些較爲複雜的演算法來重建出物體的 3D 模型。由於此法不需昂貴的儀器，且再加上現今電腦的處理能力都極爲強大，因此也成爲較多人所研究的主要對象。雖然目前已經有不少各式各樣的方法被提出，但普遍都還是具有一定的複雜性和困難度存在，故對一般人而言仍然是難以學習和應用的方法。此外，大部分方

法所重建的模型往往具有不少的缺陷，像是失真和遮蔽效應等問題，也因此間接導致其遲遲無法廣泛地應用於各方面上。這方面的論文相當的多，在此僅列舉數個例子做爲參考，例如：[10]、[11]、[12]、[13]、[14]、[15]、[16]、[17]、[18]中所提出的相關演算法。

至於本論文雖然在分類上是屬於後者，但與後者的重大差異之處在於本論文所使用的實驗器材不但便宜且方便取得，且使用的方法也極其簡單易懂，因此無論是在簡易性或是實用性等方面都明顯地具有非常高的價值性，這便是本論文的重要貢獻之所在。

第三章

簡便且快速地重建 3D 模型之流程與方法

3.1 儀器設備與實驗環境

誠如研究動機中所述，本研究的重點著重於簡便且快速地重建物體之 3D 模型，故整個儀器設備與實驗環境也都是極其簡易，既不需要使用到昂貴的儀器，也不需要花太多時間設立整個實驗環境。全部實驗所需要的器材只有以下數項：(1)一台數位相機，在此是使用 canon 公司生產的 IXUS 400 進行拍攝， (2)一張單色背景壁紙，在此我們是選用純黑色壁紙來當背景， (3)一個所欲重建 3D 模型之物體，在此使用手機、釘書機、修正帶與小時鐘等四種物體來作爲實驗的材料。

一般來說，實驗中若是有使用到相機進行攝影的話，大家通常都會先進行一項所謂的鏡頭校準(Camera calibration)之工作。不過由於目前的數位相機功能都已經相當強大且非常適合在各種環境下使用，因此我們可以略過這一步驟逕自進行實驗。雖然這的確有可能會影響實驗之效果，不過從我們之後所得的實驗結果證明了其影響極其輕微，故略過此部分繼續進行並無任何不妥之處。

有了上述三項實驗所需之器材後，我們將物體置於黑色壁紙前，

並將相機鏡頭面對物體且調整好其角度與位置後，便可開始整個一連串的拍攝程序，而整個拍攝環境就如圖 3-1 所示。由於我們必須要拍攝物體的六個面共六張影像來進行模型重建，因此有一點必須注意的是每個面向鏡頭之物體面的最突出部分需與鏡頭保持相同的距離，否則由於景深不一致的關係將會導致重建後的模型嚴重失真。不過因爲相機在拍攝過程中都是固定不動，所以我們可以在距離相機前一定的位置處繪製一條與相機平行的水平線，然後將物體平行置於此線上後，再將物體向背景壁紙處平移，直至其所要拍攝面之最突出的部分剛好在此線上爲止，這樣便可達成上述的要求。

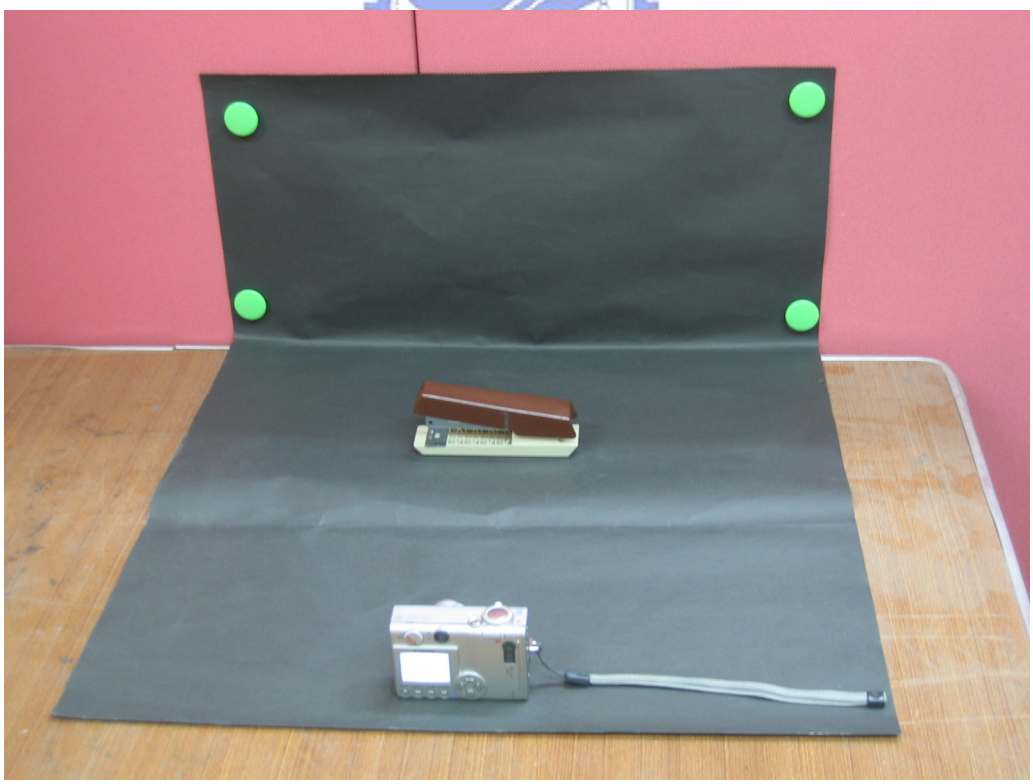


圖 3-1 拍攝環境

除了前面所提的景深問題外，還有圖像會有左右平移的問題發生。在我們整個實驗中，六個面的拍攝都是利用手來對物體進行轉動，雖然這麼做很方便且快速，但卻造成了物體之 X 軸中心點無法精確地與相機中心點對齊而產生了圖像左右位移的結果，而這將導致模型嚴重失真，因此不容我們忽視。話雖說如此，但由於在此拍攝階段實在是無法有效地解決這個問題，故我們將留待之後再進行處理，至於其詳細的處理方法則請參閱 3.4 節。

3.2 全部重建流程概略一覽

當物體之六視圖影像(前、後、左、右、上、下等六個視圖的影像)拍攝完畢後，接下來的全部工作便都是藉由電腦來運算並完成 3D 模型的重建和展示，因此無可避免的會有影像處理以及一些演算法需要進行運算。爲了方便一覽起見，我們先將整個重建的流程以及說明概略地描述於下，至於每一部分的詳細內容，則請參閱 3.3 節至 3.5 節。

3D 模型重建步驟(編號順序代表進行的步驟順序)：

1) 六視圖影像拍攝：利用 3.1 節所述之實驗器材與環境進行，每一物體皆拍攝各視圖的照片共六張。

2) 背景濾除：利用影像處理的方法將六張影像的背景濾除，並保留下物體的影像資料，以提供下一步驟所需之物體的形狀與色彩資

訊。影像處理部分使用的核心方法爲自我背景濾除法，其詳細內容請參閱 3.3 節。

3) 由物體之影像資訊建立色彩矩陣與模型矩陣：將已經濾除掉背景的六張影像依據其所代表之物體面來決定以何種方法對應至代表物體形狀的三維模型矩陣以及代表物體顏色的三維色彩矩陣中。在此步驟裡，由於每一張影像都包含著兩部分不太相同的處理方式，因此，又再度地細分爲七個處理步驟，而每張影像就各自對應一個部分。在這七個處理步驟中，大多數都是處理兩個問題，一爲影像與模型邊界找尋，另一爲影像資訊與兩矩陣間之映射關係，其詳細內容請參閱 3.4 節。

4) 3D 繪圖與動作控制：有了完整的色彩與模型矩陣後，接下來我們只要分析模型矩陣的各元素值來決定這是否代表一個點(Vertex)，便可將其矩陣維度資訊轉變爲空間中的三維座標資訊，並賦予其色彩值。最後利用 OpneGL 這套 3D 繪圖引擎將這些點以透視圖法(Perspective view)繪製到螢幕上並增加鍵盤與滑鼠等控制功能後，就能讓我們以動態的方式來展示重建後的 3D 模型成果。至於其細節內容請參閱 3.5 節。

3.3 背景濾除

由於一般數位相機所拍攝的影像尺寸都在 640*480 像素之上，若是直接使用這種尺寸的影像來進行後續處理的話，將會使得電腦的運算量過爲龐大，不論是在影像處理部分或是兩個三維矩陣處理部分，甚至是最後的 3D 繪圖部分。而每一部分皆需進行龐大運算之加成後的結果將使得整個模型重建至完成所需的時間大幅地增加，而這和我們先前所希望的能迅速地重建物體 3D 模型之目標有所抵觸，故在此我們先將影像由 640*480 縮小爲 160*120 的大小，然後再進行後續的動作。當然，若將影像縮小至 320*240 的大小來進行處理的話，經由實際實驗結果顯示其運算時間也還在可以接受的範圍內，雖然這樣將會使得影像品質提升，進一步地使得重建後的模型失真較少且看起來更細緻，但其缺點則是造成整個運算所需之時間會以非線性關係急遽地成長。因此影像品質與運算時間兩者之間必須加以取捨，而其如何決擇則視我們的需要與應用而定。

有了這六張 160*120 合適大小的影像後，接下來的工作便是濾除背景並留下物體的影像資訊。由於我們的實驗是使用黑色壁紙來作爲背景，因此一般人會直覺地想到利用臨界值法來進行影像處理，而其作法爲先拍攝一張純粹只有背景而沒有物體的影像，然後再拍攝一張包含物體與背景的影像，接著比對兩張影像各像素的色彩值。若兩者

之色彩值相減後的絕對值大於我們所定義之臨界值的話，就將其視爲物體影像資訊並複製其色彩值至欲輸出的影像中，反之則視爲背景影像並寫入我們所想要的代表色彩值至欲輸出的影像中。由於兩張影像之背景部分的色彩值差異很小，但物體與背景色彩值差異卻很大，因此，只要能設定適當的臨界值便能達成背景濾除的目的。影響此方法之成敗有兩個關鍵：一爲兩張影像背景部分之色彩成分的差異性如何？差異越小當然濾除的效果也越好。若差異過大，則將使得臨界值難以設定，甚至是無法設定，進而造成整個濾除效果不佳或失敗。因爲無論臨界值的設定爲何，總是會造成部分的背景影像被殘留下來，要不然就是發生物體某部分影像被不當濾除的現象。而另一關鍵則是臨界值的選定。要是其選的不好的話，一般而言都只是影響到背景濾除之成效而已，但更有甚者也不乏有剛剛所述之兩種狀況發生的可能性。以下所列即爲臨界值法之演算法。

演算法一：臨界值法演算法

```
FOR ALL image i= 0….numberOfImages-1 DO
  IF |BackgroundImageColor(i)-ObjectImageColor (i)|<= ThresholdColor
  THEN
      OutputImageColor(i)= BlackColor
  ELSE
      OutputImageColor(i)= ObjectImageColor(i)
  END IF
END FOR
```

在實際所進行的實驗當中，我們發現到一個現象就是同一個地方，但在不同影像中的亮度往往會有差異，更有甚者，其亮度差異已經嚴重到肉眼可輕易辨識出的地步。即便是同一張影像的背景部分，其亮度也並非是均勻分佈，而這些影像在經過臨界值法處理後，如預期般地，其結果不是背景濾除的效果很差，要不然就是有部分物體影像被濾除掉而宣告失敗，所以幾乎沒有影像能成功地達到我們的要求，而這很明顯的是因爲採用了錯誤的方法。即使我們將 RGB 的色彩空間轉換至 HSI 的色彩空間來進行處理，其結果一樣是令人失望。因此在這裡我們提出一套自創的新方法以符合我們所要求的目標，而這將在下一段中進行詳述。至於造成上述影像亮度有所差異的原因，其不外乎以下兩點因素所導致：

(i) 數位相機自動化功能：由於數位相機本身內建有白平衡校正以及曝光補償等自動化功能，因此每張影像在拍攝時，相機就會根據當時環境狀況來自動調整這些功能的相關參數。而這些內部參數是我們無法得知的，更別提想要調整或是關閉這些功能，故很遺憾的是此問題並不具有任何有效的解決方法。

(ii) 環境光源不穩定：理論上來說，本實驗應該在具有穩定且受控制的光源條件下進行，但誠如我們所強調的要以簡便的方

式來進行模型之重建，故嚴格控制光源的作法將有違當初宗旨。因此我們的實驗蓄意選在一個具有數支日光燈照明的一般室內，其光源條件完全不加以限制或控制，而這才是最通常且最容易建立的實驗環境。然而，眾所皆知的是日光燈並不是一個穩定的光源，它是以人眼所無法察覺之頻率不斷地進行閃爍來達成照明的目的。換言之，其亮度並非恆定(即亮度會隨時間而改變)，因此在多支日光燈與物體以及周遭環境之所有反射、散射以及漫射等光源條件交互加成作用下，整體的光源狀況就會變得極爲複雜且隨時異動著。

在我們所提出的自我背景濾除法中，其與臨界值法比較後的重大差異在於我們可以只要針對某一張包含物體和背景的影像進行運算便可將其背景部分直接加以濾除，因此完全不需要比對另一張只有純背景的影像便可達到我們所想要的目的。這樣做的好處有兩個，一爲我們可以少拍攝一張影像，進而避免花費多餘的運算在這上面；另一爲可以避免掉處理兩張影像間之亮度差異所造成的問題。然而無論我們選用的是何種方法，其還是存在著兩個問題需要去面對。首先第一個問題是一旦物體色彩與背景色彩很接近時，這些接近的部分就很有可能會被視爲背景而不當地被濾除掉。這個問題很遺憾的是兩種方法都無力解決，因此只能依靠手動的方式來修正影像。話雖說如此，但

