國立暨南國際大學

資訊工程　　學 系碩士班

碩士論文

以影像的二維骨骼結構為基礎之動畫產生系統

An Animation Synthesis System based on 2D Skeleton Structures of Images

研究生：余岐智

指導教授：陳履恆

中華民國九十三年七月

以影像的二維骨骼結構為基礎之動畫產生系統

An Animation Synthesis System based on 2D Skeleton Structures of Images

研究生：余岐智
指導教授：陳履恆

國立暨南國際大學

資訊工程學系碩士班

碩士論文

A Thesis
Submitted to
Computer Science and Information Engineering
National Chi Nan University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

July 2004
Puli, Nantou, Taiwan, Republic of China

中華民國九十三年七月

摘要

在現今科技的快速進步之下，數位相機與含有照相功能的手機已經越來越普遍，使得有著大量的數位影像圍繞在我們周遭。本研究的目的是發展一套動畫合成系統去進一步的利用這一些數位影像，透過簡單的方式去產生一組自然又生動的動畫。

使用二維影像來重建一個三維模型是一項非常困難的工作，而且使用傳統的電腦圖學方法去建造一個虛擬真實世界也是一個耗費時間的工程。針對以上這兩個問題，Image-based rendering 的技術提供了一個良好的解決方案。基本上，使用數張拍攝好的影像就可以合成不同視角及方向的結果。利用影像變形的技術不但可以減少重建虛擬世界的成本，也可以克服這些影像之間動作不連續性的缺點。但是如果變形後的結果沒有維持原來物件的形狀，則將會造成不合理的或是沒有意義的建立結果，尤其是人工物體。

在本研究當中，我們提出了一個簡單的保持外形的影像變形技術去產生動畫的畫格影像。首先，先從輸入的影像當中取出骨架結構。骨架化目的是將影像上的物體外形縮減成骨架的形狀，即是將一個二維的物體轉換成一維的線段來代表，並且大量地保留了物體形狀的特性。我們使用了 Robert L. Ogniewigz 所提出的 Discrete Voronoi

Skeletons 技術去計算物體的骨架。分析計算完這些輸入的影像之後，系統將會利用這些骨架的資料結構去依照某些動作樣本進行運動，如跑步或走路等等。最後，再使用 feature-based image morphing 的技術去合成出最後的畫格影像。

ABSTRACT

Together with the miniaturization of digital image capture devices and the integration of digit camera into cellular phones, there are enormous photograph images around us. The research purpose of this paper is developing an animation synthesis system to further utilize the above-mentioned image materials effectively, and produce interesting and vivid animation effects in a natural and simple way.

However, reconstructing the 3D models by using only 2D images is still a very difficult task. Also, building the virtual reality world by the traditional computer graphics methods is a highly time-consuming task. For the above problems, the image-based rendering technology provides an elegant approximate solution. Basically, the required image of arbitrary viewing position and direction is synthesized by morphing two or multiple pre-captured images at some determined camera positions. Applying the image morphing technology does not only reduce the expensive cost of world construction, but also overcome the discontinuity drawback of the pre-captured image sequence. However, if the morphing results fail to preserve the original shape of objects, it will cause a fake construction, especially when the target object is artifact.

In this research, we propose a simple shape-preserving image morphing method to produce frames of animation of images. First, the skeleton structures of the pre-captured images are extracted. The skeletonization is a process for reducing foreground regions in a digital image to a skeletal remnant that largely preserves the properties of the original region while discarding most of the original foreground pixels. We use the Discrete Voronoi Skeletons proposed by Robert L. Ogniewigz to build object's skeleton. After the pre-captured images are analyzed, the skeletons data structure of the required image are applied some forward kinematics motion patterns like walking or running. Finally, the reasonable result images are synthesized by feature-based image morphing technique.

致謝

首先感謝我的指導教授陳履恆老師，對於初次踏入電腦圖學領域的我，給予了無數的指導及幫助。陳老師的易於溝通與尊重學生的風格讓我在碩士研究的學校生活當中，留下了深刻的印象。

從資管系跨越到資工所，才驚覺領域的差異性原來是如此之大。父母親在背後的支持以及完全讓我自行決定學業生涯的規劃，使得我沒有後顧之憂，親情的鼓勵與關心原來是如此的溫暖與感動。感謝雅茵在這些年來一直不斷的鼓勵我陪伴我，讓我一直能夠打起精神努力的向前進；感謝創辦動物保護社時與我一起經營奮鬥的學妹淑萍，社團讓我的求學路途擁有特別的回憶。回憶中有笑也有淚，有歡樂也有悲傷，替我的人生注入了特別色彩；謝謝繡妮學姐曾經給予我的教誨、蕙賢學妹在論文最後衝刺的時候幫我加油打氣；謝謝穎裕學長、宗志學長、與我的同學小包和 Taco 在這個研究領域中給予我的幫助；謝謝資工系壘球隊帶給我的團隊活力；謝謝這些年來在動物保護社裡陪伴我的狗兒們、馬術課當中壯碩又可愛的馬兒，讓我擁有少數人能得到的特別經驗。

最後，這個論文能夠順利的完成，功勞是屬於大家的。在此感謝你們，沒有你們的祝福與鼓勵，就沒有今天能夠完成論文的我。

岐智，予埔里　2004 七月

目錄

表目錄

圖目錄

Chapter 1

Introduction

在現今科技的快速進步之下，數位相機與含有照相功能的手機已經越來越普遍，使得有著大量的數位影像圍繞在我們周遭。本研究的目的是發展一套動畫合成系統去進一步的利用這一些數位影像，透過簡單的方式去產生一組自然又生動的動畫。

利用傳統電腦圖學的方法來建立虛擬實境，是非常耗時的工作。而 Image-Based Rendering 的技術提供了一個良好的解決方案。基本上，藉由攝影機從不同角度所拍攝到的多張影像來做 Image Morphing 可以合成影像，其結果如同從任意視角所拍攝到的影像一樣。

影像變形(Image Morphing)已經是在視覺效果上一個強而有力的工具，在現今的許多電影及電視當中已有許多驚人的效果將一張影像很流暢的變換至另一張。它的程序是由影像扭曲及顏色的混合內插所結合。由使用者定義兩張影像之間互動的特徵，例如使用網格(mesh nodes)、線段(line segments)、曲線(curves)或點(points)。兩張影像對應的特徵將透過映射函數(mapping function)計算所有影像上的點在空間上的關連，因爲主要的工作是在扭曲，所以也稱爲扭曲函數(warp

function)。當兩張影像都完成扭曲函數而產生了中間的特徵形狀之後，原圖的顏色隨即混合內插到中間影像之中，完成結果。然而在輸入兩張相同物體但不同攝影角度(或不同運動狀態下)的影像時，在變形的過程當中將會造成物體上的扭曲失真。在本論文中，我們提出在 Morphing 時不會造成物體扭曲的方法。

3D 物體投影至 2D 的影像，要以 Image Morphing 的方式達到物件運動的效果是很困難的，主要問題有：1) Image Morphing 只能很流暢的從一張影像變形(或融解)至另一張，與實際物體運動的過程有很大的不同，並且 Image Morphing 容易造成物體的扭曲失真。2)3D 物體運動時應有深度的資訊，也就是 Z 軸。投影至 2D 影像上後並無深度資訊，無法以傳統 Morphing 方式模擬物體的運動。

透過 Robert L.提出的 Discrete Voronoi Skeletons，我們可以得到一個好的形狀表示法(Shape Representation) —骨架。它主要的程序是將影像的邊界(boundary)做 raster crack(意即分解爲點集合)後，計算 Voronoi Diagram。再透過規則化(Regularization)的函數計算所有 Voronoi edge 的權重值，最後給定一個臨界值除去多餘不需要的分支而獲得骨架。如果影像外形過於崎嶇不平或是骨架不理想時，則再使用骨架金字塔(skeleton pyramid)取得各種解析度的骨架以符合需求。這個方法是以物體的外形所表示的邊界上頂點的 Voronoi Diagram 爲

基礎，使得骨架化後的結果有 1.正確的 Euclidean Metrics 及 2.保留物體原本的連接性。有了骨架的資訊，我們就能夠透過設定骨架可變形及不可變形的部份來控制在 Morphing 的過程中的扭曲情況，以避免物體在變形的過程中造成失真。而在 3D 物體運動上，各部位可能有著深度的不同，所以我們可以依照各部位的深度切割物體爲若干區域，並將區域設定深度。深度越高者優先計算，如此一層層覆蓋至結果影像，完成模擬 3D 空間上物體在 2D 影像上的投影。

計算出各部位的骨架資訊，按照物體合理運動的狀態去設定合理的固定軸及旋轉軸，就能夠進行轉變控制(Transition Control)。利用關鍵畫面(key frame)的設定或是動作樣本(Moving Pattern)週期運動設定，最後得到一組從影像 A 合理且實際的運動動畫移動至影像 B。

第二章節介紹與本文相關的研究，首先是以外形保留爲前提的 Image Morphing 及 View Morphing 的技術做說明，然後介紹骨架化的相關技術。第三章節介紹本系統的演算法。最後章節做出結論並提供數種結果的展示。

Chapter 2

Related Background

本章節的內容將對研究背景做說明。2.1 節介紹以外形保留條件爲前提的 Image Morphing 及 View Morphing。2.2 節則是介紹骨架化(Skeletonization)的技術。

2.1 Image Morphing 與 View Morphing

2.1.1 Image Morphing

所謂的 Morphing 即是將某個物體外形平順的轉換到另一物體的形狀。而產生這樣的效果通常都是用 image-based rendering 的技術且此兩者 morphing 的物體外觀上都有幾分程度的相似，或者事先由使用者來指定變形的過程。加上以線性內插的方法來計算位於中間影像的特徵點位置。如圖 2.1 所示，在 key frame K 中的直線要轉換成如 key frame K+1 中的兩條線段。通常，在變形的過程中會造成外形的失真，如圖 2.2，且無法在相對應的攝影機角度呈現對應的外形，如圖 2.3。

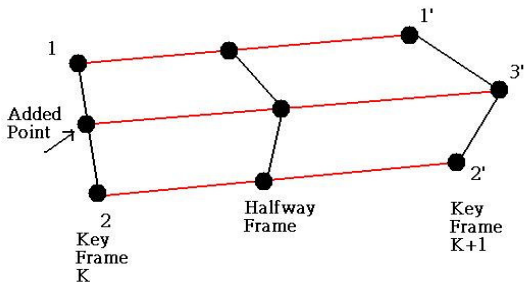


圖 2.1 利用線性內插在 key frame K 中的直線轉換
成如 key frame K+1 中的兩條線段

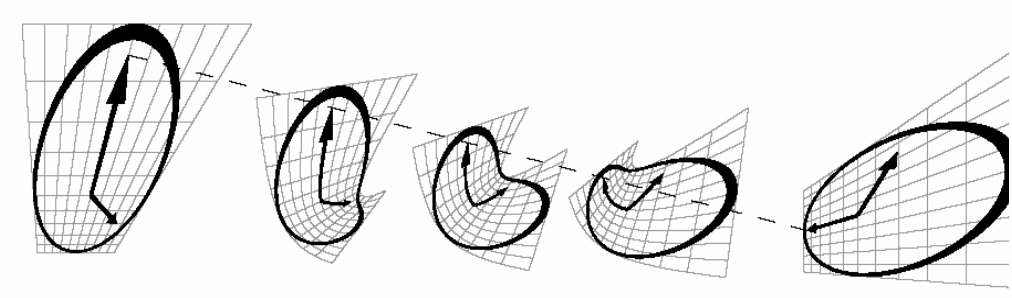


圖 2.2 Morphing 過程中所造成的外形失真

圖 2.3 Morphing 的過程中，在中間的竹蜻蜓影像縮小且彎曲，
並且旋轉軸與葉片分離

2.1.2 View Morphing

View Morphing 的技術是由 Sitez S.M.所提出的方法[6]。利用基本的投影幾何原理來處理三度空間的攝影機投影和場景的轉換，是 image morphing 技術的延伸。將攝影機從不同角度所拍攝到的多張影像，從 Computer Vision 的角度來產生影像合成的結果。這種效果可

以用圖 2.4 來說明，從左右兩邊攝影機所拍到的影像，透過 view morphing 的技術可以產生如同真正攝影機在中間所拍到的影像。就

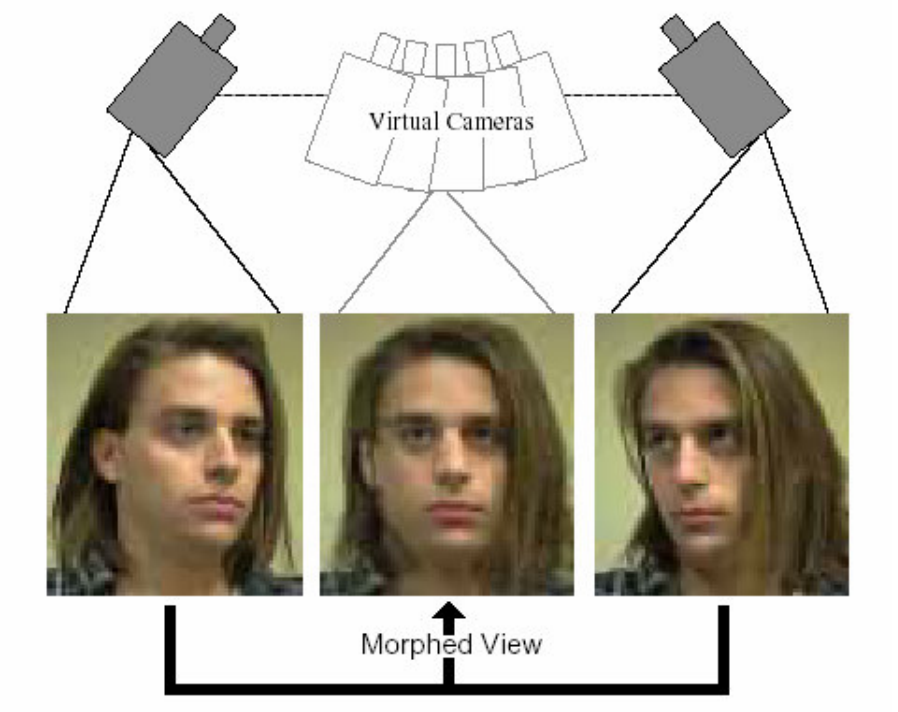


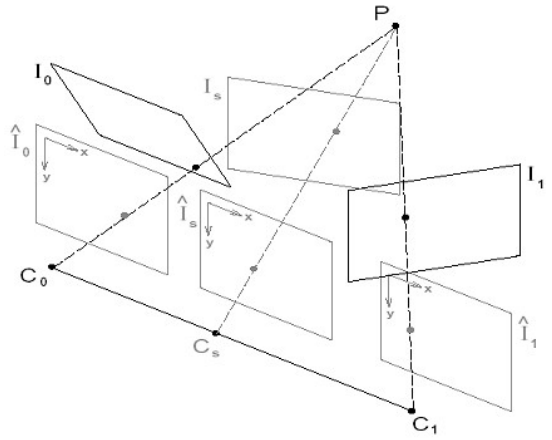
圖 2.4　以 View Morphing 來產生如同真正攝影機在中間所拍攝到的影像一般。(圖源自於[6])

two View morphing 而言，可以下三個步驟來完成，以圖 2.5 說明：

Prewarp: 校正兩張影像，使得 P 點、C0、C1 及影像上的點共平面。

Morphing: 內插產生虛擬影像。

Postwarp: 轉換成原來角度所應呈現的影像。



View Morphing in Three Steps. (1) Original images $\mathcal{I}_0$ and $\mathcal{I}_1$ are prewarped to form parallel views $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$. (2) $\hat{\mathcal{I}}_s$ is produced by morphing (interpolating) the prewarped images. (3) $\hat{\mathcal{I}}_s$ is postwarped to form $\mathcal{I}_s$.

