TR--88--036

變形物體運動模糊影像之產生

書參考
借外不

院究研央中
所究研學科訊
78.7.25
室書圖

中研院資訊所圖書室
3 0330 03 000109 8

0109

指導教授：鄭 國 揚 教授

變形物體運動模糊影像之產生

研究生：陳 茂 盛 撰

誌　謝

感謝指導教授鄭國揚博士在論文撰寫期間，一一指出所犯的錯誤，使得論文能順利完成。口試時，吳憲明教授和楊熙年教授提供的寶貴意見，使我受益匪淺。

同學王興華、史國賢、林俊仁、許丕中、舒念先等，在課業上相互砥礪，增廣見聞。研究所就讀期間，家人的鼓勵與支持，使我專心於學業，無後顧之憂。

在論文撰寫期間，中央研究院資訊科學研究所提供的優良的研究環境，使得論文順利進行。

最後，謹將本論文獻給我敬愛的父母。

目　　錄

頁 次

第一章　緒論

1.1 前言

在科技文明的今天，電腦的重要性已在人類生活中受到越來越多的肯定與重視。早期的電腦，多用於解決需要繁複計算的問題，而如今，隨著硬體技術的日新月異，以及更多人對軟體設計工作的投入，電腦對我們的影響，真可謂已到了鮑威爾在「一九八四」中所說的：「它無所不在，只不過不容易察覺出來。」

由於電腦在資料處理的方便，迅速以及電腦繪圖的出現，早在60年代就有人 [RONA 69]提出以電腦來製作動畫的構想，然由於受到技術與資金的限制，儘管之後Talbot，Carr 等人亦提出了更周詳的系統與計畫，卻都仍屬紙上談兵，不見明顯的成效。

直到1970年代 raster 顯示幕 [JAME 82]的出現，提供了高解析度的視覺顯象，以及近年來高速計算機的推陳出新，以電腦來製作動畫已不再只是空想。時下以電腦輔助動畫 (computer-aided animation)，已是一種不可避免的趨勢了。

然而，「2-D 電腦動畫」的技術儘管已日益成熟，在

真實感與美感之效果的表達上仍有其難以突破的瓶頸。因此，近來年已漸趨向於將電腦藝術 (computer art) 和動畫相結合，以產生更高品質的視覺享受。

本文目的並非在快速的得到一影像或影片，而是希望對物體運動時，因人的視覺暫留而生成之模糊 (blur) 影像以光線追跡法 (ray-tracing)作一實驗，並結合其它同是鄭國揚老師的學生之研究，以期做好「3-D 電腦動畫」的基礎工作。

1.2 研究動機

在電腦繪圖中，關於模糊 (blur) 效果的研究，多用於模擬物體 (object) 運動時的效果，即所謂的 motion blur；當然若將此一效果加之於動畫影片中，往往能使得整個影片的效果更真實且自然。

在常見的卡通動畫或廣告動畫中，物體在空間中來回穿梭，形狀多是固定的，但有時若在運動過程中兼具形狀的變化，效果可能會較生動。例如， NEC公司在其電視廣告中，若是只將 NEC三個字由遠而近，或由上而下的運動，其所給消費者的印象，應不及將 'N'這個字在運動過程中逐漸變形成為 'E'，最後再變成 'C'的作法來得生動而

有趣。雖然這種變形效果的研究，早就見於主圖像 (Key Frane Method) 方法中，但卻未見於將其運用在模糊效果的產生。因此，筆者希望結合傳統的主圖像方法和光線追跡法以對這種變形物體於運動時所產生的模糊效果作一實驗。

上述運動，若是由傳統的畫家來表達，勢必浪費許多金錢與時間，因為他們只能以紙和筆以 2-D形式來表達 3-D物體於運動時所需之連貫性 (coherence)。所以若是藉助於電腦的快速計算能力及一些數學模式勢必能使動畫拍攝成本降低，而原有的畫家也可去從事更高繪圖技巧的研究，相信只要將來技術成熟，這種作法當能使得動畫的品質更趨於完善。

1.3 擬採用之方法

基本上，本文的重點有兩項，第一是物體的變形，第二是運動模糊影像的產生。關於物體變形的處理，由於問題頗為類似，擬以 2-D物體動畫常用的主圖像法做基本架構，再配合一些電腦繪圖常用的數學模式，以求得物體的變化形狀。

至於運動模糊影像方面，則以光線追跡法 (Ray-Trac

ing)為出發點，此乃因光線追跡法是現今所有繪圖技巧中，所得效果最好的一種；儘管此法所需時間甚為可觀，但這問題可望於幾年後，以性能更佳硬體 (hardware) 設備來解決，所以這方面技巧仍是電腦繪圖常探討的重點之一。除此之外，光線追跡法於產生模糊影像上所使用的方法頗為簡潔，而原理也十分合乎自然現像，所以筆者決定結合主圖像法和光線追跡法做為本文之理論基礎，以從本實驗中觀察物體於運動過程中兼具變形的效果時，其所表現出模糊狀態。

1.4 回顧

據筆者所，現今以光線追跡法產生運動模糊 (motion blur)的研究中，效果上要以Cook於84年所提出之分佈式光線追跡法 (Distributed Ray Tracing)最為大眾所接受，因為它較前人 [Potmesil 83]所提之方法更簡潔且合乎自然現象，當然效果也更好。其實，分佈式光線追跡法的提出，更一舉的解決了模糊反射 (glass, or blurred reflection) 及景深 (depth of field) 等殊效果摹擬的問題，除了改良以往的作法 [Potmesil 82]， [Cook 81] [Nishita 82] 之外，更為傳統的光線追跡法賦與新的研

究方向，可謂一項極大的突破。而後 Robert[Robert 86] 更將多種統計分佈 (statistic distribution) 應用於光線追跡法，並作詳細的分析和驗證，予以往後的研究者確立了適當的分佈模式 (distribution model) 。

1.5 實驗範圍

在以往的分佈式光線追跡法中，於摹擬物體的運動模糊影像時，皆以固定的物體為限，因此只須根據使用者所定義的運動軌跡 (motion path)計算出該物體於某特定時刻 (time) 的所在位置與旋轉角度即可。但在本實驗中，物體的形狀是可變的，所以尚須根據使用者定義的第一個和最後一個物體形狀，以一些數學模式(mathematic model) 找出其間的形狀，如此方可達到所欲的目標。由於只是一項實驗，所以筆者先對 '掃掠物體' (objects definded by sweeping contour)[WIJK 84]作處理，若能順利完成，再對較複雜之物體做進一步研究。

1.6 章節簡介

接下來第二章將先介紹本實驗所使用到的一些數學模式。第三章則詳述本文的作法步驟。第四章則對本實驗所

使用的軟體 'Raytrace' 之使用步驟及程式架構做概略性的說明。第五章則以一些實驗結果作主題。最後於第六章做本文之總結。

第二章　理論基礎

2.1 光線追跡法之簡介

光線追跡法乃是由 Appel [Appel 68] 鑑於人的視覺是由光線反射至人眼引起而提出的一種反溯 (trace in opposite direction)光線 (ray)的電腦繪圖技巧(圖2.1)

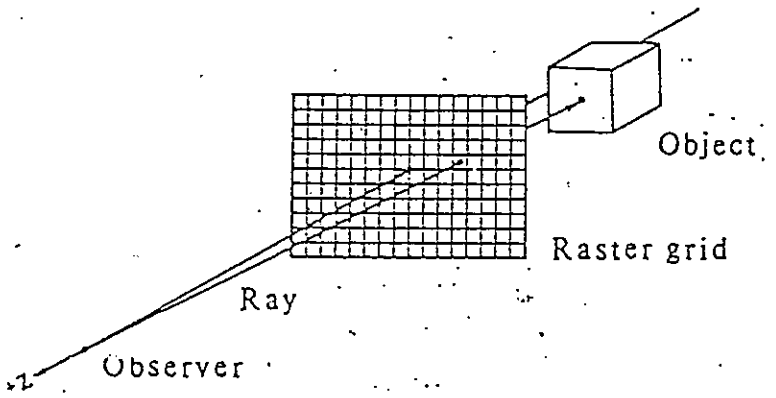


圖2.1 光線追跡法的原理

關於光線追跡法的作法大致如下：

1.產生景觀 (create scene) ：由使用者輸入景觀中各物體資料。

2.產生光線 (create ray) ：以眼睛為光線之起點，並射向螢幕 (screen) 中的每一像素點 (pixels) 成一光線。

3.求交點集合(intersection)：以數學方法求出每一條光線與景觀中的每一物體的交點及其它相關資料；如此便得到一交點集合。

4.計算採光強度：在相交集合中取出離眼睛最近的物體，若此物體的材質為：

(a) 不透明體：依採光模式 (illumiration model) 計算Id和Ia，而Is=0，It=0。

(b) 可反射體：依採光模式計算Id，Ia。

依光學原理，產生一反射光線，重覆步驟 3-4，可得Is；It=0。

(c) 透明體：依採光模計算Id，Ia。

依光學原理，產生一反射光線，重覆步驟 3-4，可得Is。

依光學原理，產生一折射光線，重覆步驟 3-4，可得It。

最後，I=Ia+Id+Is+It 為此光線 (或此像素點) 的採光值，一般稱為此像素點的 rgb (red，green，blue) 值。

5.對每一光線重覆步驟 2-4，以求得螢幕中每一像素點的 rgb 值，如此便可得到最終圖像。

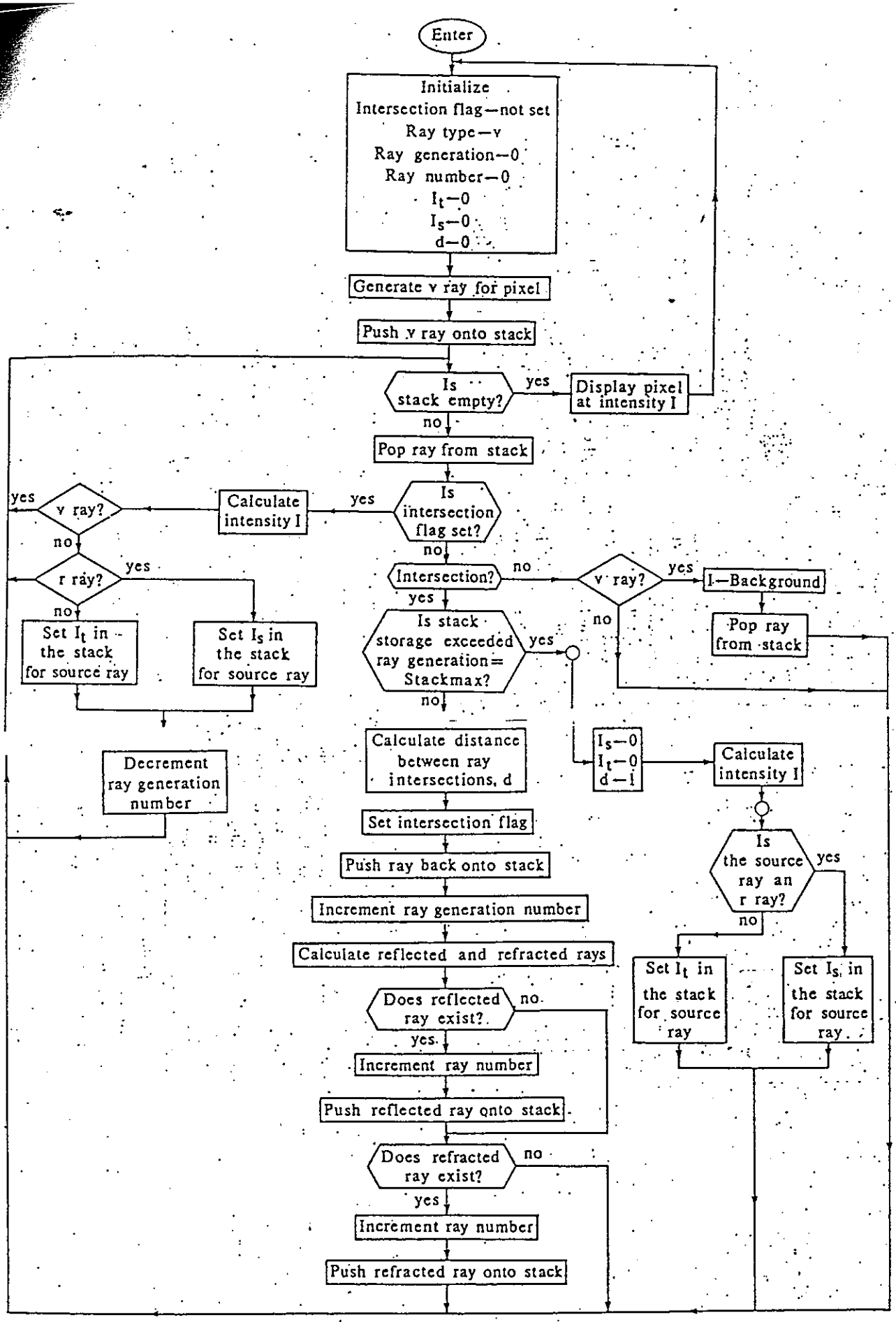


圖 2.2 光線追跡法的詳細流程

圖 2.2和 2.3為光線追跡法以 Pushdown ray stack 的方式來完成的詳細流程。

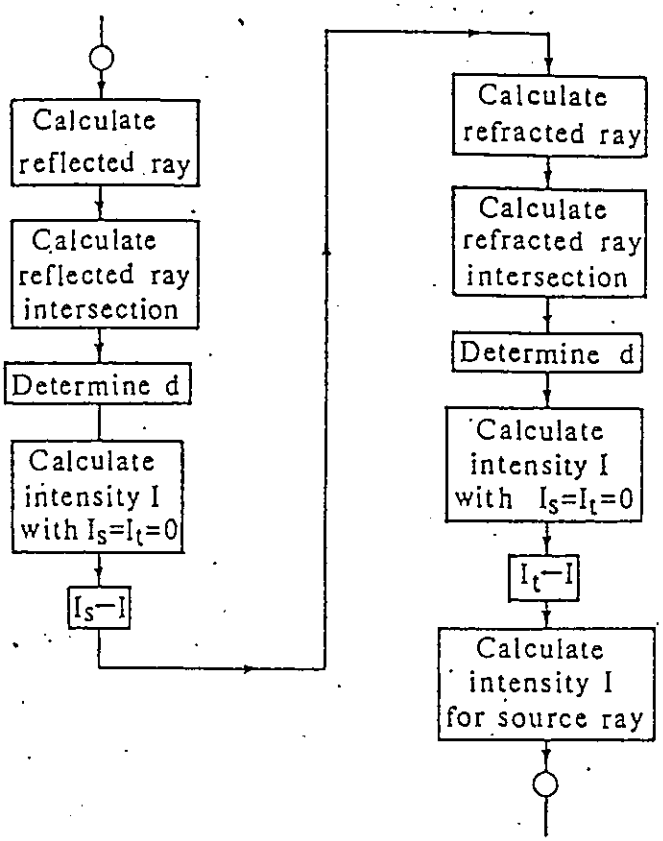


圖 2.3 光線追跡法的部分流程 (續上)

圖 2.4則是光線追跡法計算採光強度的模式

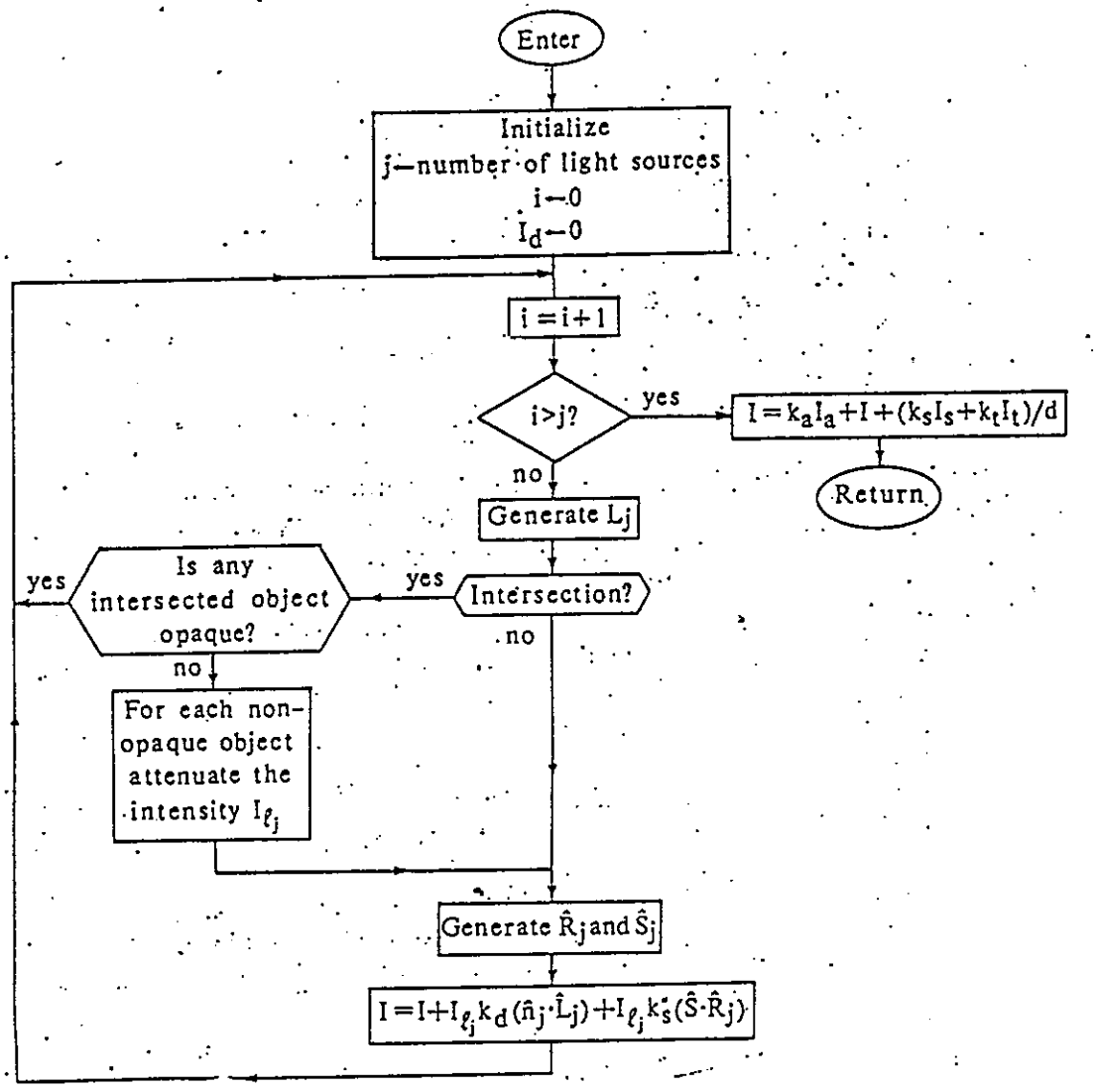


圖 2.4光線追跡法之採光模式(illumination model)

從上述的介紹，可知光線追跡法之所以效果會其它繪圖技巧來得好，最主要是因為其能做反射 (Is) 和折射效果 (It) 的摹擬，而最常見之 '掃描線法' (Scanline

technique)因不具光線的特性，所以無法求得反射光和折射光的行徑，當然也就没能做出真實狀況之反射和折射的效果了。但也因光線追跡法為了摹擬真實現象中反射光和折射光的行徑，致使過程中多衍生出許多條光線 (圖2.5)，當然也就使得 cpu為了計算某光線之如樹狀結構 (tree structure)般的 rgb值 (圖2.6)多付出了許多時間，所以這項矛盾 (tradeoff) 便成了使用者須視其硬體設備而斟酌之最大考慮事項。

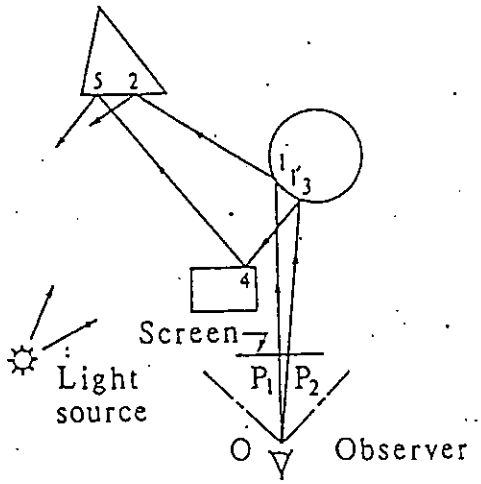


圖 2.5 反射光與折射光之衍生

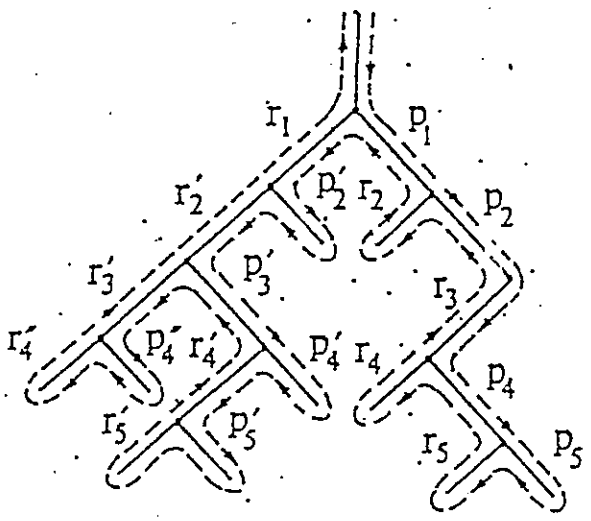


圖 2.6 Ray-Tracing tree

2.1.1 光線追跡的缺點

由於光線追跡法的原理十分簡潔，而且所得效果十分良好，所以至今仍是電腦繪圖中常被採用的技巧之一。然而這種方法的最大缺點則包括 (1)速度太慢 (time cost)，(2)別名 (aliasing) 的產生。