兩者對於其色彩接近之程度其實還是有很大的差異。就自我背景濾除法而言，其色彩要十分接近才會被濾除，然而臨界值法之狀況爲只要色彩相似就有可能被濾除掉。至於第二個問題則是同一張影像內亮度分佈不均的現象，而這也是兩者共通的毛病之一。不過同樣地，彼此仍然有決定性的差異存在。就自我背景濾除法而言，由於同一張影像之亮度分佈雖不均勻，但不至於有差異極大的狀況發生，因此我們可以只要針對這個特性來加以處理即可。反觀臨界值法不但一樣需要處理此種問題外，還得面臨不同影像間也會有亮度差異的問題，造成其變得更爲複雜且難以處理。若我們比較一下兩種方法便可發現，無論是在處理難度上亦或是最後濾除的成效上，自我背景濾除法都是優於臨界值法的。

自我背景濾除法主要的核心方法是先計算由我們所決定範圍之一部份背景影像的平均值，然後根據此平均值與其他影像資料進行比對。若是其他影像資料與此平均值之和小於我們所設定的臨界值時，就將其視爲背景影像並且與平均值一起併入計算成爲新的平均值，然後再寫入所定義的色彩值（在此我們所定義的色彩值爲黑色）至輸出影像的對應處；反之，則視爲物體影像而複製其色彩值至輸出影像的對應處。下段所列即爲自我背景濾除法之演算法。

演算法二：自我背景濾除法

```
Vnum=1
FOR ALL image i= 0….numberOfImages-1 DO
    IF i<=InputImageRowNumber TEHN
        OutputImageColor(i)= BlackColor
        ColorMeanValue= [ColorMeanValue+InputImageColor(i)]
                        / Vnum
        Vnum= Vnum+1
    ELSE
        IF InputImageColor(i)+ ColorMeanValue<= ThresholdColor
        THEN
            OutputImageColor(i)= BlackColor
            ColorMeanValue= [ColorMeanValue+InputImageColor(i)]
                            / Vnum
            Vnum= Vnum+1
        ELSE
            OutputImageColor(i)= InputImageColor(i)
        END IF
    END IF
END FOR
```

經由實際以此演算法對影像進行處理後，我們發現到大部分影像的背景濾除效果都很不錯，但仍然有少數效果並不是很理想，而這原因最主要是由於同一張影像中的亮度差異過大所導致。要解決此問題有兩種方法，一爲利用繪圖軟體進行人爲手動修正，另一爲使用更爲複雜的演算法來進行處理以使效果達到最佳化。我們在此主要採用前者的方法，因爲需要手動修正的影像不但甚少，且其還具有工作簡單、處理迅速、成效良好和不需花費額外運算量等優點。反觀後者就需要龐大的額外運算量，進而造成整個處理效率的低落，且其成效是

否能比前者還好也是相當地令人懷疑。

3.4 由物體之影像資訊建立色彩矩陣與模型矩陣

由於我們使用六張影像來進行重建，因此我們至少需要六個處理步驟來對應處理各張影像。不過由於我們所提出的這些方法會彼此互相影響，故這裡需要七個獨立的處理步驟才能處理好所有影像的對應關係，而這七個獨立處理步驟中又有四個處理步驟需要重複使用兩次，因此最後結果便是需要總計十一個處理步驟才能完成整個模型重建的工作。至於這七個獨立部分我們將分別於 3.4.1 至 3.4.6 等各小節中來詳細說明，而重建的十一個步驟則於 3.4.7 小節中詳述。

爲了方便之後的說明起見，我們在此首先定義兩個三維矩陣，一爲色彩矩陣，其各元素之值即代表某一色彩值，另一則爲模型矩陣，其元素值代表是否爲有效的物體點，若爲 1 則代表是一個有效的物體點，反之則爲 0。而這兩個矩陣其實就是構成物體 3D 模型的主要依據，模型矩陣的功能主要是用來形成物體之 3D 模型部分，至於色彩矩陣則是用來呈現模型的顏色，其類似於材質貼圖的功能，唯有兩者合併才能形成並展示出一個幾可亂真的物體模型。關於這兩個三維矩陣，我們覺得有必要先清楚地界定其座標特性以避免後續可能發生混淆的情形，因此在這裡建議將這兩個三維矩陣想像成兩個立方體，兩

者之 X 座標軸的零點在最左方，向右之座標值則依序增加；Y 座標軸的零點在最下方，向上之座標值則依序增加；Z 座標軸的零點爲最遠離視點處，向視點拉近之座標值則依序增加。整體之座標示意圖就如圖 3-2 所示。

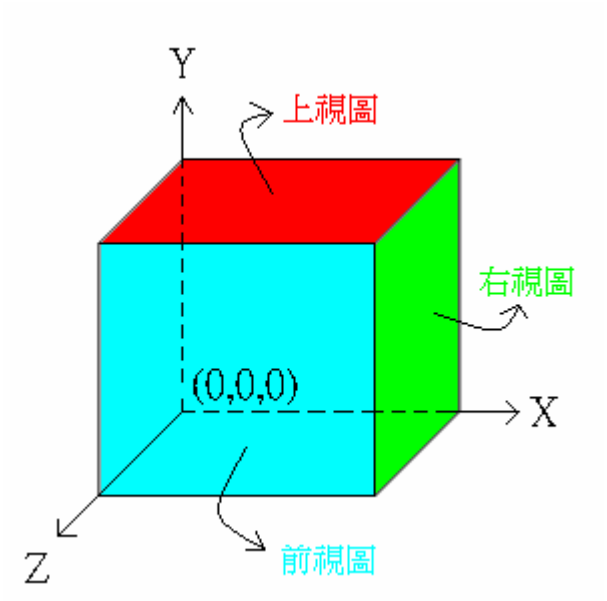


圖 3-2 座標示意圖

概略而言，以下七個小節不外乎是處理兩個問題，一是之前 3.1 節中所提及的影像位移問題，另一則爲影像與上述兩個矩陣之間的對應關係。若是以較爲具體的例子來描述的話，我們可以將其想像成如何將一塊原本是長方體的木頭（模型矩陣）雕刻出我們所要的物體形狀（3D 模型），然後再將數張材質貼紙（色彩矩陣）移動到正確的地方並貼到已雕刻好的木頭表面上去，而這就是以下各部分大致上所要進行的工作。

3.4.1 前視圖處理

由於此部分爲所有步驟的開頭，因此在還沒有任何 3D 模型的狀況下，我們可以略去處理影像位移的問題而直接將重心放在影像與兩矩陣間的對應關係即可。既然我們在此部分所使用的是前視圖的影像，那麼彼此之間的對應關係自然就是影像的 X 座標軸零點順向對應兩矩陣的 X 座標軸零點，且 Y 座標軸零點順向對應兩矩陣的 Y 座標軸零點，而兩矩陣中具有相同 XY 座標值但不同之 Z 座標值的全部元素則根據對應的影像資訊同時改變成某一相同的數值。在這個對應關係下，若影像的色彩值大於我們所設定之臨界值的話，換言之即代表物體影像，則將色彩矩陣中相對應位置處但不同 Z 座標值之所有的元素寫入此色彩值，且同時將模型矩陣中相對應位置處但不同 Z 座標值之所有元素的值設定爲 1；反之，則在與上述相同之處將兩矩陣之元素值設定爲 0。至於詳細之演算法則如下所示。此處所指順向對應的意思爲若其中一方之值增加的話，則另一方也以相同的數值增加；反之，若其中一方之值減少，則另一方也以相同的數值減少。同樣的，所謂反向對應的意思是其中一方之值增加的話，則另一方就以相同的數值減少；反之，若其中一方之值減少，則另一方就以相同的數值增加。

演算法三：前視圖處理演算法

```
FOR FrontViewImage a= 0….numberOfImageColumns-1 DO
    FOR FrontViewImage b= 0….numberOfImageRows-1 DO
        IF FrontViewImageColor(a,b)<= Threshold TEHN
            FOR ChangingAxis c= 0….numberOfImageColumns-1 DO
                ModelMatrix(a,b,c)= 0
                ColorMatrix(a,b,c)= 0
            END FOR
        ELSE
            FOR ChangingAxis c= 0….numberOfImageColumns-1 DO
                ModelMatrix(a,b,c)= 1
                ColorMatrix(a,b,c)= FrontViewImageColor(a,b)
            END FOR
        END IF
    END FOR
END FOR
```

3.4.2 左視圖處理

由於 3D 模型一樣仍然尚未建立好，所以我們還是可以繼續略去處理影像位移的問題。整體而言，其跟前視圖的方法其實非常相近，差別只有在於影像與兩矩陣間對應的方式有所改變而已，而這也是理所當然的，因爲我們在此是使用了左視圖的影像，因此有必要再將對應關係重新進行調整，使得影像的 X 座標軸零點順向對應兩矩陣的 Z 座標軸零點，且 Y 座標軸零點順向對應兩矩陣的 Y 座標軸零點，而兩矩陣中具有相同 YZ 座標值但不同之 X 座標值的全部元素則根據對應的影像資訊同時改變成某一相同的數值。同樣地，在這種對應前提下，若影像的色彩值大於我們所設定之臨界值，則將色彩矩陣中相對

應位置處但不同 X 座標值之所有元素寫入此色彩值，且同時將模型矩陣中相對應位置處但不同 X 座標值之所有元素的值設定爲 1；反之，則在與上述相同之處將兩矩陣的元素值設定爲 0。至於詳細之演算法則如下所示。

演算法四：左視圖處理演算法

```
FOR LeftViewImage a= 0….numberOfImageColumns-1 DO
    FOR LeftViewImage b= 0….numberOfImageRows-1 DO
        IF LeftViewImageColor(a,b)<= Threshold TEHN
            FOR ChangingAxis c= 0….numberOfImageColumns-1 DO
                ModelMatrix(c,b,a)= 0
                ColorMatrix(c,b,a)= 0
            END FOR
        ELSE
            FOR ChangingAxis c= 0….numberOfImageColumns-1 DO
                ModelMatrix(c,b,a)= 1
                ColorMatrix(c,b,a)= FrontViewImageColor(a,b)
            END FOR
        END IF
    END FOR
END FOR
```

3.4.3 後視圖處理

有了前視圖和左視圖兩部分的處理後，物體的 3D 模型其實已經初步形成了，因此自此開始以下的所有部分就會與前兩部分差異很大，而其差異點便在於由此之後都必須要處理影像位移的問題。針對這個問題，我們的處理方法分爲找尋影像邊界與找尋模型邊界等兩部分來進行。前者主要是找尋影像中屬於物體影像的最左下方邊界，而

其具體作法也一樣包含了兩個方面，即找尋屬於物體影像之最左方以及最下方的邊界。在找尋最左方邊界之作法爲由零點開始依序順向掃瞄影像中 X 座標軸的部分，若相同 X 座標但不同 Y 座標位置的全部像素之色彩值小於或等於我們所設定之臨界值，此即代表背景部分，則繼續向下一個 X 座標的所有不同 Y 座標像素進行掃瞄；反之，若是在其之中有任何一個像素的色彩值大於我們所設定之臨界值的話，此即代表物體部分，則將此 X 座標值寫入至影像最左邊界值中。至於在找尋最下方邊界之作法上與找尋最左方邊界之作法相近，一樣是由零點開始依序順向掃瞄影像中 Y 座標軸的部分，若相同 Y 座標但不同 X 座標位置的全部像素之色彩值小於或等於我們所設定之臨界值，則繼續向下一個 Y 座標的所有不同 X 座標像素進行掃瞄；反之，若是在其之中有任何一個像素的色彩值大於我們所設定之臨界值的話，則將此 Y 座標值寫入至影像最下邊界值中。

至於在找尋模型邊界的部分上，也一樣是分成兩個方面，即找尋模型矩陣中屬於物體部分的最右方以及最下方之邊界，這與找尋影像邊界的差異之處在於多出了一個座標軸需要掃瞄而已。因此在找尋模型最右方邊界之作法上，可以參考找尋影像邊界的做法並調整如下：由模型矩陣之 X 座標最大值處開始反序掃瞄，若相同 X 座標但不同 Y 座標和 Z 座標的所有元素值爲 0，此即代表不存在有效的物體

點，則繼續對下一群具有相同的 X 座標但不同之 Y 座標和 Z 座標的所有元素進行掃瞄；反之，若是在其中有任何一個元素值爲 1 的話，此即代表有效的物體點，則將此 X 座標值寫入至模型最右邊界值中。類似的作法同樣可用於找尋模型最下方邊界上，其方法仍是由零點開始依序順向掃瞄模型矩陣之 Y 座標軸部分，若相同 Y 座標但不同 X 座標和 Z 座標之所有元素值爲 0，則繼續對下一群具有相同的 Y 座標但不同之 X 座標和 Z 座標的所有元素進行掃瞄；反之，若是在其中有任何一個元素值爲 1 的話，則將此 Y 座標值寫入至模型最下邊界值中。

在接著更進一步地說明處理方法之前，我們還是必須先將影像與兩矩陣間的對應關係予以清楚的定義。因爲這裡採用的是後視圖影像，再加上又有影像位移的問題需一起加以處理，故其必然會跟其他視圖的對應關係有所差異。在此的對應關係爲影像之 X 座標軸的最左邊界點反向對應兩矩陣之 X 座標軸的最右邊界點，且影像之 Y 座標軸的最下邊界點則順向對應兩矩陣之 Y 座標軸的最下邊界點，而兩矩陣相同 XY 座標值但不同之 Z 座標值的元素則根據所對應的影像資訊同時改變成相同的數值。這裡有一個地方必須要加以說明的是，剛剛所言之兩矩陣的最右邊界點和最下邊界點其實就是上段中所提到模型最右邊界值和最下邊界值所對應到的座標點。