圖 2.5　View Morphing 的三個步驟：Prewarp，Morphing，Postwarp (圖源自於[6])。

2.1.3　Comparing Result

利用圖 2.6 可以比較 image morphing 與 view morphing 的差異性。View morphing 在變形的過程中，保留了物體外形的完整；而 image morphing 在變形的過程中，造成了物體外形的失真。

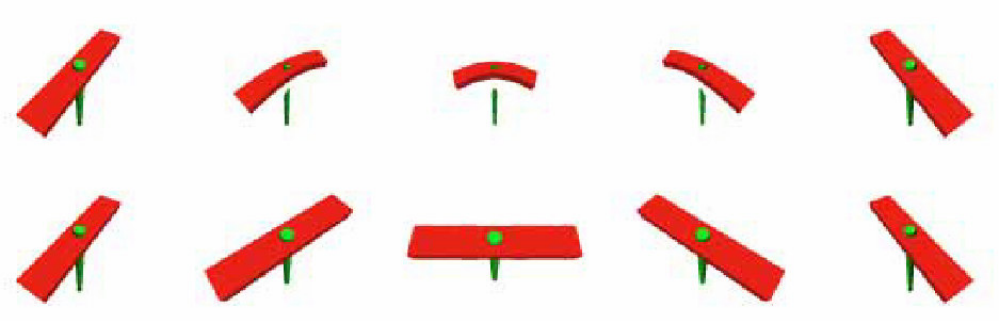


圖 2.6　上面是 Image Morphing 的結果，造成中間過程的物體外形變形縮小彎曲；下面是 View Morphing 的結果，在變形的過程中保持物體的外形(圖源自於[6])。

2.1.4 Common Problems

Image morphing 及 view morphing 皆會有 holes 的問題存在。如果左右兩張的影像拍攝的角度過大，皆會造成變形的結果失真；另外，若是對移動中的物體做變形，則會因速度及時間的問題而難以處理。

2.2 Skeletonization

骨架化目的是爲了取出以區域爲主的外形的特徵(region-based shape feature)來表示一個物體的整體外形(general form); 即是將一個 2D 的物體轉換成 1D 的線來代表，結果就像是柴枝的外形。

骨架(Skeleton) 的概念是由 H. Blum 以 Media Axis Transform (MAT) 或 Symmetry Axis Transform (SAT) 所推導出來的。對每一個在物體中的點，MAT 可以決定其最接近的邊界點(boundary point); 如果該點有超過 2 個以上的邊界點，那麼此點就屬於骨架上的點。

2.2.1 Technology of Skeletonization

骨架化的方法大致上可以分爲三類：

- Skeletons form Voronoi diagrams
- Medial axis extraction from a distance map
- Topological thinning

由於 thinning 的方法是漸漸的剝去物體輪廓直到再也不能剝去爲止(否則就會使得骨架分離)，並不能確切的保證骨架是否正確；而以 distance map 取中間軸的手法又容易使得骨架的結果爲分離狀態，必須要另外連接起來。因此進行骨架化的工作使用 Voronoi Diagram 爲基礎的方法已經越來越爲多人所採用。

2.2.2 Discrete Voronoi Skeletons

Discrete Voronoi Skeleton 的優點在於保留物體的連接性及幾何性，比起以往的演算法更加健全且有效率。Discrete Voronoi Skeleton 的建立，首先是依據物體外形的邊界點所建立的 Voronoi Diagram，以測量函數移除雜訊的干擾，建立一個結構化的骨架組織；如圖 2.7 所示，若中間軸(media axis)在物體較內部，則受到邊界點的影響而改變的輻度較小。就圖中的 m_A 與 m_C 而言，m_C 的 anchor distance 較短，所以，比較有可能因爲雜訊的影響而改變；m_A 在物體較內部的地方，且 anchor distance 較長，所以，有較大的機會成爲物體的主要骨架之一。所謂 DVMA 的邊的 anchor distance 是指兩個邊界上的點(anchor

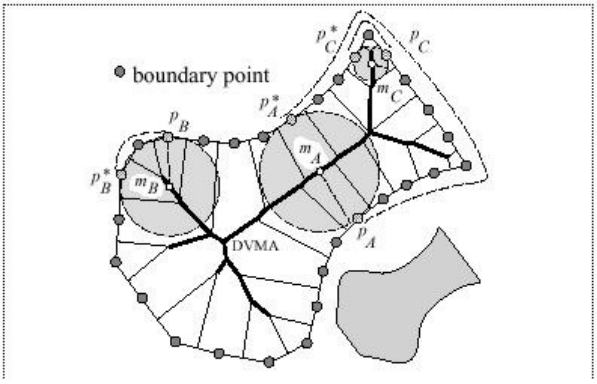


圖 2.7 exploiting the distance between point sites [1]

point)的距離，在物體的外形邊界上是最短的。

爲了有效計算 anchor distance，進而決定物體中的每個 Voronoi Diagram 主要的 Skeleton，必須推導出一套擷取骨架的規則。以下三

小節將介紹三種不同的規則化函數 Potential Residual、Circularity Residual 以及 Chord Residual。透過規則化函數計算出每個骨架 edge 的權重，再利用臨界值的設定就可以刪除多餘不必要的骨架，建立出 DVSK。

2.2.2.1 Potential Residual

定義： $\Delta R_P(e) \stackrel{def}{=} w_{AB} \stackrel{def}{=} dist^B(e), e \in DVMA(S)$

Potential Residual 是對每個邊給予一個權重值，表示其 anchor point 的集合 P_A 與 P_B 之間的最短邊界路徑長度 W_{AB}。找出 W_{AB} 是很直覺容易的，只要將邊界上的點依序給予一個連續編號值就可以求出。

2.2.2.2 Circularity Residual

如圖 2.8 所示，P_A 及 P_B 兩點爲圓與邊界的切點，m 爲圓心，r 爲半徑。Circularity residual 即是由比較邊界路徑的長度 與圓周長度 W_{AB} 來取得。藉由 Circularity Residual 的計算可以得到初步的 DVSK。Circularity Residual 如下定義：

$$\Delta R_C(e,m) \stackrel{def}{=} W_{AB} - b_{AB} = \Delta R_P(e) - b_{AB}, e,m \in DVMA(S)$$

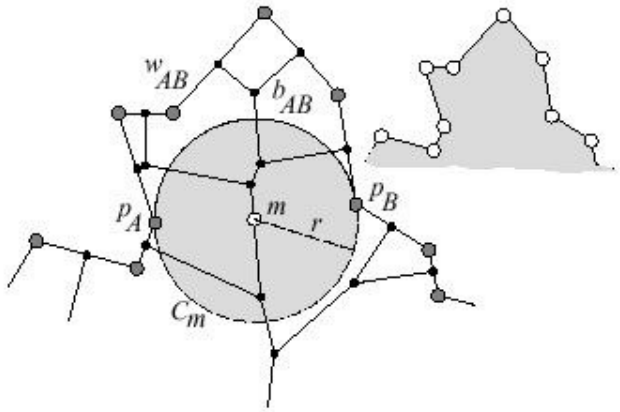


圖 2.8 Circularity Residual (圖源自[1])

2.2.2.3 Chord Residual

如圖 2.9 所示，S_{AB} 爲弦長 P_AP_B 的長度。藉由 Chord Residual 的計算可以刪除不必要的骨架。

Chord Residual 定義：

$$\Delta R_H(e) \stackrel{def}{=} W_{AB} - S_{AB} = \Delta R_P(e) - S_{AB}, e \in DVMA(S)$$

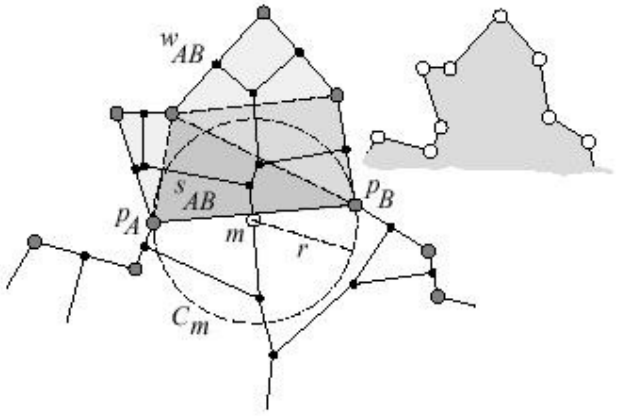


圖 2.9 Chord Residual (圖源自[1])

2.2.2.4 The Skeleton Pyramid

一般骨架化都會有的共同問題，即是有部份的骨架不能利用骨架分支的長度或是骨架對物體邊界之間的距離可以判斷的，所以提出 Pyramid 的方法可另外以其他的方式考慮是否爲骨架(圖 2.10)。它主

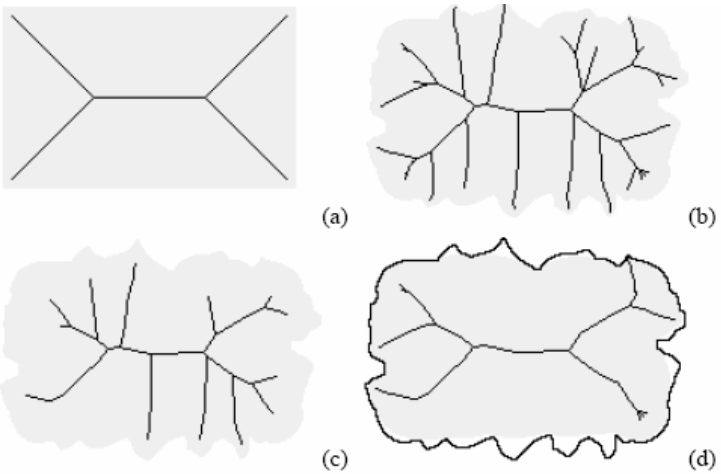


圖 2.10 (a)簡單的長方形與它的骨架，臨界值 T=3.0，chord residual。(b)鋸斷狀的長方形與它的骨架，T=3.0，chord residual。(c)提高 T 至 20.0 並沒有刪除多餘的分支，卻造成有意義的骨架被刪除。(d)使用 Skeleton Pyramid 技術。(圖源自於[1])

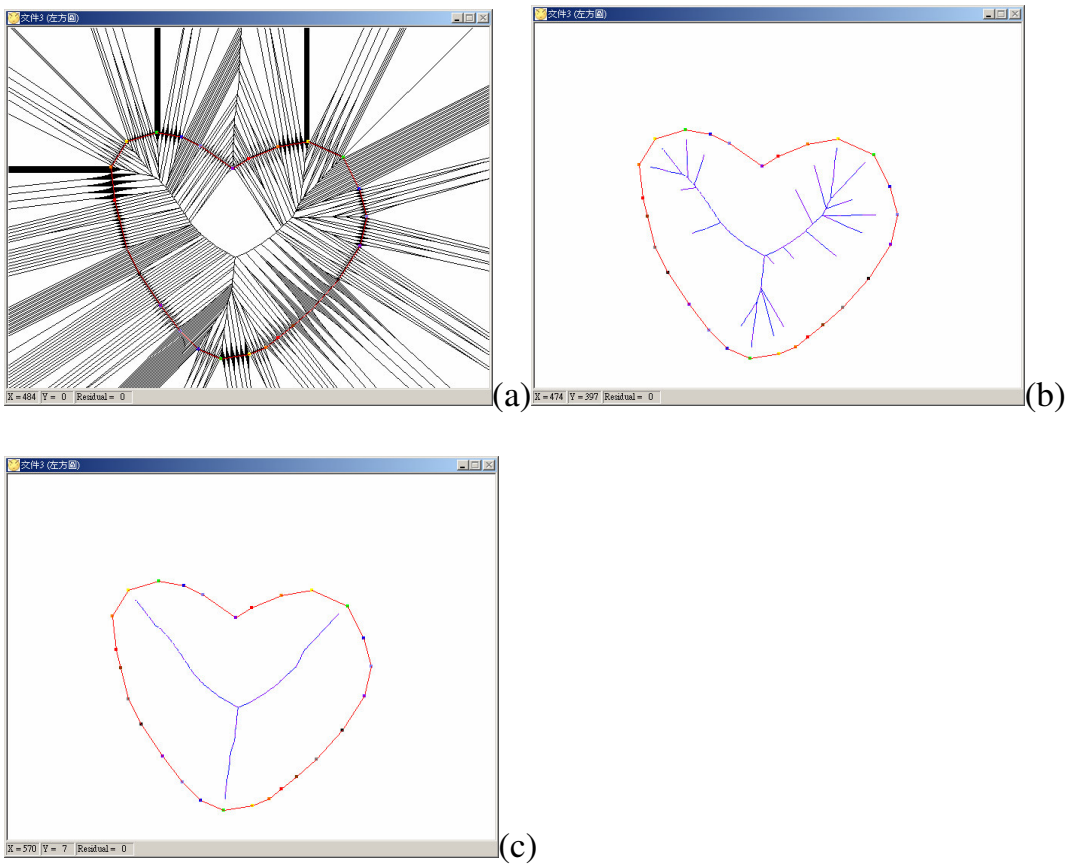
要的步驟是從規則函數最大的骨架開始進行骨架的拜訪並記錄到一個表格中，然後對此表格中每個骨架邊以規則函數值由大至小排序，最後使用比例臨界值決定最後所保留之骨架，也稱爲 First Order Skeleton。

2.2.2.5 DVSK Conclusion

透過以上的介紹得知，取得 DVSK 骨架的步驟依序爲：

1. 計算物件邊界點的 Voronoi Diagram。
2. 使用規則函數、設定臨界值而取得骨架。
3. 使用 Skeleton Pyramid 取得更好的骨架結果。

使用此技術不但可僅僅使用臨界值的設定就取得好的骨架，並且透過 Pyramid 技術可以取得各種不同解析度的骨架資訊(圖 2.11)。



(a) (b) (c)

圖 2.11 (a)是做 VD 計算後的結果，也稱為 Discrete Voronoi Medial Axis。(b)DVSK，threshold=10.0，並未使用 Pyramid 技術。(c)使用 Pyramid 方式所產生的結果，threshold = 0.7。

Chapter 3

Algorithm

在本章中將分成以下數小節來介紹本系統的架構與演算法的流程圖以及所使用的相關技術。依照順序首先 3.1 先解釋整個系統的架構，接著介紹 3.2 Voronoi Diagram 的演算法實作、3.3 Discrete Voronoi Skeletons 的實作、3.4 與 3.5 合併骨架 edge 與邊界點跟骨架的對應計算方式，以及 3.6 Feature-based image morphing 的實作，最後是 3.7 動作樣本的腳本建立方法。

3.1 System Configuration

本系統的構想如圖 3.1 所示。首先將輸入的影像做影像處理的工作，並且替物體進行有層次深度的邊框。接下來計算出該物體的骨架資訊，設定運動的關節點之後套用系統提供的動作樣本，取得改變後的骨架。利用改變後的骨架圖求得物體對應的動作位置，進行貼圖，完成一張畫格的結果。