(1) 速度太慢

如眾所皆知，光線追跡法的核心在於 '求光線和物體的交點'，儘管已有許多關於多種形狀物體與光線相交的數學求解法，然而一旦應用於光線追跡法中，卻因所需計算次數過多而導致此技巧所需的時間十分可觀。根據 Whitted [Whitted 80] 統計，在整個光線追跡法過程中，求

光線與物體的交點這個步驟往往會耗掉整個光線追跡法 95% 的時間；所以針對這項問題以提高執行速度的研究，便成為現今有關論文的探討重點之一。

(2) 別名 (aliasing) 的產生

由於每一條光線只傳回相對應的像素點之 rgb值，所以若所使用的硬體之解析度不是很高，那麼所得到的影像便往往會於物體的邊緣處發生如鋸齒般的不自然狀況，這種現象我們通稱為別名 (aliasing)；因此如何降低別名 (antialiasing) 以提高所得圖像的真實感便也成了相當熱門的研究問題之一。

2.1.2 加快光線追跡法速度的方法

基本上關於增快光線追跡法之執行速度的方法大致可分為以下兩類：

(1) 提高 '求交點' 這個步驟的速度。

(2) 減少 '求交點' 的次數。

以上兩類方法各有其優點及適用範圍，而依使用者的所需而定。

(1) 提高 '求交點' 的速度。

(a) 硬體

由於光線追跡法基本上是一種迴路式 (Do-Loop)的程序，因而很適合將整個影像切割成多個子影像(subimages)，再交由多個 Processing Unit分別執行 (圖2.7)最後再將這些子影像整合在一起即可 [Nishimura 83] 。由圖2.8 可看出於 mulitprocessor 系統中，光線追跡法的 performance 提高了不少；當然所需資源也是很可觀的，如圖2.9。所以未來的多處理器 (mulitprocessors) 系統的技巧若發展成熟時，光線追跡法應更能為大家接受。

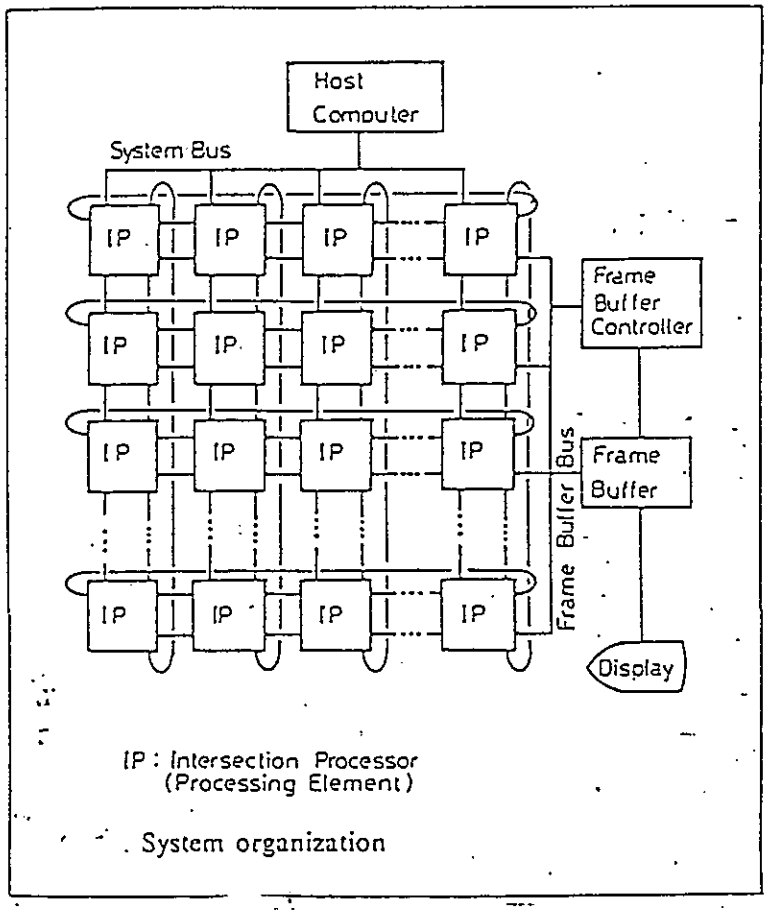


圖 2.7 多處理器系統架構

(b) 軟體

常見的作法就是對每一個物體造出其相對應的,'包圍空間' (bounding volume) ，然後每一條光線在求與物體的交點之前，先判斷其是否通過該物體的 bounding volume，唯有通過，才進一步作求交點的計算。為了方便計算，'包圍空間' 常採球形 (bounding sphere) (如圖 2.10)；或平行六面形 (bounding box)(如圖 2.11)。

至於當物體多聚集在一起時，則可將這些常可能重疊的 bounding volume聚成 (cluster) 一個大的 bounding volume，如此，光線便可先對這一大的 bounding volume作 '通過測試' ，若是，再對其內的小 bounding volume作測試 (如圖 2.12)，這樣作往往可減少許多計算時間。

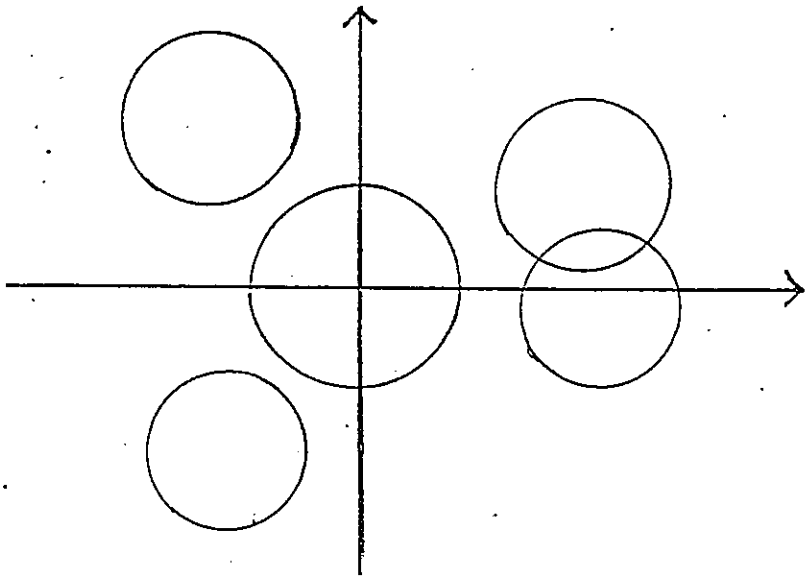


圖 2.10 bounding sphere

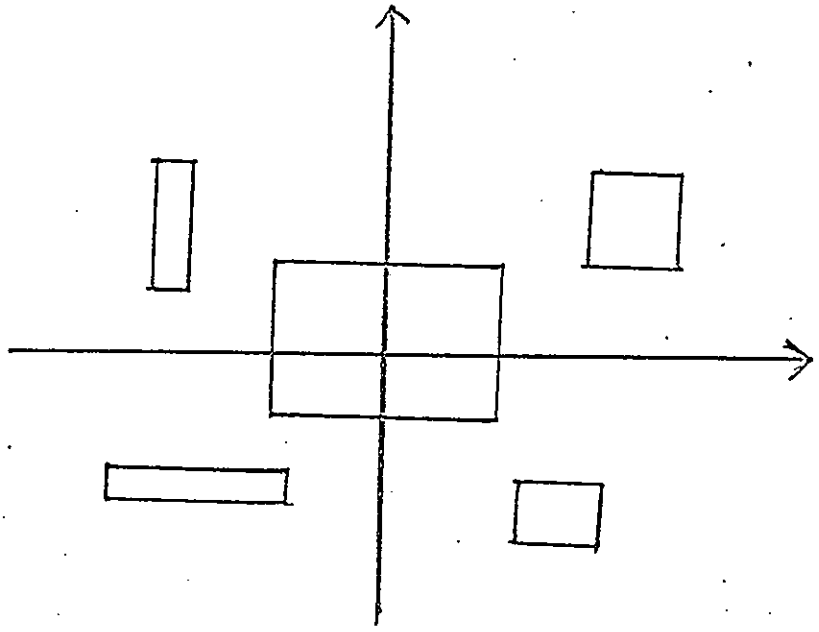


圖2.11 bounding box

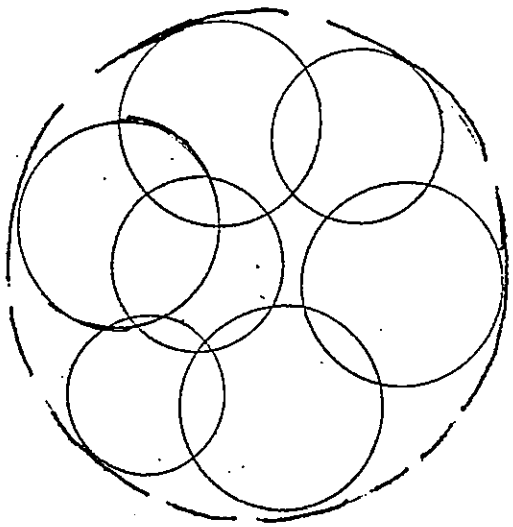


圖2.12 clusters bounding volume

(2) 減少 '求交點' 的次數

一般而言，常見的方法中，可分為以下兩類

(a) 連續性 (coherence)的利用：

這種方法乃利用兩個相鄰的像素點往往會具有相近之光線行徑 (圖2.13，圖2.14) 的特性，以節省所產生光線的數目，其作法是每隔 n(n≧2)個像素點才求一次光線與物體的交點；至於其間像素點的 rgb值則是用內插法求得。當然這種方法能節省不少時間，但多只適用於 '單調' 的景觀，若是顏色變化複雜的景觀(complex scence)，往往會弄巧成拙，所以須小心使用方可 [Badt 88]。

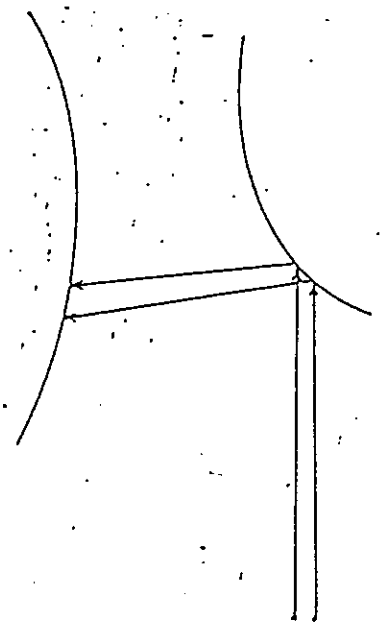


圖2.13 光線行徑之連續性

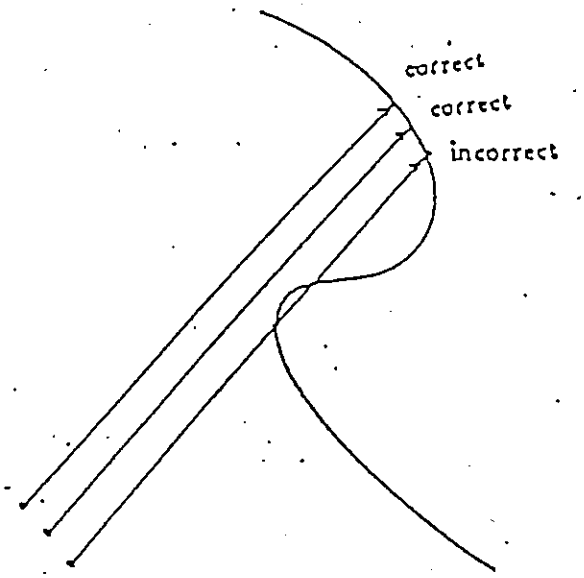


圖2.14 光線行徑之連續性(二)

(b) 子空間的切割 (space subdivion)

將原有空間切割成多個子空間 (如圖2.15，2.16)，並記錄各子空間的物體資料，如此只須先對離光線最近的子空間作 '求交點' 的計算，若有交點，便可忽略其他的子空間，否則再對下一個子空間作交點的測試。所以往往可達到增快速度的目的 [Glassner 84]；此法尤其適用於景觀複雜的情況中，如空間中有上百個球時，此法確能收到立竿見影之效果。

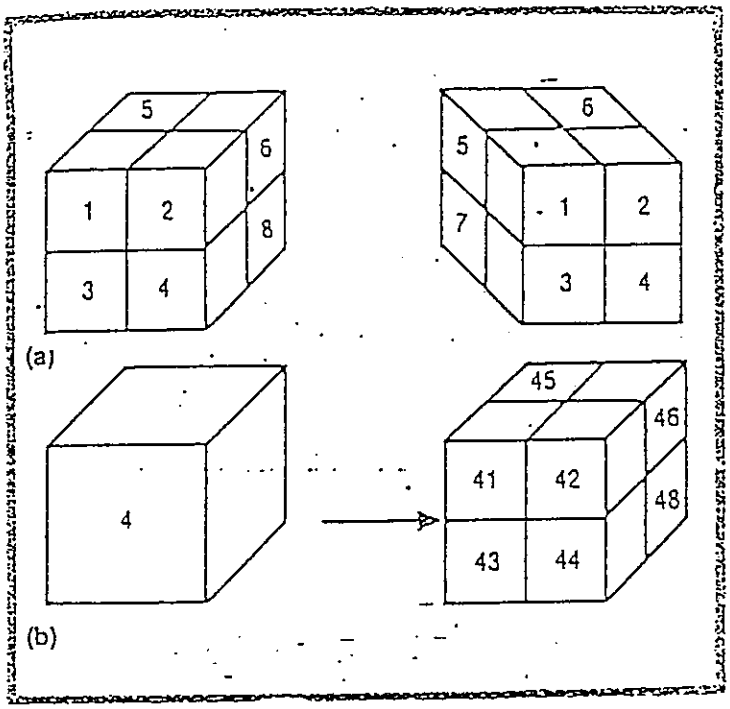


圖2.15　空間的切割

```
                LISTING 1


findnode(x,y,z) {
    node = 1;
    WHILE ( node_subdivided  is TRUE ) {
        IF (x > node_center_x)
            IF (y > node_center_y)
                IF (z > node_center_z)
                      node = (node*10) + 6;
                ELSE  node = (node*10) + 2;
            ELSE
                IF (z > node_center_z)
                      node = (node*10) + 8;
                ELSE  node = (node*10) + 4;
        ELSE
            IF (y > node_center_y)
                IF (z > node_center_z)
                      node = (node*10) + 5;
                ELSE  node = (node*10) + 1;
            ELSE
                IF (z > node_center_z)
                      node = (node*10) + 7;
                ELSE  node = (node*10) + 3;
    }
    RETURN ( node )
}
```

圖2.16　尋找某特定子空間之方法

2.1.3 降低別名的方法 (Antialiasing)

基本上，電腦繪圖可視為一取樣過程 (sampling process) ：而取樣的頻率便決定了電腦解析度的大小。當所欲繪製的景觀 (scene)的顏色變化頻率大於電腦的取樣頻率時，便會發生所謂的別名 (aliasing) 之現象。如圖2.17，雖然a和c是兩個不同頻率的訊號，但經過取樣後卻成了同一種訊號 (b和d) 。

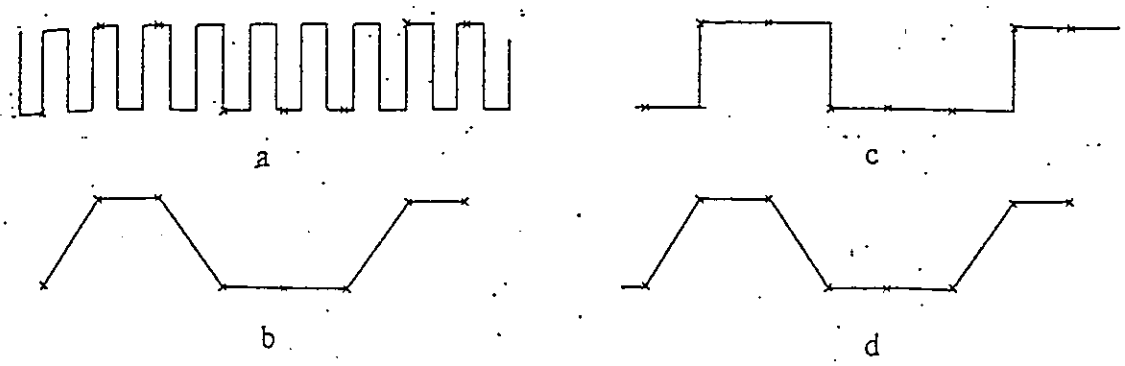


圖2.17 取樣和別名

因此便有人提出一些解決這種現象的技巧，通稱為 '反別名'(antialiasing)， 常見的方法有以下幾種：

(1) Filtering ：藉用 '傅立葉轉換' 算出各像素點所得之 rgb值的比例係數，由於過程頗為複雜，所以常耗去許多時間[Brigham 74]。

(2) Supersampling ：將各像素點切割成多個子像素點，

對每個子像素點取樣後，再結合成原來的像素點，如此便可虛擬式地提高硬體的解析度，但所需時間將會更多。

(3) Adaptive Sampling ：在物體的邊緣處 (edge) ，提高該處像素點的解析度 [Whitted 80] ；這種做法的速度當然會較前一種來得快，但Algorithm會較複雜。

(4) Stochastic Sampling ：利用統計學上各種分佈模式的特性來作取樣的依據，雖然速度最快，但往往所得效果較差。

2.2 曲線的簡介

2.2.1 曲線之應用

在電腦繪圖、動畫、影像處理及模式識別的研究上，常需藉助數學上畫曲線的方法。一般畫曲線的方法依其特性大致可分為：1.通過所有控制點 (或稱為參考點) 的內

插法 (interpolation method)。 2.逼近控制點的相近法 (approximation method) 。內插法最常見的就是拉哥藍基內插多項式 (Legrange interpolation Polynomial) [Burden 81] ，此方法雖然簡單，卻因振盪性 (oscillatory nature) 較大，即區域性的改變會對整段曲線有較大的影響。所以一般在畫曲線時多採用逼近法；在逼近法中，又以片段式多項式逼近法 (piecewise polynomial approtimation) [burden 81] 為較多為人所使用，其方法就是將一區間 (interval)畫分成多個子區間 (subinterval)，再依子區間的控制點及邊界條件 (boundary condition) 以高次多項式來逼近。

由於本文的探討範圍是掃掠物體 (sweeping object) 為主，所以需先藉助一些曲線來定義物體形狀，所以下兩個子節將介紹本實驗所用到的兩種畫曲線法：(1) Bezier曲線和(2) B-spline曲線；其目的在說明這些曲線的特性，以便使得使用者在定義系統所需之Sweeping物體時對曲線之特性有較深的認識，當然也才能較易得到使用者所想要之曲線。

2.2.2 Bezier曲線

Bezier曲線是由法國人Bezier [Bzei 74]所提出之畫面曲線法，其以控制點來推算曲線方程式的作法如下：

$$\vec{P}(u) = \sum \vec{P}_k B_{kn}(u), 0 \leq u \leq = 1.$$

$$\vec{B}_{kn}(u) = \frac{n!}{k!(n-k)!} u^k (1-u)^{n-k}.$$

其中$B_{kn}(u)$稱作 Bernstein Ploynomial!

根據上適公式，我們可推得 Bezier 曲線會具有下列性質

(1) $\sum B_{kn}(u) = 1, B_{kn}(u) \geq 0$

(2) $B_{0n}(0) = 1$ and $B_{nn}(1) = 1$

(3) $B^p_{kn}(0) = 0$ when $p < k$ and $B^k_{kn}(0) = \frac{k!}{(n-k)!}$.