就兩矩陣的性質而言，色彩矩陣可以視爲模型矩陣的裝飾物。換言之，模型矩陣是主而色彩矩陣是從，因此理所當然的就是由模型矩陣來決定整個模型的邊界值。在這種對應前提下，若影像的色彩值小於或等於我們所設定之臨界值，則將色彩矩陣與模型矩陣中相對應位置處但不同 Z 座標值之所有元素的值設定爲 0；反之，則先找尋模型矩陣中與影像相同對應位置處之元素值爲 1 的最小 Z 座標值，然後根據這些資訊將影像的色彩值複製到對應座標的色彩矩陣中，如此即完成後視圖部分的處理。至於全部完整之演算法則如下所示。

演算法五：後視圖處理演算法

```
FOR BackViewImage a= 1….numberOfImageColumns-2 DO
    FOR BackViewImage b= 0….numberOfImageRows-1 DO
        IF BackViewImageColor(a-1,b)<= Threshold &&
            BackViewImageColor(a,b)> Threshold    &&
            ImageBoardXFind== 0
        TEHN
            ImageBoardX= a
            ImageBoardXFind= 1
        END IF
    END FOR
END FOR

FOR BackViewImage b= 1….numberOfImageRows-2 DO
    FOR BackViewImage a= 0….numberOfImageColumns-1 DO
        IF BackViewImageColor(a,b-1)<= Threshold &&
            BackViewImageColor(a,b)> Threshold &&
            ImageBoardYFind== 0
        TEHN
            ImageBoardY= b
```

```
            ImageBoardYFind= 1
        END IF
    END FOR
END FOR

FOR ModelMatrix m= numberOfMatrixColumns-2….0 DO
    FOR ModelMatrix n= 0…. numberOfMatrixRows-1 DO
        FOR ModelMatrix o= 0…. numberOfMatrixDepths-1 DO
            IF ModelMatrix (m+1,n,o)== 0 &&
                ModelMatrix (m,n,o)== 1 && ObjectBoardXFind== 0
            TEHN
                ObjectBoardX= m
                ObjectBoardXFind= 1
            END IF
        END FOR
    END FOR
END FOR

FOR ModelMatrix n= 1…. numberOfMatrixRows-1 DO
    FOR ModelMatrix m= 0…. numberOfMatrixColumns-1 DO
        FOR ModelMatrix o= 0…. numberOfMatrixDepths-1 DO
            IF ModelMatrix (m,n-1,o)== 0 &&
                ModelMatrix (m,n,o)== 1 && ObjectBoardYFind== 0
            TEHN
                ObjectBoardY= n
                ObjectBoardYFind= 1
            END IF
        END FOR
    END FOR
END FOR

FOR p= 0…. numberOfImageColumns- ImageBoardX -1 DO
    Flag= 0
    FOR q= 0…. numberOfImageRows- ImageBoardY -1 DO
        Flag= 0
        IF BackViewImageColor(p+ImageBoardX,q+ImageBoardY)
            <= Threshold
        THEN
```

```
                FOR r= 0…. numberOfMatrixDepths-1 DO
                    ModelMatrix(ObjectBoardX-p,ObjectBoardY+q,r)= 0
                END FOR
            ELSE
              FOR r= 1…. numberOfMatrixDepths-1 DO
                IF ModelMatrix(ObjectBoardX-p,ObjectBoardY+q,r-1)==0
                    &&
                    ModelMatrix(ObjectBoardX-p,ObjectBoardY+q,r)==1
                    &&
                    Flag== 0
                THEN
                    Flag= 1
                    ColorMatrix(ObjectBoardX-p,ObjectBoardY+q,r)
                    = BackViewImageColor(p+ImageBoardX,
                                         q+ImageBoardY)
                END IF
              END FOR
            END IF
        END FOR
END FOR
```

3.4.4 右視圖處理

在此部分所進行的處理其實與後視圖處理部分非常的相似，對於影像位移問題也一樣必須加以解決，只是由於對應關係有所改變而必須加以調整。在找尋影像之邊界部分，其與後視圖的作法完全一致，故不再贅述。而在找尋模型之邊界部分，其一樣是分成兩個部分來找尋物體模型部分的最下方以及最前方之邊界。在找尋最下方邊界之作法上與後視圖部分作法完全相同。至於在找尋最前方邊界之作法上，則是由 Z 座標最大值處開始依序反向掃瞄模型矩陣之 Z 座標軸的部分，若相同 Z 座標但不同 X 座標和 Y 座標之所有元素值爲 0，

則繼續對下一群具有相同的 Z 座標但不同 X 座標和 Y 座標的所有元素進行掃瞄；反之，若是在其之中有任何一個元素值爲 1 的話，則將此 Z 座標值寫入至模型最前邊界值中。

在此部分的對應關係爲影像之 X 座標軸的最左邊界點反向對應兩矩陣之 Z 座標軸的最前邊界點，且影像之 Y 座標軸的最下邊界點順向對應兩矩陣之 Y 座標軸的最下邊界點，而兩矩陣相同 YZ 座標值但不同之 X 座標值的元素則根據所對應的影像資訊同時改變成相同的數值。在這種對應前提下，若影像的色彩值小於或等於我們所設定之臨界值，則將色彩矩陣與模型矩陣中相對應位置處但不同 X 座標值之所有元素值設定爲 0；反之，則先找尋模型矩陣中與影像相同對應位置處之元素值爲 1 的最大 X 座標值，然後根據這些資訊將影像的色彩值複製到對應座標的色彩矩陣中，如此即完成右視圖部分的處理。至於全部完整之演算法則如下所示。

演算法六：右視圖處理演算法

```
FOR RightViewImage a= 1….numberOfImageColumns-2 DO
    FOR RightViewImage b= 0….numberOfImageRows-1 DO
        IF RightViewImageColor(a-1,b)<= Threshold &&
            RightViewImageColor(a,b)> Threshold &&
            ImageBoardXFind== 0
        TEHN
            ImageBoardX= a
            ImageBoardXFind= 1
```

```
            END IF
        END FOR
    END FOR

    FOR RightViewImage b= 1….numberOfImageRows-2 DO
        FOR RightViewImage a= 0….numberOfImageColumns-1 DO
            IF RightViewImageColor(a,b-1)<= Threshold &&
                RightViewImageColor(a,b)> Threshold &&
                ImageBoardYFind== 0
            TEHN
                ImageBoardY= b
                ImageBoardYFind= 1
            END IF
        END FOR
    END FOR

    FOR ModelMatrix o= numberOfMatrixDepths-2….0 DO
        FOR ModelMatrix n= 0…. numberOfMatrixRows-1 DO
            FOR ModelMatrix m= 0…. numberOfMatrixColumns-1 DO
                IF ModelMatrix (m,n,o+1)== 0 &&
                    ModelMatrix (m,n,o)== 1 && ObjectBoardZFind== 0
                TEHN
                    ObjectBoardZ= o
                    ObjectBoardZFind= 1
                END IF
            END FOR
        END FOR
    END FOR

    FOR ModelMatrix n= 1…. numberOfMatrixRows-1 DO
        FOR ModelMatrix m= 0…. numberOfMatrixColumns-1 DO
            FOR ModelMatrix o= 0…. numberOfMatrixDepths-1 DO
                IF ModelMatrix (m,n-1,o)== 0 &&
                    ModelMatrix (m,n,o)== 1 && ObjectBoardYFind== 0
                TEHN
                    ObjectBoardY= n
                    ObjectBoardYFind= 1
                END IF
```

```
            END FOR
        END FOR
END FOR

FOR p= 0…. numberOfImageColumns- ImageBoardX -1 DO
    Flag= 0
    FOR q= 0…. numberOfImageRows- ImageBoardY -1 DO
        Flag= 0
        IF RightViewImageColor(p+ImageBoardX,q+ImageBoardY)
            <= Threshold
        THEN
            FOR r= 0…. numberOfMatrixColumns-1 DO
                ModelMatrix(r,ObjectBoardY+q,ObjectBoardZ-p)= 0
            END FOR
        ELSE
            FOR r= numberOfMatrixColumns-2…. 0 DO
                IF ModelMatrix(r+1,ObjectBoardY+q,ObjectBoardZ-p)==0
                    &&
                    ModelMatrix(r,ObjectBoardY+q,ObjectBoardZ-p)==1
                    &&
                    Flag== 0
                THEN
                    Flag= 1
                    ColorMatrix(r,ObjectBoardY+q,ObjectBoardZ-p)
                    = RightViewImageColor(p+ImageBoardX,
                                          q+ImageBoardY)
                END IF
            END FOR
        END IF
    END FOR
END FOR
```

3.4.5 上視圖處理

在此部分所進行的處理一樣與後視圖部分相似，仍舊必須解決影像位移問題和調整對應關係。在找尋影像之邊界部分，其與後視圖

的作法完全一致，因此我們不再贅述。而在找尋模型之邊界部分，其照例是分成兩個部分來找尋物體模型的最左方以及最前方之邊界。在找尋最左方邊界之作法爲由 X 座標零點處開始依序順向掃瞄模型矩陣 X 座標軸的部分，若相同 X 座標但不同 Y 座標和 Z 座標之所有元素值爲 0，則繼續對下一群具有相同的 X 座標但不同之 Y 座標和 Z 座標的所有元素進行掃瞄；反之，若是在其之中有任何一個元素值爲 1 的話，則將此 X 座標值寫入至模型最左邊界值中。至於若要找尋最前方邊界的話，則與右視圖作法完全相同。

而在此部分的對應關係爲影像之 X 座標軸的最左邊界點順向對應兩矩陣之 X 座標軸的最左邊界點，且影像之 Y 座標軸的最下邊界點反向對應兩矩陣之 Z 座標軸的最前邊界點，而兩矩陣相同 XZ 座標值但不同之 Y 座標值的元素則根據對應的影像資訊同時改變成相同的數值。在這種對應前提下，若影像的色彩值小於或等於我們所設定之臨界值，則將兩矩陣中相對應位置處但不同 Y 座標值之所有元素的值設定爲 0；反之，則找尋模型矩陣中相對應影像位置處之元素值爲 1 的最大 Y 座標值，然後根據這些資訊將影像的色彩值複製到對應座標的色彩矩陣中，如此即完成上視圖部分的處理。至於全部完整之演算法則如下所示。

演算法七：上視圖處理演算法

```
FOR TopViewImage a= 1….numberOfImageColumns-2 DO
    FOR TopViewImage b= 0….numberOfImageRows-1 DO
        IF TopViewImageColor(a-1,b)<= Threshold &&
            TopViewImageColor(a,b)> Threshold &&
            ImageBoardXFind== 0
        TEHN
            ImageBoardX= a
            ImageBoardXFind= 1
        END IF
    END FOR
END FOR

FOR TopViewImage b= 1….numberOfImageRows-2 DO
    FOR TopViewImage a= 0….numberOfImageColumns-1 DO
        IF TopViewImageColor(a,b-1)<= Threshold &&
            TopViewImageColor(a,b)> Threshold &&
            ImageBoardYFind== 0
        TEHN
            ImageBoardY= b
            ImageBoardYFind= 1
        END IF
    END FOR
END FOR

FOR ModelMatrix o= numberOfMatrixDepths-2….0 DO
    FOR ModelMatrix n= 0…. numberOfMatrixRows-1 DO
        FOR ModelMatrix m= 0…. numberOfMatrixColumns-1 DO
            IF ModelMatrix (m,n,o+1)== 0 &&
                ModelMatrix (m,n,o)== 1 && ObjectBoardZFind== 0
            TEHN
                ObjectBoardZ= o
                ObjectBoardZFind= 1
             END IF
        END FOR
    END FOR
END FOR
```

```
FOR ModelMatrix m= 1…. numberOfMatrixColumns-1 DO
    FOR ModelMatrix n= 0…. numberOfMatrixRows-1 DO
        FOR ModelMatrix o= 0…. numberOfMatrixDepths-1 DO
            IF ModelMatrix (m-1,n,o)== 0 &&
               ModelMatrix (m,n,o)== 1 && ObjectBoardXFind== 0
            TEHN
              ObjectBoardX= m
              ObjectBoardXFind= 1
            END IF
        END FOR
    END FOR
END FOR

FOR p= 0…. numberOfImageColumns- ImageBoardX -1 DO
   Flag= 0
   FOR q= 0…. numberOfImageRows- ImageBoardY -1 DO
      Flag= 0
      IF TopViewImageColor(p+ImageBoardX,q+ImageBoardY)
         <= Threshold
      THEN
         FOR r= 0…. numberOfMatrixRows-1 DO
            ModelMatrix(ObjectBoardX+p,r,ObjectBoardZ-q)= 0
            END FOR
      ELSE
         FOR r= numberOfMatrixRows-2…. 0 DO
            IF ModelMatrix(ObjectBoardX+p,r+1,ObjectBoardZ-q)==0
               &&
               ModelMatrix(ObjectBoardX+p,r,ObjectBoardZ-q)==1
               &&
               Flag== 0
            THEN
               Flag= 1
               ColorMatrix(ObjectBoardX+p,r,ObjectBoardZ-q)
               = TopViewImageColor(p+ImageBoardX,
                                   q+ImageBoardY)
            END IF
         END FOR
      END IF
```

END FOR
END FOR

3.4.6 下視圖處理

下視圖部分所進行的處理與前幾個視圖的處理方法略有差異。由於在先前的前視圖處理部分已經獲得影像最左邊界以及模型最左邊界與最前邊界等資訊，因此我們只要直接取用這些資訊即可而不用再進行重複的運算，所以剩下的工作就只有找尋影像最上邊界與設定對應關係而已。由於上視圖與下視圖最主要的差別在於前者的影像最下邊界反向對應模型的最前邊界，而後者則是影像最上邊界順向對應模型的最前邊界，因此我們只要找尋影像最上邊界並調整對應關係即可。至於其他邊界找尋部分則是直接由上視圖部分所運算完的相關結果直接匯入至此處，故我們不需再花費額外的運算在那些部分，而這將使得整體的處理時間大幅地縮短。