本系統的實際操作流程則爲圖 3.2。

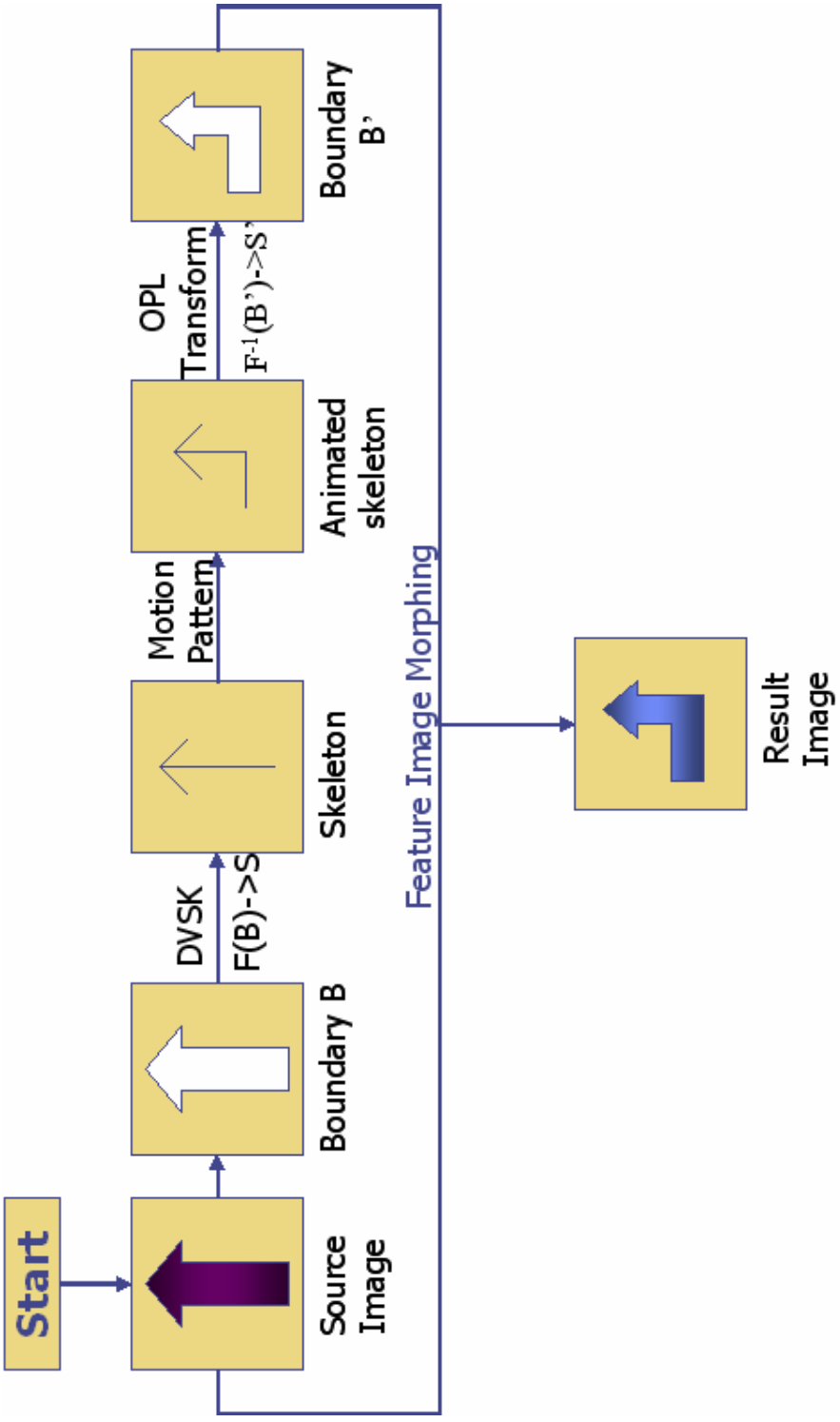

Start
Source Image
Boundary B
DVSK
F(B)->S
Skeleton
Motion Pattern
Animated skeleton
OPL Transform
F-1(B')->S'
Boundary B'
Feature Image Morphing
Result Image

圖 3.1　系統架構圖


Input images and bounding object
Shape skeletonization
Set the joints of skeletons
Skeleton merge
Motion pattern selection
Image morphing

圖 3.2　系統操作流程圖

3.2 Voronoi Diagram Algorithm

爲了取得物體的分離的 Voronoi 骨骼，則必須先要實作出 Voronoi Diagram 的演算法。以下爲演算法的詳細介紹。在平面上求點集合的 VD 演算法，可分爲三類: 累加法(增加)，分而治之法(分並且克服)，以及掃瞄法(sweepline) 。累加法是從一個點(場所) 開始，一次輸入一個點去計算出對應的 VD，此種演算法複雜度爲 $O(n^2)$。而令人感興趣的掃瞄法及分而治之法，時間複雜度皆爲 O(nlogn)，不過掃瞄法對於平行的點和共圓的點都要以特例作另外的處理，而分而治之的方式困難在於合併時的實作難度很高。Robert L.所採用的演算法是一種累加法及分而治之法的混合，於是它爲一個有累加法的簡單與分而治之的樣式所折衷而成的演算法，它對於平行或垂直的點集合可以輕易的作處理，適合以圖形形狀產生的點集合特性。

這個演算法有以下幾個要實作的程序：

- NEXTRAY operation：取得 VD 的下一個射線
- DISCARD operation：加入新 site 後校正 VD 的程序
- Compute convex hull
- VOROLINK procedure：合併 VD 的程序

我們將分成數個小節詳細介紹本演算法的概念與以上的程序。

3.2.1 Main Procedure

這個實作借用了分而治之很好的想法，基本的概念如圖 3.3：

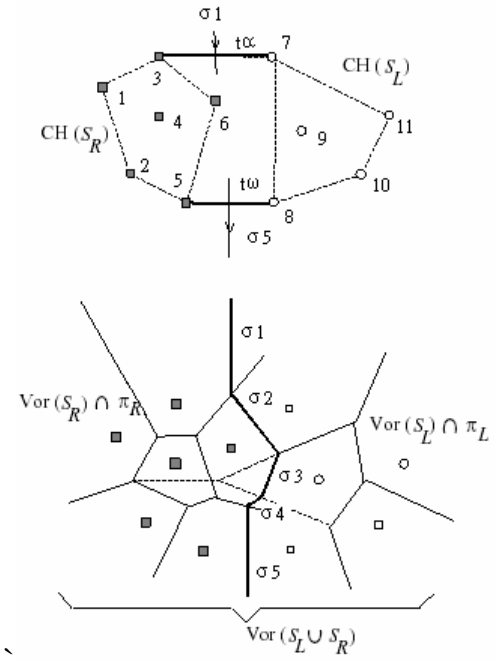


圖 3.3　Divided-and-Conquer 的合併方法

(b)舉例了以σ邊界線方向的左半邊(p7-11) 及右半邊(p1-p6) 合併兩者 VD 時的情形。在我們能建立σ邊界線開始合併之前，我們必須要先找出σ起點與終點的元素。我們發現在 b 圖中起點σ1 與終點σ5 都是射線，意即兩者都是 open Voronoi polygon 的終止元素。更進一步來說，open Voronoi polygon 的點(site) 都會是 convex hull 的點集合中的成員，於是找出σ進去及出來的線段就可透過找出兩者間 convex hull 的 supporting lines tα和 tω來達成(圖 3.3(a))。

找出 tα 和 tω 後，tα 的兩個端點就可以建立出第一個 σ 的元素，然後開始搜尋左半部或右半部的 VD 與 σ 相交，建立出 σ ，同時建立過程中會將左右兩半的 open polygon 封閉住，並且這些(open polygon 的)site 會變成合併後 convex hull 的內部點。就這樣不停地曲折行走，直到最後 σ 跨過了 tω。

我們要實作的演算法與上述的技術最大不同在於，我們是從數位影像中一次截取一條平行上的點。所以，這些排序好的點會送進演算法中，並且一次處理一條平行上的所有點，來代替分而治之中兩兩相併的方式。這個演算法最主要的缺點在於他的時間複雜度，如果每條掃瞄線都只有一個點存在的話，這個演算法基本上就會變成與累加法相同，時間複雜度則會變成 $O(n^2)$ 。不過如果真的每條掃瞄線只存在一個點的話，整張平面上的點數量也會很少，相較於影像的大小，演算法仍然會在合理的時間內完成。整個演算法的概要就如表 3.1 所描述，在程序 VORONOI 中最重要的工作就在於第十行的 VOROLINK。

VOROLINK 函式合併了先前完成的 Vor(Si-1)以及現在掃瞄線正在處理的 VD Vor(Li)。第四行的 VOROPRESCAN 則在處理如果所有的點皆共線時，直到偵測出有非共線的點，才跳至第六行開始進行主要的迴圈。

procedure VORONOI

(* S_i: 第 i 個掃瞄線{L_i}的點集合, 0 ≦ i ≦ i_{max}, Vor(S_i): S_i的 Voronoi diagram *)

1. **begin** i:=0; S_i :=$\emptyset$;
2. **while**(i<i_{max} ∧ S_i ⊏ 直線) **do begin**
   (* 建立出初始的 VD 直到有一個正規 Vor(S_i)，如 Vor(S_i)至少包含了一個 Voronoi 的頂點 *)
3. 累加 i; 抓取 第 i 個掃瞄線 L_i;
4. Vor(S_i) := VOROPRESCAN(Vor(S_{i-1}), L_i);
   (* VOROPRESCAN: adapted version of procedure VOROLINK at line 10 *)
5. **end**;
6. **while** (i< i_{max}) **do begin**
7. 累加 i; 抓取 第 i 個掃瞄線 L_i;
8. 計算出 tα 與 tω: dvivding chain σ 的進入與跳出線段
9. Vor(L_i) := L_i的中垂線圖 (* 像梳子般形狀的 VD 圖 *)
10. Vor(S_i) := VOROLINK(Vor(S_{i-1}), Vor(L_i));
    (* link 'comb' with VD computed so far *)
11. **end**;
12. **end;**

表 3.1 VORONOI 主程序

3.2.2 The NEXTRAY Operation

在合併過程中，其中一個工作是爲了要找出與σ相交的邊，所以會掃瞄 Voronoi polygon。而我們知道與σ相交的 Voronoi polygon 都是開放的，所以掃瞄就會從射線開始，而且會沿著下一個射線做尋找直到遇見合適的射線，如圖 3.4。因此就必須有一個能取得下一個射線的函式。表 3.2 爲 NEXTRAY 的函式虛擬碼。

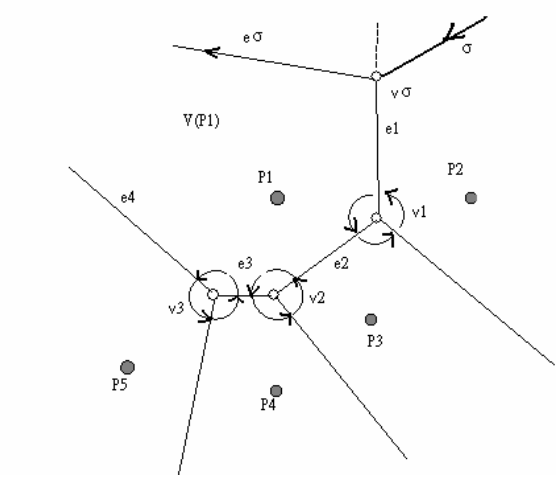


圖 3.4　NEXTRAY 工作示意圖

```
function   NEXTRAY (V(PQ)，ecur)

(*starting at edge ecur∈V(PQ)，proceeds to the end of (open) polygon
   V(PQ)，returns semi-infinite ray eterm which terminates V(PQ)*)
1.  begin
2.      ecur :=initial location in V(PQ)
3.      Vref :=(reference) vertex of ecur which is closer to eterm
            Assuming a clockwise scan of V(PQ)
4.      repeat
5.        if(tail_vertex(ecur) = Vref ) then
6.          ecur := tail_edge(ecur)
7.        else
8.            ecur :=head_edge(ecur);
9.        if(tail_vertex(ecur) = Vref) then
10.         Vref :=head_vertex(ecur)
11.       else
12.           Vref :=tail_vertex(ecur)
13.   until(rank(ecur)＜2);    (* Open end of V(PQ) encountered *)
14.   return ecur ≡ eterm;
15. end.
```

表 3.2 NEXTRAY 函式

3.2.3 The Discard Operation

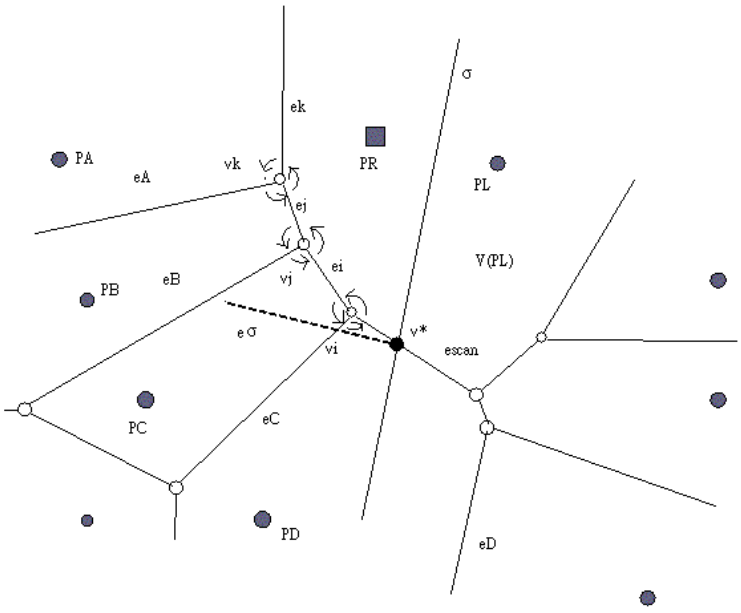


圖 3.5　DISCARD 工作示意圖

假設 σ 的右邊都沒有另外的邊存在，那麼我們就不需要從 DECL 中做刪除邊或端點的工作，但是如果有像圖 3.5 的情況發生，就表示有一些原先已經計算好的 VD 必須要做更改。在圖 3.5 當中，PR 是將要加入的點，在 Voronoi polygon V(PL) 中是不符合意義的一部份，即爲邊 ei，ej，ek， 和端點 vi，vj，vk 。所以多角形 V(PL)，V(PA)，V(PB)，V (PC) 和 V (PD) 跟它們的 DECL 結構資料都要重建，而這項工作就是由 DISCARD 程序來完成。(表 3.3，3.4)

圖 3.6 是處理完成後的情形。

```
function DISCARD (V(p_Q), e_cur)

(* starting at edge e_cur (must be semi-infinite ray)
   in V(p_Q), discards all edges between e_cur
   and edge e_cut intersected by dividing chain σ,
   inclusive e_cur, returns e_cut *)

1.  begin
2.    e_cur := edge where discard operation should begin;
        (* e.g., e_k in Fig. A.3 *)
3.    if (R_σ(e_cur) = ∅) then goto line 44;
4.    v_ref := tail_vertex(e_cur);
5.    e_next := tail_edge(e_cur);
6.    repeat (* while no intersection with V(p_Q) found *)
7.      repeat (* rotate about reference vertex v_ref *)
8.        H-stack ⇐ e_next;
            (* put e_next onto the top of H-stack, the stack
               of future 'intersection' candidates *)
9.        if (tail_vertex(e_next) = v_ref) then begin
10.         e_after := tail_edge(e_next);
11.         if (rank(e_next) = 2) then
12.           reverse orientation of e_next;
              (* exchange head and tail data *)
13.         rank(e_next) := rank(e_next) - 1;
14.       end
15.       else begin
16.         e_after := head_edge(e_next);
17.         rank(e_next) := rank(e_next) - 1;
18.       end;
19.       e_next := e_after;
20.       if (tail_vertex(e_after) = v_ref) then
21.         e_after := tail_edge(e_after);
22.       else
23.         e_after := head_edge(e_after);
24.     until (e_after = e_cur);
25.     E-stack ⇐ e_cur; (* put e_cur onto the top
          of E-stack of free edge references *)
26.     V-stack ⇐ v_ref; (* put v_ref onto the top
          of V-stack of free vertex references *)
27.     e_cur := e_next;

28.     if (R_σ(e_cur) = ∅) then
          goto line 38; (* exit outer repeat loop *)

29.     if (tail_vertex(e_cur) =v_ref) then begin
30.       v_ref := head_vertex(e_cur);
31.       e_next := head_edge(e_cur);
32.     end
33.     else begin
34.       v_ref := tail_vertex(e_cur);
35.       e_next := tail_edge(e_cur);
36.     end;
37.   until (false); (* i.e., forever, matches repeat at line 7 *)
38.   if (tail_vertex(e_cur) = v_ref
39.     ∧ rank(e_cur) = 2) then begin
40.     reverse orientation of e_cur;
41.     rank(e_cur) := rank(e_cur) - 1;
42.   end
43.   else rank(e_cur) := rank(e_cur) - 1;

44.   e_cut := e_cur; return e_cut;
45. end.
```