$B^q_{kn}(1) = 0$ when $q < n-k$ and $B^{n-k}_{kn}(1) = (-1)^{n-k} \frac{n!}{k!}$

由於在一般繪圖應用上，三次的 Bezier 曲線便足夠合乎所需，因此接下來用三次的 Bezier 曲線來對Bezier曲線作進一步說明：

在空間中給定 4 點 (如圖2.18)，我們欲求出一三次多項式通過第一點及第 4 點，並於該兩點保持一次微分連續 (C′)：

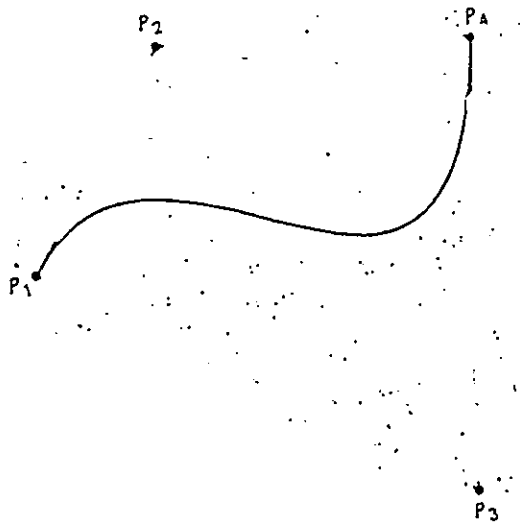


圖2.18 Bezier 曲線

由於上述曲線於起點和終點都具有一次微分連續，所以多個此種曲線組合而成的組合曲線 (composite curve) (如圖 2.19)亦具備一次微分連續性，因此 Bezier曲線常被用於定義一些使用者所欲之曲線 (如圖2.20)。

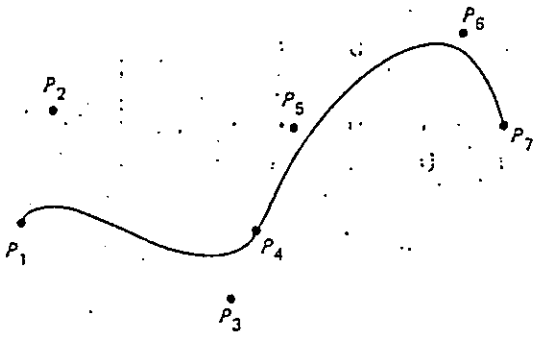


圖 2.19 組合Bezier曲線(compositive Bezier curve)

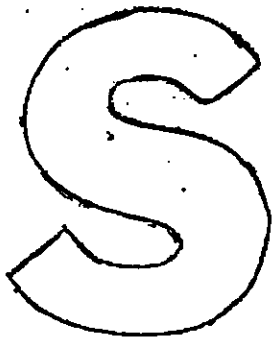


圖 2.20 以 Bezier曲線定義之圖像

至於 Bezier 曲線多項式的求法如下：

$$M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad T = [t^3\ t^2\ t\ 1] \quad 0 \le t \le 1$$

$$P_{\cdot} = \begin{bmatrix} p_{1\cdot} \\ p_{2\cdot} \\ p_{3\cdot} \\ p_{4\cdot} \end{bmatrix}$$

$$X(t) = T\,M_b\,P_x = [t^3\ t^2\ t\ 1] \begin{bmatrix} p_{1x} \\ p_{2x} \\ p_{3x} \\ p_{4x} \end{bmatrix}$$

$$= \sum_0^3 C_i^3 t^i (1-t)^{3-i} P_{ix}, \quad 0 \le t \le 1.$$

$$Y(t) = \sum_0^3 C_i^3 t^i (1-t)^{3-i} P_{iy}, \quad 0 \le t \le 1$$

$$Z(t) = \sum_0^3 C_i^3 t^i (1-t)^{3-i} P_{iz}, \quad 0 \le t \le 1$$

關於上述之Mb之值的求法，可參閱 [K.Y.Cheng]

2.2.3 B-spline

B-spline 為 spline 中於電腦繪圖的應用裡最常用的一種，其求逼近控制點的公式如下：

$$\vec{P}(u) = \sum_{0}^{n} \vec{P}_k \, N_{km}(u) .$$

where N_{km} have the following properties :

- polynomial of degree m—1 (indenpent of n)
- Nonzero only part of total range of u . Thus provides more local control over curve design than Bezier.

其中混合函數 (Blending Functions) Nkm(u)為：

$$N_{k1}(u) = \begin{cases} 1 \text{ if } u_k \le u \le u_{k+1} \\ 0 \text{ otherwise .} \end{cases}$$

$$N_{km}(u) = \frac{u - u_k}{u_{k+m-i} - u_k} N_{k,m-1}(u) + \frac{u_{k+m} - u}{u_{k+m} - u_{k+1}} N_{k+1,m-1}(u)$$

理論上 B-spline 可提供任意次方的多項式，並可達

到任意高次連續性的要求；但本實驗只需得一次或兩次連續性的曲線即可，所以只需採用三次 B-spline ，否則較高次的多項式將會成為電腦計算的沉重負荷，而且較高的振盪性也會令使用者感到不便。所以我們仍以三次 B-spline 做進一步說明：

在平面上，使用者給定 n 個控制點座標 (如圖2.21)，我們欲求出逼近這些控制點的多項式

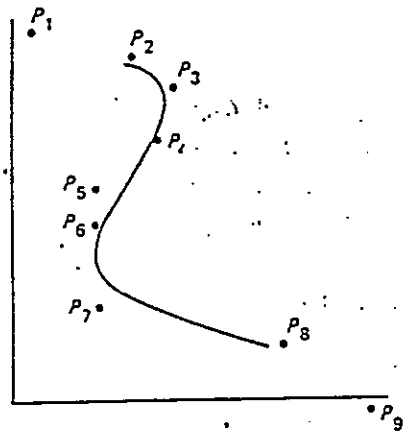


圖2.21 B-spline

由於我們只需得到三次多項式即可，所以接下來將每相鄰 4 點的座標代入計算公式 (如 $P_1P_2P_3P_4, P_2P_3P_4P_5, \cdots\cdots$ $\cdots\cdots, P_{n-3}P_{n-2}P_{n-1}P_n$) 即可 (圖2.22) ，同理若要得到 k (K≧3)次多項式，也只要將每相鄰的 K+1點座標代入相類似公式 (公式內之矩陣須重新導過) 即可。

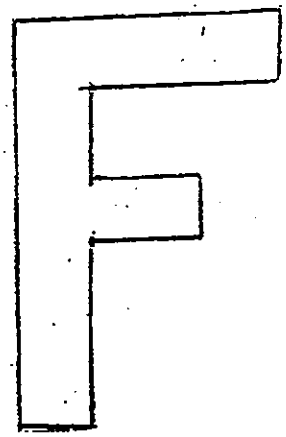


圖 2.22 三次 B-spline

關於 B-spline 的計算公式如下：

三次 B-spline 通常以 t 為其參數，且 t 的範圍為 $0 \leq t \leq 1$，其一般式則為

$$X(t) = T\ Ms\ Gsx$$

$$Y(t) = T\ Ms\ Gsy$$

其中

$$T = [t^3\ t^2\ t\ 1]$$

$$Ms = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

$$G_{sx}^{i} = \begin{bmatrix} x_{i-1} \\ x_i \\ x_{i+1} \\ x_{i+2} \end{bmatrix}$$

$$G_{sy}^{i} = \begin{bmatrix} y_{i-1} \\ y_i \\ y_{i+1} \\ y_{i+2} \end{bmatrix}$$

$2 \leqq i \leqq n-2$

$(X_1,Y_1),(X_2,Y_2),\ldots\ldots\ldots\ldots (X_n,Y_n)$ 為n個控制點座標。

至於為什麼每段曲線之間的交接點會有兩次微分連續性，其原因為下：

我們可以以 G_s^i ，G_s^{i+1} 兩種種矩陣式求出第 P_{i+1} 點的第一及第二微分：

先以 t =1代入 G_{sx}^{i}

$$X^i(1) = \frac{x_i + 4x_{+1} + x_{i+2}}{6}$$

$$\left.\frac{d\,x^i}{d\,t}\right|_{t=1} = \frac{-x_i + x_{i+2}}{2}$$

$$\left.\frac{d^2x^i}{d^2t}\right|_{t=1} = x_i - 2x_{i+1} + x_{i+2}$$

再以 t =0代入 G_{sx}^{i+1}

$$X^{i+1}(0) = \frac{x_i + 4x_{i+1} + x_{i\ 2}}{6}$$

$$\left.\frac{d\ x^{i+1}}{d\ t}\right|_{t=0} = \frac{-x_i + x_{i+2}}{2}$$

$$\left.\frac{d\ x^{i+1}}{d\ t}\right|_{t=0} = \frac{-x_i + x_{i+2}}{2}$$

由上即可驗證，在 x -分量上，這兩種曲線的交接點的切向量和曲率的函數都是連續的；關於 y -分量上，甚至 z -分量上的推導都和上述過程相同，不再詳述。

至於 B-spline 的特性，大致有以下幾點：

1.曲線通常都不會通過控制點。

2.所求得之 n 次多項式必具備(n-1)次的連續性。

3.欲通過某控制點，則需連續性出現 3 次以上方可 (此指三次 B-spline)。

4.n 個控制點，可得到(n-3)段曲段 (指三次多項式)。

5.若改變某個控制點座標，只會影響鄰近的幾段曲線，所以適合作局部曲線修改。

6.具備凸形殼曲線，即曲線行徑不會超過由 4 個控制點所連成的四邊形區域 (如圖2.23) ；故使用者不難想

像其行徑；此一性質 Bezier 曲線亦具備，所以此兩種方法皆適合給使用者作定義曲線之用。

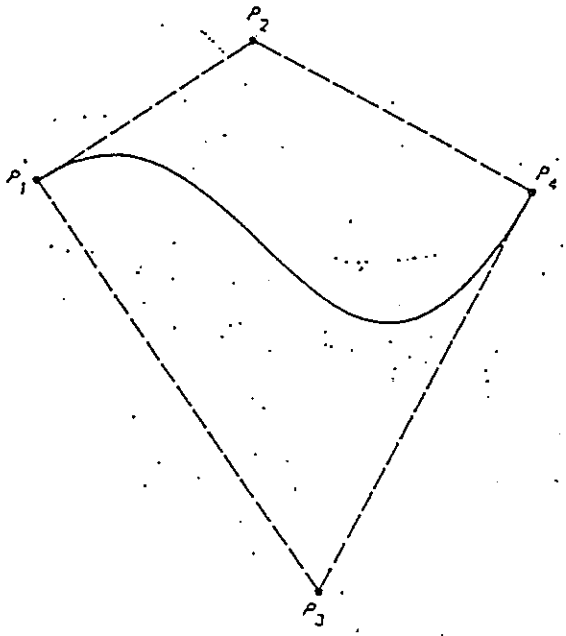


圖 2.23 凸形殼

2.3 掃掠物體 (Qbject Obtained by Sweeping)

本實驗所討論的對象，乃是由掃掠 (Sweeping) 2D曲線面得到的3D物體為主，關於掃掠 (sweeping) 的方式則有以下三種：

A.平移掃掠 (Translation Sweeping) ：將一封閉曲線沿一直線掃掠，所經過之處，皆即為所定義的物體，如圖2.24。

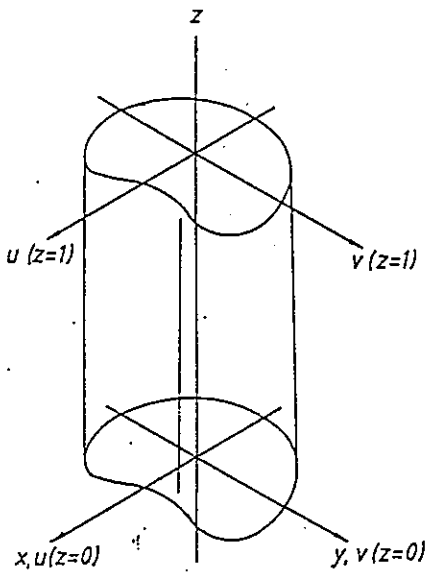


圖 2.24 平移掃掠

B.卡尼克掃掠 (Conic Sweeping) ：在平移掃指的過程中，並對該封閉曲線作等比例的放大或縮小，如圖2.25。

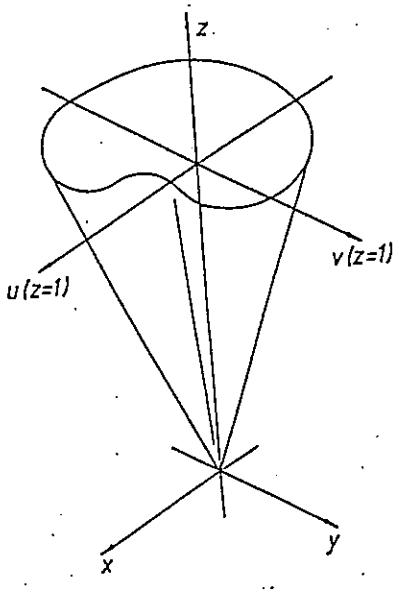


圖 2.25 卡尼克掃掠

C.旋轉掃掠 (Rofational Sweeping)：將一封閉曲線沿一

直線軸作旋轉，如圖2.26。

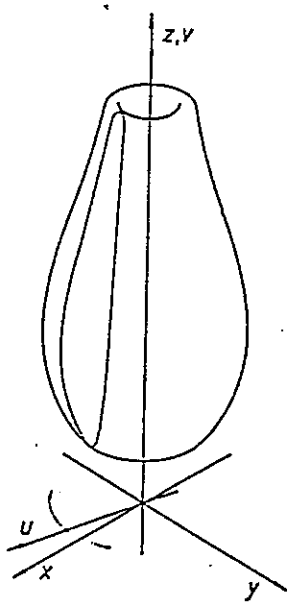


圖 2.26 旋轉掃掠

至於有關使用者在定義 '掃掠物體' 時，所需給定的資料如下：

1.物體所在之座標位置。

2.封閉曲線的控制點（本人在PC上有設計一介面程式，可讓使用者做交談式的修改）。

3.掃掠方式（平移、卡尼克or旋轉）及長度。

2.4 赫米特 (Hermite)曲線及曲面

在求取運動過動中物體的變形狀況，我們將需藉助赫米特曲面(Hermite Patch)，關於其原理則分述於下兩節：

2.4.1 赫米特曲線 (Hermite curve):

在空間中給定兩點，及點上運動軌跡的切向量，欲求出通過此兩點的三次曲線，且曲線上此兩點的切向量需和給定之值相同，如圖 2.27 所示。

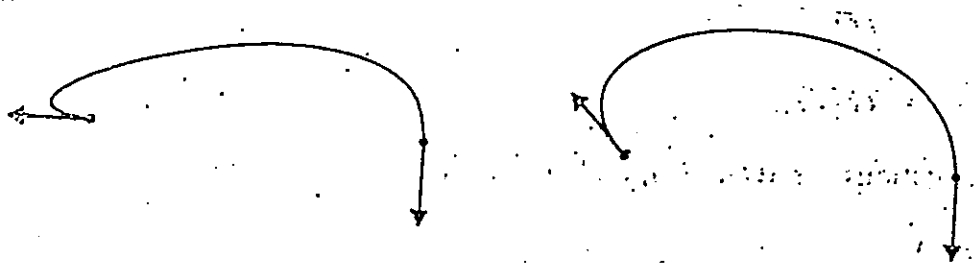


圖2.27 Hermite曲線

同 Bezier 曲線一樣， Hermite曲線亦具備一次微分連續的特性，所以多條此種曲線所連接而成的組合曲線亦具備一次微分連續，亦即整個曲線是相當平滑的(smooth)如圖 2.28 所示。

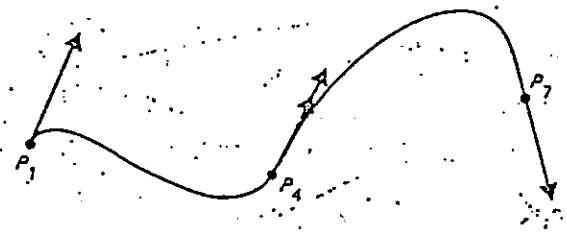


圖 2.28 組合 Hermite曲線

欲求得 Hetmite曲線的多項式，其公式如下：

已知點 P_1 ，P_2 及 R_1，R_2 （ P_1，P_2 的切向量）

$$T = [\,t^3\ t^2\ t\ 1\,]$$

$$Mh = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$X(t) = T\,M_h\,P_x$$
$$= P_{1x}(2t^3-3t^2+1)+P_{2x}(-2t^3+3t^2)+R_{1x}(t^3-2t^2+t)+R_{2x}(t^3-t^2)$$

$$Y(t) = T\,M_h\,P_y$$
$$= P_{1y}(2t^3-3t^2+1)+P_{2y}(-2t^3+3t^2)+R_{1y}(t^3-2t^2+t)+R_{2y}(t^3-t^2)$$

$$Z(t) = T\,M_h\,P_z$$
$$= P_{1z}(2t^3-3t^2+1)+P_{2z}(-2t^3+3t^2)+R_{1z}(t^3-2t^2+t)+R_{2z}(t^3-t^2)$$

欲知其原理及證明，可參閱 [Foley 82]

2.4.2 Hetmite 曲面 (patch)

上述之 Hermite曲線可推廣至成兩個參數之三度空間上的曲面，即將兩個 Hetmite曲線之間的運動軌跡用另一參數 S 同樣以 Hermite曲線來定義，由此我們可得一個二

元三次的曲面 (如圖 2.29)。所以只要給定一特定時間，我們便可得到該時間的 Hermite曲線。

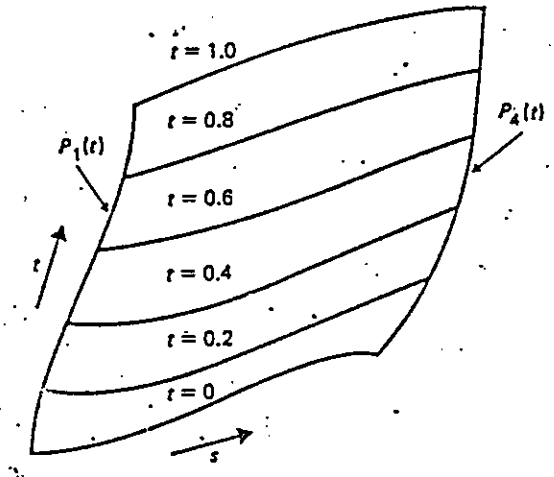


圖2.29 Hermite曲面

至於曲面的公式如下：

$S = [s^3\ s^2\ s\ 1]$

M_h：同上節之 M_h

M_h^t：M_h 的轉置矩陣 (tranpose matrix)

T^t ：T 之轉置矩陣

$T = [t^3\ t^2\ t\ 1]$

$$Q=\begin{bmatrix} p_{00} & p_{01} & dp/dt_{00} & dp/dt_{01} \\ p_{10} & p_{11} & dp/dt_{10} & dp/dt_{11} \\ dp/ds_{00} & dp/ds_{01} & dp/dsdt_{00} & dp/dsdt_{01} \\ dp/ds_{10} & dp/ds_{11} & dp/dsdt_{10} & dp/dsdt_{11} \end{bmatrix}$$

$X(s,t)=SM_hQ_xM_h^tT$

$Y(s,t)=SM_hQ_yM_h^tT$

$Z(s,t)=SM_hQ_zM_h^tT$

(見圖2.30)

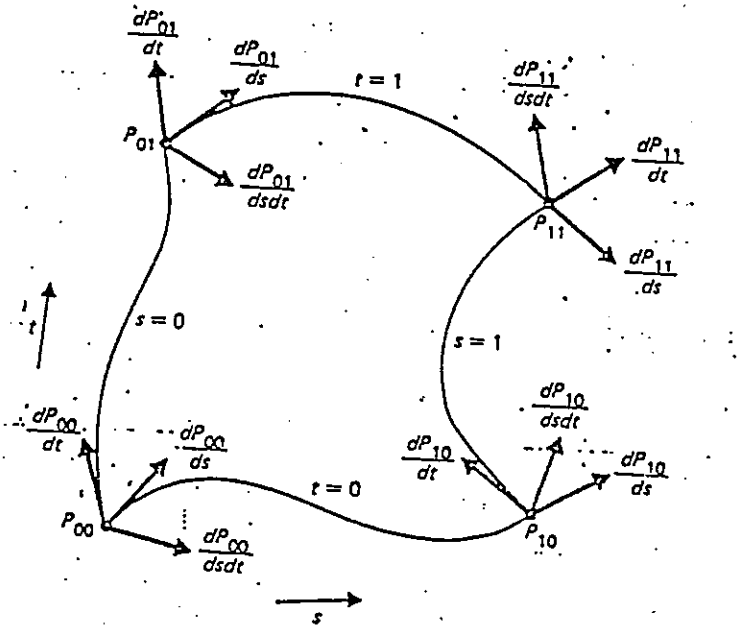


圖 2.30 Hermite曲面的控制參數，其中每一向量亦為三維的向量座標

2.4.3 Hermite、 Bezier和 B-spline 三種曲線模式(Form)之比較

事實上， Hermite曲線也可和Bezier曲線或B-spline一樣用於定義掃掠物體所需之平面曲線，至於這三種模式的利弊如下：

Hermite 曲線所需條件為控制點和切向量，所得到之曲線將較有利於真實曲線的逼近，但卻因一般使用者對切向量的行為較不易掌握，所以較不利於作為交談式曲線定義之用。