在找尋影像最上方邊界的作法爲由 Y 座標最大值處開始依序反向掃瞄影像中 Y 座標軸的部分，若相同 Y 座標但不同 X 座標之所有像素的色彩值小於或等於我們所設定的臨界值，則繼續對下一群具有相同的 Y 座標但不同 X 座標的所有像素進行掃瞄；反之，若是在其中有任何一個像素的色彩值大於我們所設定之臨界值的話，則將此 Y 座標值寫入至影像最上邊界值中。至於最後的對應關係則爲影像之 X

座標軸的最左邊界點順向對應兩矩陣之 X 座標軸的最左邊界點，且影像之 Y 座標軸的最上邊界點順向對應兩矩陣之 Z 座標軸的最前邊界點，而兩矩陣相同 XZ 座標值但不同 Y 座標值之元素則根據所對應的影像資訊同時改變成相同的數值。在這種對應前提下，若影像的色彩值小於或等於我們所設定之臨界值，則將兩矩陣中相對應位置處但不同 Y 座標值之所有元素的值設定爲 0；反之，則先找尋模型矩陣中與影像相同對應位置處之元素值爲 1 的最小 Y 座標值，然後根據這些資訊將影像的色彩值複製到對應座標的色彩矩陣中，如此即完成後視圖部分的處理。至於全部完整之演算法則如下所示。

演算法八：下視圖處理演算法

```
FOR BotViewImage b= numberOfImageRows-2….0 DO
    FOR BotViewImage a= 0….numberOfImageColumns-1 DO
        IF BotViewImageColor(a,b+1)<= Threshold &&
            BotViewImageColor(a,b)> Threshold &&
            ImageBoardYFind== 0
        TEHN
            ImageBoardY= b
            ImageBoardYFind= 1
        END IF
    END FOR
END FOR

FOR p= 0…. numberOfImageColumns- ImageBoardX -1 DO
    Flag= 0
    FOR q= ImageBoardY…. 0 DO
        Flag= 0
        IF BotViewImageColor(p+ImageBoardX,q)<= Threshold
```

```
        THEN
            FOR r= 0…. numberOfMatrixRows-1 DO
                ModelMatrix(ObjectBoardX+p,r,
                                ObjectBoardZ+q-ImageBoardY)= 0
            END FOR
        ELSE
            FOR r= 1…. numberOfMatrixRows-1 DO
                IF ModelMatrix(ObjectBoardX+p,r-1,
                                    ObjectBoardZ+q-ImageBoardY)==0 &&
                    ModelMatrix(ObjectBoardX+p,r,
                                    ObjectBoardZ+q-ImageBoardY)==1 &&
                    Flag== 0
                THEN
                    Flag= 1
                    ColorMatrix(ObjectBoardX+p,r,ObjectBoardZ+q-Image
                        BoardY)= BotViewImageColor(p+ImageBoardX,q)
                END IF
            END FOR
        END IF
    END FOR
END FOR
```

3.4.7 前視圖再修正處理

在經歷了以上各視圖的處理後，模型將因爲被不當切割而造成前視圖的對應部分產生錯誤，進而使得整個模型的外觀嚴重失真，因此我們需要一個額外的處理程序來修正相關問題，而這就是爲什麼會多出此一處理部分的原因。至於這部分的做法一樣分爲影像邊界找尋、模型邊界找尋以及處理對應關係等三方面。在影像邊界找尋方面，其與後視圖部分一樣是要找尋影像的最左和最下邊界，故不再贅述。而在模型邊界找尋方面，則是要找尋模型最左以及最下邊界，而

這兩方面也在之前所提過的的上視圖處理部分以及右視圖處理部分中有詳細描述，因此也不再進行重複地敘述。至此，僅剩下調整對應關係的工作需要完成，其做法爲將影像之 X 座標軸的最左邊界點順向對應兩矩陣之 X 座標軸的最左邊界點，且影像之 Y 座標軸的最下邊界點順向對應兩矩陣之 Y 座標軸的最下邊界點，而兩矩陣相同 XY 座標值但不同 Z 座標值之元素則根據所對應的影像資訊同時改變成相同的數值。在上述對應前提下，若影像的色彩值小於或等於我們所設定之臨界值，則將兩矩陣中相對應位置處但不同 Z 座標值之所有元素的值設定爲 0；反之，則先找尋模型矩陣中與影像相同對應位置處之元素值爲 1 的最大 Z 座標值，然後根據這些資訊將影像的色彩值複製到對應座標的色彩矩陣中，如此即完成前視圖部分的再修正處理。至於全部完整之演算法則如下所示。

演算法九：前視圖再修正處理演算法

```
FOR FrontViewImage a= 1….numberOfImageColumns-2 DO
    FOR FrontViewImage b= 0….numberOfImageRows-1 DO
        IF FrontViewImageColor(a-1,b)<= Threshold &&
          FrontViewImageColor(a,b)> Threshold &&
           ImageBoardXFind== 0
        TEHN
           ImageBoardX= a
           ImageBoardXFind= 1
        END IF
    END FOR
END FOR
```

```
FOR FrontViewImage b= 1….numberOfImageRows-2 DO
    FOR FrontViewImage a= 0….numberOfImageColumns-1 DO
        IF FrontViewImageColor(a,b-1)<= Threshold &&
           FrontViewImageColor(a,b)> Threshold &&
        ImageBoardYFind== 0
        TEHN
            ImageBoardY= b
            ImageBoardYFind= 1
        END IF
    END FOR
END FOR

FOR ModelMatrix m= 1…. numberOfMatrixColumns-1 DO
    FOR ModelMatrix n= 0…. numberOfMatrixRows-1 DO
        FOR ModelMatrix o= 0…. numberOfMatrixDepths-1 DO
            IF ModelMatrix (m-1,n,o)== 0 &&
               ModelMatrix (m,n,o)== 1 && ObjectBoardXFind== 0
            TEHN
              ObjectBoardX= m
              ObjectBoardXFind= 1
            END IF
        END FOR
    END FOR
END FOR

FOR ModelMatrix n= 1…. numberOfMatrixRows-1 DO
    FOR ModelMatrix m= 0…. numberOfMatrixColumns-1 DO
        FOR ModelMatrix o= 0…. numberOfMatrixDepths-1 DO
            IF ModelMatrix (m,n-1,o)== 0 &&
                ModelMatrix (m,n,o)== 1 && ObjectBoardYFind== 0
            TEHN
                ObjectBoardY= n
                ObjectBoardYFind= 1
            END IF
        END FOR
    END FOR
END FOR
```

```
FOR p= 0…. numberOfImageColumns- ImageBoardX -1 DO
    Flag= 0
    FOR q= 0…. numberOfImageRows- ImageBoardY -1 DO
        Flag= 0
        IF FrontViewImageColor(p+ImageBoardX,q+ImageBoardY)
            <= Threshold
        THEN
            FOR r= 0…. numberOfMatrixDepths-1 DO
                ModelMatrix(ObjectBoardX+p,ObjectBoardY+q,r)= 0
            END FOR
        ELSE
            FOR r= numberOfMatrixDepths-2…. 0 DO
                IF ModelMatrix(ObjectBoardX+p,ObjectBoardY+q,r+1)==0
                    &&
                    ModelMatrix(ObjectBoardX+p,ObjectBoardY+q,r)==1
                    &&
                    Flag== 0
                THEN
                    Flag= 1
                    ColorMatrix(ObjectBoardX+p,ObjectBoardY+q,r)
                    = FrontViewImageColor(p+ImageBoardX,
                                          q+ImageBoardY)
                END IF
            END FOR
        END IF
    END FOR
END FOR
```

3.4.8 色彩與模型矩陣之建立順序

有了以上全部視圖之處理方法後，接下來只要經過一連串的處理便能將物體的色彩與模型矩陣完整的建立起來，而其處理順序爲：前視圖處理->左視圖處理->後視圖處理->右視圖處理->上視圖處理->下視圖處理->前視圖再修正處理->後視圖處理->右視圖處理->上視圖

處理->下視圖處理。當以上的十一個處理步驟都完成後，我們其實可以說物體的 3D 模型重建工作已經宣告完成。只是若還想更進一步地將其展示在螢幕上的話，則需多出一個額外的步驟來利用 3D 繪圖引擎將其繪製於電腦螢幕上，並加上一些控制功能來對螢幕上的 3D 模型進行旋轉或移動等動作，而這部分將留待下一節再來進行詳述。

3.5 3D 繪圖與動作控制

在此我們選擇 OpenGL 做爲我們的 3D 繪圖引擎，並使用三種緩衝區來繪圖，分別爲雙重緩衝區、色彩緩衝區以及景深緩衝區。雙重緩衝區主要是額外使用一個預繪緩衝區做預先繪製的動作，等到其繪製完後再轉變成顯示緩衝區顯示在螢幕上，而原來的顯示緩衝區則轉爲預繪緩衝區以繪製下一個場景，如此不斷地交替循環著。這麼一來，當我們移動或轉動模型時所看到的 3D 模型顯示便能較爲順暢與自然，而不會出現一邊做動作還一邊繪製的不協調現象。至於色彩緩衝區主要是使用最爲一般的 RGB 三色來作爲色彩的表示方法，而最後的景深緩衝區則具有將模型的景深正確地呈現在螢幕上之功用。若說的更明白點，那就是螢幕上應該只能看到距離眼睛較近(即景深較淺)的物體點而已，一旦物體點之景深較深且被其它較淺之物體點所遮掩的話，螢幕上將不會顯示出來，而這也才符合我們眼睛實際上所看到的狀況。至於與 OpenGL 有關之繪製參數則設定如下（在此僅列

出重點部分且無關乎順序）[19]：

```
gluLookAt(0.0f, 0.0f, 100.0f, 0.0f, 0.0f, -100.0f, 0.0f, 100.0f, 0.0f);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glShadeModel(GL_SMOOTH);
glEnable(GL_POINT_SMOOTH);
glHint(GL_POINT_SMOOTH_HINT,GL_NICEST);
glPointSize(1.5);
glRotatef(roty, 0.0, 1.0, 0.0);
glRotatef(rotx, 1.0, 0.0, 0.0);
glEnable(GL_DEPTH_TEST);
glBegin(GL_POINTS);
glViewport(0, 0, w, h);
gluPerspective(130.0f, w/h, 10.0, 200.0);
glutReshapeFunc(OnSize);
glutDisplayFunc(OnDraw);
glutKeyboardFunc(OnKey);
```

由上面的參數其實已經可以看得出來模型是採用點繪法，而非一般常用的三角形繪法。其最主要的理由爲：若是利用點繪法，我們只要將模型矩陣與色彩矩陣中的各元素成分一一對應至各個不同的點即可，而不用再設法找尋三個點以形成一個三角形，且另一方面也可以省去材質貼圖的處理，如此將能以較爲簡易和快速的方式來將物體的 3D 模型繪製出來。至於對應的方法則爲判斷模型矩陣各元素的值是否爲 1，若爲 1 則將此點繪製於螢幕上，而其色彩值就由對應座標的色彩矩陣之元素值來決定；反之，則不描繪此點。如此全部的彩色物體點集合起來所呈現在螢幕上的便是一個很擬真的 3D 物體模型了。若還想要讓模型進行移動或轉動等功能的話，只需要再額外加上

滑鼠或鍵盤的控制參數即可。以此爲例，我們是使用鍵盤上的 a、d、w、s 等鍵來分別控制模型進行向左、向右、向上以及向下等轉動的動作，而每次所轉動的角度皆爲 10 度。當然這些控制功能是可以視實際需求加以變更的，因此要是能善加利用這些功能的話，那麼做出一部簡易的 3D 動畫也將會是一件輕鬆的事。

3.6 重建步驟總結

雖然之前已經就各部份之重建流程做過詳盡的敘述，但由於整個第三章部分爲本論文之核心所在，故在此特地再將整個模型重建步驟予以整理摘錄出來，以便使讀者能更快速地一覽所有的重建流程。至於整個重建流程則如圖 3-3 中所示。

3D 模型最終重建步驟總結：

第一步：利用數位相機拍攝物體之六面視圖影像。

第二步：將此六張影像縮小至合適尺寸，並使用自我背景濾除法來濾除影像的背景部分。

第三步：分別建立適當尺寸的三維模型矩陣與色彩矩陣。

第四步：利用前視圖演算法來處理前視圖與兩矩陣間的對應關係。

第五步：利用左視圖演算法來處理左視圖與兩矩陣間的對應關係。

第六步：利用後視圖演算法來處理後視圖與兩矩陣間的對應關係。

第七步：利用右視圖演算法來處理右視圖與兩矩陣間的對應關係。

第八步：利用上視圖演算法來處理上視圖與兩矩陣間的對應關係。

第九步：利用下視圖演算法來處理下視圖與兩矩陣間的對應關係。

第十步：利用前視圖再修正演算法來修正前視圖與兩矩陣間的對應關係。

第十一步：再度利用後視圖演算法來修正後視圖與兩矩陣間的對應關係。

第十二步：再度利用右視圖演算法來修正右視圖與兩矩陣間的對應關係。

第十三步：再度利用上視圖演算法來修正上視圖與兩矩陣間的對應關係。

第十四步：再度利用下視圖演算法來修正下視圖與兩矩陣間的對應關係。

第十五步：利用 OpenGL 將模型矩陣中爲 1 的元素以點繪法繪製於螢幕上，而其色彩值則由對應位置之色彩矩陣的元素值來決定，至此便完成整個物體 3D 模型的重建。此外，還可以額外加上一些控制功能以便模型能進行動作的展示。