表 3.3、3.4　DISCARD 函式的虛擬碼 (from [1])

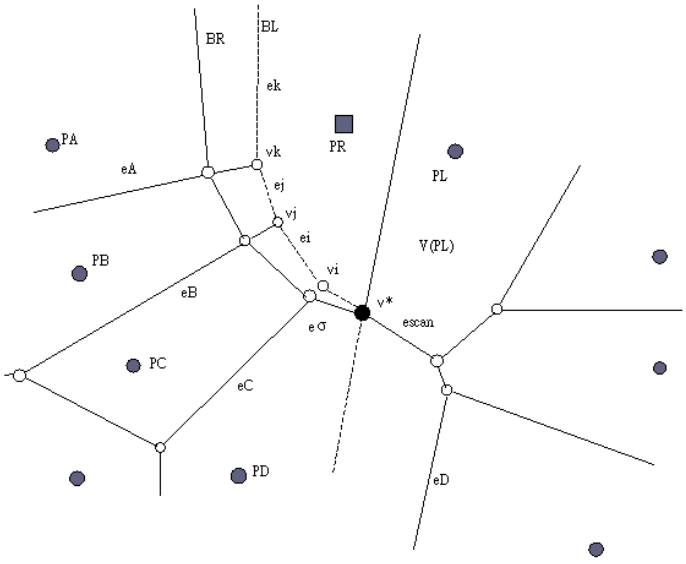


圖 3.6　修正後的結果

3.2.4 Computing Convex Hull

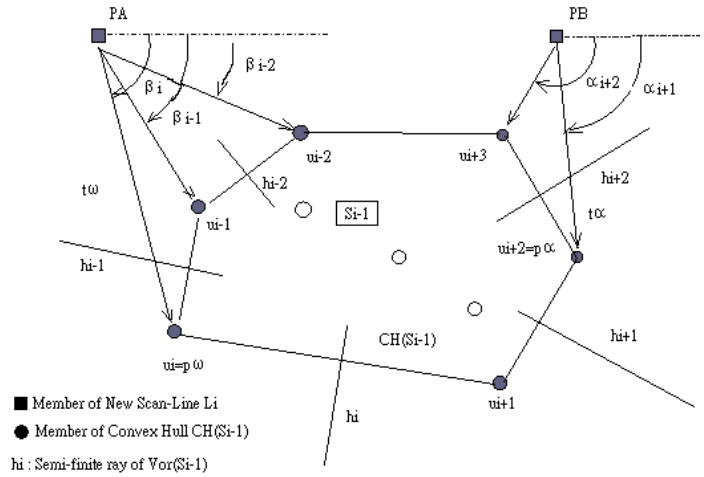


圖 3.7　計算 Convex Hull

計算點集合 S 的 convex hull CH(S)是計算 VD 的另一個重要工作。根據先前圖 3.3 所提及的，爲了要找出 supporting line tα 和 tω，所以我們必須計算出 CH(Si-1)與 CH(Li)。圖 3.7 描寫出一個簡單的解決方式，當 Vor(Si-1) 計算完成後，將掃瞄線 Li 的最左與最右邊的點(在 Li 唯一的 supporting line 兩個候選點)與 CH(Si-1)中的每個點計算出夾角，則 max(αk) 與 min(βk)分別可找出 Pα 與 Pω(注意 αk，βk<0) 。但是只找出 supporting segment 是不足以做合併動作的，還必需要找出開始及結束會掃瞄到的 Voronoi polygons。所以我們就要想出尋找 Vor(Si-1)射線的方式來實作，以下有三個不同的解決方案來達成怎麼儲存 convex hull 的目的：

解決 1.(直接從 DECL 中取出 convex hull) 直接使用 NEXTRAY 的程序依逆時針找出 convex hull，但有兩個缺點：必須再做另一個類似的程序來達成順時針找 convex hull，另一個缺點則是 NEXTRAY 會一直拜訪 edge 一直到找到爲止。不過因爲它不需要另外儲存 convex hull 資訊，所以也算是一個很好的做法。

解決 2.(射線儲存在雙向串列中) 將 open polygon 的射線直接存在雙向串列中則可以任意方向的尋找資料，並且這樣將可以廢棄 NEXTRAY 而不需使用。但串列必須要在處理 dividing chain 時做同步的更新。

解決 3.(射線儲存在雙向佇列中) 第三個方式就是使用雙向佇列來儲存 convex hull 資料。這樣的方式雖然在建立σ時還是需要 NEXTRAY，但是透過此方式來搜尋 supporting segments 變得更有效率。依照圖 3.8 來說，要計算 tα時將會向佇列要求資料依序爲 6，5，4，3，2(tω則爲 2，3，4，5，6)。這種佇列要更新也很簡單，只要在運算完σ後插入進入及結束的兩條射線即可。因此我們最後選擇此種方式來儲存 convex hull 的資訊。

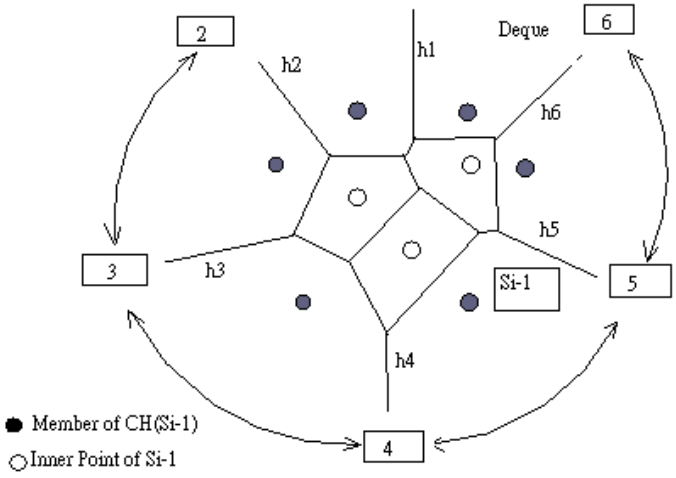


圖 3.8　使用雙向佇列儲存 Convex Hull 的方法

3.2.5 The VOROLINK Operation

在前面我們已經提出了幾個重要的工作程序後，最後我們要介紹控制 dividing chain σ 建立的程序 VOROLINK。

爲了要解釋 VOROLINK 的機制，讓它看起來不會很煩瑣麻煩，所以將其分成幾個小部份來列表。在表 3.5 中，第八行初始化之後，就開始做主要的迴圈工作，第九到二十五行之間在反覆的運行直到 σ 到達了左邊的 supporting segment t_ω。第十一行的 if-condition 在控制著 Vor(Si-1)的掃瞄。如果 e_σ 還沒進入到「終止」的 Voronoi polygon $V(P_\omega)$，就會測試 Vor(Si-1)的後一個邊有無和 e_σ 相交。

如果第十一行的條件沒有成立，就表示 Vor(Si-1)的整個合併工作已經完成，於是就只剩下跟 Vor(Li)有相交要做合併的可能了(第二十四行)。

第十二與第十九行的判斷是 VOROLINK 的動作跟 DISCARD 的 H-STACK 更新做同步，當正在處理 H-STACK 中的 edge 時，則不需要呼叫 NEXTRAY 或是 DISCARD 程序。

Function VOROLINK(Vor(S_{i-1}), Vor(L_i))

(* 合併現在掃瞄線 L_i 上的 Vor(L_i)與先前已經做好的 Vor(S_{i-1}) *)
(* P_R: point site of currently modified Voronoi polygon V(P_R) on the right side of dividing chain σ
P_L: left counterpart of P_R
e_{scan}: edge of Vor(S_{i-1}) currently checked for intersection with σ
e_{comb}: currently tested element of Vor(L_i)('comb')
v_{ref}: reference vertex with respect to e_{scan}
eσ : current element of σ , vσ : previous vertex on σ
e_L: edge of Vor(P_L) to the left of σ which was previously modified or inserted
e_i﹀H-stack denotes that edge e_i was taken off the H-stack during the last access to the H-stack *)

1. begin
2. PR := upper(lager y-coordinate) endpoint of right-hand supporting segment tα ;(* ex: p10 *)
3. PL := lower endpoint of tα ; (* ex: p5 *)
4. Pω := lower endpoint of tω ; (* p1 *)
5. escan := infinite ray r from V(PL) which has PL on its right side;
(* ex: e1 *)
6. ecomb := rightmost edge of Vor(Li);
7. vref := tail_vertex(escan);
8. eL:=**nil**; eR:=**nil**; vσ :=**nil**;
9. **while**($p_L \notin t_\omega \ \vee \ p_R \notin t_\omega$) **do begin**
(* repeat until σ passes across tω *)
10. eσ :=bisector(PL PR);

表 3.5 VOROLINK 函式

```
(* execute lines 11-22 until the whole affected portion of Vor(Si-1) has been
processed *)
11.  if(e_scan ⌢ H-stack ∨ Top(H-stack) ≠ ∅ ∨ p_L ≠ p_σ) then begin
12.       if(not(e_scan ⌢ H-stack) ∨ Top(H-stack) ≠ ∅ ∨ rank(escan)>0) then
               (* search for intersection of σ with V(PL) *)
13.            escan := DISCARD(V(PL), escan);
14.       end;

          (* determine whether I(escan,eσ) of I(ecomb,eσ) are closer to
          vσ (i.e., which of V(PL) or V(PR) was first entered by σ) and update
          the DCEL accordingly; *)
15.       UPDATE-DCEL(Vor(Si-1), Vor(Li), σ ,escan, ecomb, eσ);
16.       if(x(I[e_scan, e_σ]) ≥ x(I[e_comb, e_σ])) then begin (* intersection with Vor(Si-1)
          comes first, due to monotonic property of σ only test of x-coordonates *)
17.            if(Top(H-stack)≠ ∅) then
18.                 escan ⇐ H-stack;
                      (* take off the top edge reference of the H-stack *)
19.            else if(not(e_scan ⌢ H-stack) ∨ rank(escan)>0) then
20.                 escan := NEXTRAY(V(PL), escan);
21.       end;
22.   end (* if, line 11 *)
23.   else (* only intersection with Vor(Li): case R i.e., escan=nil *)
24.  UPDATE-DCEL(Vor(Si-1), escan);
25.   end; (* while, lime 9 *)

26. create last element of dividing chain:
          semi-infinite ray ⊂ bisector(PL PR), starting av vσ ;
27.  store first and last element of σ ; (* for computation of CH(Si) in order to
     determine new ta and tw *)
28.  return Vor(Si-1) = Vor(Si);
29.  end.
```

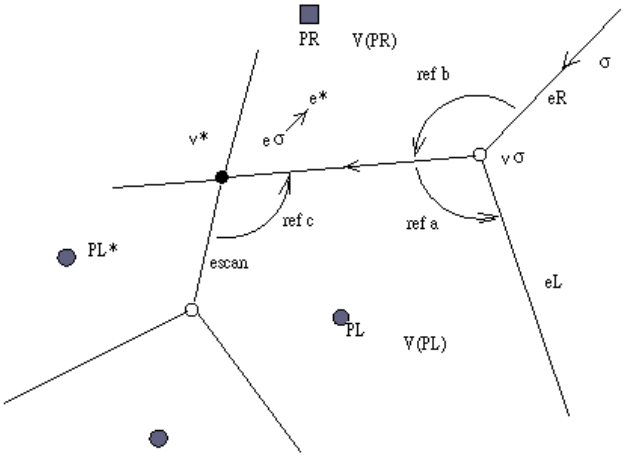
表 3.6　VOROLINK 函式(續)

當一個相交點已經被求出後，就必需要插入一個端點(vertex)並且成爲 dividing chain 的一部份，而在 UPDATE-DCEL 中完成。最後表 3.8 到 3.11 描寫出了在合併中 DECL 更新時可能的四種情形。

```
procedure UPDATE–DCEL(Vor(S_{i-1}), Vor(L_i), σ, e_scan, e_comb, e_σ)
1.  begin
      (* determine whether I(e_scan, e_σ) or I(e_comb, e_σ) are closer to v_σ:
        four different situations: *)
2.    if (x(I[e_scan, e_σ]) > x(I[e_comb, e_σ])) then
        (* intersection with V(p_L) ⊂ Vor(S_{i-1}) *)
3.      if (I[e_scan, e_σ] ≠ v_σ) then
4.        case L; (* exactly three cocircular points *)
5.      else
6.        case C; (* more than three cocircular points
                    from S_{i-1} ∪ L_i encountered*)
7.    else
8.    if (x(I[e_scan, e_σ]) < x(I[e_comb, e_σ])) then
9.      case R; (* intersection with V(p_R) ⊂ Vor(L_i) *)
10.   else (* x(I[e_scan, e_σ]) = x(I[e_comb, e_σ]) *)
11.     case LR; (* intersection with both V(p_L) and V(p_R)
                    at the same vertex: more than three cocircular points
                    from S_{i-1} ∪ L_i encountered *)

12.   Handle the four intersection types according
        to Listings A.8, A.9, A.10, and A.11;
        (* Note: whenever a (new) edge or vertex is inserted into
          the DCEL of Vor(S_{i-1}), the E-stack or V-stack, respectively,
          is searched for free reference numbers, otherwise
          the global edge or vertex counter is incremented *)
13. end.
```

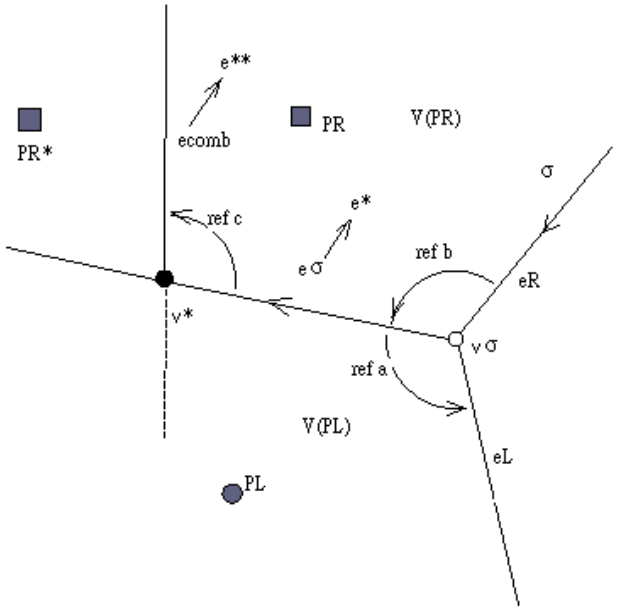
表 3.7　依照四種交點情況分別有不同的
更新 DCEL 步驟 (from [1])。



case L: (* next intersection with polygon $V(p_L)$
below/left of σ *)