Bezier曲線之幾何概念因僅需以控制點來規劃使用者所需之曲線，所以避開了 Hermite曲線所需之切向量的頭痛問題；故若只需一次連續性的曲線，這種模式應可提供使用者一方便的交談式工具；它因具備了凸形殼的特性並通過的前後兩個控制點，更可讓使用者很方便地想像並定義所欲之曲線。

B-spline曲線因可滿足切向量，曲率變化都具連續性，所以較前兩種模式來得平滑、順暢，故常被繪圖人員用於設計飛機和船隻的表面，以計算所用的金屬條所需之高次連續性，以期達到高度的韌性。但也因為這種特性使得其所需之計算較前兩種模式來得多，且曲線通常不會通過控制點，儘管亦具備了凸形殼的特性，畢竟較不利於使用者作為交談式的繪圖工具，所以若非必要， Bezier 曲線應可

列為優先考慮。

2.5 Laguerre's Method

在光線追跡法中，'求取物體和光線的交點'這個步驟算是最重要的了，由於這一步驟往往佔去了整個方法的95%時間，所以交點之求解方法的優劣便決定了所需時間之長短。

本文在求取光線和掃掠物體的交點時，於化簡後，所得之一元n次方程式，其根的求取所採用之方法，原則上是採用 Laguerre's Method，因其具備了：

(1) 穩定性 (stabliy)

(2) 快收斂性 (fast convergence)

(3) 可求出所有根，包括實數根和複數根。

至於 Laguerre's Method的推導結大致如下[Ralston 78]

f(x)為一元n次多項次，其根的求法為

$$X_{i+1} = X_i - \frac{n\, f(x_i)}{f'(x_i) - \sqrt{H(x)}}$$

x_0 為任意起始值

其中

$$H(x) = (n-1)\{(n-1)[f'(x)]^2 - nf(x)f''(x)\}$$

2.6 分佈式光線追跡法 (Distributed Ray Trauing)

模糊影像之產生乃是因物體快速移動，而使得人的眼睛因視覺暫留而產生對物體的影像有模糊且連續的效果，最常見的例子便是將照像機對一移動的景物作長期曝光的處理所得之結果。

83年 Potmesil 提供用電腦來模擬Blur效果的方法，雖然效果還算不錯，但由於過程繁瑣且原理頗為牽強，所以並未得到多數人的認同。

到了84年， Robert [Robert 84] 提出了將統計分佈的理論應用於傳統的光線追蹤法，或稱之為分佈式光線追蹤法 (Distributed Ray Tracing)，是較具實用效果及合

乎自然的作法，所以本實驗便採用這種技巧作為產生模糊影像的方法。以下幾個章節，將簡述實際的作法和原理。

2.6.1 傳統的作法

在傳統的光線追跡法中，物體必須是固定的，所以很顯然的若要應用於動畫效果的模擬，勢必得藉助傳統所謂'慢拍快放'的原理將多張影像快速放送不可；至於要達到Blur的效應，更須使前後的影像儘量連貫且密合；故而這種以'時間換取效果'的作用常會耗掉許多的金錢與人力。

2.6.2 如何移動物體

如前節所述，傳統的作法具有'物體必須固定'的瓶頸；所以接下來便須'移動'物體； Robert 所提出之分佈式光線追跡法的作法是賦與每一條光線 (ray)一個特定時間 (time) ，然後根據這一時間和物體既定的移動軌跡 (motion path)，算出該物體在這時間的位置(position)，大小 (scaling)，最後再依傳統的光線追跡法，對物體做'碰撞' (intersection)的處理。

2.6.3 以 Supersampling來表現 motion

Supersampling 原是一種解決別名 (aliasing) 的技巧，即是將螢幕上的每一個像素點，切割成多個子像素點 (subpixel)以邏輯式 (logically) 地提高螢幕的解析度 (resolution)，最後再經過 '過濾器' (filter)算出該像素點的顏色 (GRB)。

前節所提到將每一光線賦與一時間，當然也可視為將每一pixel 賦與一時間，但因每一pixel 只能表現一種顏色，所以若只對每一pixel 作一次光線追蹤，那麼勢必無法表現物體運動的連貫性；因而必須藉助 supersampling 由每一subpixel代表一持定之time frame最後再結合起來，如此每一個 pixel便能因人的視覺限制表現出物體移動的連貫性；如圖2.32是一方塊作水平式的運動。

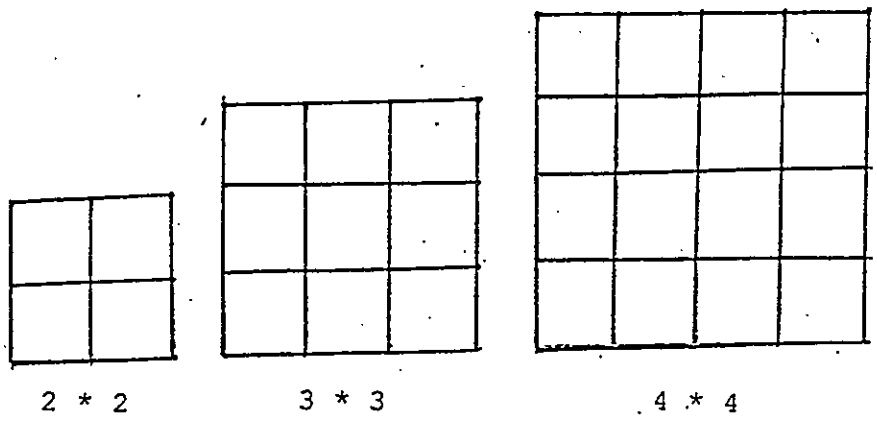
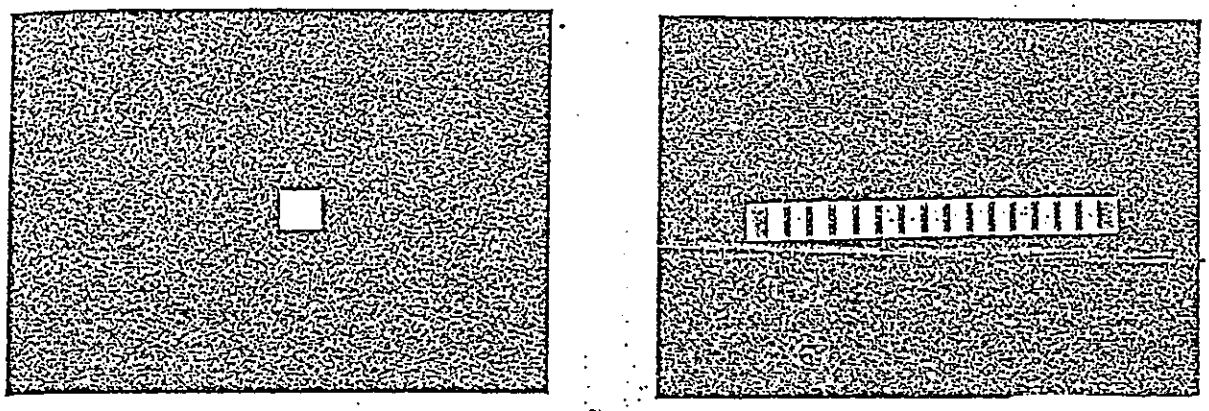


圖 2.31 Supersampling



(a) One sample per pixel, no jitter

Sixteen samples per pixel, no jitter

圖 2.32 物體移動的連貫性

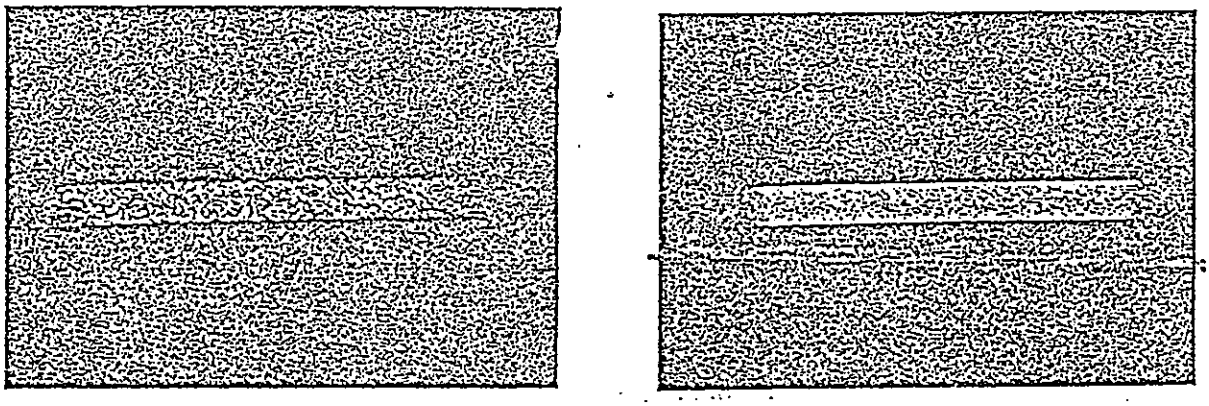
2.6.4 以統計分佈來表現模糊效果

至於每一(sub)pixel的時間如何決定呢？通常是依一亂數分佈 (random distribution)表 '初步' 決定其物特定時間。如圖2.33是一 4 × 4 的樣本模式 (sampling pattern) ，為了節省時間，通常建成 lookup 表以加快執行速度。

7	11	3	14
4	15	13	0
10	1	8	12
6	10	5	2

圖 2.33 4 × 4 樣本模式

然而上述每一條光線，所得到的時間，其結果只能表現連貫性，若要達到Blur的效果尚須透過一統計分佈模式對前述時間作一擾亂 (noise)，即所謂 jittering，方能得最後的確定時間 [Robert 86]。圖2.34是圖2.32的Blur效果。



(b) One sample per pixel, with jitter　　(d) Sixteen samples per pixel, with jitter

圖 2.34 方塊移動的Blur效果

2.6.5 中間極限定理

理論上，用 Poisson disk Distribution 來決定jittering是最理想的；但為了方便起見，本實驗根據統計學上的 CLT定理 [Mood 78]，以 Normal distrbution 來摹擬之。至於 CLT的內容如下：

[Central Limit Theorem]

Central-limit theorem If for each positive integer n, $X_1, \ldots, X_n$ are independent and identically distributed random variables with mean μ_X and variance σ_X^2, then for each z

$$F_{Z_n}(z) \text{ converges to } \Phi(z) \text{ as } n \text{ approaches } \infty, \qquad (31)$$

where

$$Z_n = \frac{(\bar{X}_n - \mathcal{E}[\bar{X}_n])}{\sqrt{\text{var}\,[\bar{X}_n]}} = \frac{\bar{X}_n - \mu_X}{\sigma_X/\sqrt{n}}.$$

第三章　變形物體運動模糊影像之產生

繼第二章介紹了一些本文所會用到的電腦繪圖技巧及數學模式之後，本章將結合言些理論基礎和常用的動畫技巧：'主圖像法'(Key Frame Method)說明本文一'變形物體運動模糊影影像之產生'的作法。

3.1 主圖像法之簡介

所謂主圖像法(Key Frame Method)是以往由畫家繪製圖案來產生動畫的時代所發展出的製作方法。其作法乃是以多人協同作業來完成，亦即由資深作畫員繪出動畫流程中的主要變化點，稱之為主圖像(Key drawing),然後由多為較資淺的作畫員繪出主圖像間的圖像，稱為中間圖像(in-between drawing)，最後再整合所有的圖像(圖3.1)以慢拍快放的方式，表現出動畫的效果。

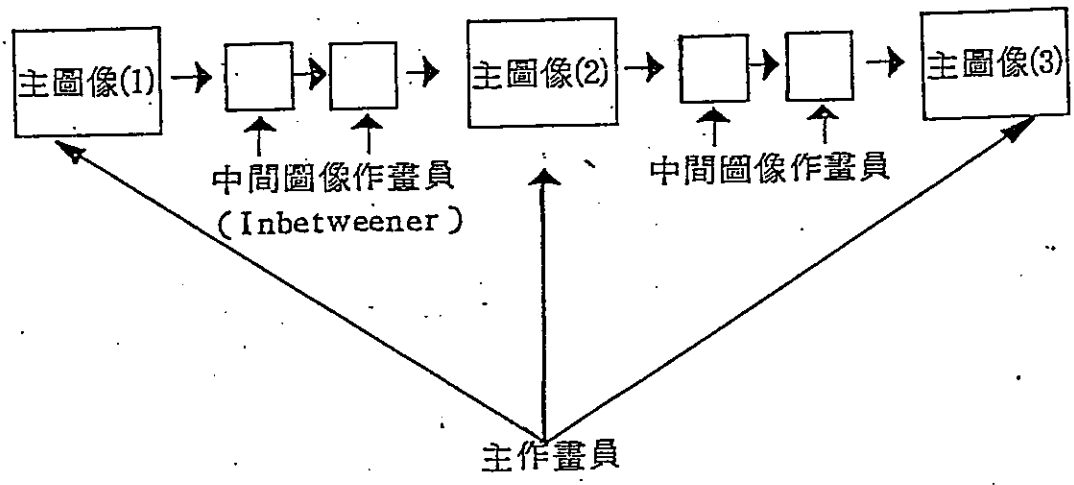


圖3.1 主圖像法的概略圖

而這種方法應用於電腦動畫的製作上，更具備了以下幾個優點[張76]:

(1) 可做高速檢查，在過去如果不繪製中間圖像就無法檢查角色的動作，但使用電腦來作動畫時，可自動產生中間圖像並加以檢查，並於必要時以交談的方式進行角色之動作的修改。

(2) 用電腦作的主圖像法所製作的主圖像，很容易完成其後的自動著色處理。

(3) 可作出精密的角色動作，因為可利用電腦依對象物體的正確物理原理計算其動作並予以顯示。

所以主圖像法至今仍為電腦製作動畫時的常用技巧之一。

3.2 中間圖像之產生

由於主圖像法於產生中間圖像的過程和本文頗為類似，所以本實驗決定仿照主圖像法並做適度的修正，以做為此步驟之理論基礎。

至於本系統產生中間圖像的步驟大致如下：

1. 輸入圖像(input drawings)
2. 確定物體運動路徑 (specify the motion of on

object along an path)

3. 產生中間圖像(produce in-between drawings)

若再加上

4. 產生背景(create a background)

5. 錄影(initiate the recording of a sequence on film)

6. 配上聲效(synchronize with sound)

應該就能達到動畫的目的了[Daniel 85]

以下幾個子節將詳述上述三個步驟。

3.2.1 輸入圖像

由於第二章所介紹之Bezier曲線和B軟樑分別具備了一次及二次微分連續性，因此本文選擇了這兩種曲線作為使用者定義首尾兩張圖像的工具。所以使用者若對此二種特性不是很清楚，可先瀏覽本文第二章的曲線簡介一節，以了解它們的特性，如此便較易定義出所欲的圖像。

關於輸入的方法，則可以PS300所提供之tablet或本人於PC上所作之介面以交談式的方式輸入控制點，而computer 便會根據使用者指定的曲線(Bezier曲線或B軟樑)公式及時地劃出所定義的曲線形狀，由於是以交談式的方式進行，所以使用者應能很方便的得到所要之曲線形狀，在

系統得到此曲線的控制點座標後，曲線便算定義完成。

於曲線定義完成之後，電腦便得到了所需要之控制點座標，接下來使用者再輸入此張圖像之掃掠方式：

1. 平行掃掠

2. 卡尼克掃掠

3. 旋轉掃掠

及物體的座標與旋轉角度，便完成了'輸入圖像'這項步驟。

3.2.2 確定物體運動路徑

路徑也可稱之為軌跡，軌跡可以運用在主圖像的運動，也可以運用在參考點的變化，路徑本身具有獨特的資料結構，可以讓主圖像選用；至於常用的路徑，則可存於檔案形成路徑庫，以方便使用者選用。

路徑的相關研究，在1970年由N. BURTNYK和M. WEIN所提出的方法大致包括等速、加速度和減速三個種類，其主圖像的移動方式如圖3.2所示

```
function LAW(OP:INTEGER; MAXVAL,FRACT:REAL):REAL;
begin
   case OP of
      1: (* constant velocity *)
         LAW := MAXVAL*FRACT;
      2: (* acceleration *)
         LAW := MAXVAL*(1-COS(PI*FRACT/2));
      3: (* deceleration *)
         LAW := MAXVAL*SIN(PI*FRACT/2);
      4: (* acceleration then deceleration *)
         LAW := MAXVAL*(1-COS(PI*FRACT))/2
   end
end.
```

圖3.2 主圖像移動方式

圖3.3所示則是常見的主圖像移動法則：

```
create OBJECT(...);
for IMAGE := 1 to NBFRAMES do
begin
   FRACTION := IMAGE/NBFRAMES;
   VAL := LAW(OPERATION,MAXVAL,FRACTION);

   .. transformation using VAL to modify OBJECT into OBJ2 ..

   draw OBJ2;
   TAKEPICTURE;
   PAGEGRAPH
end;
```

圖3.3 主圖像移動法則

路徑的定義：

在本文中，使用者當然也可在速度上依自己所需，選擇其中一種。接下來是更重之運動軌跡的定義：

1. 在本系統中，若是使用者忽略此項定義，那麼系統就會依物體的起始位置(S)和結束位置(E)，作等速直線運動。而旋轉角度的變化也會依直線內插法進行。

   如： position(S) : (x_1,y_1,z_1)

   angle(S) : $(\theta_{x1},\theta_{y1},\theta_{z1})$

   position(E) : (x_2,y_2,z_2)

   angle(E) : $(\theta_{x2},\theta_{y2},\theta_{z2})$

   則任意時間t(0≦t≦1)的所在位置及旋轉角度為

   position(t) : $(x_1+(x_2-x_1)*t, y_1+(y_2-y_1)*t, z_1+(z_2-z_1)*t)$

   angle(t) : $(\theta_{x1}+(\theta_{x2}-\theta_{x1})*t,\ \theta_{y1}+(\theta_{y2}-\theta_{y1})*t,\ \theta_{z1}+(\theta_{z2}-\theta_{z1})*t)$

2. 否則可以依Bezier曲線的輸入方式將所欲定義的運動軌跡告訴系統：即除了起始與結束位置之外，再於其中加上兩個控制點(3,4)

   如： position(S) : (x_1,y_1,z_1)

   angle(S) : $(\theta_{x1},\theta_{y1},\theta_{z1})$

   position(E) : (x_2,y_2,z_2)

angle(E) : $(\theta_{x2},\theta_{y2},\theta_{z2})$

加上 position(3) : (x_3,y_3,z_3)

position(4) : (x_4,y_4,z_4)

根據(S),(E),(3),(4)四個座標，系統便可得到一Bezier曲線：

Bx : $a_xt^3+b_xt^2+c_xt+d_x$

By : $a_yt^3+b_yt^2+c_yt+d_y$

Bz : $a_zt^3+b_zt^2+c_zt+d_z$

故任意時間 $0 \le t_1 \le 1$ ，物體所在位置即為

position(t_1) : ($B_x(t_1),B_y(t_1),B_z(t_1)$)

關於角度變化的軌跡，若有必要，使用者亦可依上例得到

angle(t_1) : $(B_{xa}(t_1),B_{ya}(t_1),B_{za}(t_1))$

3.2.3 中間圖像之求法一

截至上一小節，本系統根據那些軌跡函數便可表現出一般常見物體的運動過程。但本文希望達到變形的效果，所以尚須推算出其變形的曲線方程式：

由於本文對象為sweeping object，而object 之曲線

又是Bezier或B-spline曲線定義；故objcet便可將其視為平面上其二次連續性的組合曲線 (compsite curve)，(如圖 3.4)。

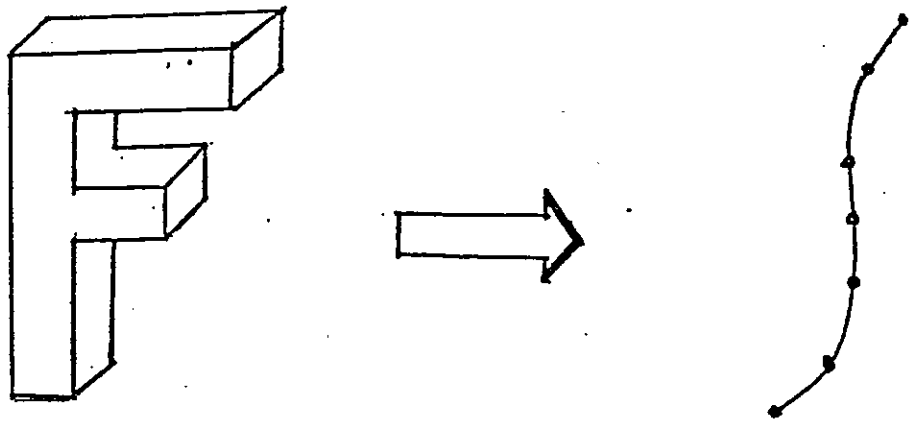


圖3.4 將sweeping object 視為一曲線

所以整個變形過程，可以由下述分析，以Hermite patch來摹擬[圖3.5]:

1. 將首尾圖像分別視為Hermite patch中t=0與t=1的軌跡。
2. 將使用者所定義的運動軌跡視為首尾圖像起點與終點(即S=0與S=1時)之運動軌跡。
3. 根據(1)和(2)，我們可得到下述有關Hermite patch之資料

   (a)端點座標：即首尾圖像之起點與終點(圖3.5之 P_{00}, P_{01},P_{10},P_{11})

(b)端點上S方向之切向量：由各端點於曲線上之軌跡微分求得(圖3.5之 $dP_{00}/ds, dP_{01}/ds, dP_{10}/ds, dP_{11}/ds$)

(c)端點上t方向之切向量：由各端點於運動軌跡微分求得(圖3.5之 $dP_{00}/dt, dP_{01}/dt, dP_{10}/dt, dP_{11}/dt$)

(d)最後為了簡化起見我們設此曲面之扭曲度為0[即 $dP/dsdt=0$]，便可合乎Hermite Patch之模式

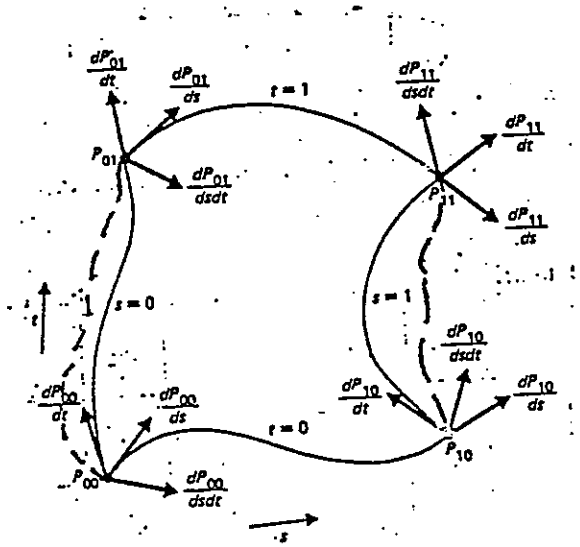


圖3.5 以Hermite Patch 摹擬曲線變形過程

至於摹擬過程可分為以下幾個步驟：

(a) 首尾主圖像段數(number of segment)的一致化

由於本文是採用主圖像法來求取運動過程的中間圖像，所以第一步驟便需使首尾兩主圖像的段數(nember of segment)一致，如此方能帶入適當的數學模式，以內插法求出任意時間的中間圖像。

關於這個問題，Burtnyk [Burtnyk 71]早已提出了解決之道，其方法如下：

If both figures have the same number of segments (NS1=NS2), do nothing else

if NS1 > NS2 (for NS2 < NS1 , the algorithm is the same with the NS1 and NS2 interchanged)

then the following numbers are computed :

RT := (NS1–1) div (NS2–1);

RS := (NS1–1) mod (NS2–1);

RT points are added to the first RS segments, and the (RT–1) to the others.