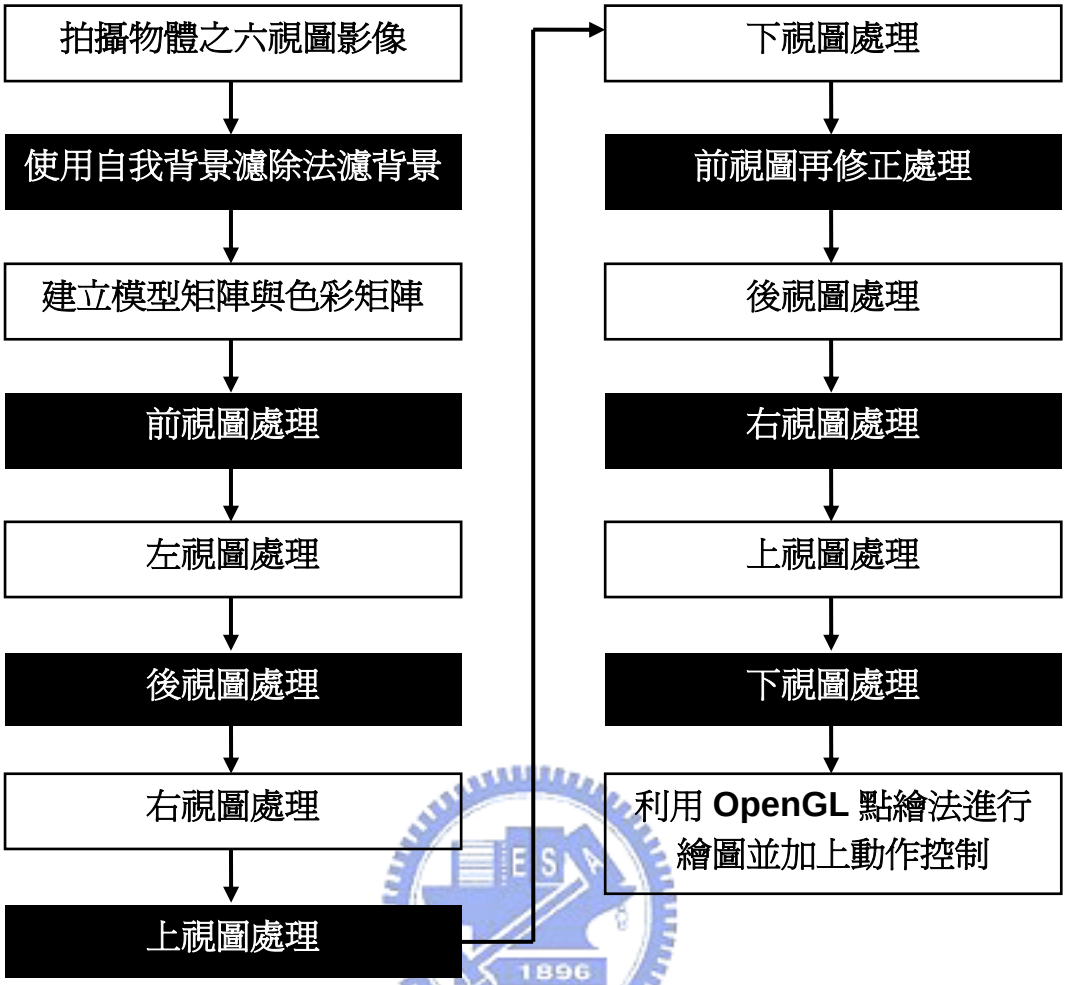

拍攝物體之六視圖影像
使用自我背景濾除法濾背景
建立模型矩陣與色彩矩陣
前視圖處理
左視圖處理
後視圖處理
右視圖處理
上視圖處理
下視圖處理
前視圖再修正處理
後視圖處理
右視圖處理
上視圖處理
下視圖處理
利用 OpenGL 點繪法進行繪圖並加上動作控制

圖 3-3 完整的重建流程

第四章

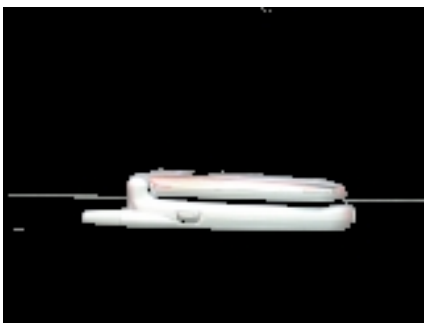
3D 模型重建結果展示

4.1 相關規格說明

爲了要驗證我們上述所提出來的方法是否真能重建物體之 3D 模型，因此我們選了四個物體分別爲手機、釘書機、修正帶以及小時鐘來進行試驗。所有試驗條件都一樣是使用 160*120 像素的影像來進行運算處理，而且全部的運算工作都是交由了一部配備 AMD 2800+ (1.99 GHz)之中央處理器，512 MB DDR400 之記憶體以及 RADEON 9800 PRO 128 MB 之顯示卡的個人電腦來進行處理。