1. **begin**
2. new vertex $v^* := I(e_{\text{scan}}, e_\sigma)$;
3. $\text{DCEL}(\text{Vor}(S_{i-1})) \hookleftarrow v^*$;
4. **if** $(\text{rank}(e_{\text{scan}}) < 1)$ **then** $\text{ref}_{\text{DCEL}}{}^{\text{tail}}(e_{\text{scan}}, v^*)$
5. **else** $\text{ref}_{\text{DCEL}}{}^{\text{head}}(e_{\text{scan}}, v^*)$;
6. $\text{rank}(e_{\text{scan}}) := \text{rank}(e_{\text{scan}}) + 1$;
7. new edge $e^* :\sim e_\sigma$;
8. $\text{rank}(e^*) := 2$;
9. $\text{ref}_{\text{DCEL}}{}^{\text{head}}(e^*, v^*)$;
10. $\text{ref}_{\text{DCEL}}{}^{\text{tail}}(e^*, v_\sigma)$;
11. $\text{DCEL}(\text{Vor}(S_{i-1})) \hookleftarrow e^*$;
12. $ref_a = \text{ref}_{\text{DCEL}}(e^*, e_L)$;
13. $ref_b = \text{ref}_{\text{DCEL}}(e_R, e^*)$;
14. $ref_c = \text{ref}_{\text{DCEL}}(e_{\text{scan}}, e^*)$;
15. $p_L :=$ point on other side of e_{scan} (* here: p_L^* *);
16. $e_L := e_{\text{scan}}$; $e_R := e^*$; $v_\sigma := v^*$;
17. **end**;

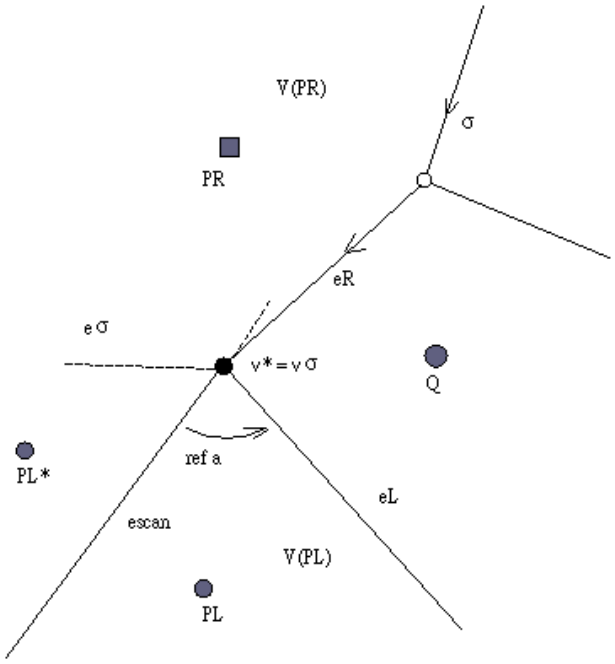
表 3.8　case L，與先前計算好的 VD 子集圖之
射線有交點時(虛擬碼摘自[1])。



case R: (* next intersection with polygon $V(p_R)$
above/right of σ *)

1. **begin**
2. new vertex $v^* := I(e_{\text{comb}}, e_\sigma)$;
3. DCEL(Vor(S_{i-1})) $\hookleftarrow v^*$;
4. new edge $e^* :\sim e_\sigma$;
5. rank(e^*) := 2;
6. $\text{ref}_{\text{DCEL}}^{\text{head}}(e^*, v^*)$; $\text{ref}_{\text{DCEL}}^{\text{tail}}(e^*, v_\sigma)$;
7. DCEL(Vor(S_{i-1})) $\hookleftarrow e^*$;
8. new edge (semi-infinite ray) $e^{**} :\sim e_{\text{comb}}$;
9. rank(e^{**}) := 1;
10. $\text{ref}_{\text{DCEL}}^{\text{head}}(e^{**}, 0)$; $\text{ref}_{\text{DCEL}}^{\text{tail}}(e^{**}, v^*)$;
11. DCEL(Vor(S_{i-1})) $\hookleftarrow e^{**}$;
12. $ref_a = \text{ref}_{\text{DCEL}}(e^*, e_L)$;
13. $ref_b = \text{ref}_{\text{DCEL}}(e_R, e^*)$;
14. $ref_c = \text{ref}_{\text{DCEL}}(e^*, e^{**})$;
15. p_R := point on other side of e_{comb} (* here: p_R^* *);
16. $e_{\text{comb}} :=$ edge $\in$ Vor(L_i) to the left of e_{comb}; (* can be **nil** *)
17. $e_L := e^*$; $e_R := e^{**}$; $v_\sigma := v^*$;
18. **end**;

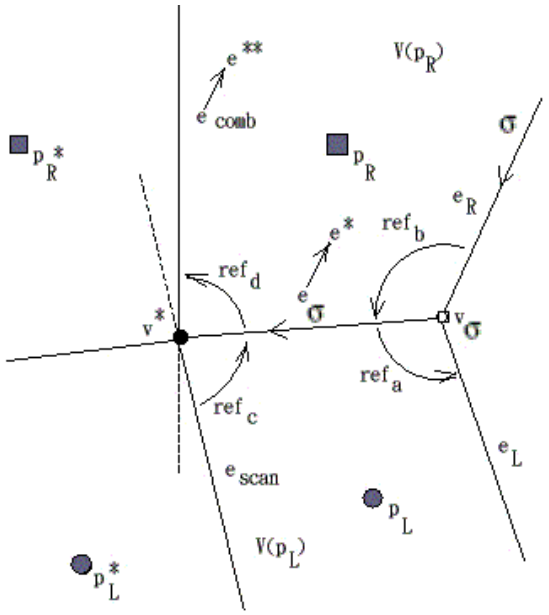
表 3.9　case R，與 comb edge 有交點時。Comb edge 即為輸入的平行點之 Voronoi edge(虛擬碼摘自[1])。



case C: (* more than three points are cocircular,
namely p_L, p_L^*, p_R, and Q;
no new edge or vertex created *)

1. **begin**
2. **if** (rank(e_{scan}) < 1) **then** $\text{ref}_{\text{DCEL}}{}^{\text{tail}}(e_{\text{scan}}, v_\sigma)$
3. **else** $\text{ref}_{\text{DCEL}}{}^{\text{head}}(e_{\text{scan}}, v_\sigma)$;
4. rank(e_{scan}) := rank(e_{scan}) + 1;
5. $ref_a = \text{ref}_{\text{DCEL}}(e_{\text{scan}}, e_L)$;
6. p_L := point on other side of e_{scan} (* here: p_L^* *);
7. $e_L := e_{\text{scan}}$;
8. **end**;

表 3.10　case C，與 VD 子集圖之頂點相交，使得射線消失成為一共圓的情況(虛擬碼摘自[1])。



case LR: (* intersection at vertex v^* with both
polygons $V(p_L)$ and $V(p_R)$ below resp. above σ *)

1. **begin**
2. new vertex $v^* := I(e_{\text{scan}}, e_\sigma)$; DCEL(Vor($S_{i-1}$)) $\hookleftarrow v^*$;
3. **if** (rank(e_{scan}) < 1) **then** ref$_{\text{DCEL}}^{\text{tail}}(e_{\text{scan}}, v^*)$
4. **else** ref$_{\text{DCEL}}^{\text{head}}(e_{\text{scan}}, v^*)$;
5. rank(e_{scan}) := rank(e_{scan}) + 1;
6. new edge $e^* :\sim e_\sigma$; rank(e^*) := 2;
7. ref$_{\text{DCEL}}^{\text{head}}(e^*, v^*)$; ref$_{\text{DCEL}}^{\text{tail}}(e^*, v_\sigma)$;
8. DCEL(Vor(S_{i-1})) $\hookleftarrow e^*$;
9. new edge (semi-infinite ray) $e^{**} :\sim e_{\text{comb}}$; rank($e^{**}$) := 1;
10. ref$_{\text{DCEL}}^{\text{head}}(e^{**}, 0)$; ref$_{\text{DCEL}}^{\text{tail}}(e^{**}, v^*)$;
11. DCEL(Vor(S_{i-1})) $\hookleftarrow e^{**}$;
12. ref_a = ref$_{\text{DCEL}}(e^*, e_L)$; ref_b = ref$_{\text{DCEL}}(e_R, e^*)$;
13. ref_c = ref$_{\text{DCEL}}(e_{\text{scan}}, e^*)$; ref_d = ref$_{\text{DCEL}}(e^*, e^{**})$;
14. p_L := point on other side of e_{scan} (* p_L^* *);
15. p_R := point on other side of e_{comb} (* p_R^* *);
16. e_{comb} := edge $\in$ Vor(L_i) to the left of e_{comb}; (* can be **nil** *)
17. $e_R := e^{**}$; $e_L := e_{\text{scan}}$; $v_\sigma := v^*$;
18. **end**;

表 3.11　case LR，與 ecomb 及 VD 子集圖同時
交於一點時之情況(虛擬碼摘自[1])。

3.3 Discrete Voronoi Skeletons

使用 Robert L.所提出的 DVSK 骨架化方法，大致上分成三個主要的步驟：(1) 計算以物體輪廓爲輸入點的 Voronoi Diagram，也稱爲 Discrete Voronoi Medial Axis。(2) 刪減 DVMA 資料的規則化 (Regularization) 方法。(3) 階層式管理骨架資訊的金字塔(The Skeleton Pyramid)。

在 3.2 中我們已經詳細的介紹了 Voronoi diagram 的演算法，在下兩節當中我們將依序的介紹 Regularization 及 The Skeleton Pyramid。

3.3.1 DVSK：Regularization

在規則化的工作當中，我們使用了 Potential Residual 的定義，將每一個 edge 賦予一個 residual value。這一個權重值是來自於 edge 的左右兩個 site 繞行於物體邊界上的最短路徑。則所有 DVMA 的 edge 皆有一個權重值，設定一個臨界值後就能刪除不必要的骨架資訊。

3.3.2 DVSK：The Skeleton Pyramid

雖然規則化的 pruning 技術能夠產生一個公平且穩定的骨架，但是還是沒有辦法解決一些骨架大量分支的問題。因此使用 Pyramid 技術可以階層式的管理複雜的骨架圖型，進而產生更簡潔並且更有意義的骨架圖。圖 3.9 以及表 3.12 說明了 Pyramid 技術。

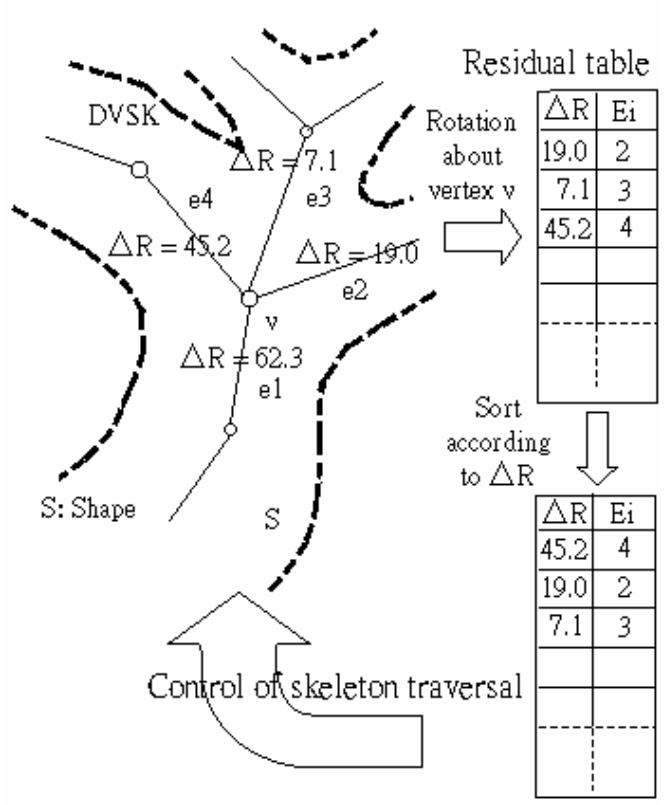


圖 3.9　Pyramid 使用了 Residual Table 進行骨架的分類

```
procedure SKELETON–PYRAMID (v_i, e_cur, h_s, T')

(* Assigns each edge an integer number denoting the level
   of the hierarchy this edge belongs to *)
1.  begin
2.    e_ref := e_cur;
3.    R-tab := ∅; (* empty residual table R-tab *)
4.    hierarchy(e_cur) := h_s;
5.    e_cur := VRot (v_i, e_cur, T');

6.    while (e_cur ≠ e_ref ∧ e_cur ≠ nil) do begin
7.      if (tail_vertex(e_cur) = v_i) then
8.        r := Δ_R(tail_vertex(e_cur))
9.      else
10.       r := Δ_R(head_vertex(e_cur));
11.     R-tab ⇐ {e_cur, r}; (* insert pair {e_cur, r} into R-tab *)
12.     e_cur := VRot (v_i, e_cur, T');
13.   end; (* while *)

14.   Sort (R-tab); (* sort R-tab according to the r values
        in descending order *)

      (* visit all incident branches ∈ Ske(S, T') on v_i *)
15.   while (Top (R-tab) ≠ ∅) do begin (* R-tab empty ? *)
16.     e_cur ⇐ R-tab; (* get next entry from R-tab *)
17.     if (tail_vertex(e_cur) = v_i) then
18.       SKELETON–PYRAMID(head_vertex(e_cur), e_cur, h_s, T')
19.     else
20.       SKELETON–PYRAMID(tail_vertex(e_cur), e_cur, h_s, T');
21.     if (Δ_R(e_cur) < w_∞) then (* else e_cur has been generated
            by two disjoint boundary segments, w_∞ ≫ 1 *)
22.       h_s := h_s + 1; (* increment level of the hierarchy *)
23.   end; (* while *)
24. end.
```

表 3.12　Pyramid 程序的虛擬碼(摘自[1])

以深度拜訪的程式結構爲基礎，從 residual value 最大的骨架 edge 的端點(vertex)爲起點開始移動，遇到分支點時，將所有的分支以 residual value 由大到小記錄在 residual table 中，則我們可以依其順序替每一個分支編上代號 h，第一個分支 h=1，第二個分支 h=2…依此

類推。最後依照使用者設定 h 的臨界值，決定保留的分支，進而達到階層式骨架的功能，也就是符合了 multi-resolution 的特性。

SKELETON-PYRAMID 一開始給予四個參數，ecur 爲最大 residual value 的 edge，vi 爲 ecur 的其中一個端點(也就是行進方向)，h=1 代表是最有意義的 skeleton，T 則爲 Regularization 時的臨界值。

VRot(vi， ek， T)爲一函式，可以以 vi 端點，取得 ek 以逆時針方向的下一個 edge。

3.4 The Skeleton Edge Merge

當我們有了 DVSK 的結果之後，由於骨架的 edge 資料過於細小瑣碎，不便於系統的利用，因此爲了簡化角度相同或近似之的骨架 edge，則需要有一個合併骨架的方法。

因爲在物件內部之骨架一定是線段，則可以計算出其向量，藉由兩個向量之內積公式：$\cos\theta = (a \bullet b) / (|a| \bullet |b|)$ 求出兩個鄰近的骨架 edge 的夾角 θ 。而合併的演算法有兩種考慮：