這方法原用於解決多邊形(polygon) 的問題，但由於情況頗為類似，所以亦適用於解決的曲線段數擴增問題，上述Algorithm ，不僅簡單，而且快速，所以頗為眾人所採用。但其缺點就是不很合乎真實情況，觀其精神，可知這方法乃是將所需增加的段數，很平均地加在各原有的曲線(見圖3.6) 上，因此本文擬以下法做比較：

```
dist1[1] := 0;  dist2[1]:=0;
for i:=2 to NS1 do
 dist1[i]:=(distance(p[i–1],p[i])+dist1[i–1])/length(curve1);
```

```
for i:=2 to NS2 do
  dist2[i]:=(distance(p[i-1],p[i])+dist1[i-1])/length(curve2);
 minus:=0;
for i:=2 to NS1 do
 begin
  nu:=0;
  for j:=1 to NS2 do
   if (dist1[i] < dist2[j]) and (dist1[i] ≥ dist2[j-1]) then
    nu:=nu+1;
   nu:=nu+minus-1;
   if nu > 0 then
    begin
     minus:=0;
     if i < NS1 then nu points are added to the ith segment
     else (nu-1) points are added to the segment
    end;
   else minus:=minus + nu;
 end;
```

其中P[i]為各段曲線的起點座標

上述Algorithm乃是根據各segment的起點至曲線起點的長度占總曲線長線長度的百分比，以對首尾曲線做相對應位置的比較，所以較合乎自然現象(如圖3.7)

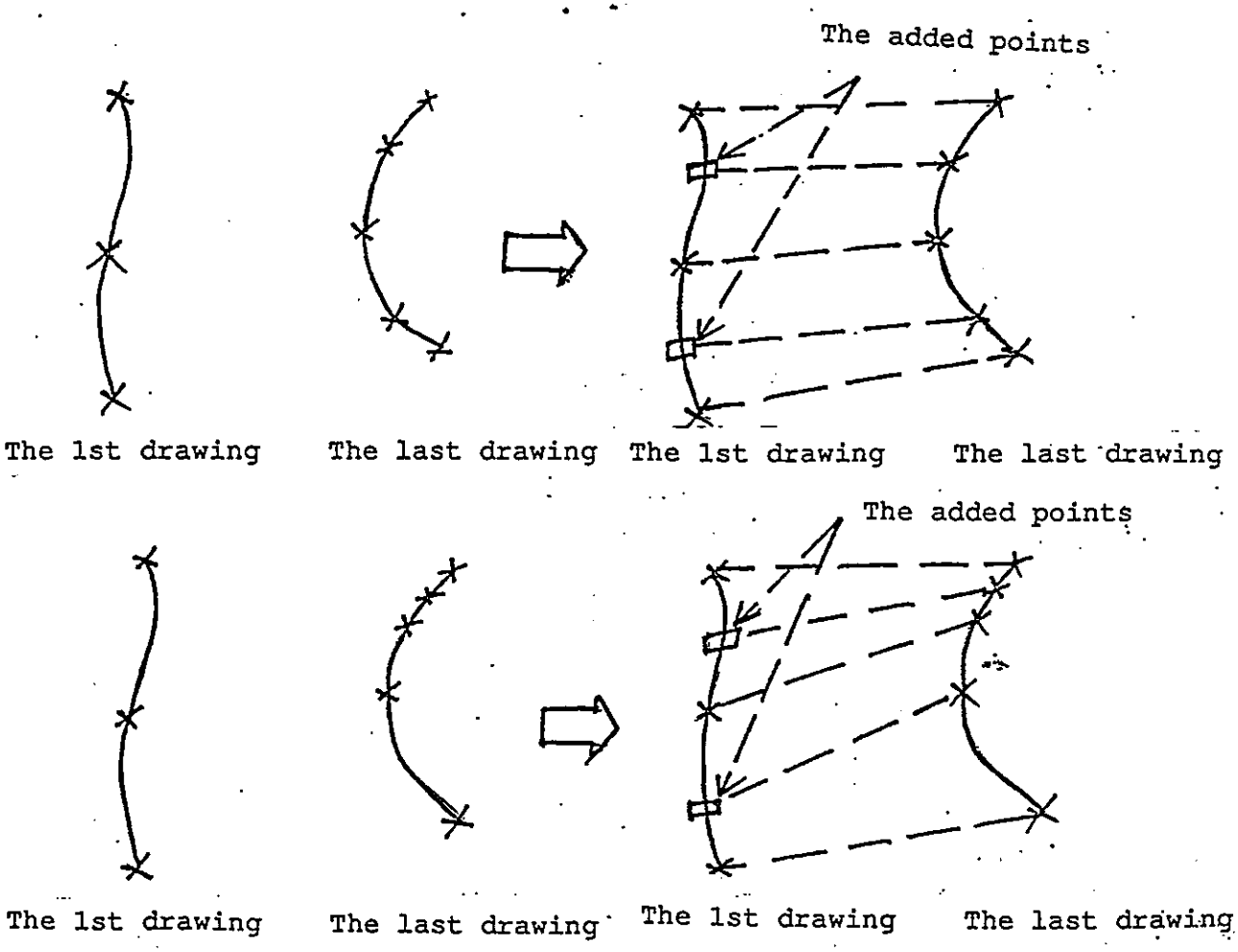


圖3.6 Burtnyk' Algorithm

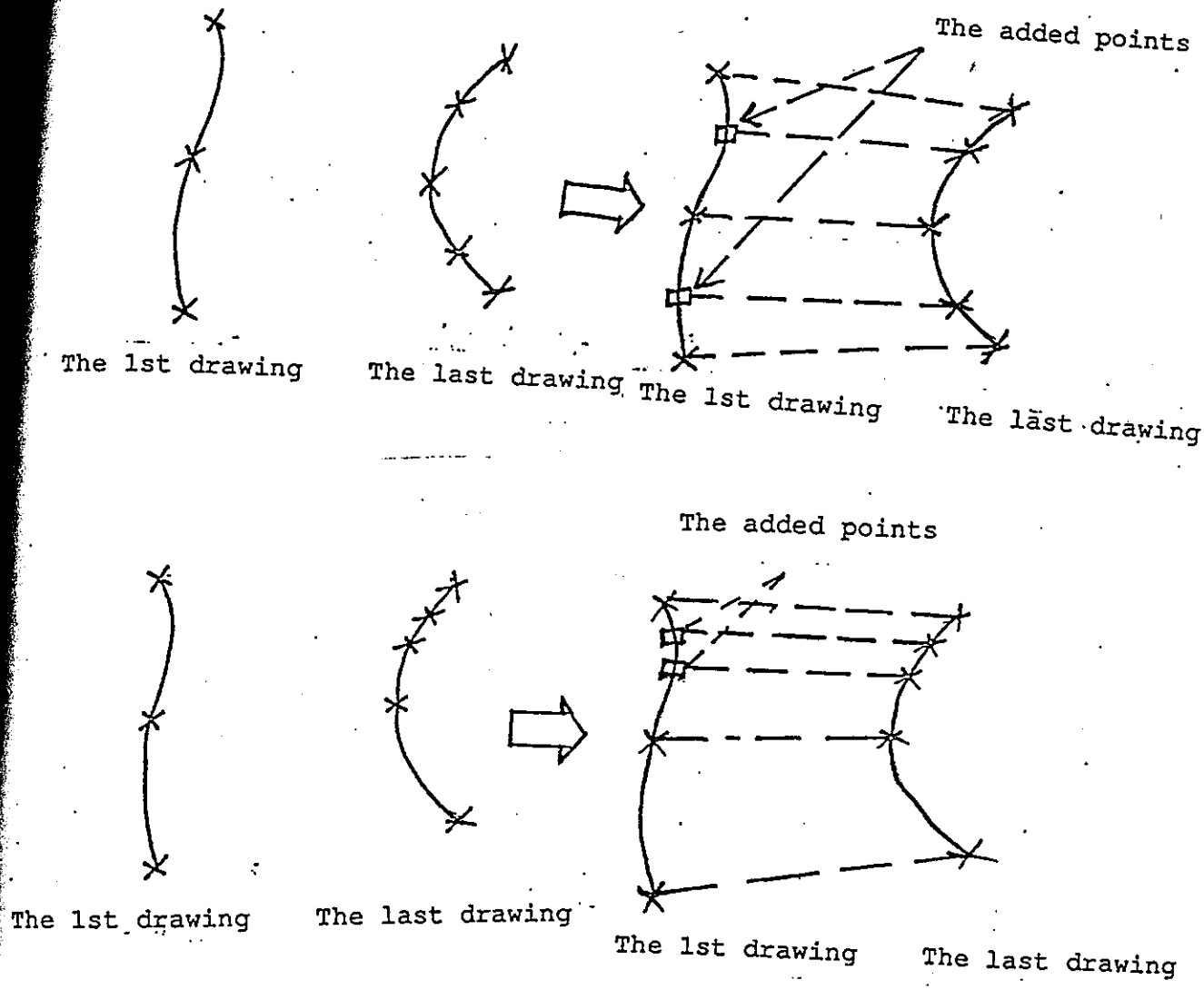


圖3.7 本 paper 之 Algorithm

在首尾兩條曲線的段數相等之後，我們便可依序做一對一的對應，以進一步的求出各段起點的運動軌跡，進而得到某特定時間該物體的變化過程。

(b) 求取各段曲線起始點的路徑

於步驟(a)做完首尾圖像段數一致化之後，接下來的

我們須求出各段曲線起始點的變形軌跡，以進一步求出各段曲線的變形過程。

若有需要，使用者可自行定義各段曲線的變形軌跡，但一般而言，普通的使用者很難想像這種變形過程，所以本系統是以使用者輸入的運動軌跡，作為曲線之首尾兩(因為曲線是封閉的）的變形軌跡。又為了使各點的路徑不致一樣，系統亦設定曲線的中點的變形軌跡為此運動軌跡。接下來將首尾圖像之起點與中點視為Hermite patch 的四個端點，配合3.2.3節的說明，將這端點的座標，曲線軌跡、運動軌跡代入Hermite patch的模式中，便可得以下兩個函數：

$$x(s,t)=a_{11}s^3t^3+a_{12}s^3t^2+a_{13}s^3t+a_{14}$$
$$+a_{21}s^2t^3+a_{22}s^2t^2+a_{23}s^2t+a_{24}$$
$$+a_{31}s\,t^3+a_{32}s\,t^2+a_{33}s\,t+a_{34}$$
$$+a_{41}\,t^3+a_{42}\,t^2+a_{43}\,t+a_{44}$$

$$y(s,t)=b_{11}s^3t^3+b_{12}s^3t^2+b_{13}s^3t+b_{14}$$
$$+b_{21}s^2t^3+b_{22}s^2t^2+b_{23}s^2t+b_{14}$$
$$+b_{31}s\,t^3+b_{32}s\,t^2+b_{33}s\,t+b_{34}$$
$$+b_{41}\,t^3+b_{42}\,t^2+b_{43}\,t+b_{44}$$

其中t代表曲線於時間0到1的運動軌跡，s代表曲線於時間0到1的變形過程。

得到上述方程式，我們便可用此Hermite 曲面來求得各段曲線起始點的運動軌跡：將各segment 的起始點至曲線起點的長度占總曲線長度的百分比代入 S，即可得到前半段各段曲線之起始點的運動軌跡。其幾何現象，則可以下圖表示：

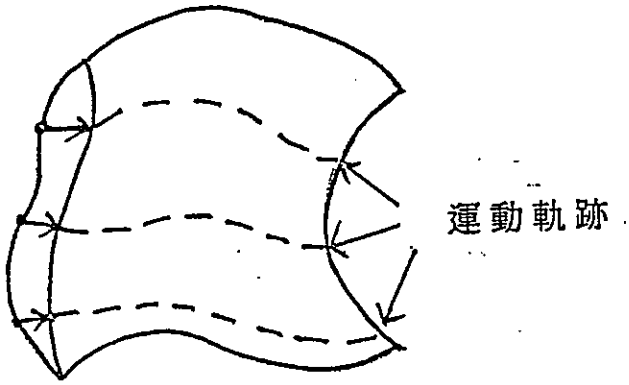


圖3.8 各起始點之運動軌跡

再將首尾曲線的中點及最後一點視為 Hermitepatch 的四個端點，並將其相關資料代入Hermite 模式中，重覆上述步驟，便可得到曲線後半部各段曲線起始點的運動軌跡。

(c) 求取中間圖像

由上節之方法得到各段曲線起始點的運動軌跡之後，

整個曲線的變化過程便可視為一個多頁Hermite曲面相接合的曲面，如下圖所示

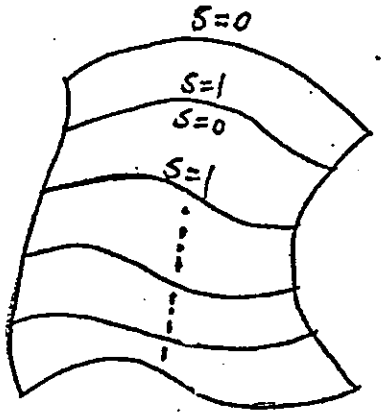


圖3.9 多頁小Hermite曲面相接合之曲面

由於Hermite曲面如同Hermite曲線般亦於接合處具有一次微分連續的特性，所以接下來我們可根據一時間t代入所有的Hermite曲面方程式，便可得到一個三次方的連續曲線；這便是我們所欲求的中間圖像(如圖3.10)。

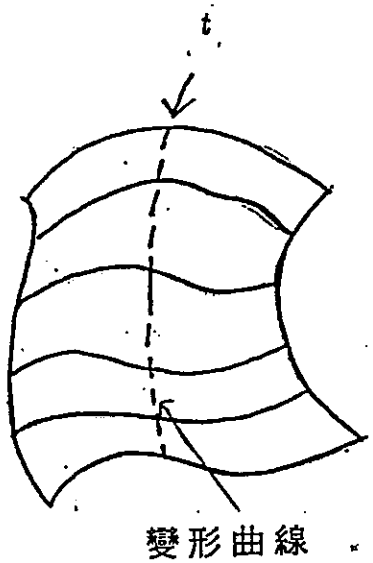


圖3.10 中間曲線的求得

3.3 Bounding Box 之計算

在第二章中我們曾經提過光線追跡法的最大缺點之一便是速度問題；尤其本文所探討的問題中，除了傳統的計算之外，尚需根據每條光線的時間，推算出各物體在該時間的變形狀況，所以更需借助一些方法來加快執行速度。

本實驗中為了簡化資料結構，只採用 bounding box 這種技巧來解決速度過慢這個問題：

(1) 先計算各張圖像的bounding box：

不論使用者是由鍵盤或tablet輸入各張圖像的控制點座標，在系統讀入座標的同時，便決定相對應的bounding box,如圖3.11所示

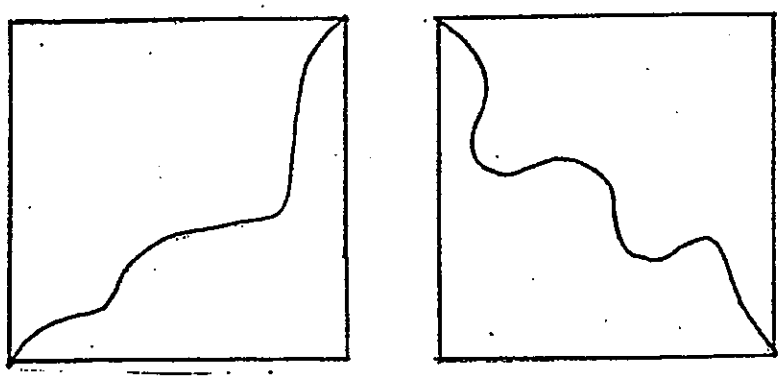


圖3.11 首尾兩張圖像bounding box

(2) 變形過程中的bounding box

決定了各張圖像的bounding box之後，將首尾兩張圖

像的bounding box結合在一起，或說聯集起來，便可視為變形過程中各張圖像的bounding box，但因本實驗乃採用Hermite 內插法來計算中間變形過程的圖像，所以於計算變形過程的同時並對各段曲線做極值(Extrema)的測試以調整bounding box的大小(如圖3.12)

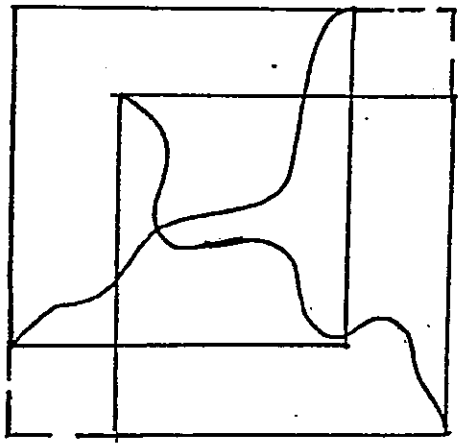


圖3.12 合併首尾兩張圖像之bounding box

3.4 模糊影像之產生

以3.3節所提之法求得首尾兩張圖像之間物體的變形過程之後，再配合第二章中所介紹之分佈式光線追跡法，以產生運動過程的模糊影像：

(a) 決定每一像素點的取樣數目(如圖3.13)