4.2 手機之 3D 模型重建結果展示

圖 4-1 顯示手機之六視圖影像。

	原始影像	已濾除背景之影像
前視圖		

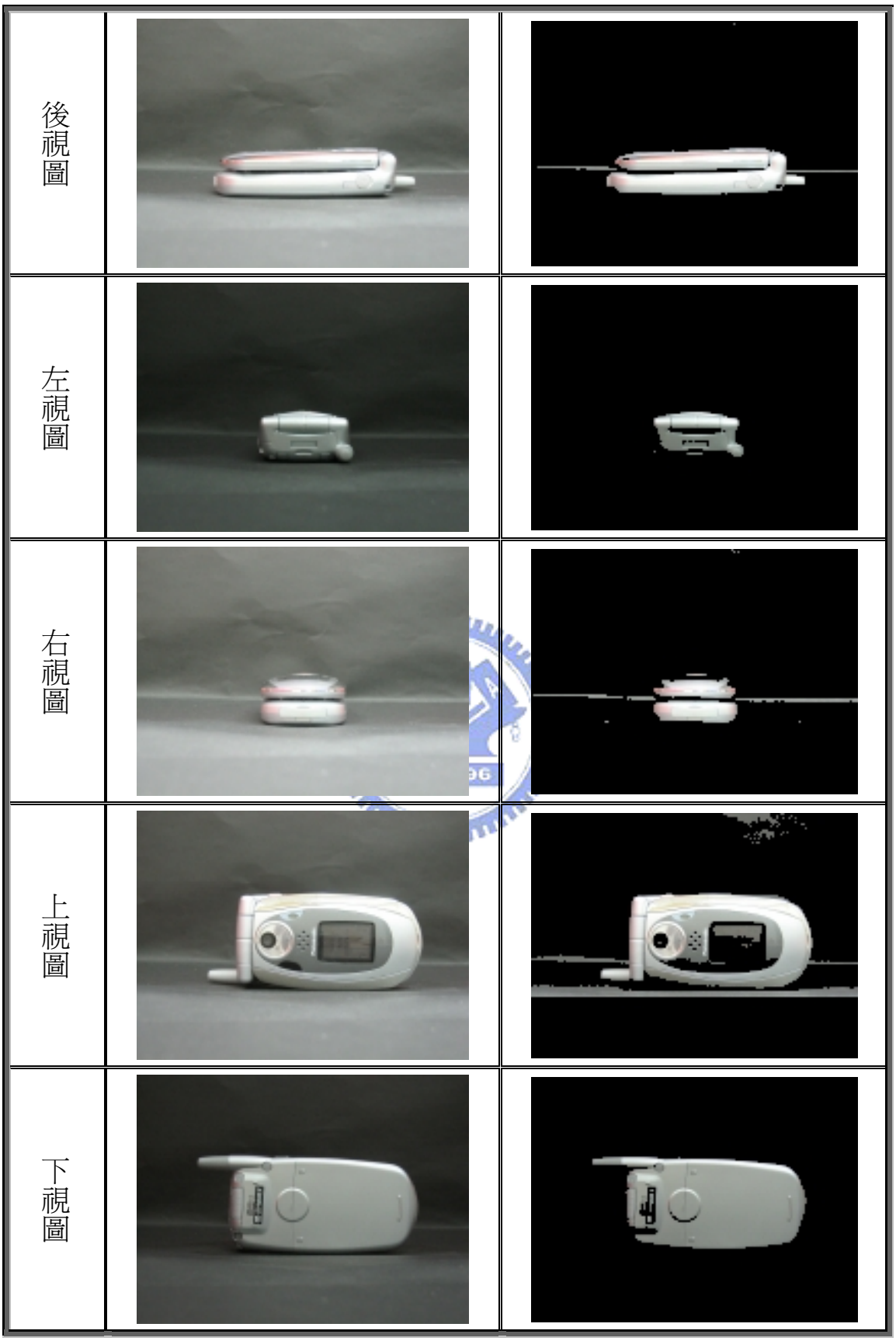


圖 4-1 手機之六視圖影像

圖 4-2 爲實際利用 OpenGL 平台在螢幕中進行展示的部分畫面截圖：

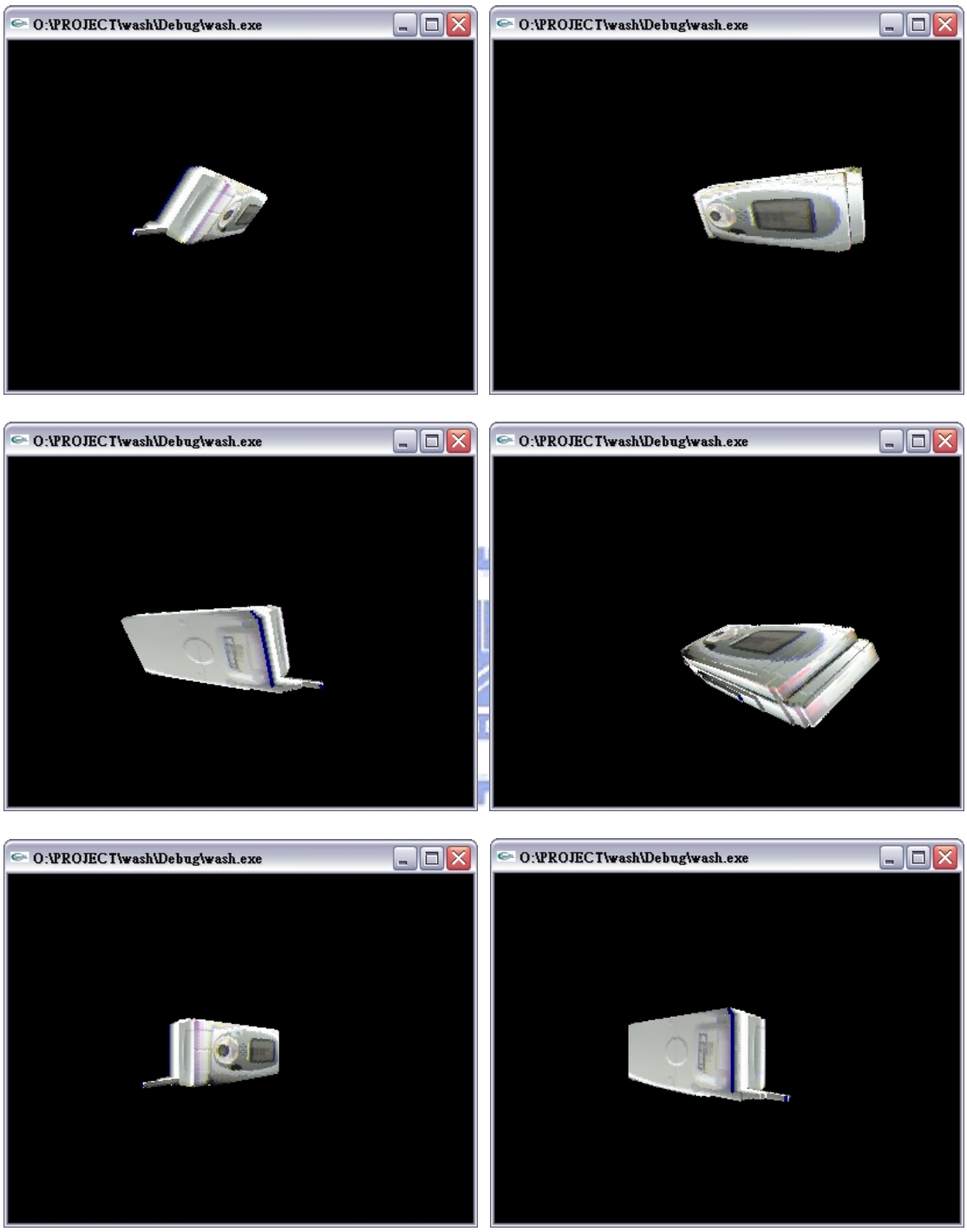


圖 4-2 手機展示之部分畫面截圖

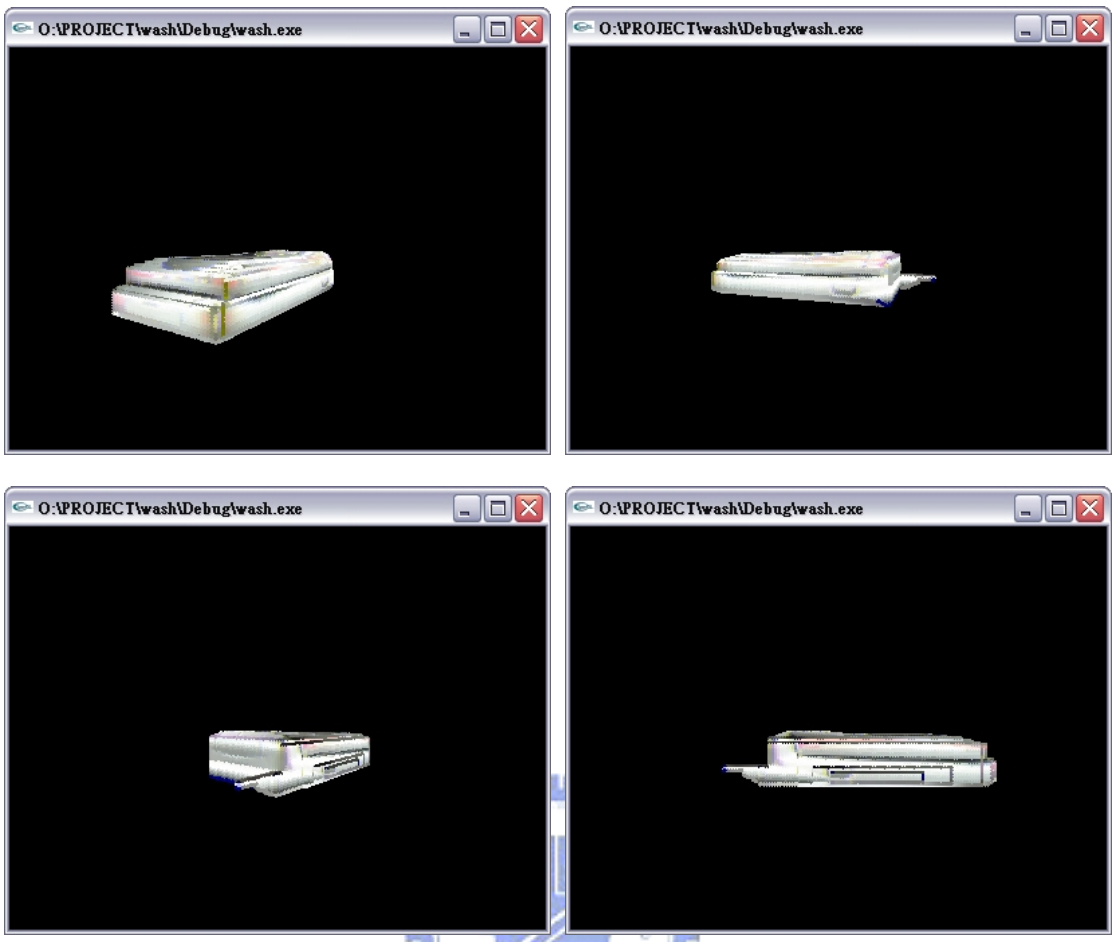
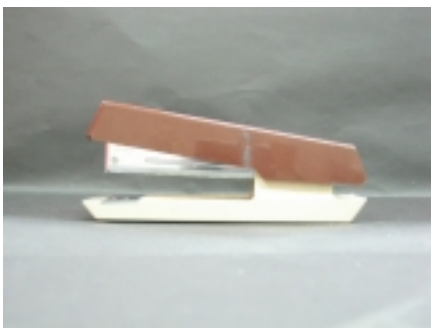
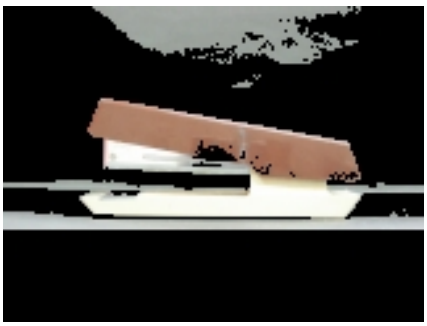


圖 4-2 手機展示之部分畫面截圖（續）

4.3 釘書機之 3D 模型重建結果展示

圖 4-3 顯示釘書機之六視圖影像。

	原始影像	已濾除背景之影像
前視圖		

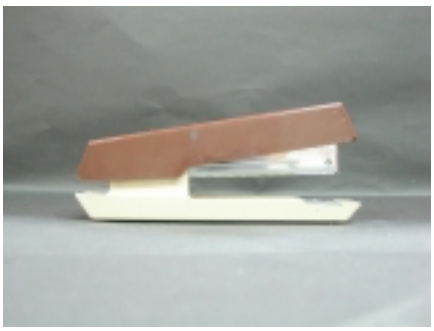
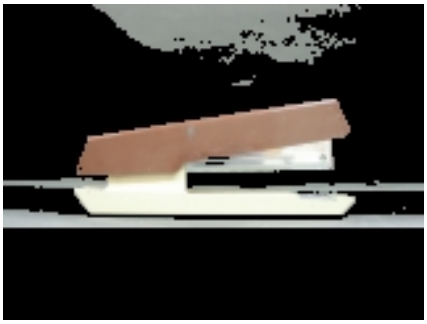
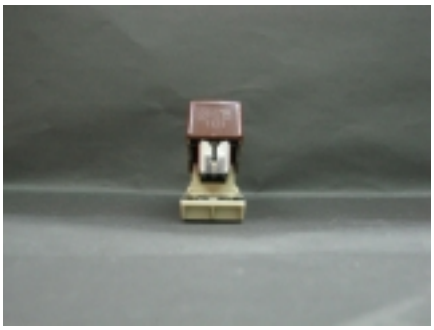
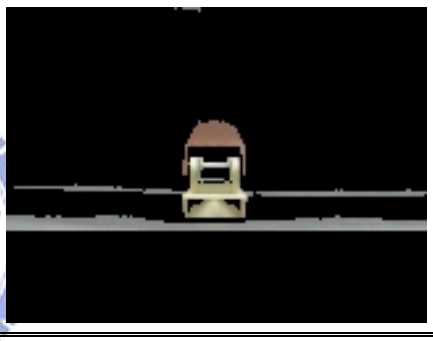
後視圖		
左視圖		
右視圖		
上視圖		
下視圖		