以本身與下一線段直接判斷是否可合併，如果可以則合併後再以合併後的新線段與下一線段繼續判斷。

以本身與下一線段判斷是否可合併，如果可以則繼續判斷再下一個，直到角度累積至臨界值後一起合併。

圖 3.10 所示，由於 θ 不會超過 0～180 度，則夾角不一定會同方向，所以要以第二種方式判斷時比較不便（因爲要考慮到累積正負夾角），因此以第一種方式實作。以 depth-first search 拜訪所有骨架 edge，每拜訪到一個 edge 時，則進入副程式 DoMerge( ) 開始處理合併問題。

圖 3.10　夾角示意圖

```
Procedure DoMerge(ecur,vcur)
begin
    if(Vertex_Rotate(ecur,vcur) != 2)
    /* vcur如爲branch交界端點則不予處理 */
        return 0;
    Edge enext = VRot(vcur,ecur);     /* 取得下一條 Edge */
    計算兩條Edge：(ecur, enext)之內積。
    計算出角度θ，如果超過90度則 θ＝180－θ。
    if(θ< MergeDegree)            /* θ小於合併臨界值則進行合併 */
    begin
        產生一新Edge newEdge，設定其DCEL結構。
        設定上下兩條 Edge 指向 newEdge。
        delete (ecur, enext);
    return(enext := newEdge)      /* 回傳新合併Edge */
    end
    return 0;
end
```

3.5 Sum up Boundary Points to Merged Skeleton Edges

爲了讓計算出的骨架在進行運動時，物體的邊界點(Boundary Points)也能相對的移動至正確的位置，所以必須先計算出每個骨架 Edge 所應有之邊界點。當邊界點所組成的邊框進行了取樣(raster crack)之後，其後一連串的步驟所計算出的 Merged Skeleton Edges 變失去了左右點(Site)的資訊，必須透過一組演算法以歸納其應有的邊界點。

利用在合併前之 Pyramid Edge 所擁有之 Site 資訊，將合併成同一條 Merged Edge 的所有細碎 Pyramid Edges 的 residual value 最大的 Edge 之兩端 Site 紀錄下來。合併工作完成後，以 Merged Edges 的 residual value 從小至大爲順序，從左 Site 爲起點右 Site 爲終點，行走最短路徑拜訪所經過之邊界點，歸納至此 Merged Skeleton Edge，如拜訪之邊界點已被歸納過，則不予歸納。尚未歸納的邊界點直接指定至規則函數最大值的物件骨架線段。

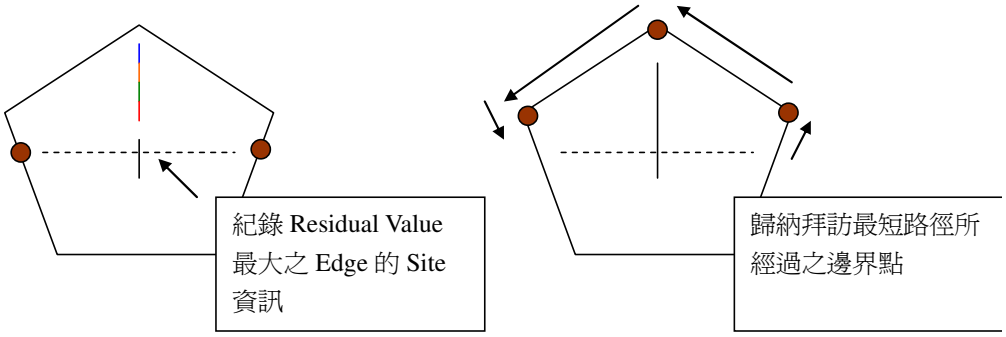


圖 3.11　歸納邊界點的方法示意圖

3.6 Feature-based Image Morphing

由 Thaddeus Beier 所提出的 Feature-based Image Morphing 是一個以影像爲特徵點爲基礎的影像變形技術。透過向量指定兩張影像的特徵，輕易的產生出中間變化的影像。Feature-based Image Morphing 的演算法包括了 Transformation with One Pair of Lines 與 Transformation with Multiple Pairs of Lines，利用前者可以計算出骨架以及邊界點的對應位置，後者幫助我們進行貼圖(Texture mapping)的工作。

3.6.1 One Pair of Lines Transformation

以圖 3.12 解釋，求目標影像的 X 像素(pixel)所對應的來源影像 X’像素資訊，則計算方式如表 3.14。而 v.perpendicular 代表 v 向量的相同長度之垂直向量，雖然這個垂直向量會有兩個不同的方向，但是只要演算法所使用的都是同一個方向的垂直向量就可以任意的擇一使用。

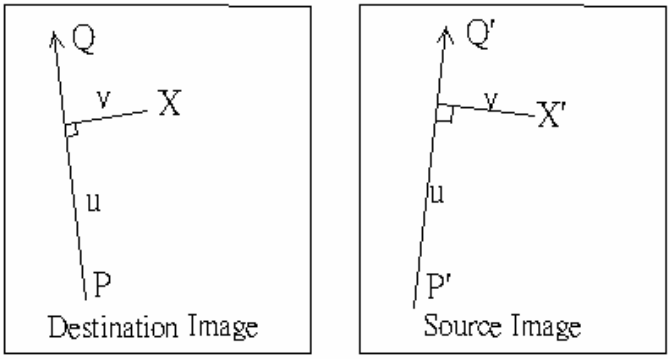


圖 3.12 One Pair of Lines Transformation 的概念

$$u = \frac{(X - P) \cdot (Q - P)}{\| Q - P \|^2} \quad (1)$$

$$v = \frac{(X - P) \cdot \text{Perpendicular}(Q - P)}{\| Q - P \|} \quad (2)$$

$$X' = P' + u \cdot (Q' - P') + \frac{v \cdot \text{Perpendicular}(Q' - P')}{\| Q' - P' \|} \quad (3)$$

表 3.14　(u, v)座標的計算式

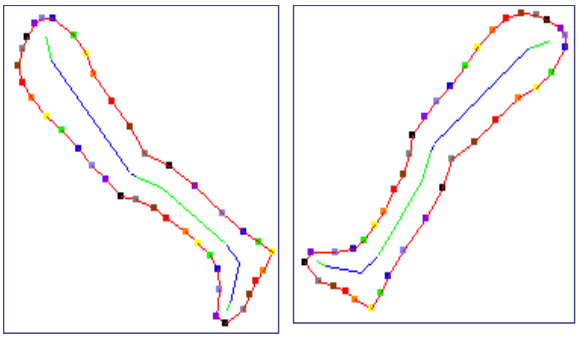


圖 3.13　利用(u, v)座標轉換使得對應於骨架的邊界點跟隨著骨架移動

利用(u， v)向量的表示式，我們則可以應用至骨架與邊界點的對應計算，如圖 3.13。

3.6.2 Multiple Pairs of Lines Transformation

Multiple Pairs of Lines Transformation 因爲要考慮的特徵向量不止爲一對，所以計算上稍微複雜一點，演算法如表 3.15 所示。對於目的影像的 X 像素，會以每一對向量去求得個別的 X_i'。位移 D_i 是每一個 X_i'與 X 的相減結果，最後除以權重 weight 得到一個平均位移值，加上 X 則爲最後的 X_i'結果。在計算權重值時，length 是向量的長度，dist 是點到向量的距離，而 a，b 與 p 則爲權重值的調整參數。一般情況下，a 會設定大於 0，b 爲 0.5 到 2 之間，p 則在 0 到 1 之間。

```
For each pixel X in the destination
    DSUM = (0,0)
    weightsum = 0
    For each line PiQi
        calculate u,v based on PiQi
        calculate X'i based on u,v amd Pi'Qi'
        calculate Di = Xi' – Xi for this line
        dist = shortest distance from X to PiQi
        weight = (length^p / (a + dist))^b
        DSUM += Di * weight
        weightsum += weight
    X' = X + DSUM / weightsum
    destinationImage(X) = sourceImage(X')
```

表 3.15　Multiple Pairs of Lines Transformation 的演算法

圖 3.14(b)則爲利用 Feature-based Image Morphing 所計算出運動後的兩張影像。

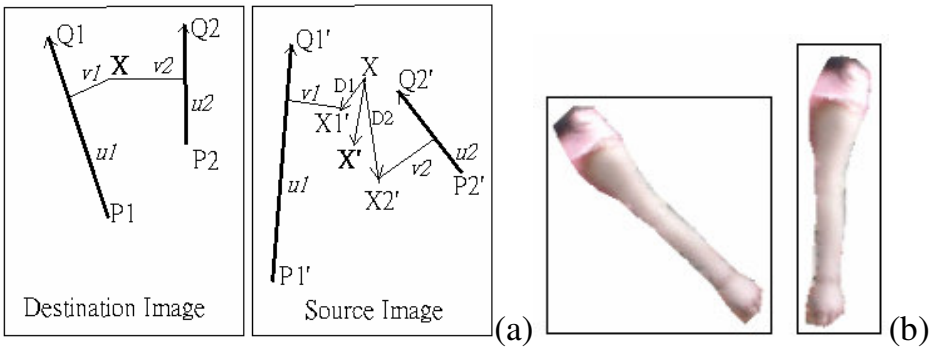


圖 3.14　(a) Multiple Pairs of Lines Transformation 的概念，
(b) 使用 Multiple Pairs of Lines Transformation 所進行的貼圖結果

3.7 Forward Kinematics Scripts of Motion Pattern

系統所提供的動作樣本是由另一個程式 Script Builder(圖 3.15)所製作而成。首先找出要輸入的動畫，利用 Script Builder 程式對此動畫進行處理，產生 Script 的工作，最後將製做好的 Script 檔案匯入至本系統中，提供使用者選取套用。表 3.16 爲 script 的檔案格式。

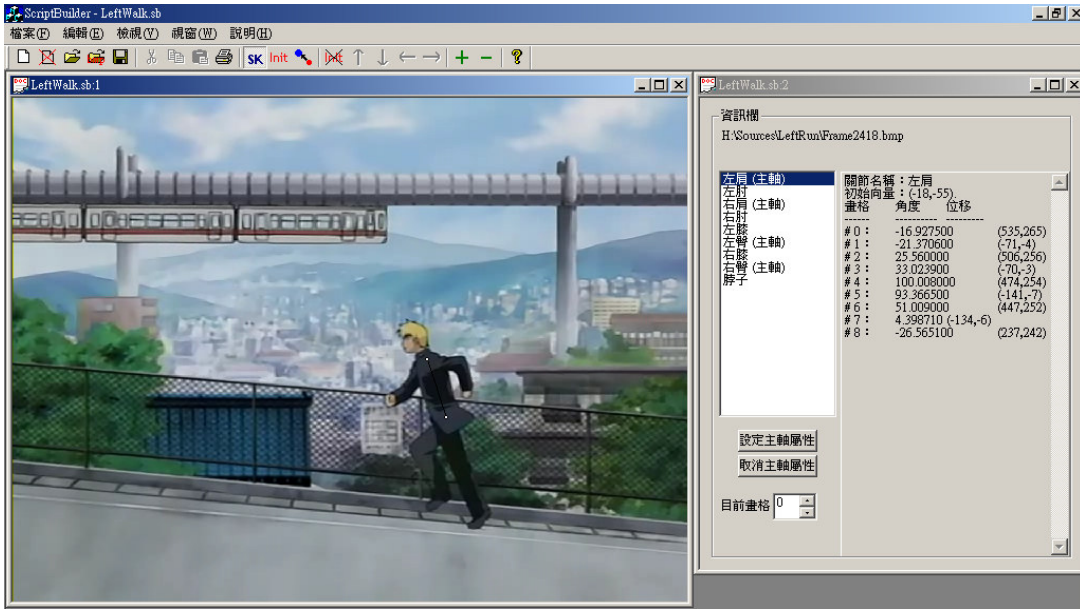


圖 3.15 Script Builder 程式畫面

Chapter 4

Conclusion

在本研究當中，我們所實作出的系統成功的從(一張或兩張的)2D 影像當中合成出相片般真實的、自然且有趣的動畫。

透過物體的骨架資訊，使得可以利用動作樣本的設定讓物體做出合理以及各式各樣的運動，而不會破壞物體的形狀或扭曲變形。

使用動作樣本的設計不但可以重覆使用於不同的影像上，並且也具有高度的擴充性，可以將任何有趣的動作匯入至系統中，使得原本單純且靜止的 2D 影像生動活潑了起來，甚至做出物體原本無法做到的運動效果。

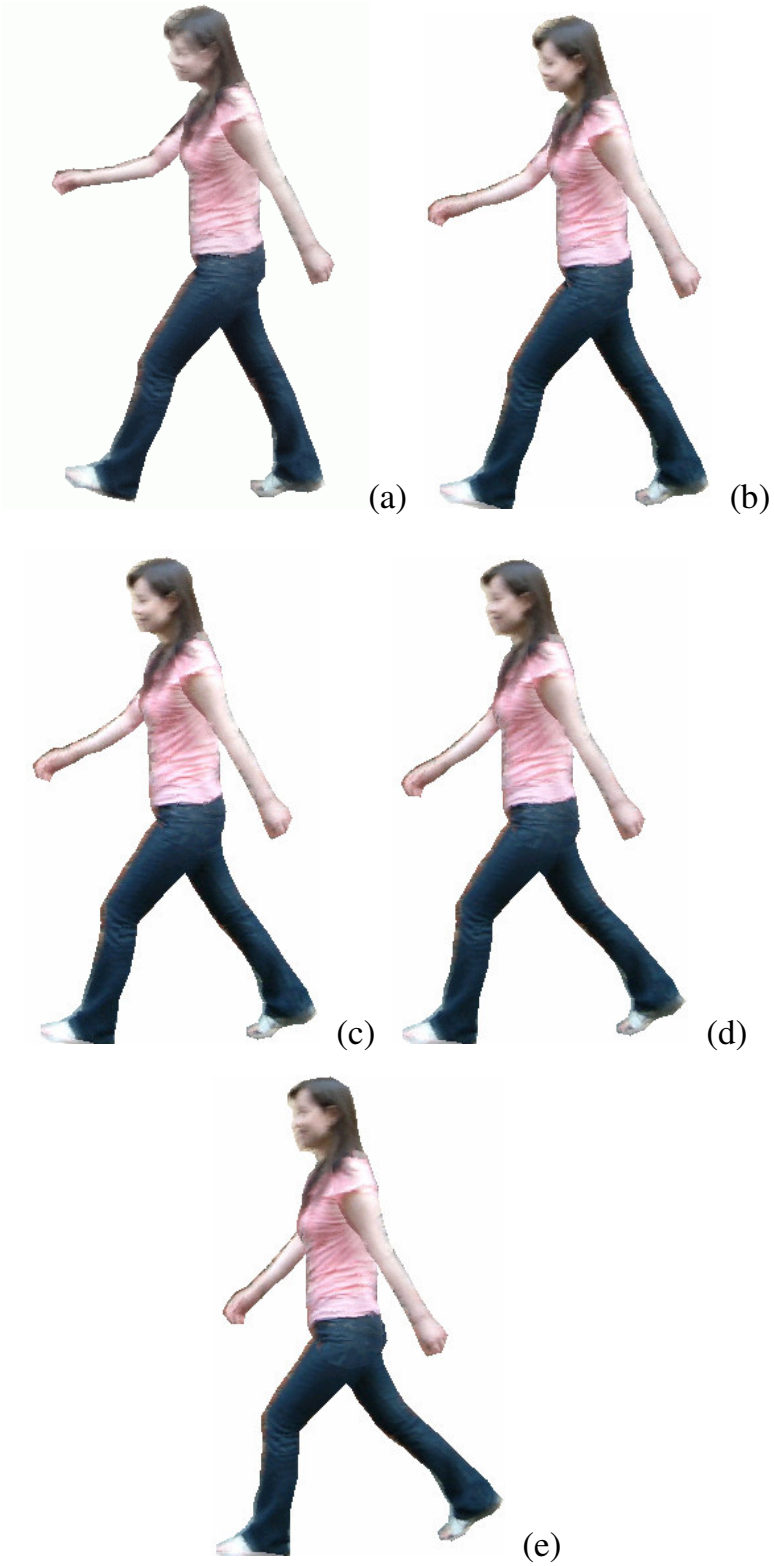


圖 4.1　(a)-(e) There are result images of interpolate sequences, interpolate value 0.1, 0.3, 0.5, 0.7, and 0.9