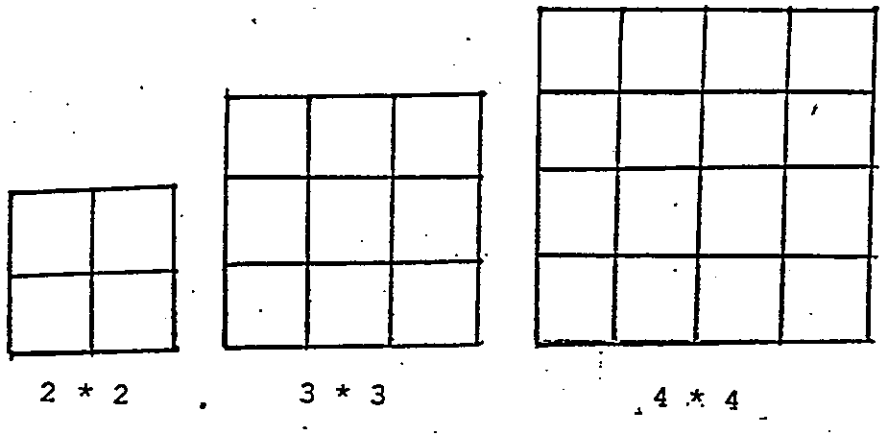


圖3.13 取樣模式

(c) 決定模糊影像的強度

本系統是以中間值定理來摹擬Normal distrbution，所以使用者只須輸入變數 n ，便可決定各Normal distribution 的分佈狀況

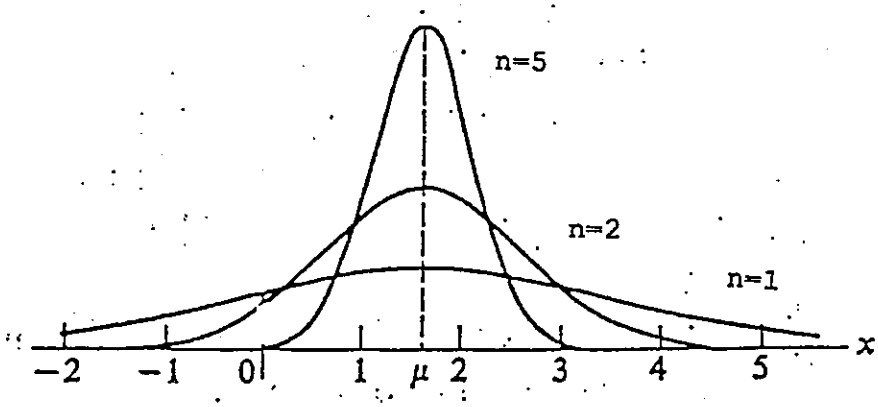


圖3.14 以變數 n 決定Normal disribution

根據上述(a)，(b)兩項條件，系統便可依分佈式光線追跡法產生所欲之模糊影像。

第四章　系統簡介

為了驗証本文所提之理論，筆者利用了中央研究院資訊研究所所提供之設備作了些實驗，本章將作簡略的說明和介紹。

(一)硬體設備

本實驗是在SUN3的工作站上進行的，所使用的 OS 是UNIX 3.5版，並以 E&S公司的PS390作為顯像器(dislay monitor)，其解析度為1024×864。

(二)軟體設備

軟體方面，則採用美國科羅拉多大學 (University Colorado) 於84年以PASCAL語言於Unix環境下所開發之軟體，名叫 'Raytrace'。由於此套軟體是以machine independent 為其出發點，所以只要接上適當的介面程式，便可適用於其它機器，如本實驗便是於SUN 工作站完成後，將所產生的raster file透過psout這個介面送到 PS390顯示(如圖4.1)

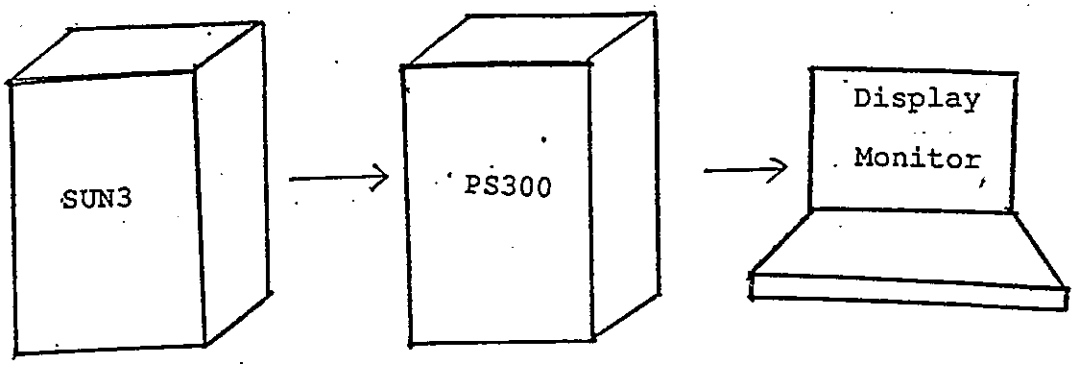
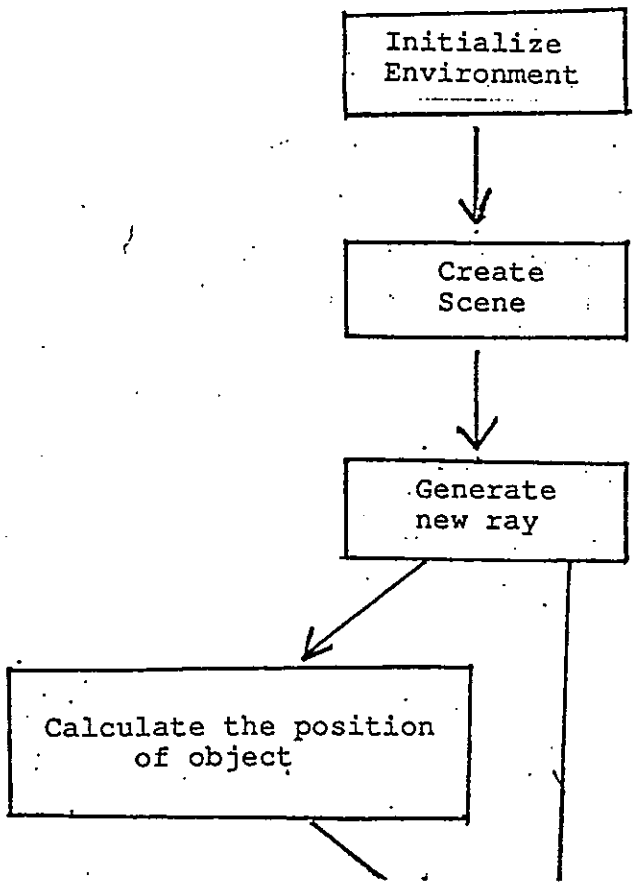


圖 4.1 硬體架構

4.1 'Raytrace'軟體簡介

關於 'Raytrace'這套軟體的基本架構大致如下：



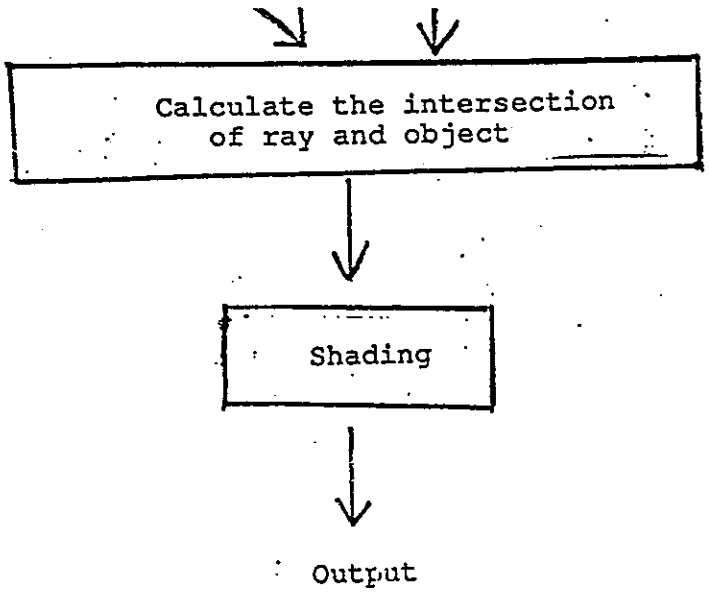


圖4.2 Raytrace之基本架構

至於上圖各步驟的內容分述於下：

(a) Initialize environment

此步驟的主要工作包括

(1)清除raster file

(2)讀入並設定一些變數之值，以供往後的步驟使用

(3)建立lookup table以做為產生blur時所需之統計數據

(b) Create scene

這部分的工作則是由使用者輸入所欲表現之物體的有關資料；其資料結構如下：

```
string    = packed array [1 .. stringlength] of char;
axis       = (xaxis, yaxis, zaxis);

dualarray = array [1..4, 1..4] of real;
matrix = ^ dualarray;

primitiveclass    = (block, cylinder, sphere, prism,translation,conic,rotation

surfaceattribute  = (transparent, mirrored, opaque, background, light,
                     crystal);
colors            = (red, green, blue);
intensity         = array [red .. blue] of real; { 0 - 1 }

surfaceptr = ^ surfacerecord;
surfacerecord = record
  color : intensity;  (* color of object under diffuse *)
  ka,kd,ks,kt : real; (* tweeking constants *)
  Iobject: intensity; (* ambient reflection color of object *)
  surface : surfaceattribute;
  prcntrfl: real;     (* for mirrored in raytrace *)
  end;

points = record
           x, y, z : real
         end;

coeff = record
          a,b,c,d,max,min,first,last:real;
        end;

objectptr = ^ objectrecord;
objectrecord = record
    name     : string;
    class    : primitiveclass;
    surface  : surfaceptr;
    transfm  : matrix;
    inverse  : matrix;
    sphereo  : points;
    angle    : points;
    radius   : real;
    case primitiveclass of
       cylinder : (cylradius : real;
                   orient    : axis;
                   top,bottom,left,right : real);
       translation ,conic,rotation: (orientt : axis ;
                      segment : integer;
                      u,v : array [1.. 60] of coeff);
       polyhydra: (   orientp   : axis;
                      vnu,pnu   : integer;
                      vertex    : array [1..vertexnumber] of points;
                      box       : boundingbox;
                      polygon   : array [1..polygonnumber] of polygontype;  )
  end;
```

圖4.3 物體之資料結構

(c) Generate new ray

顧名思義，光線追跡法的主要角色便是'光線'，關於光線的種類大致如下：

(1)普通光線：以使用者所定義之光源為光線起點，並通過螢幕上像素點的中心點，此為普通光線。故而解析度為512×512的螢幕，便可產生512×512條普通光線。

(2)反射光線：當光線碰到具反射性質的物體(如鏡子)時，便會產生反射光，則反射光乃是根據物理學上的反射定理求出的。

(3)折射光線：當光線碰到具有折射性質的物體時（如玻璃），便會產生折射光，此折射光及是根據物理學上的折射定理求出的。

至於光線的資料結構如下：

```
ray=record
    x0,x1,y0,y1,z0,z1:real;
end;
```

(d) Blur Or Not:若使用者欲產生變形物體之運動模糊效果，則於此步驗根據所輸入的運動軌跡及亂數表計算出物體的變形軌跡及所在位置。

(e) Calculate the intersection of ray and object

此部分乃是光線追跡法的核心，至於有關本系統所提供之物體，大致有下列幾種：

(1)球

(2)方塊

(3)圓柱體

(4)prism

(5)sweeping object [WIJK 84]

而光線與物體碰撞到後，其所傳回之資料結構如下：

```
intersectionptr = ^ intersectionrecord;
intersectionrecord = record
    status        : boolean;
    dx,dy,dz      : real;     { normal vector }
    px,py,pz      : real;     { fudged out intersection point }
    insidepx,
    insidepy,
    insidepz      : real;     { fudged in intersection point }
    surface       : surfaceptr;
    parameter     : real;
    attenuation   : real;
    distance      : real;
  end;
```

圖4.3 交點之資料結構

(f) Shading

根據步驟(d)所得到的資料，再根據Whitted[Whitted 80]之採光模式：

$$I = Ia + K_d\Sigma(N \cdot \hat{L}_j) + K_sI_s + K_tI_t$$

以得到此光線之最終rgb值。

(g) Output raster file

對所有光線做完追跡之後，便可產生一raster檔，再將此檔再透過psout送到PS390顯現。

第五章　實驗結果

本章則以幾張於軟體 "Raytrace" 上執行所得之圖片為本文做進一步說明：由於受限於硬體設備，本實驗只將一像素點切成 4個子像素點，即以2×2的 supersampling 做實驗；而每張圖片的解析度為256×256，再以照相機作適當的取景並顯現。

圖 5.1: 此乃一球依使用者所定義的路徑做運動，為了方便說明，本圖並未加入jittering 所以只有"運動"的效果，而無模糊的影像。

圖 5.1　球之運動

圖 5.2: 本圖為一撞球之運動模糊影像之摹擬，由於球之運動範圍極小，所以給人的感覺是晃動的效果。

圖 5.2 撞球之運動模糊

圖 5.3: 本圖的物體是於表面貼上其它圖片的raster file之立方體，再根據運動路徑，產生其運動模糊影像。本圖旨在摹仿前人已做過之結果[Foley 82]，以驗証程式之優劣。

圖 5.3 立方體表面貼上圖片後之運動模糊

圖 5.4: 和上圖一樣，只是將物體形狀換成圓柱體。

圖 5.4 圓柱體表面貼上圖片後之運動模糊

圖 5.5: 於立方體的表面上貼上 NEC的字樣，產生其做運動模糊的影像，以達到廣告效果。但或許會稍嫌單調。

圖 5.5 立方體表面貼上 NEC後做運動模糊

圖 5.6: 平行掃掠方式產生之物體。

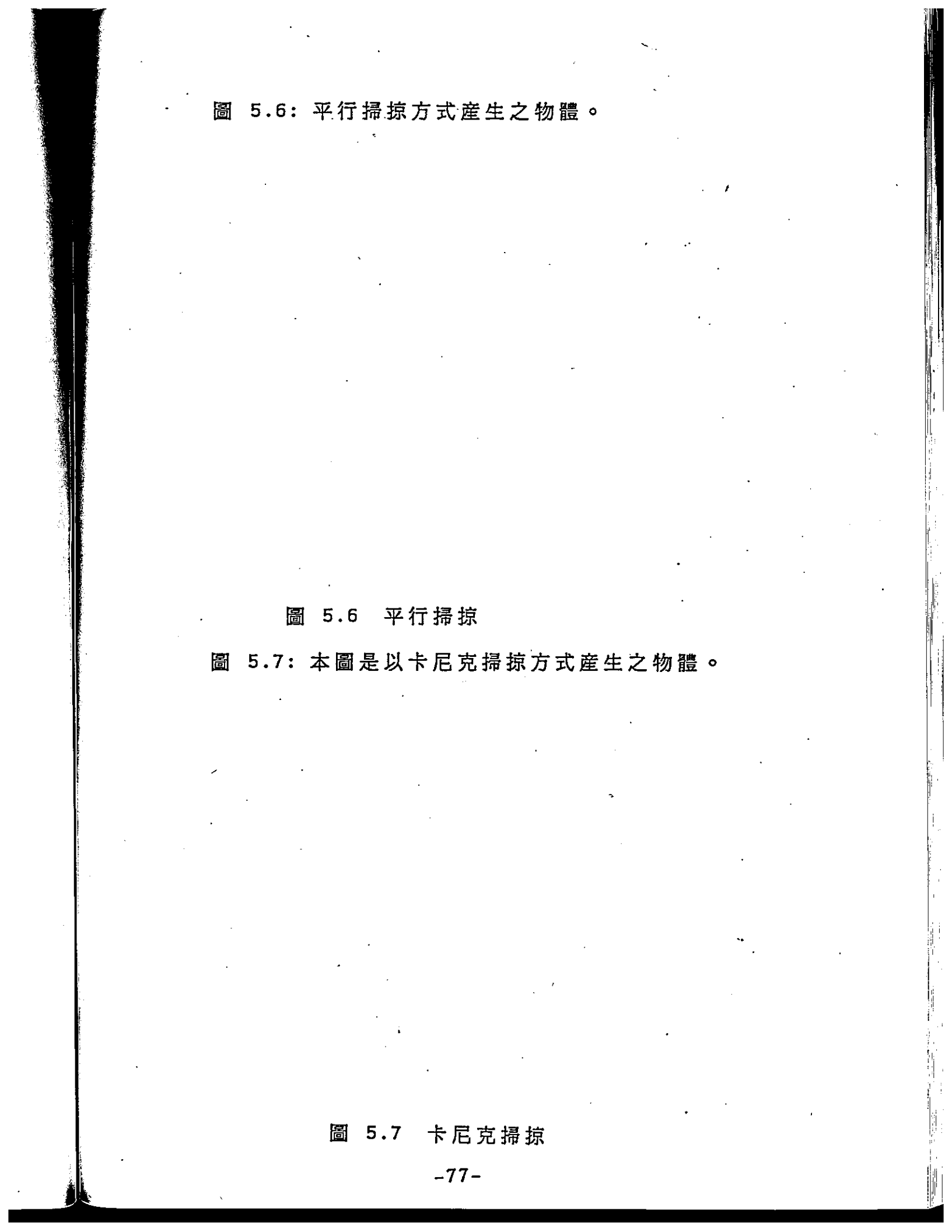


圖 5.6 平行掃掠

圖 5.7: 本圖是以卡尼克掃掠方式產生之物體。

圖 5.7 卡尼克掃掠

圖 5.8: 本圖則是以旋轉掃掠方式產生之物體。

圖 5.8 旋轉掃掠

圖 5.9: 以平行掃掠方式定義 NEC三個字樣，接下來以由前而後的運動方式產生其運動模糊影像；這是較常見的廣告方式。

圖 5.9 NEC由前而後之運動模糊影像

圖 5.10:由使用者以平行掃掠方式定義出首尾圖線的形狀，再根據其運動路徑產生其變形的模糊影像。

圖 5.10 變形物體之運動模糊影像 (一)

圖 5.11 變形物體之運動模糊影像 (二)

圖 5.12:由使用者以旋轉掃掠方式定義出首尾圖像的形狀，再根據其運動路徑產生其變形的模糊影像。

圖 5.12 變形物體之運動模糊影像

第六章　總　結

6.1 結論

經過多年的研究發展，電腦繪圖於真實感與美感的表達上，在某些方面已經頗為完善了；再加上電腦的速度愈來愈快，現今結合動畫技巧及電腦繪圖方法以產生高品質的電腦動畫雖然尚言之過早，但一些電腦輔助教學及電腦模擬訓練之軟體的現況，卻給了我們無比的信心，相信這種計畫的實現，決不只是緣木求魚，而是指日可待的。

以往關於模糊效果的研究，多只見於電腦繪圖的領域中，若將其加之於動畫影片中，或許更能增加其真實感，當然所製出之片子也就更具說服力。本文所做之實驗，主要是針對掃掠物體，以動畫製作上常用的主圖像法為架構，配合Hermite Patch 模式來求得物體的變形過程，如此使用者若有需要，只需輸入首尾兩張圖像的形狀，人便可得到物體由第一張圖像變化到最後一張圖像過程之任一時間的圖像；最後再依分佈式光線追跡法以統計分佈模式對運動的取樣時間加以擾亂 (jittering)，如此便可產生物體於運動過程中兼具變形效果的模糊影像。所以一些以掃掠方式產生的 3-D之物體，若欲產生其變形效果的運動模

糊影像，本系統大致還能提供一項不難使用的工具。

6.2 未來的發展

由於時間不足與硬體設備的限制，本文只對掃掠物體做探討，因而所能提供的物體形狀也就十分有限。但因光線追跡法十分適合以Multiprocessor系統來加快其運算速度；雖然現今尚未達到成熟階段，但筆者相信不久的將來，光線追跡法應能更為大家所接受。屆時有關本文的研究對象，必能更為提昇。例如 Parametric Surface, Soft object 等物體，只要克服其有關運動路徑的瓶頸，欲產生其中間之變形圖像當是可行的，當然若再加上模糊影像或紋路(texture) 所拍得之動畫品質，當是更佳的。

參考文獻

[張 76] 張希誠譯，"電腦動畫原理精析"，台北，第三波文化事業公司，民國76年 2月。

[Appel 68] Allel A "Smoe Technique for Shading Machine Rendering of Solids " , Proc SJCL , AFIPS ,Reston , Va 32:37-45

[Badt 88] Sig Badt Jr, " Two Algorithm for taking advantage of temperal coherence in ray tracing ",Visual Computer., 1988:4.

[Bezi 74] Bezier , P , "Mathematics an Practical Possibilities of UNISURF",in R.E. Barnhill and R.F.Riesenfeld (eds) , Computer Aided Geometric Design , Academy, New York , 1974.

[Brigham 74] Brigham , E. Oran , The Fast Fourier Translation, Prentice-Hall,Englewood Cliffs,1974.

[Burden 81] Richard L. Burden,Numerical Analysis,@nd edition.Wad sworth Intersectional Student Edition.

[Burtnyk 71] Burtnyk N , Wein M, " Computer-Generated Key-Frame Animation" Journal of Society for Motion Picture and Television Engineering,80:149-153.

[Cook 81] Cook,Robert 1. and Kenneth E.Torrance ," A Reflection Model for Computer Graphics " , ACM Translations on Graphics , Vol 1 ,No 1, pp 7-24,January 1982.

[Foley 82] Foley,James D & Van Dam ,Andries , " Fundementals of Interactive Computer Graphics ", Addition-Wesley Publ ishing Company,1982.

[Glassner 84] Andrew S.Glassner,"Space Subdivision for Fast Ray Tra cing", IEEE CG&A , 1984 October.

[Deniel 85] Nadia Magnenat-Thalmann , Daniel Thalmann ", Computer Animation , 1985 , page 42.

[James 82] James D. Foley & Van Dam ,"Fundamental of Interactive Computer Graphics ",Addison-Wesley Publishing Company ,1982.

[Mishita 82] Mishita,Tomoyuki,Isao Okamura ,and Eihachiro Makamae, SIGGRAPH Art Show ,1982 .