圖 4-3 釘書機之六視圖影像

圖 4-4 爲實際利用 OpenGL 平台在螢幕中進行展示的部分畫面截圖：

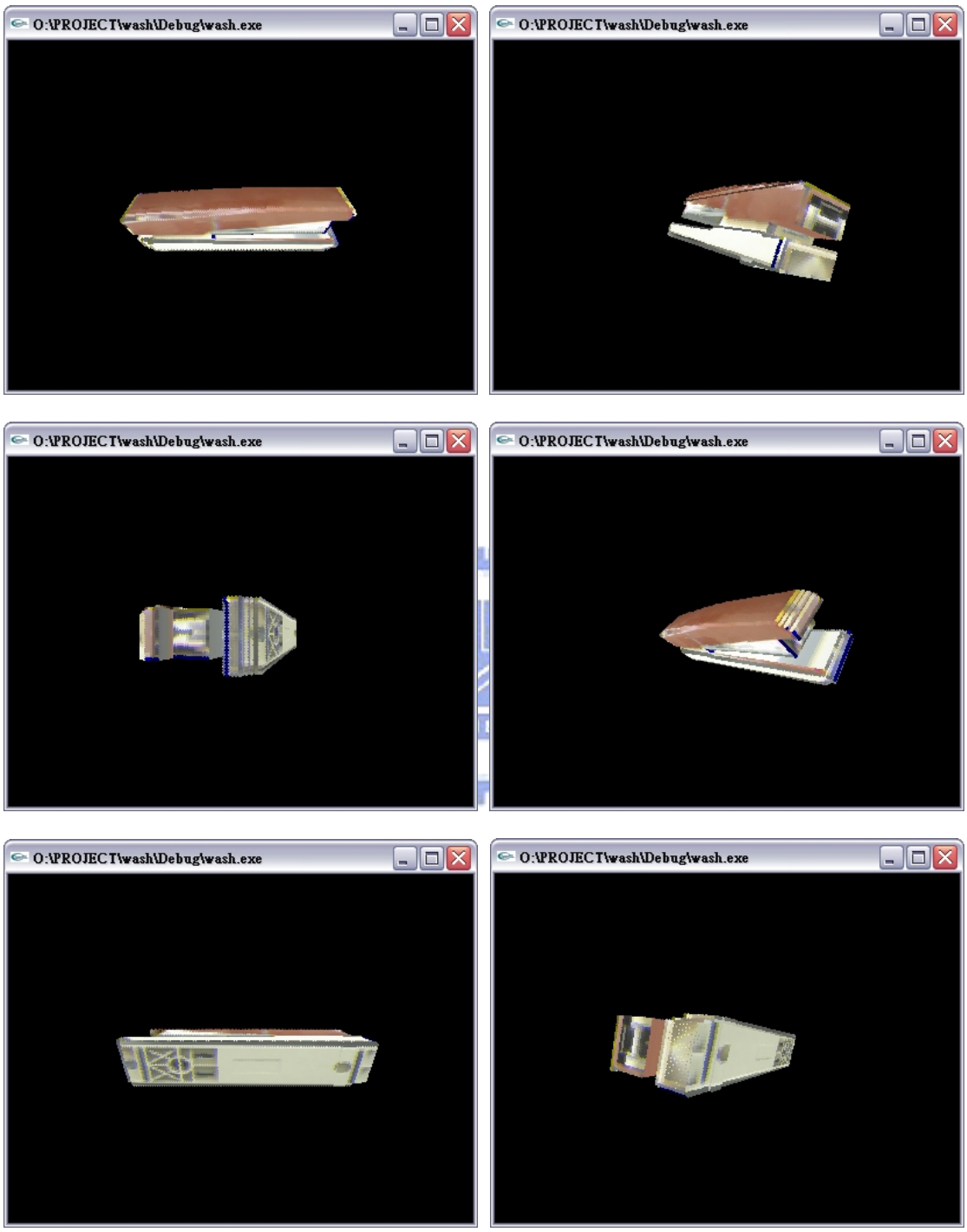


圖 4-4 釘書機展示之部分畫面截圖

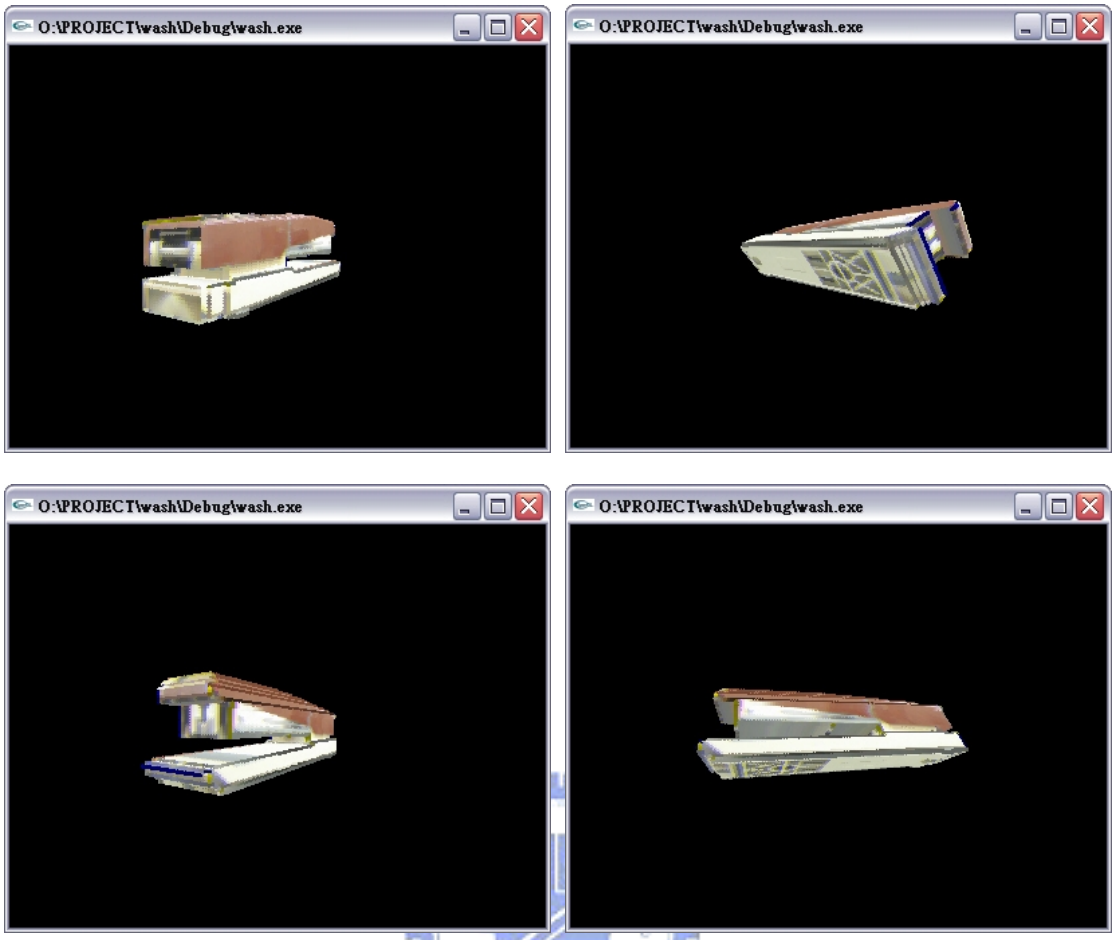


圖 4-4 釘書機展示之部分畫面截圖（續）

4.4 修正帶之 3D 模型重建結果展示

圖 4-5 顯示修正帶之六視圖影像。

	原始影像	已濾除背景之影像
前視圖		

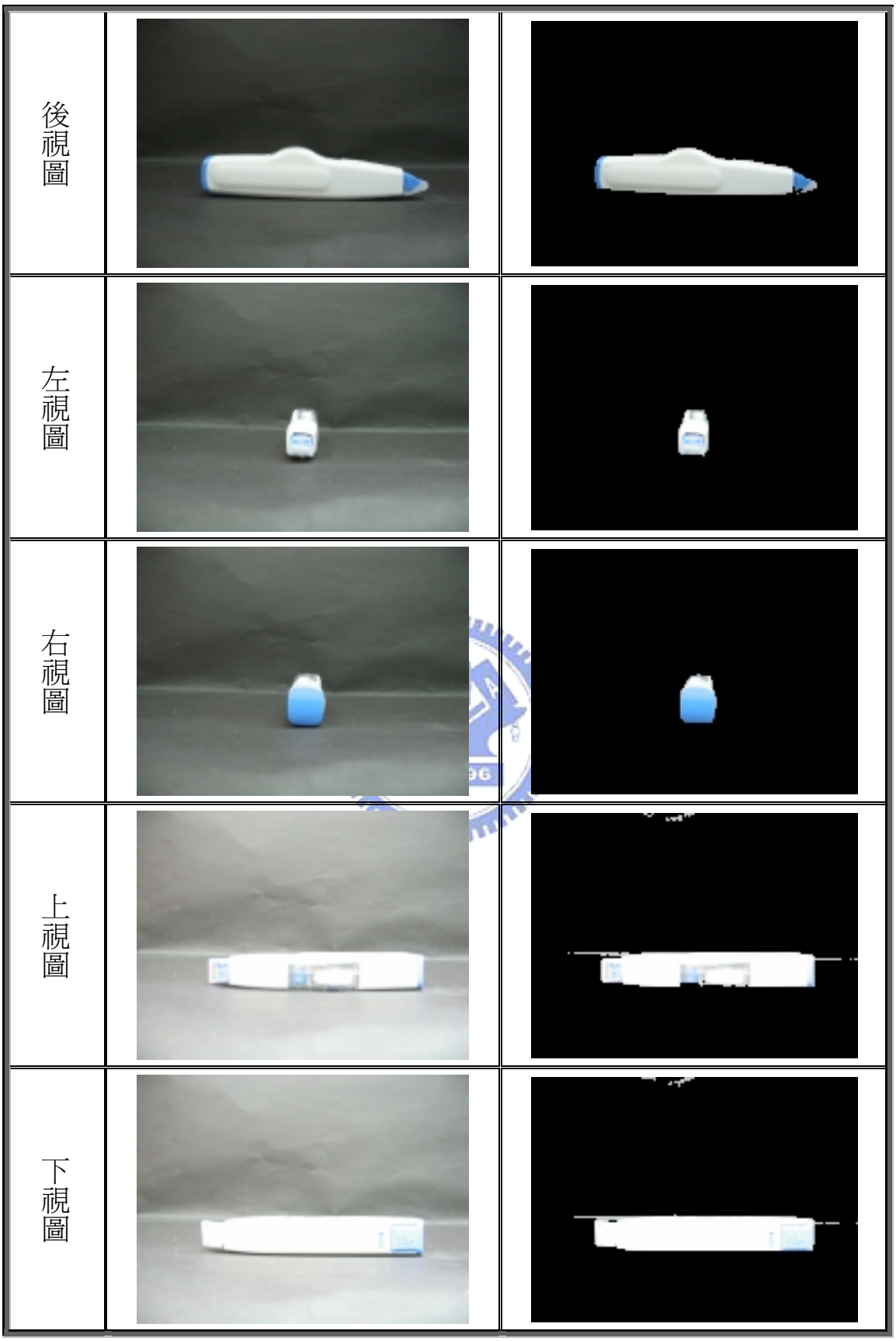


圖 4-5 修正帶之六視圖影像

圖 4-6 爲實際利用 OpenGL 平台在螢幕中進行展示的部分畫面截圖：

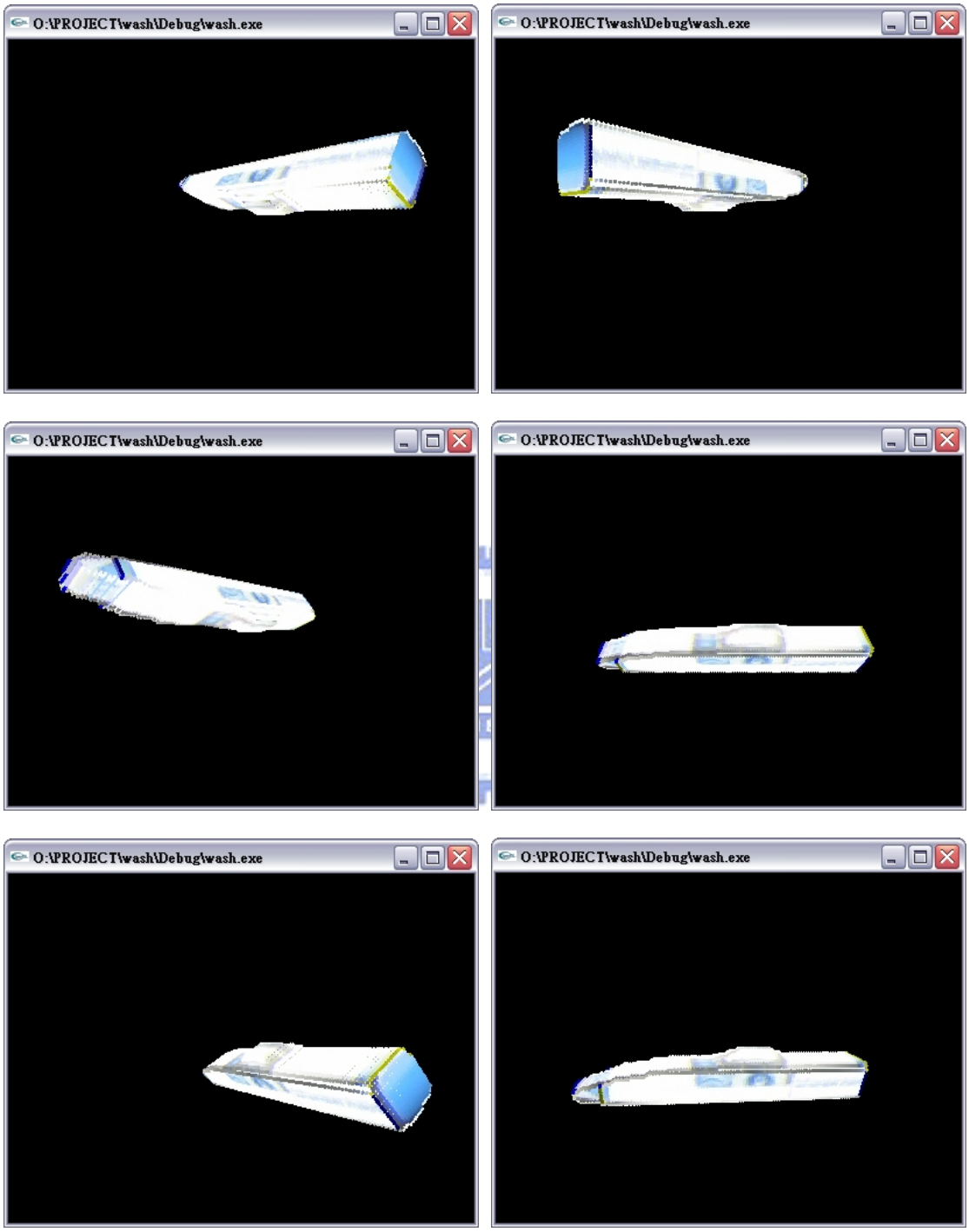


圖 4-6 修正帶展示之部分畫面截圖

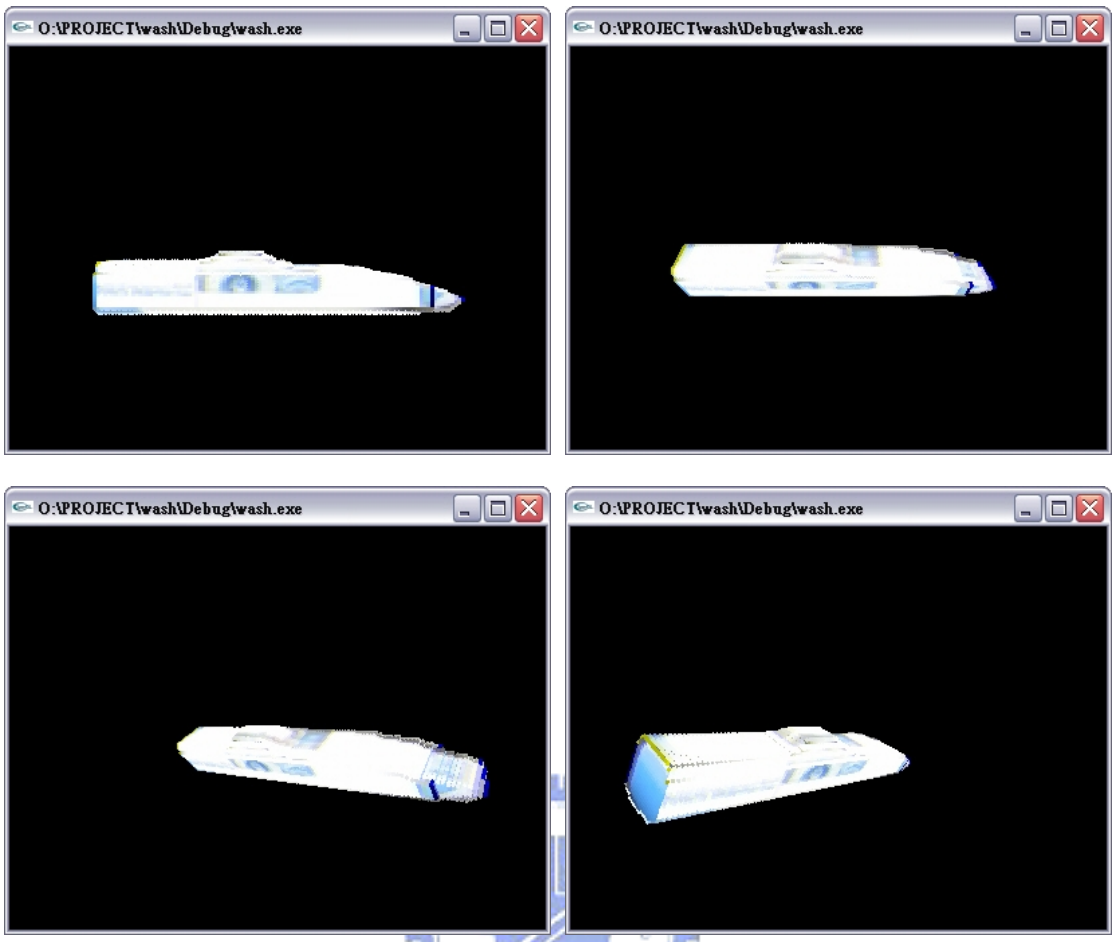


圖 4-6 修正帶展示之部分畫面截圖（續）

4.5 小時鐘之 3D 模型重建結果展示

圖 4-7 顯示小時鐘之六視圖影像。

	原始影像	已濾除背景之影像
前視圖		

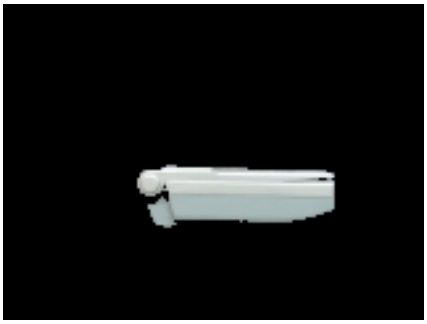
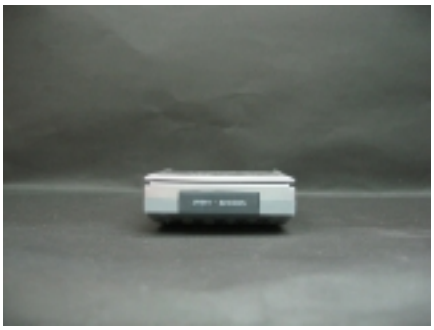
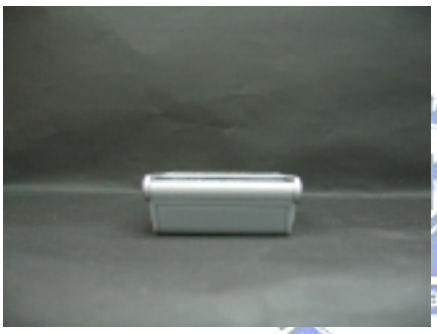
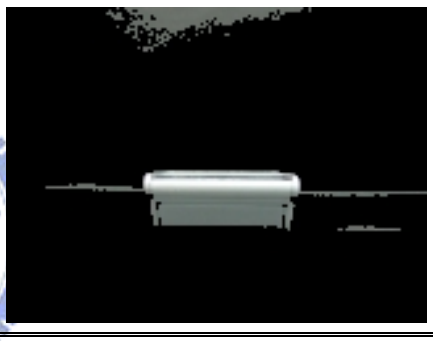
後視圖		
左視圖		
右視圖		
上視圖		
下視圖		