(a)

(b)

(c)

圖 4.2　(a)輸入的兩張影像 (b)套用跑步的動作樣本 (c)貼上背景的動畫。

Future Work

由於影像上的物體形狀是 3D 物體投射至 2D 平面的結果，所以物體之於影像的投射平面角度產生變化時，就會因爲缺乏像素的資訊以及暫時沒有考慮物體自轉的繪圖技術而使得效果不盡理想，也使得目前的動作樣本並無法提供物體自轉的相關運動。但是系統所提供的物體邊框工作中，已經含有基本的影像深度紀錄功能(如圖 4.3)，所以有著所謂的 2.5D 骨架資訊，可以模擬 3D 運動時的情況，對於解決 2D 影像要進行 3D 的運動時，有很大的幫助。

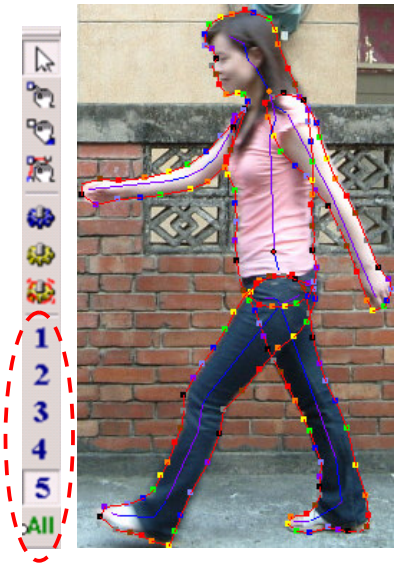


圖 4.3　程式的工具列與階層式的骨架結果

未來的工作內容爲：

- 將座標系統從二維更改爲三維，使得物體可以進行自轉的運動。
- 新增物體影像的貼圖(texture mapping)管理功能。給予物體幾個關

鍵的攝影機角度影像，就能夠生成任意攝影機角度之影像結果。對於動畫的生成，可以得到更正確的貼圖結果。

- 設計更多的動作樣本提供給使用者利用。
- 實作 Inverse Kinematics。使用 IK 的方式讓物體自動且流暢的從 A 動作變換至 B 動作，不但可以減輕在製作 Script 的負荷，更能夠將各種不同的動作樣本連接起來，使得能夠顯示的動畫更多采多姿。

A. Appendix

在本研究中，我們採用了 Microsoft Visual C++ 6.0 作爲開發工具，利用 MFC 及 STL 的函式庫，實作系統的演算法以及程式介面。在本附錄中我們以 Step by Step 圖例方式介紹系統的使用方法及步驟。

Step1: New Project

新建一個文件，此時會開啓兩個子視窗分別爲左圖與右圖(圖A.1)。

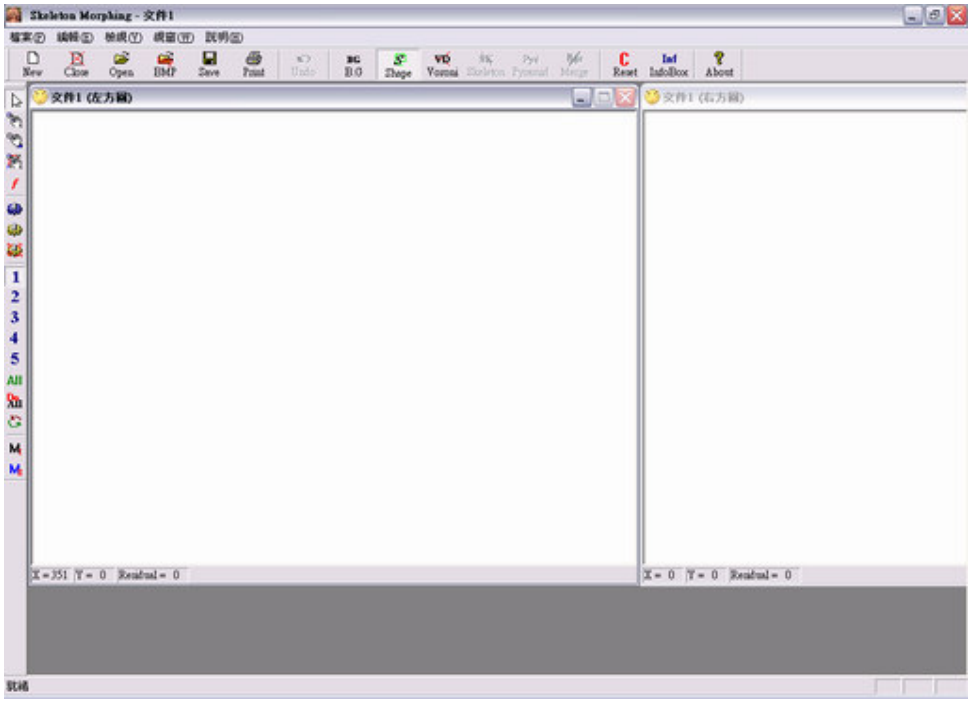


圖 A.1 新增一文件。

Step2：Load Images

載入影像檔準備處理。系統彈出對話盒 1 要求載入左圖；再彈出對話盒 2 要求載入右圖。載入完成後即便會顯示在視窗中(圖 A.2)。

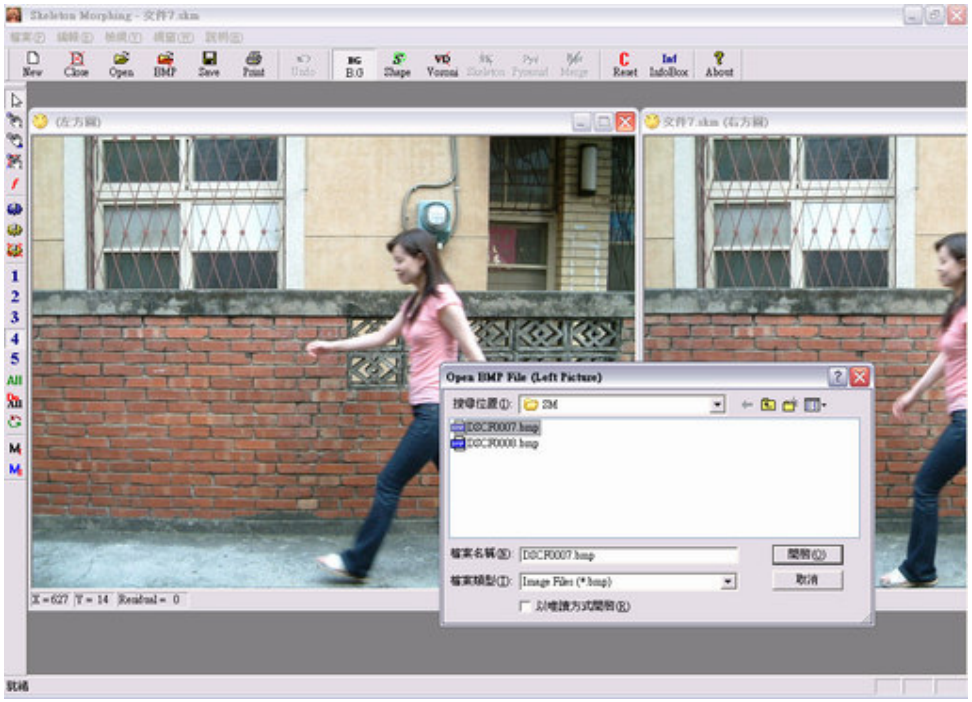


圖 A.2 載入點陣圖形。

Step3：Bound Shape

選擇 Add Point 模式(左方工具列，或快速鍵 ALT+A)，將影像中待處理的物體上邊框。依照物體深度的順序，將物體拆開來由遠至近進行處理；最深遠的部位請使用 Layer1。在兩者影像當中，請選擇較複雜的形狀來做邊框的動作(圖 A.3)。