[Potmesil 82] Potmesil,Michael and Indranil Chakravarty ,"Synthetic Image Generation with a Lens and Aperature Camera Mod el" , ACM Translations on Graphics , Vol 1 ,No 2 , pp 85-108 , april 1982 .

[Potmesil 83] Potmesil,Michael and Indranil Chakravarty ," Modeling Motion Blur in Computer-Generated Images " , Computer Graphics, Vol 17,No 3,pp 389-399,July 1983 .

[Ralston 78] Ralston,A.,and Rabinowitz,P." A First Course in Numerical analysis,and ed,McGraw-Hill,New York,1978,pp380 -383.

[Robert 84] Robert L. Cook,"Distributed Ray Tracing",ACM Computer Graphics , 1984 , page 137-144.

[Robert 86] Robert 1. Cook,"Stochastic Sampling in Computer Graphics ", ACM Translations on Graphics , vol 5, No 1, January 1982 .

[Rogers 85] David F. Rogers , " Procedual Elements for Computer Graphics",page 374-377.

[Rona 69] Ronald M . Baecker , " Picture-Driven Animation " , Preceedings of Spring Joint Computer Conference,1969, AFIPS Press.

[Thalmann 87] Nadia Magnenat-Thalmann," Image Synthesis ",Page 176.

[Whitted 80] Whitted, T." An illumition model for shaded display." Comm ACM 23,6(June 1980),343-349.

[Wijk 84] Wijk , J , Van Jarke," Ray tracing Objects Defined By Sweeping Planar Cubic Spline " , ACM Translations on Graphics,Vol 3,No 3,July 1984.

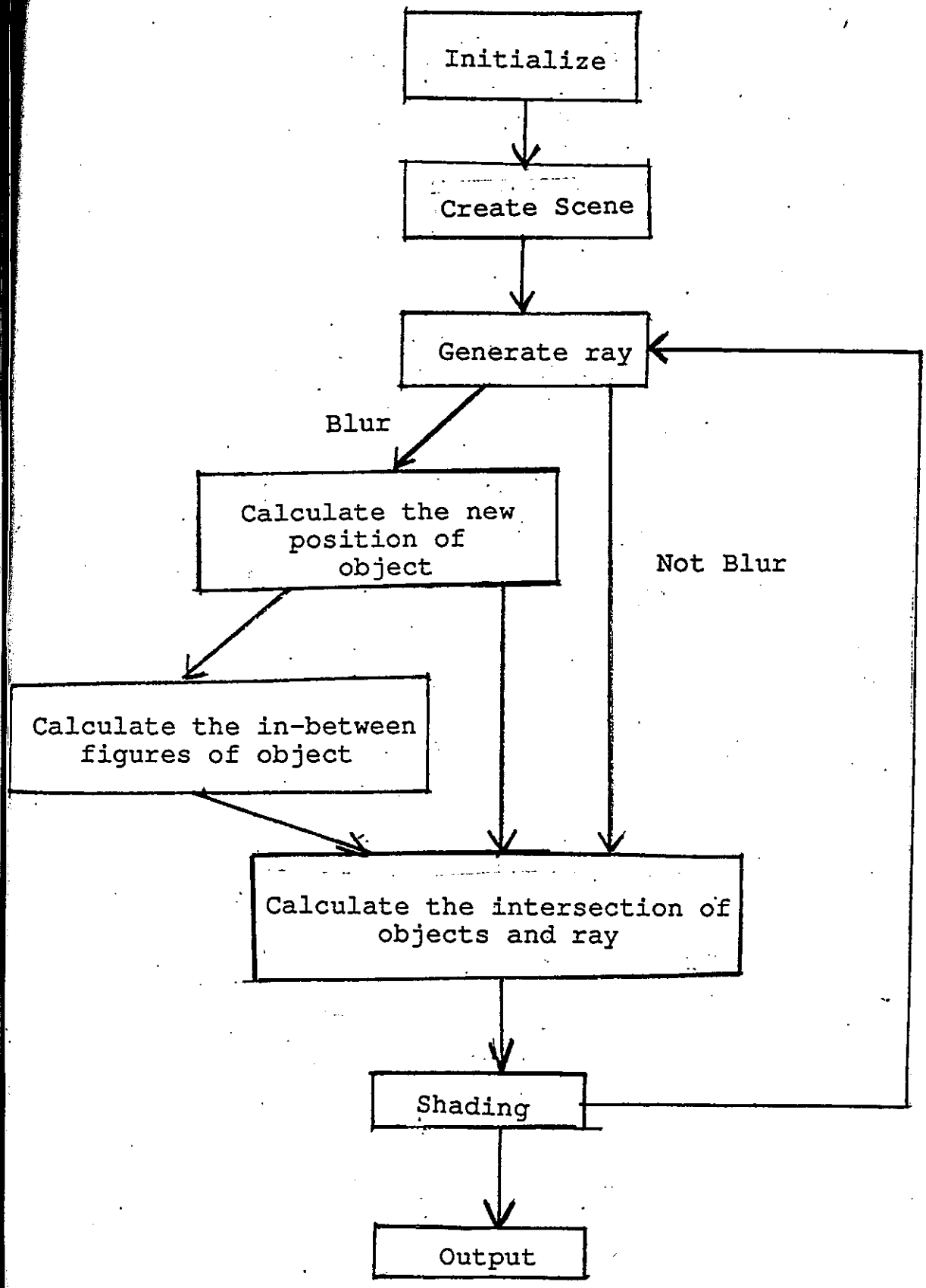
Document

本系統是以 PASCAL 所寫的程式，大致流程見下頁，若使用者欲轉換'c'，可參考'ptoc'指令，但有些限制，所以最好先 backup 一份再作測試。

次頁之流程，虛線框內以外是一般'靜態'物體之 Raytrace 的流程，可參考論文後 Reference [Rogers 85] 關於運用者欲作 motion blur 及重的物体之处理時，流程就會進入框內，亦即本論文之重点。主要參考文獻包括：

1. [Foley 82]
2. [Robert 86]
3. [Robert 86]
4. [Foley 82]

關於各 procedure 及 function 的功能，將於以下之頁作進一步的 document。


Initialize
Create Scene
Generate ray
Blur
Calculate the new position of object
Not Blur
Calculate the in-between figures of object
Calculate the intersection of objects and ray
Shading
Output

procedure floorect(x,y):

目的：在立方体表面着色以成地板样式。

procedure startest(x,y,z:real; cav:integer):

目的：在球表面上着上'星星'状的图案

procedure floorstar(x,y:real):

目的：在一平面上着上'星星'状的图案

procedure laguerre(----):

目的：求取一n次方程式的所有实根

Reference: [Ralston 78] and [Wijk 82]

sorting:

目的：对roots之record依root.a值sorting

並記錄roots.a>=tmax的位置

Ref: [Wijk 82]

opline:

目的：求取Bezier or B-spline的多项式

決定Bounding-box的範圍

outrectp:

目的：先作bounding box的測試(for rotational object)

Ref: [Wijk 82]

rootfeating:

目的：判断一元n次方程式之根的个数，以作適当的

增減

motionblur：

目的：1. 对'object'根据 noise time 作平移

2. if paper=1 则作景深的处理

Ref：[Robert 84] [Robert 86]

boundvolume：

目的：测试一'ray'是否通过 bounding box

deptRoffield：

jittering ：Ref—[Robert 84]

Isblur：

目的：判断 tableindex 是否在 motionmatrix 中；

expanding:

目的：使obj1 and obj2的段數一致，以便作進一步的一對一處理

Ref：[Daniel 87]

more:

目的：擴增obj的段數

saving:

目的：將obj之spline方程式作backup

merge:

目的：將p1和p2的各段spline之bounding box作一對一的merge

goRead:

目的：若使用者欲作表形物体之处理，則進入本proc.

1. obj1和obj2之段數的一致化(expand(2))
2. 求取表的曲線

Ref：論文之第3.2節

getmatrix:

目的：求取Hermite patch的方程式

Ref：[Foley 87]

coor:

目的：求取spline P之起点座標

Pvalue:

目的：求取spline P上一点 t 之一次微分值

multiply:
目的：求取矩陣 $m_1 \times m_2$ 之結果

Hermite:
目的：求取中間繼線
Ref：論文 3.2 節

initM:
目的：initialize MR and MR^T
Ref：[Foley 書]