圖 4-7 小時鐘之六視圖影像

圖 4-8 爲實際利用 OpenGL 平台在螢幕中進行展示的部分畫面截圖：

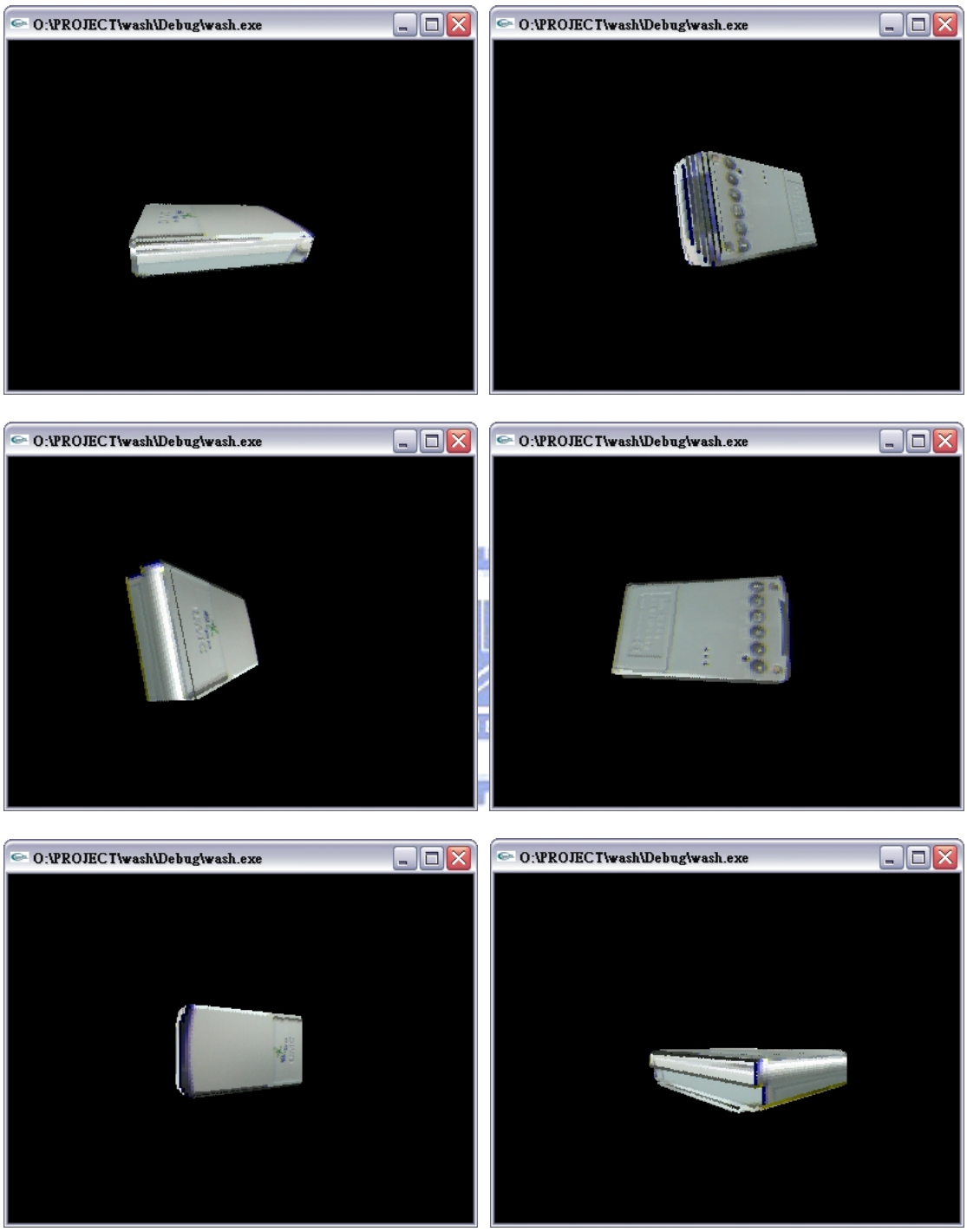


圖 4-8 小時鐘展示之部分畫面截圖

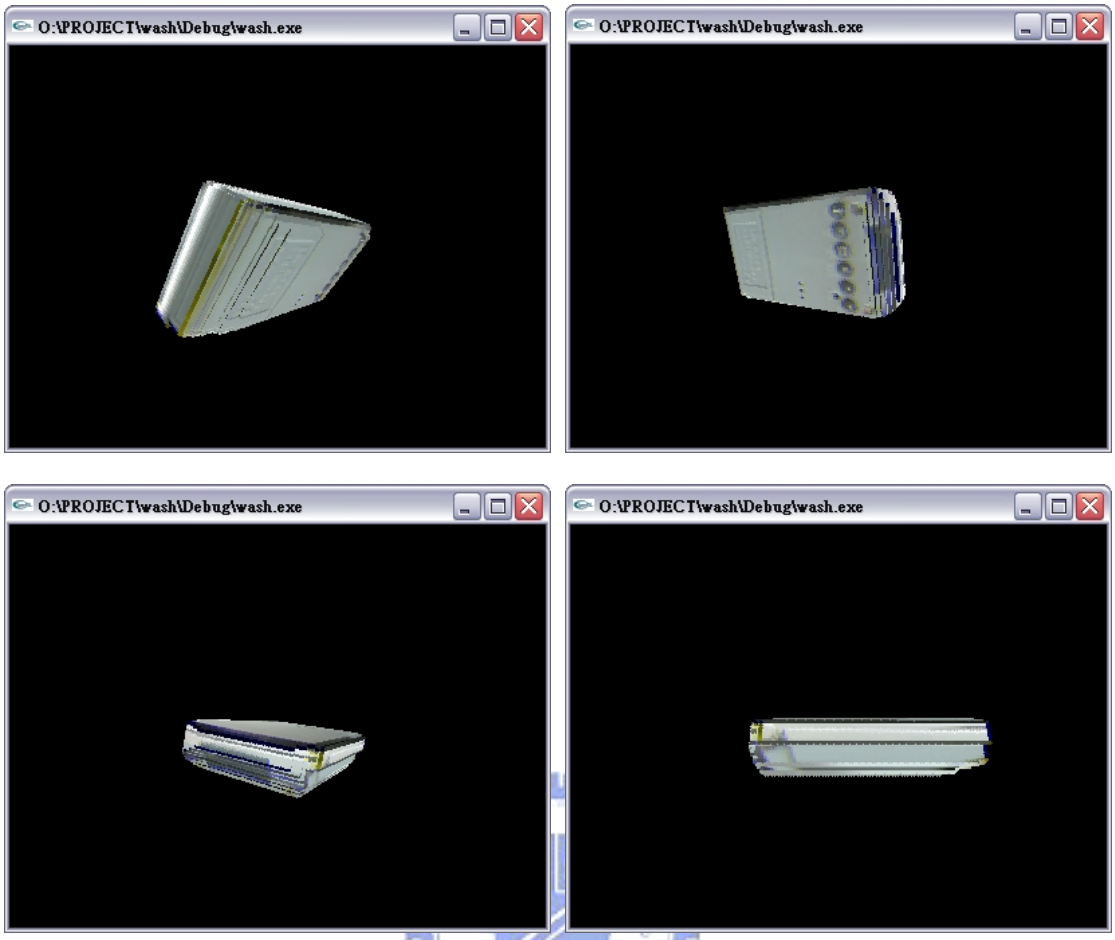


圖 4-8 小時鐘展示之部分畫面截圖（續）

4.6 結果比較與討論

在相同的硬體條件下，將四個物體所需之重建運算時間（在此的重建運算時間是指由輸入已濾除背景之影像一直到完成 3D 模型重建與展示的過程所花費的時間）進行比較的話，可得結果如下：若影像尺寸爲 160*120 大小時，約需費時 4 秒左右；若影像尺寸放大至 320*240 大小時，則約需費時 1 分 40 秒左右。由此可以觀察到模型解析度與重建運算時間並非呈線性關係，所以如何取捨就要視使用者的考量來決定了。此外，無論模型採用的是何種解析度，整體的重建

運算時間也都在可以接受的範圍之內，因此可以說是相當地迅速。

我們接著就以上各物體的重建結果來進行討論。在手機部分所重建出來的 3D 模型可以說相當完美，並沒有任何瑕疵出現；而在釘書機部分所重建出來的 3D 模型也算相當的成功，不過仔細觀察仍然可以發現模型的前視面部分會出現數條黑線，只是這種小瑕疵所造成的影響倒是十分地輕微而可予以忽視；至於在修正帶部分所重建出來的 3D 模型也算不錯，只是瑕疵稍嫌多了一點而已，例如在邊界部分會出現明顯的不連續現象以及修正帶頭部附近的模型有略微失真的現象等等。最後在小時鐘部分所重建出來的 3D 模型一樣是與實際物體差不多逼真，不過在其左側黑色大按鈕的部分，由於其顏色太過接近背景色，因此重建出來的模型便會因爲背景濾除效應而產生嚴重失真，且更進一步地造成模型左側出現許多不連續的黑色線條。若我們對於以上這些小瑕疵的部分並不是很在意的話，那麼大體上而言我們的方法的確是相當的快速、簡便且有效，而重建出來的 3D 模型也跟原物體極其相似，因此我們所提出的相關重建流程與方法確實能成功地達成先前所期望的目標。

第五章
結論與未來可研究之方向

5.1 結論

在本論文中我們僅使用了簡便的器材來驗証所提出一些不算複雜的演算法進行整個 3D 模型的重建工作，比較起其他論文所提出的方法而言，我們既無須使用複雜且昂貴的儀器，且也不使用高深與複雜的演算法來做爲核心，因此我們所提的方法確實比其他方法更具有實用的價值，而這就是本論文的主要貢獻所在。特別是在實際利用四個物體來進行重建後，其結果不但呼應了之前所述的迅速、簡便與有效之特性，且重建後的 3D 模型也可觀察到與原物體十分地逼近，縱有些許瑕疵亦不至於嚴重影響整體的表現。因此整體而言，我們所提的方法確實是相當的有效且能符合一開始所規劃的目標，一旦未來有需要時，將能迅速且廣泛地應用於各個領域中。

5.2 未來可研究之方向

我們在本論文中所提的方法雖然已經可以說是極爲不錯，不過若還想要更進一步深入研究的話，仍有許多方面值得我們去留意和思考。首先是我們可以考慮將傳統的三角形繪法和材質貼圖一起合併來取代方法中所使用的點繪法，甚至進一步加上光源照明來使得整個模型的展示更爲逼近我們所想要的場景狀況。再者也可以使用更多種和

更複雜的演算法來改善背景濾除效果或是模型的重建結果，以降低整個模型的失真程度。甚至我們也可以考慮結合 3D 掃描器或是他人所提的演算法來使整個重建的過程更爲迅速和簡便。因此其實還是有許許多多的方法和可能性等待我們去發掘，以上只不過是略舉數例罷了。總之，凡是能成功地重建出物體 3D 模型的方法就是好方法，其彼此之間的差異之處只有在於重建所花的時間以及重建的複雜度罷了，不過這當然也會連帶地影響其實用性，因此如何取捨就看應用面而定了。

參考文獻

[1] Soon-Yong Park and Murali Subbarao, “Automatic 3D model reconstruction using voxel coding and pose integration,” In Proceedings of 2002 International Conference on Image Processing, Vol. 2, pp. II-533 - II-536, Sept. 2002.

[2] V. Sequeira, J.G.M. Goncalves, M.I. Ribeiro, “High-level surface descriptions from composite range images,” In Proceedings of International Symposium on Computer Vision, Vol. 2, pp. 163-168, Nov. 1995.

[3] Huei-Yung Lin and Murali Subbarao, “Multiple base-angle rotational stereo for accurate 3D model reconstruction,” In Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis, Vol. 2, pp. 931-935, Sept. 2003.

[4] C. Fruh, A. Zakhor, “3D model generation for cities using aerial photographs and ground level laser scans,” In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2, pp. II-31 - II-38, 2001.

[5] F. Bernardini, H. Rushmeier, I.M. Martin, J. Mittleman, G. Taubin, “Building a digital model of Michelangelo's Florentine Pieta,” IEEE Transactions on Computer Graphics and Applications, Vol. 22, Issue 1, pp. 59-67, 2002.

[6] L. Blonde, M. Buck, R. Galli, W. Niem, Y. Paker, W. Schmidt, G. Thomas, “A virtual studio for live broadcasting: the Mona Lisa project,” IEEE Transactions on Multimedia, Vol. 3, Issue 2, pp. 18-29, 1996.

[7] S. Ferrari, N.A. Borghese, “A portable modular system for automatic acquisition of 3D objects,” IEEE Transactions on Instrumentation and Measurement, Vol. 49, Issue 5, pp. 1128-1136, Oct. 2000.

[8] C. Fruh, A. Zakhor, “Constructing 3D city models by merging aerial and ground views,” IEEE Transactions on Computer Graphics and Applications, Vol. 23, Issue 6, pp. 52-61, 2003.

[9] H. Saito, S. Baba, T. Kanade, “Appearance-based virtual view generation from multicamera videos captured in the 3-D room,” IEEE Transactions on Multimedia, Vol. 5, Issue 3, pp. 303-316, Sept. 2003.

[10] A.Y. Mulayim, U. Yilmaz, V. Atalay, “Silhouette-based 3-D model reconstruction from multiple images,” IEEE Transactions on

Systems, Man and Cybernetics, Vol. 33, Issue 4, pp. 582-591, Aug. 2003.

[11] M. Potmesil, “Generating octree models of 3-D objects from their silhouettes in a sequence of images,” Computer Vision Graphic Image Processing, vol. 40, no. 1, pp. 1–29, Oct. 1987.

[12] R. Szelisky, “Rapid octree construction from image sequences,” Computer Vision Graphic Image Processing, vol. 58, no. 1, pp. 23–32, July 1993.

[13] E. Boyer, “Object models from contour sequences,” In Proceedings of Europe Conference on Computer Vision, 1996, pp. 109–118.

[14] J. Y. Zheng, “Acquiring 3-d models from a sequence of contours,” IEEE Transactions on Pattern Analysis Machine Intell., vol. 16, pp. 163–178, Feb. 1994.

[15] P. Eisert, E. Steinbach, B. Girod, “Multi-hypothesis, volumetric reconstruction of 3-d objects from multiple calibrated camera views,” In Proceedings of International Conference on Acoustics Speech Signal Processing, pp.3509–3512. Mar. 1999.

[16] M. Pollefeys, R. Koch, M. Vergauwen, L. Van Gool, “Automated reconstruction of 3-D scenes from sequences of images,” J, Photogramm. Remote Sensing, vol. 55, no. 4, pp. 251–267, 2000.

[17] A. Fusiello, “Uncalibrated euclidean reconstruction: A review,” Image and Computer Vision, vol. 18, no. 67, pp. 555–563, May 2000.

[18] C. H. Esteban and F. Schmitt, “Multi-stereo 3-D object reconstruction,” In Proceedings of International Symposium on 3-D Data Processing Visualization Transmission, pp.159–166, June 2002.

[19] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner, OpenGL Programming Guide, Addison Wesley, 3rd, ISBN 0201604582, 1999.