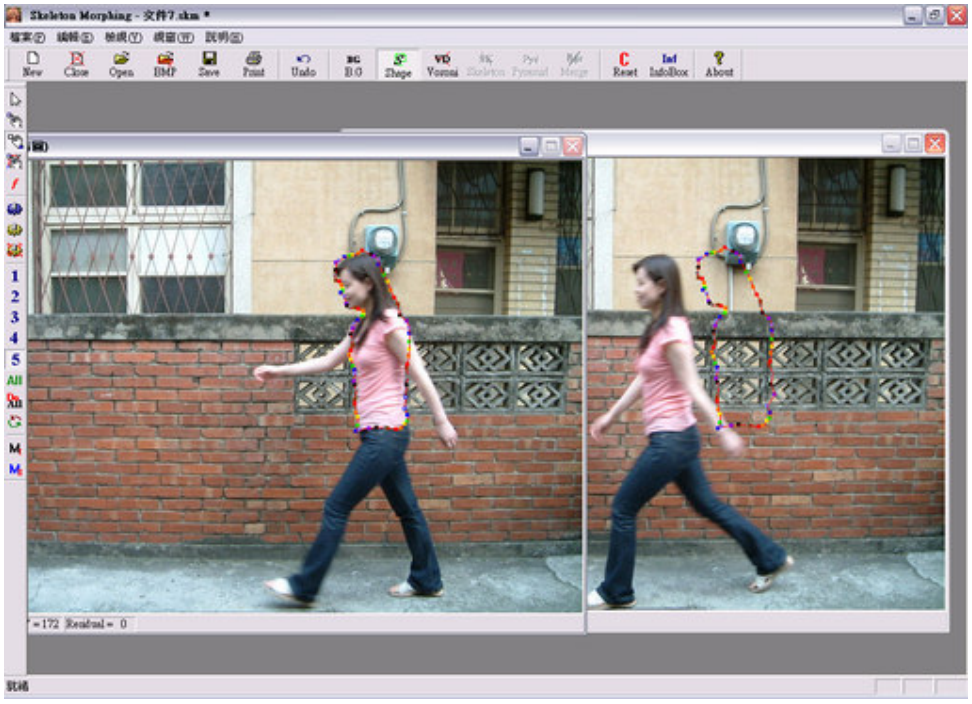


圖 A.3 將其中一張圖進行邊框，另一張同時也會出現邊界點。

Step4：Moving Boundary Points of Another Image

利用[Move Point]及[Move Boundary]的工具移動另外一張同步出現的邊框點，使得邊框點完全對應至物體的輪廓上(圖 A.4)。

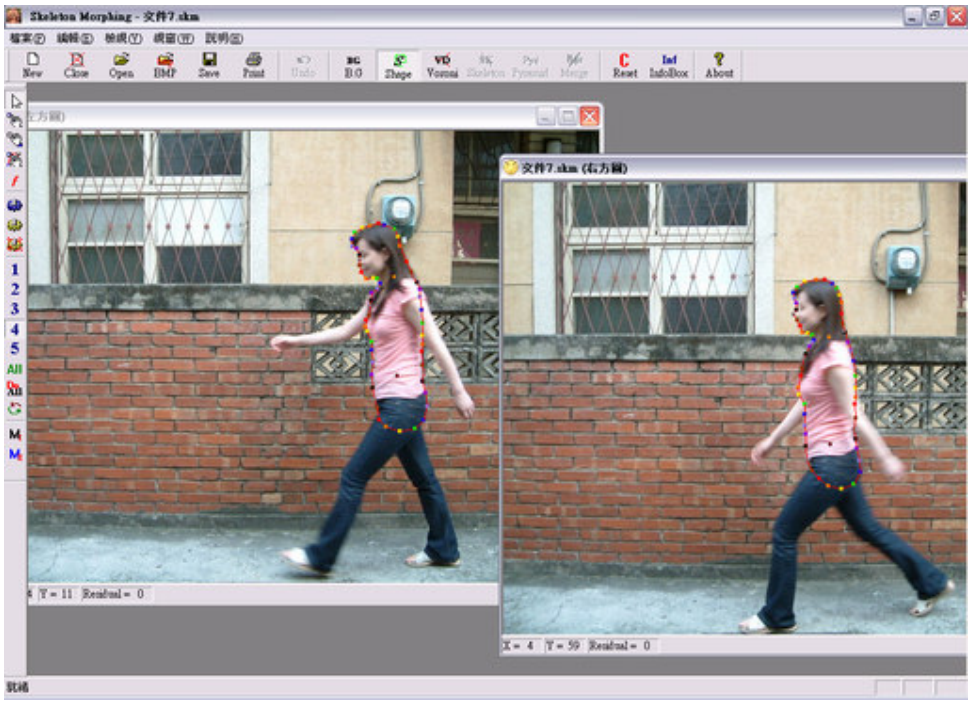


圖 A.4 將邊界點移到對應的輪廓上。

Step5：Re-do Until Bound Shape Complete

重覆 Step3、Step4，按照物體各部位的深度，由遠至近依序利用 Layer 完成物體所有部位的邊框動作，Layer 至多可使用五層。

Step6：Compute Voronoi Diagram of Raster Crack of Boundary Points

計算左右兩圖的 Voronoi Diagram(圖 4.5)。

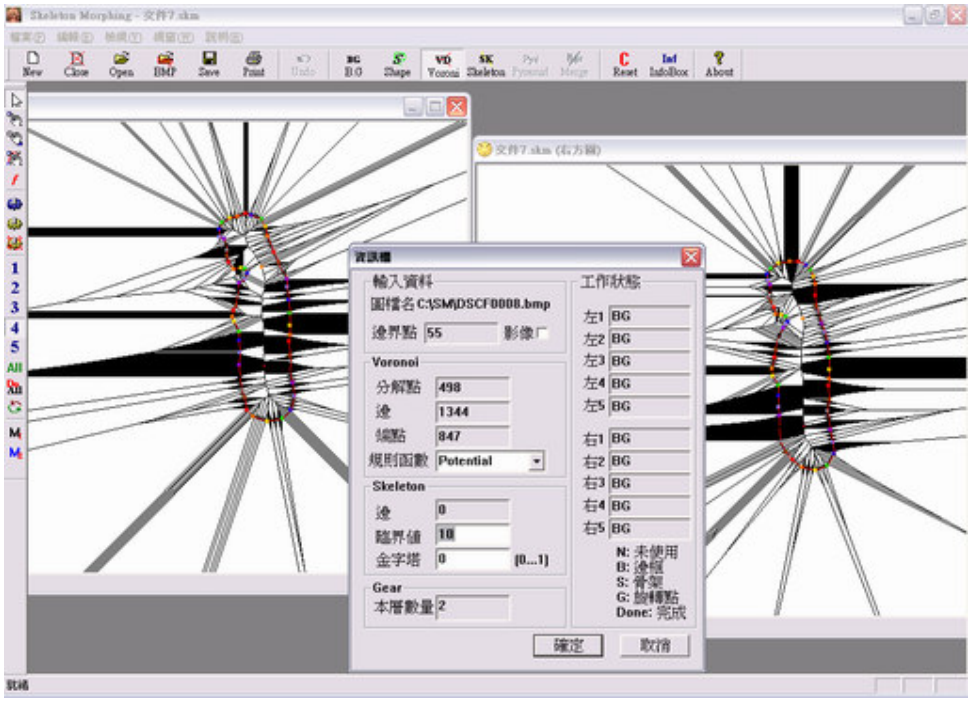


圖 A.5 計算左右兩圖的 VD，中間對話盒為資訊欄。

Step7：Compute Discrete Voronoi Skeletons with Potential Function

以 Potential Function 爲規則函數，計算 DVSK 骨架。

Step8：Compute Skeleton Pyramid

計算階層式骨架。如果對於骨架結果不甚滿意，可開啓資訊欄調整 Skeleton 的臨界值及金字塔參數(圖 A.6)。

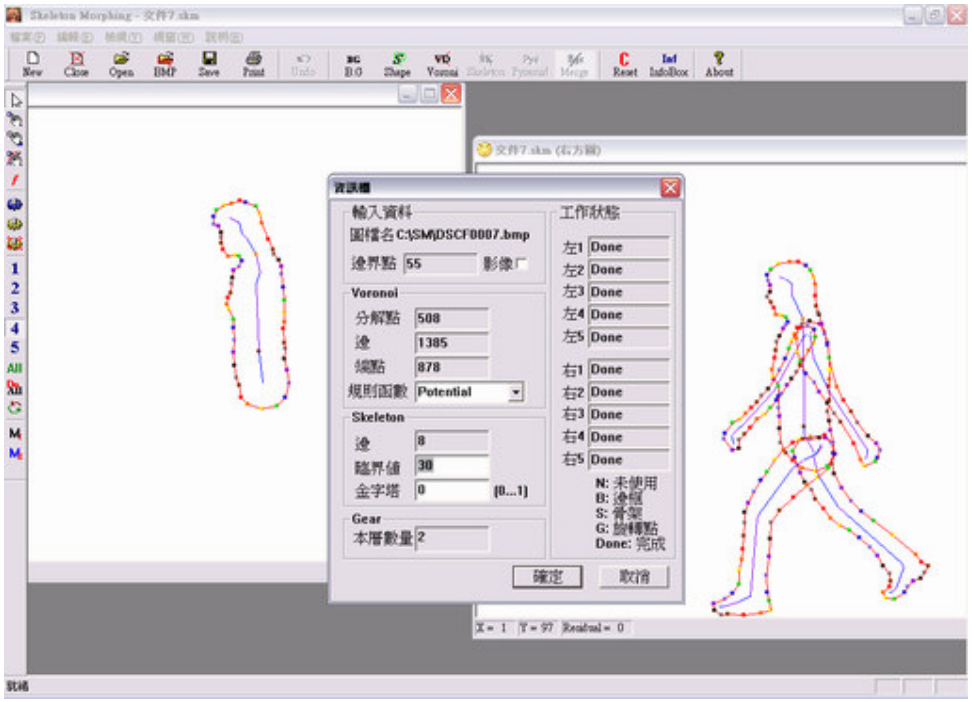


圖 A.6 進行工具列上的 Skeleton 及 Pyramid 功能。取得一組優良的骨架圖，並可以透過臨界值的設定自由調整需求。

Step9：Install Shape Gears

爲了確定物體的移動方式，利用左方工具列的[主軸]及[活動軸]功能，爲所有 Layer 設定關節或可運動的部位(主軸：此層部位移動的支點。活動軸：非支點的運動部位。)；點選時儘量靠近骨架以免誤差太大。每一層至少要有一個主軸的設定(圖 A.7)。

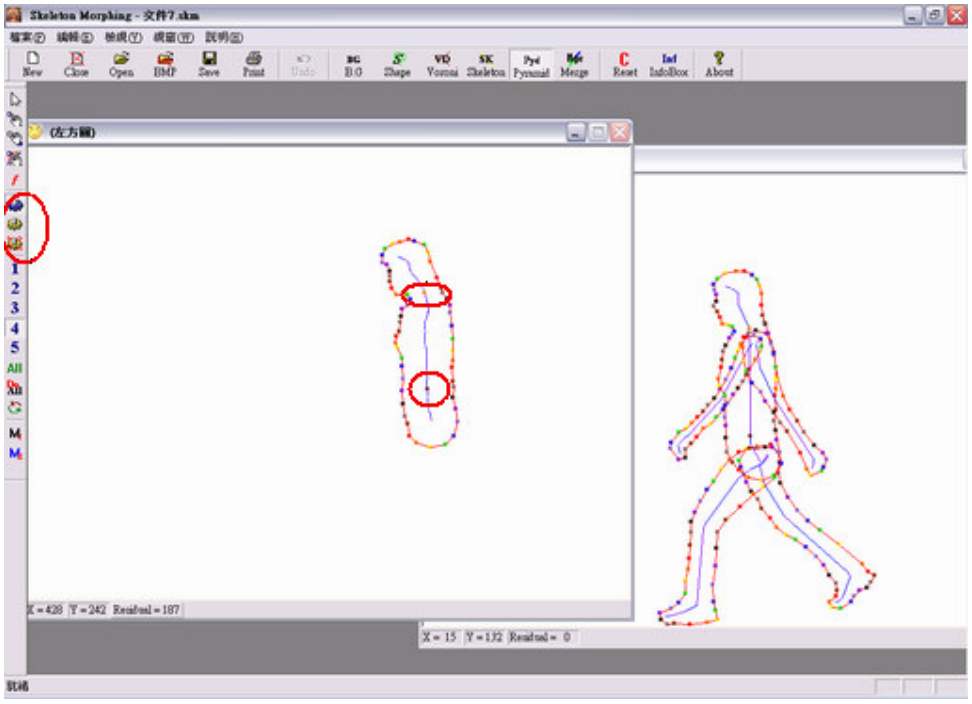


圖 A.7 設定運動軸。另一張圖則透過移動的方式將同步出現的運動軸移到對應的位置上。

Step10：Merge Skeletons

合併以簡化骨架資料。在合併的過程中，系統會自動將運動軸設定至骨架線段對應的端點上，同時進行邊界點的歸納(圖 A.8)。

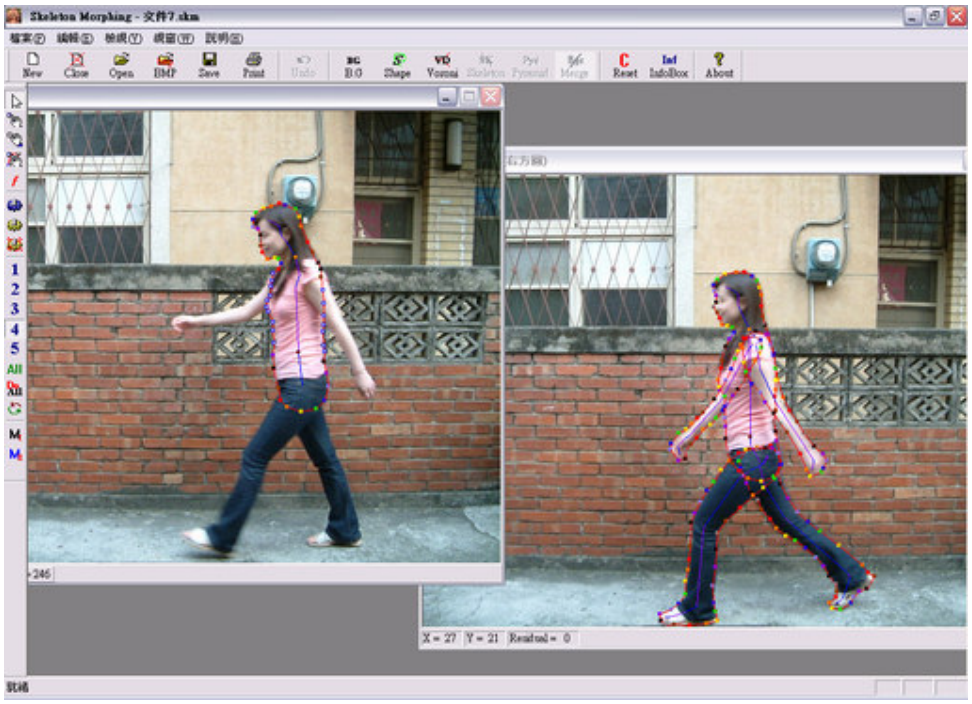
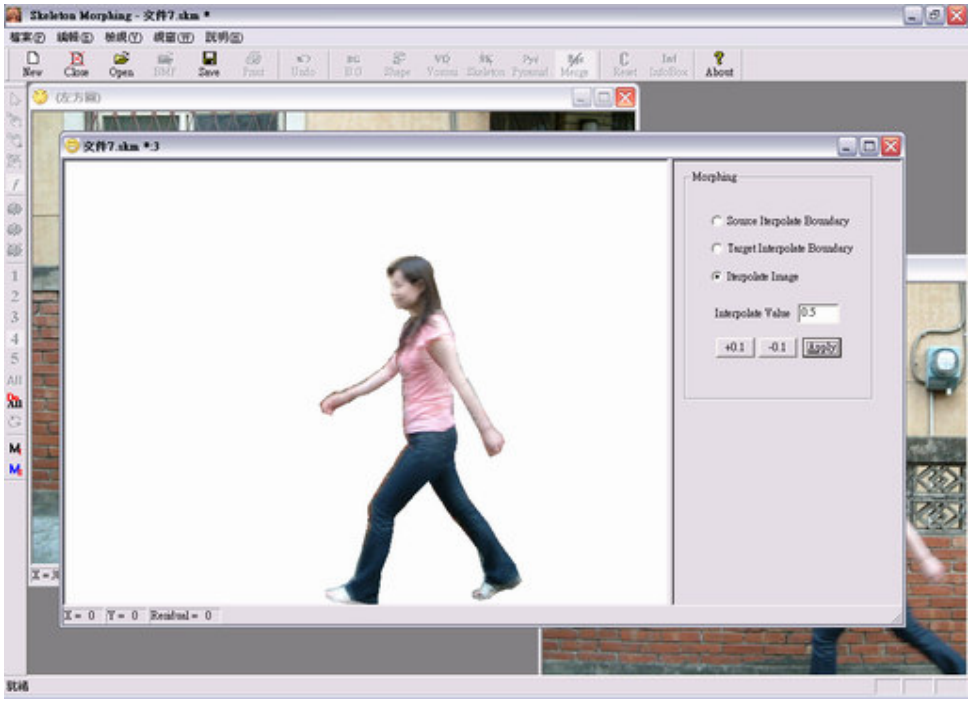


圖 A.8 合併後骨架不再為瑣碎的小線段組成，並且邊界點已經歸納至對應的骨架上(左圖之反白邊界點即為中間骨架所屬之邊界點)。

Step11：Image Morphing

取得充份的資訊後即可進行 Morphing 的動作，將左右圖的骨架分別進行內插計算，移動至對應位置，再計算邊界點之相對的位置，取得結果(圖 A.9)。



- 圖 A.9 Image Morphing。右方表單選擇顯示形變後的左圖或右圖，並且可以設定內插值。

Reference

[1] R.L. Ogniewicz, Discrete Voronoi Skeletons, Hartung-Gorre Verlag, Konstanz, Germany, (1993), Revised and extended version of Ph.D. thesis No. 9876, ETH-Zurich, Switzerland.

[2] R. L. Ogniewicz, and O.Kubler, Hierarchic Voronoi Skeletons, Pattern Recognition, Vol. 28, no. 3, pp. 343-359, 1995.

[3] H. Blum, A transformation for extracting new descriptors of shape, in W. Wathen-Dunn, editor, Models for the Perception of Speech and Visual Form, MIT Press, Cambridge MA, (1967)

[4] P. Felkel and S. Obdrzalek. Straight skeleton computation. In Lazlo Szirmay-Kalos, editor, Spring Conference on Computer Graphics, pages 210-218, Budmerice, Slovakia, April 23-25 1998.

[5] P. Felkel and S. Obdrzalek. Straight skeleton Implementation, 1998.

[6] Sitez S. M., Image-based Transformation of Viewpoint and Scene Appearance, A Dissertation Submitted in Partial Fulfillment of The Requirements For The Degree of Doctor of Philosophy, University of Wisconsin – Madison, 1997.

[7] Seitz S. M. and Dyer C. R., View Morphing, Proc. SIGGRAPH 1996, 21-30

[8] T. Beier and S. Neely. Feature-based Image Metamorphosis. SIGGRAPH'92 Proceedings, pp. 33-42, 1992

[9] G. Wolberg. Image Morphing Survey. The Visual Computer, 14, 8/9, 1998

[10] S. J. Fortune, A sweepline algorithm for Voronoi diagrams, Algorithmica, 2:153{174, 1987.

[11] C. Roman, Construction of Voronoi diagrams using Fortune's method : A look on an Implementation, http://www.cg.tuwien.ac.at/studentwork/CESCG/CESCG99/RCuk/ , 1999

[12] Alan Watt, Mark Watt. Advanced Animation and Rendering Techniques, ACM Press, 1992

[13] Alan Watt, 3D Computer Graphics 3rd Edition, Addison-Wesley, 2000

[14] Chen S. E. (1995). QuickTime VR-An Image based approach to virtual environment navigation. Proc. SIGGRAPH 1995, 29-38

[15] Williams L. and Chen S. E., View Interpolation for Image Synthesis, Proc. SIGGRAPH 1993, 279-88

[16] R.C.T. Lee, Introduction to the Design and Analysis of Algorithms, second edition, FLAGE publishing company.

[17] Paul S. Heckbert, Pixar, Survey of Texture Mapping, IEEE Computer Graphics

and Application, 1986, 56-67
[18] Eugene Kain, The MFC Answer Book, Addison-Wesley, 1998
[19] Ivor Horton's, Visual C++ 6, Wrox, 1998
[20] 侯捷、黃俊堯譯，泛型程式設計與 STL，碁峯，2000
[21] L. Paul Chew. Constrained Delaunay Triangulations. Algorithmica 4(1):97–108, 1989.
[22] Graig Gotsman, Vitaly Surazhsky. Shape Blending Guaranteed intersection-free polygon morphing, Computer & Graphics PERGAMON, 25(2001)67-75
[23] Alexa M, Cohen-Or D, Levin. As-rigid-as-Possible Shape Interpolation, Proceedings of the SIGGRAPH'2000, New Orleans, LA USA.
[24] Tal A, Elber G. Image Morphing with feature preserving texture. Computer Graphics Forum(Eurographics'99 proceedings) 1999;339-48.
[25] Mark De Berg (Editor), et al, Computational Geometry: Algorithms and Applications, Springer, (Hardcover - May 2000)
[26] Paul S. Heckbert, Fundamentals of Texture Mapping and Image Warping, thesis, Dept. of Electrical Engineering and Computer Sicence, University of California, Berkley, CA94720
[27] Mark A. Ruzon Carlo Tomasi. Edge, Junction, and Corner Detection Using Color Distributions, IEEE Transactions on pattern analysis and machine intelligence, vol. 23, no. 11, 2001.
[28] I. Pitas, Digital Image Processing Algorithms and Application,Wiley-Interscience
[29] J. R. Parker, Algorithms for Image Processing and Computer Vision, Wiley Computer.
[30] S. Y. Lee, K. Y. Chwa, S. Y. Shin, and G. Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. SIGGRAPH'95, pp. 439-448, 1995
[31] Alan Watt, Fabio Policarpo, "The Computer Image", Addison-Wesley

Curriculum Vitae

余岐智，民國六十八年三月於台北縣出生(台灣)。

1984-1989　台北縣秀朗國民小學
1990-1993　台北縣永和國民中學
1994-1998　私立華夏工商專校電子科
1999-2000　國立雲林科技大學資管系
2001-2003　國立暨南國際大學資工所