getdistance:
目的：求各段 spline 起点至曲線起点弧長占整條曲線長之比例，並將其值置於 array dist 中

getpath:
目的：求取各段 spline 起点之運动路徑

delete:
目的：在 table 中去掉 table^.id = name 的 record

saving1:
目的：將 u, v 存入 uu, vv 中

do3d:
目的：當 time=0，則作第一圖像的 raytracing
time=1 時，則作下張圖像的 ray-traing

plus：对段数较少的object作段数的擴增

Ref：論文3.2節

more2：

目的：依本人之algorithm作「擴增段數」的处理

Ref：論文3.2節

expand2：

目的：目的和expand相同，但algorithm不同，以作比較

Ref：論文3.2節

```
#include "pconst.h"
#include "sconst.h"
#include "const.h"
#include "ptypes.h"
#include "stypes.h"
#include "type.h"
#include "pvars.h"
#include "svars.h"
#include "var.h"
#include "pextprocs.h"
#include "sextprocs.h"
#include "proc.h"

procedure rotatetest{(var rae:ray;angle:real)};
  begin
    with rae^ do
     begin
       x1:=x1*cos(angle*3.141592654/180)-z1*sin(angle*3.141592654/180);
       z1:=x1*sin(angle*3.141592654/180)+z1*cos(angle*3.141592654/180);
    end;
  end;

procedure floortest{(x,y:real)};
  /*
      purpose : to paint floor patern on block
  */
  var tx,ty:real;
      tz :integer;
  begin
    tx := x * lengthx;
    ty := y * lengthy;
    tx := round(tx) mod 2;
    ty := round(ty) mod 2;
    tz := round(tx + ty) ;
    if (tz = 0) or (tz =2) then painting:=floor1;
 end;

procedure startest{(x,y,z:real;cas:integer;)};
  /*
     purpose : to test a point is in the area of star or not
  */
  var t:real;
  begin
    t:=sqrt(3);
    if cas=1 then
    begin
      if (z-t/9 < 0) and ((t*x + z + 2*t/9) > 0) and ((t*x -z -2*t/9) < 0)
then painting:=star1
      else if (z+t/9 > 0) and ((t*x + z - 2*t/9) < 0) and ((t*x -z +2*t/9) > 0)
then painting:=star1
    end;

    if cas=2 then
    begin
      if (y-t/9 < 0) and ((t*x + y + 2*t/9) > 0) and ((t*x -y -2*t/9) < 0)
then painting:=star1
```

```
      else if (y+t/9 > 0) and ((t*x + y - 2*t/9) < 0) and ((t*x -y +2*t/9) > 0)
then painting:=star1
   end;
end;

procedure floorstar{(x,y:real)};
  /*
      purpose : to paint 'star' pattern on block
  */
  var tx,ty:real;
      xx,yy,t:integer;

  begin
    tx:=x*lengthx;
    ty:=y*lengthy;
    xx:=0;  yy:=0;
    while (tx > 3) do
      begin
        tx:=tx-3;
        xx:=xx+1;
      end;
    while (ty>3) do
      begin
        ty:=ty-3;
        yy:=yy+1;
      end;
    if odd(xx+yy) then
      begin
    if ((tx*15+ty*10-27.5) >= 0) and (ty <=2) and ((tx*1.5-ty-1.75) <=0)
      then  painting:=star1
    else
    if ((tx*15+ty*10-47.5) <= 0) and (ty >=1) and ((tx*1.5-ty+0.25) >=0)
      then painting:=star1
    else painting:=star1;
      end
  end;

procedure laguerre;{(n:integer;a:col;max,min:real;var total:integer;root:col)}
  /*
     purpose : to find the roots of equation
     ref     : [ Ralston 78 ]
  */
  var h,temp,temp1,f1,f2,f3,xi,x2:real;
      iteration,i,j,flag:integer;
  begin
    temp:=0.98/n;temp1:=temp;
    total:=0;
    for i:=1 to 10 do
      root[i].a:=0;
    while (temp < 1 ) do
      begin
        xi:=temp;
        iteration:=1;
        while (iteration <= 4) do
        begin
        if n=3 then
```

```
          begin
            f1:=xi*xi*(a[3]*xi+a[2])+xi*a[1]+a[0];
            f2:=xi*(3*a[3]*xi+2*a[2])+a[1];
            f3:=6*a[3]*xi+2*a[2];
          end;
        if n=6 then
          begin
    f1:=a[0]+xi*(a[1]+xi*(a[2]+xi*(a[3]+xi*(a[4]+xi*(a[5]+a[6]*xi)))));
    f2:=a[1]+xi*(2*a[2]+xi*(3*a[3]+xi*(4*a[4]+xi*(5*a[5]+6*a[6]*xi))));
    f3:=a[2]*2+xi*(6*a[3]+xi*(12*a[4]+xi*(20*a[5]+30*a[6]*xi)));
         end;
            h:=(n-1)*f2*f2-n*f1*f3;
            h:=(n-1)*h;
            if h >= 0 then
            begin
              h:=sqrt(h);
              if ((f2 < 0) and ((f2-h)<>0)) then
                x2:=xi-n*f1/(f2-h)
              else
                if (((f2+h) <> 0) and (f2>0))then x2:=xi-n*f1/(f2+h);
            end
            else   x2:=xi-n*f1/f2;
            if (abs(x2-xi) < 0.0001) and (x2 <=max) and (x2 >=min) then
              begin
                i:=1;
                if (x2 < 0.0000001) then x2 :=0;
                if (total>=1) then
                  while i <= total do
                    begin
                      if (abs(root[i].a-x2) > 0.001) then
                        i:=1+i
                      else i:=100;
                    end;
                if (i<>100) then
                     begin
                       total:=total+1;
                       root[total].a:=x2;
                     end;
                iteration := 6;
              end;
            if (x2 > 100) or (x2<-100) then iteration:=6;
            xi:=x2;
            iteration:=1+iteration;
          end;
        temp:=temp+temp1;
      end;
     if total > 0 then
       for i:=1 to total do
         begin
           xi:=root[i].a;
           root[i].b:=a[1]+xi*(a[2]*2+xi*a[3]*3);
         end;
end;

procedure sorting;{(tmax:real;number:integer;var roots:co2;pos:integer)}
  /*
```

```
    purpose : to sort the roots
    ref     : [ Wijk 84 ]
  */
  var i,j:integer;
      temp,temp1,temp2:real;
  begin
    for i:=1 to number-1 do
      for j:=i+1 to number do
        begin
          if roots[i].a > roots[j].a then
            begin
              temp:=roots[i].a;
              temp2:=roots[i].b;
              roots[i]:=roots[j];
              roots[j].a:=temp;
              roots[j].b:=temp2;
            end;
        end;
    for i:=1 to number do
      if roots[i].a=tmax then pos:=i;
   end;

procedure spline;{(Bezier:integer;one,two,three,four:real;var u:coeff;class:primit
  /*
      purpose : get the equaltion of B-spline or Berzier curve
  */
  var temp1,temp2,temp3,temp4:real;
  begin
    if Bezier=0 then
      begin
        u.a:=(one*(-1)+two*3+three*(-3)+four)/6;
        u.b:=(one*3+two*(-6)+three*3)/6;
        u.c:=(one*(-3)+three*3)/6;
        u.d:=(one+two*4+three)/6;
      end
    else
      begin
        u.a:=(one*(-1)+two*3+three*(-3)+four);
        u.b:=(one*3+two*(-6)+three*3);
        u.c:=(one*(-3)+two*3);
        u.d:=(one);
      end;
    u.first:=0;
    u.last:=1;
    if one > two then
      begin
        temp1:=two;
        temp2:=one;
      end
    else
      begin
        temp1:=one;
        temp2:=two;
      end;
    if three > four then
      begin
```

```
      temp3:=four;
      temp4:=three;
    end
  else
    begin
      temp3:=three;
      temp4:=four;
    end;
  if temp1<temp3 then u.min:=temp1
  else u.min:=temp3;
  if temp2<temp4 then u.max:=temp4
  else u.max:=temp2;
  if flag=1 then
    case class of
      translation : begin
                      if u.max > tbox.xmax then tbox.xmax:=u.max;
                      if u.min < tbox.xmin then tbox.xmin:=u.min;
                    end;
      conic       : begin
                      if u.max > cbox.xmax then cbox.xmax:=u.max;
                      if u.min < cbox.xmin then cbox.xmin:=u.min;
                    end;
      rotation    : begin
                      if u.max > rbox.xmax then rbox.xmax:=u.max;
                      if u.min < rbox.xmin then rbox.xmin:=u.min;
                    end;
    end
  else if flag=2 then
    case class of
      translation : begin
                      if u.max > tbox.ymax then tbox.ymax:=u.max;
                      if u.min < tbox.ymin then tbox.ymin:=u.min;
                    end;
      conic       : begin
                      if u.max > cbox.ymax then cbox.ymax:=u.max;
                      if u.min < cbox.ymin then cbox.ymin:=u.min;
                    end;
      rotation    : begin
                      if u.max > rbox.zmax then rbox.zmax:=u.max;
                      if u.min < rbox.zmin then rbox.zmin:=u.min;
                    end;
    end;
end;

procedure outrectp;{(u,v:coeff;a,b,c,d,hu,hv:real;var outrect:boolean)}
  function inshyp(u,v:real):boolean;
    begin
      if (((a*v+b)*v+c-d*u*u) < 0) then
        inshyp:=true
      else
        inshyp:=false;
    end;
  begin
    if u.max < hu then outrect:=true
    else
```

```
    if v.min > hv then
      begin
        if inshyp(u.max,v.min) then
          begin
            if u.min > hu then
              outrect := inshyp(u.min,v.max)
            else outrect:=false
          end
        else outrect:=true;
      end
    else
      begin
        if v.max < hv then
          begin
            if inshyp(u.max,v.max) then
              begin
                if u.min > hu then
                  outrect:=inshyp(u.min,v.min)
                else outrect:=false;
              end
            else outrect:=true;
          end
        else
          if inshyp(u.min,v.min) then
            outrect:=inshyp(u.min,v.max)
          else outrect:=false;
      end;
  end;

procedure roottesting{(var number:integer;var root:co2;temp:integer)};

  var i,temp2,flag:integer;
      temp1: real;
  begin
    flag:=0;
    if number = 1 then
      begin
        root[temp+2]:=root[temp+1];
        flag:=1;
      end
    else if odd(number) then
      begin
        temp1:=root[temp+1].c;
        root[number+temp+1]:=root[temp+1];
        flag:=1;
        for i:=temp+2 to number+temp do
          if abs(root[i].c) < abs(temp1) then
            begin
              temp1:=root[i].c;
              root[number+temp+1]:=root[i];
            end;
      end;
    if flag=1 then number := number+1;
    if odd(number) then writeln('**** impossible ****');
  end;
```

```
#include "pconst.h"
#include "const.h"
#include "ptypes.h"
#include "type.h"
#include "stypes.h"
#include "drttype.h"
#include "papertype.h"
#include "pvars.h"
#include "var.h"
#include "papervar.h"
#include "drtvar.h"
#include "pextprocs.h"
#include "drt.h"
#include "paper.h"

procedure motionblur{(r:rayrecord;int : integer;temp1,temp2:by4;var object:objectpt
    /*
       purpose :   processing the motion blur
       reference : " Distributed Ray Tracing " ,ACM Computer Graphics 84.
    */

  function boundvolume(r:rayrecord;class:primitiveclass):boolean;
    /*
       purpose :   to test the input ray wether passing the bounding box or not
    */

    var vx,vy,vz,temp1,temp2,tail,tail2:real;
        test:boolean;
        box : boundingbox;
    begin

      case class of
        translation : box:=tbox;
        conic       : box:=cbox;
        rotation    : box:=rbox;
      end;

      with r do
        begin
          vx:=x1-x0;
          vy:=y1-y0;
          vz:=z1-z0;
          tail:=vy*x0-vx*y0;
          tail2:=vz*x0-vx*z0;
        end;
        if (vx*vy) >= 0 then
          begin
            temp1:=vy*box.xmin-vx*box.ymax-tail;
            temp2:=vy*box.xmax-vx*box.ymin-tail;
            if (temp1*temp2) <= 0 then test:=true
            else test:=false;
          end
        else
          begin
            temp1:=vy*box.xmax-vx*box.ymax-tail;
            temp2:=vy*box.xmin-vx*box.ymin-tail;
```

```
            if (temp1*temp2) <= 0 then test:=true
            else test:=false;
          end;

        if test then
          begin
            if (vx*vz) >= 0 then
              begin
                temp1:=vz*box.xmin-vx*box.zmax-tail2;
                temp2:=vz*box.xmax-vx*box.zmin-tail2;
                if (temp1*temp2) <= 0 then test:=true
                else test:=false;
              end
            else
              begin
                temp1:=vz*box.xmax-vx*box.zmax-tail2;
                temp2:=vz*box.xmin-vx*box.zmin-tail2;
                if (temp1*temp2) <= 0 then test:=true
                else test:=false;
              end;
          end;

        boundvolume:=test;
      end;

  var time:real;
      i,j,k,row,col:integer;
      rae:ray;
      raec:rayrecord;
      flag:boolean;
  begin
    flag:=msc;
    col := (scanline mod masksize ) +1;
    row := (scancol mod masksize ) +1;
    time := pattern[row,col]*0.1;

    mindex := mindex +1;
    if mindex > maxindex then mindex := 1;

    with object^ do
      begin
        for i:=1 to 4 do
          for j:=1 to 4 do
            begin
              transfm^[i,j]:=temp1.a[i,j];
              inverse^[i,j]:=temp2.a[i,j];
            end;

        sphereo:=temp1.sphereo;

        if paper=1 then
          begin
            sphereo.x:=sphereo.x+time*diff[int].x;
            sphereo.y:=sphereo.y+time*diff[int].y;
            sphereo.z:=sphereo.z+time*diff[int].z;
            rotate(name,xaxis,0,0,0,angle.x);
```

```
        (*   rotate(name,yaxis,0,0,0,angle.y+diffangle[int].y*time);
             rotate(name,zaxis,0,0,0,angle.z+diffangle[int].z*time);   *)
             translate(name,sphereo.x,sphereo.y,sphereo.z);
        (*   if not msc then
               begin
                 rae:=transformray(r,inverse);
                 raec:=rae^;
                 msc:=boundvolume(raec,class);
               end;
         *)
           end
        else
          begin
            sphereo.x := sphereo.x+time*6;
            sphereo.y := sphereo.y;
            sphereo.z := sphereo.z;
            rotate(name,xaxis,0,0,0,270*time*50);
            translate(name,sphereo.x,sphereo.y,sphereo.z);
          end;

    if (paper=1) then
      begin
        if (time=0) or (time=1) then
          do34(time,int,object)
        else
          Hermite(int,time,object);
      end;

    end;
 end;

/* the following procedure are not completed , if you want to know more about
   it, please reference "Distributed Ray Tracing ", ACM Computer Graphics,84
*/

procedure depthoffield{(x0,y0:real;var z0,x1,y1,z1)};
  var t:real;
  begin
    t:=(focuslength-y0)/(y1-y0);
    x1:=x0+(x1-x0)*t;
    y1:=y0+(y1-y0)*t;
    z1:=z0+(z1-z0)*t;

    z0:=z0+fradius*noise[dindex]*2;
    dindex:=dindex+1;
    if dindex > maxindex then dindex :=1;
  end;

procedure jittering{(x0,y0,z0,r:real;var a,b,c:real)};
  var len,xi,yi,zi,t:real;
  begin
    len:=sqrt(a*a+b*b+c*c);
    a:=a/len; b:=b/len; c:=c/len;

    xi:=x0+a*5; yi:=y0+b*5; zi:=z0+c*5;
```

```
    if a <> 0 then
      begin
        t:=sqrt(a*a/(2*a*a+sqr(b+c)));
        a:=xi-(b+c)*r/a;
        b:=yi+r*t;
        c:=zi+r*t;
      end
    else if b<>0 then
      begin
        t:=sqrt(b*b/(2*b*b+sqr(a+c)));
        a:=xi+r*t;
        b:=yi-(a+c)*r/b;
        c:=zi+r*t;
      end
    else if c <> 0 then
      begin
        t:=sqrt(c*c/(2*c*c+sqr(b+a)));
        a:=xi+r*t;
        b:=yi+r*t;
        c:=zi-(a+b)*r/c;
      end;
  end;

function Isblur;{(:boolean;)}
  var i:integer;
      ok : boolean;
  begin
    i:=1;
    ok:=true;
    while (i<=motionnumber) and (ok) do
      begin
      if motionmatrix[i]=tableindex then
        ok:=false;
      i:=i+1;
      end;
     Isblur := ok;
  end;
```

```
#include "pconst.h"
#include "const.h"
#include "ptypes.h"
#include "type.h"
#include "stypes.h"
#include "papertype.h"
#include "pvars.h"
#include "var.h"
#include "papervar.h"
#include "paper.h"

procedure expanding{(int:integer;var obj1,obj2:objectptr)};
  /*
     purpose : to let the segments of obj1 and obj2 are equal
     ref     : [ Deniel 85 ]
  */
  var div1,mod1:integer;
  begin
    writeln(debugf,'expanding');
    if obj1^.segment < obj2^.segment then
     begin
       div1 := obj2^.segment div obj1^.segment;
       mod1 := obj2^.segment mod obj1^.segment;
       more(obj1,div1,mod1,p1u[int],p1v[int]);
       saving(obj2,p2u[int],p2v[int]);
     end
    else
     begin
       div1 := obj1^.segment div obj2^.segment;
       mod1 := obj1^.segment mod obj2^.segment;
       saving(obj1,p1u[int],p1v[int]);
       more(obj2,div1,mod1,p2u[int],p2v[int]);
     end;
     if obj1^.segment <> obj2^.segment then
       begin
         writeln(' ********  error ************');
         writeln(obj1^.segment,obj2^.segment);
       end
     else writeln(' equal ');
  end;

procedure more{(var obj:objectptr;div1,mod1:integer;var uu,vv:sdirection)};
  /*
    purpose : adding control points to the object whose segment is less
    ref     : [ Deniel 84 ]
  */

  var i,j,k:integer;
      t1,t:real;
  begin
    j:=1;
    with obj^ do
      begin
        if mod1 > 0 then
          for i:=1 to mod1 do
            begin
```

```
              uu[j]:=u[i];
              vv[j]:=v[i];
              j:=1 + j;
              t:=1/(div1+1);
              t1:=t;
              for k:=1 to div1 do
                begin
                  uu[j]:=u[i];
                  vv[j]:=v[i];
                  uu[j].first:=t;
                  vv[j].first:=t;
                  uu[j-1].last:=t;
                  vv[j-1].last:=t;
                  t:=t+t1;
                  j:=j+1;
                end;
            end;

        for i:=(mod1+1) to segment do
          begin
            uu[j]:=u[i];
            vv[j]:=v[i];
            j:=1 + j;
            t:=1/(div1);
            t1:=t;
            for k:=1 to div1-1 do
              begin
                uu[j]:=u[i];
                vv[j]:=v[i];
                uu[j].first:=t;
                vv[j].first:=t;
                uu[j-1].last:=t;
                vv[j-1].last:=t;
                t:=t+t1;
                j:=j+1;
              end;
          end;
          segment:=j-1;
      end; (* end for with *)
  end;


procedure saving{(obj:objectptr;var uu,vv:sdirection)};
  /*
     purpose : backup the equations of curve of obj
  */
  var i:integer;
  begin
    with obj^ do
      for i:=1 to segment do
      begin
        uu[i]:=u[i];
        vv[i]:=v[i];
      end;
 end;
```

```
procedure merge{(var p1,p2:sdirection;nu:integer)};
  /*
     purpose : merge the bounding box of the corresponding bounding box
  */

  var i:integer;
  begin
    for i:=1 to nu do
      begin
        if p1[i].max < p2[i].max then
          p1[i].max:=p2[i].max
        else p2[i].max:=p1[i].max;
        if p1[i].min > p2[i].min then
          p1[i].min:=p2[i].min
        else p2[i].min:=p1[i].min;
      end;
  end;


procedure goahead{(int :integer;var obj1,obj2:objectptr)};
  /*
     purpose : to calculate the in-between figure from obj1 and obj2
  */
  var i,j,k,snu,mid:integer;
      Hu,Hv:Mh;
  procedure difference(po1,po2:points;var dif:points);
    begin
      dif.x:=po2.x-po1.x;
      dif.y:=po2.x-po1.y;
      dif.z:=po2.x-po1.z;
    end;

  begin
    initM;
    if obj1^.segment <> obj2^.segment then
      if expander=1 then
        expanding(int,obj1,obj2)
      else
        expanding2(int,obj1,obj2)
    else
      begin
        saving(obj1,p1u[int],p1v[int]);
        saving(obj2,p2u[int],p2v[int]);
      end;

    tbox.xmax := tbox.xmax+2;
    tbox.xmin := tbox.xmin-2;
    tbox.ymax := tbox.ymax+2;
    tbox.ymax := tbox.ymax-2;

    difference(obj1^.sphereo,obj2^.sphereo,diff[int]);
    difference(obj1^.angle,obj2^.angle,diffangle[int]);

    snu:=obj1^.segment;

    mid:=snu div 2;
```

```
    merge(p1u[int],p2u[int],snu);
    merge(p1v[int],p2v[int],snu);
    saving1(snu,p1u[int],p1v[int],p3u[int],p3v[int]);
    saving1(snu,p2u[int],p2v[int],p4u[int],p4v[int]);

    getdistance(p1u[int],p1v[int],snu,dist1);
    getdistance(p2u[int],p2v[int],snu,dist2);

    with pathx[1] do
      begin
        a:=0;
        b:=0;
        c:=obj2^.sphereo.x-obj1^.sphereo.x;
        d:=obj1^.sphereo.x;
      end;

    pathx[mid]:=pathx[1];

    with pathy[1] do
      begin
        a:=0;
        b:=0;
        c:=obj2^.sphereo.z-obj1^.sphereo.z;
        d:=obj1^.sphereo.z;
      end;

    pathy[mid]:=pathy[1];

    getmatrix(p1u[int][1],p1u[int][mid],p2u[int][1],p2u[int][mid],pathx[1],pathx[mi
    getmatrix(p1v[int][1],p1v[int][mid],p2v[int][1],p2v[int][mid],pathy[1],pathy[mi
    getpath(1,mid,Hu,dist1,pathx);
    getpath(1,mid,Hv,dist1,pathy);

    getmatrix(p1u[int][mid],p1u[int][1],p2u[int][mid],p2u[int][1],pathx[mid],pathx[
    getmatrix(p1v[int][mid],p1v[int][1],p2v[int][mid],p2v[int][1],pathy[mid],pathy[
    getpath(mid,snu+1,Hu,dist1,pathx);
    getpath(mid,snu+1,Hv,dist1,pathy);

    for i:=1 to snu do
      with pathx[i] do
        writeln(debugf,'path : ',a:3:3,'  ',b:3:3,'  ',c:3:3,'  ',d:3:3);

    for i:=1 to snu-1 do
      begin
        getmatrix(p1u[int][i],p1u[int][i+1],p2u[int][i],p2u[int][i+1],pathx[i],path
        getmatrix(p1v[int][i],p1v[int][i+1],p2v[int][i],p2v[int][i+1],pathy[i],path
      end;

    i:=snu;
    getmatrix(p1u[int][i],p1u[int][1],p2u[int][i],p2u[int][1],pathx[i],pathx[1],Hmx
    getmatrix(p1v[int][i],p1v[int][1],p2v[int][i],p2v[int][1],pathy[i],pathy[1],Hmy

    delete(obj2^.name);
end;
```

```
procedure getmatrix{(p00,p10,p01,p11,path1,path2:coeff;var H:Mh)};
  /*
      purpose : get the matrix of Hermite Patch
  */

  var i,j:integer;
  begin
    H[1,1]:=coor(p00);
    H[1,2]:=coor(p01);
    H[2,1]:=coor(p10);
    H[2,2]:=coor(p11);
    H[1,3]:=Rvalue(path1,0);
    H[1,4]:=Rvalue(path1,1);
    H[2,3]:=Rvalue(path2,0);
    H[2,4]:=Rvalue(path2,1);
    H[3,1]:=Rvalue(p00,p00.first);
    H[4,1]:=Rvalue(p10,p10.first);
    H[3,2]:=Rvalue(p01,p01.first);
    H[4,2]:=Rvalue(p11,p11.first);
    H[3,3]:=0;
    H[3,4]:=0;
    H[4,3]:=0;
    H[4,4]:=0;

    H:=multiply(M1,H);
    H:=multiply(H,M2);
  end;

function coor{(p:coeff):real;};
  begin
    with p do
      coor:=d+first*(c+first*(b+a*first));
  end;

function Rvalue{(p:coeff;t:real):real};
  begin
    with p do
      Rvalue:=c+t*(2*b+t*3*a);
  end;

function multiply{(m1,m2:Mh):Mh};
  var i,j:integer;
      K:Mh;
  begin
    for i:=1 to 4 do
      for j:=1 to 4 do
        K[i,j]:=m1[i,1]*m2[1,j]+m1[i,2]*m2[2,j]+m1[i,3]*m2[3,j]+m1[i,4]*m2[4,j];
    multiply:=K;
  end;

procedure Hermite{(int:integer;t:real;var obj:objectptr)};
  /*
      purpose : calculate the in-between figure of every segment of obj
  */
```

```
  function coor2(u:coeff;co:real):real;
    var t:real;
    begin
      with u do
        begin
          t:=(first+last)*co;
          coor2:=d+t*(c+t*(b+a*t));
        end;
    end;

  function coor1(u:coeff):real;
    begin
      with u do
        coor1:=d+last*(c+last*(b+a*last));
    end;

  function Rvalue1(p:coeff;t:real):real;
    begin
      with p do
        Rvalue1:=2*b+t*6*a;
    end;

  var x1,x2,x3,x4,x5,x6,t2,t3:real;
      i,j:integer;
      pt : array [1..8] of real;
  begin
    t2:=t*t;
    t3:=t2*t;
    with obj^ do
      for i:=1 to segment do
        begin
u[i].a:=Hmx[int,i][1,1]*t3+Hmx[int,i][1,2]*t2+Hmx[int,i][1,3]*t+Hmx[int,i][1,4];
u[i].b:=Hmx[int,i][2,1]*t3+Hmx[int,i][2,2]*t2+Hmx[int,i][2,3]*t+Hmx[int,i][2,4];
u[i].c:=Hmx[int,i][3,1]*t3+Hmx[int,i][3,2]*t2+Hmx[int,i][3,3]*t+Hmx[int,i][3,4];
u[i].d:=Hmx[int,i][4,1]*t3+Hmx[int,i][4,2]*t2+Hmx[int,i][4,3]*t+Hmx[int,i][4,4];
v[i].a:=Hmy[int,i][1,1]*t3+Hmy[int,i][1,2]*t2+Hmy[int,i][1,3]*t+Hmy[int,i][1,4];
v[i].b:=Hmy[int,i][2,1]*t3+Hmy[int,i][2,2]*t2+Hmy[int,i][2,3]*t+Hmy[int,i][2,4];
v[i].c:=Hmy[int,i][3,1]*t3+Hmy[int,i][3,2]*t2+Hmy[int,i][3,3]*t+Hmy[int,i][3,4];
v[i].d:=Hmy[int,i][4,1]*t3+Hmy[int,i][4,2]*t2+Hmy[int,i][4,3]*t+Hmy[int,i][4,4];
          u[i].first:=0;
          u[i].last:=1;
          u[i].max:=plu[int][i].max+2;
          u[i].min:=plu[int][i].min-2;
          v[i].first:=0;
          v[i].last:=1;
          v[i].max:=plv[int][i].max+2;
          v[i].min:=plv[int][i].min-2;
        (*
          pt[1]:=coor(u[i]);
          pt[2]:=coor(v[i]);
          pt[3]:=coor2(u[i],0.13);
          pt[4]:=coor2(v[i],0.13);
          pt[5]:=coor2(u[i],0.9);
          pt[6]:=coor2(v[i],0.9);
          pt[7]:=coor1(u[i]);
          pt[8]:=coor1(v[i]);
```

```
          for j:=1 to 8 do
            write(debugf,pt[j]:1:3,' ');
          writeln(debugf);
        *)
        end;
end;

procedure initM;
  /*
     purpose : initialize the value of Mh and transpose(Mh)
  */
  begin
    M1[1,1]:=2;
    M1[1,2]:=-2;
    M1[1,3]:=1;
    M1[1,4]:=1;
    M1[2,1]:=-3;
    M1[2,2]:=3;
    M1[2,3]:=-2;
    M1[2,4]:=-1;
    M1[3,1]:=0;
    M1[3,2]:=0;
    M1[3,3]:=1;
    M1[3,4]:=0;
    M1[4,1]:=1;
    M1[4,2]:=0;
    M1[4,3]:=0;
    M1[4,4]:=0;

    M2[1,1]:=2;
    M2[1,2]:=-3;
    M2[1,3]:=0;
    M2[1,4]:=1;
    M2[2,1]:=-2;
    M2[2,2]:=3;
    M2[2,3]:=0;
    M2[2,4]:=0;
    M2[3,1]:=1;
    M2[3,2]:=-2;
    M2[3,3]:=1;
    M2[3,4]:=0;
    M2[4,1]:=1;
    M2[4,2]:=-1;
    M2[4,3]:=0;
    M2[4,4]:=0;
  end;

procedure getdistanoe{(u,v:sdirection;nu:integer;var dist:distarray)};
  var i:integer;
      u1,u2,v1,v2,sum:real;
  begin
    u2:=coor(u[1]);
    v2:=coor(v[1]);
    u1:=coor(u[nu]);
    v1:=coor(v[nu]);
```

```
    sum:=sqrt(sqr(u2-u1)+sqr(v2-v1));

    dist[nu]:=sum;
    for i:=2 to nu do
      begin
        u2:=coor(u[i]);
        u1:=coor(u[i-1]);
        v2:=coor(v[i]);
        v1:=coor(v[i-1]);
        dist[i-1]:=sqrt(sqr(u2-u1)+sqr(v2-v1));
        sum:=sum+dist[i-1];
      end;

    dist[0]:=0;

    for i:= 1 to nu do
      dist[i]:=dist[i]/sum+dist[i-1];
 (*
    for i:=0 to nu do
      writeln(debugf,dist[i]:3:3);
    writeln(debugf);
 *)
  end;

procedure getpath{(n1,n2:integer;H:Mh;dist:distarray;var path:sdirection)};
  /*
      purpose : get the motion path of every starting point of segment
  */

  var i,j:integer;
      s1,s2,s3:real;
  begin
(*  writeln(debugf,'   H ');
    for i:=1 to 4 do
      begin
        for j:=1 to 4 do
          write(debugf,H[i,j]:3:3,'  ');
        writeln(debugf);
      end;
    writeln(debugf);
*)
    for i:=n1+1 to n2-1 do
      begin
        s1:=dist[i-1]-dist[n1-1];
        s2:=s1*s1;
        s3:=s2*s1;
        path[i].a:=s3*H[1,1]+s2*H[2,1]+s1*H[3,1]+H[4,1];
        path[i].b:=s3*H[1,2]+s2*H[2,2]+s1*H[3,2]+H[4,2];
        path[i].c:=s3*H[1,3]+s2*H[2,3]+s1*H[3,3]+H[4,3];
        path[i].d:=s3*H[1,4]+s2*H[2,4]+s1*H[3,4]+H[4,4];
     (*
        with path[i] do
          writeln(debugf,s1:3:3,' : ',a:3:3,'  ',b:3:3,'  ',c:3:3,'  ',d:3:3);
     *)
      end;
  end;
```

```
  procedure delete{(name : string)};
  var i, j : integer;
  begin
    i := 1;
    while (table[i]^.id <> name) and (i < nextentry) do i := i + 1;

    if i = nextentry
      then writeln('     no object by that name exists          ')
      else begin
          for j := i to nextentry do table[j] := table[j+1];
          table [tablesize] := nil;
          nextentry := nextentry - 1;
        end;
  end;

procedure saving1{(n:integer;u,v:sdirection; var uu,vv:sdirection)};
  /*
      purpose : backup the equations of curve
  */
  var i:integer;
  begin
    for i:=1 to n do
      begin
        uu[i]:=u[i];
        vv[i]:=v[i];
      end;
  end;

procedure do34{(time:real;int:integer;var obj:objectptr)};
  /*
      purpose :  if time = 0 then the in-between figure is the curve of object1
  */

  var i:integer;
  begin
    with obj^ do
      for i:=1 to segment do
        begin
          if time=0 then
            begin
              u[i]:=p3u[int][i];
              v[i]:=p3v[int][i];
            end
          else begin
                 u[i]:=p4u[int][i];
                 v[i]:=p4v[int][i];
               end;
        end;
 end;


procedure plus{(nu:integer;pu,pv:coeff;var tu,tv:sdirection;var k:integer)};
  /*
     purpose : add more points to the object whose number of segments is less
  */
```

```
  var i:integer;
      time,time2:real;
  begin
    if nu < 0 then writeln(' error : plus. nu:',nu)
    else
      begin
        tu[k]:=pu;
        tv[k]:=pv;
        time:=1/(nu+1);
        time2:=time;
        for i:=1 to nu do
          begin
            tu[k+i]:=pu;
            tv[k+i]:=pv;
            tu[k+i].first:=time; tu[k+i-1].last:=time;
            tv[k+i].first:=time; tv[k+i-1].last:=time;
            time:=time+time2;
          end;
        k:=k+nu+1;
      end;
  end;

procedure more2{(d1,d2:distarray;n1,n2:integer;var pu,pv:sdirection)};
  var i,j,k,add,minus,sum:integer;
      tu,tv:sdirection;
      temp : array [1..70] of integer;
      ok : boolean;
  begin
    k:=1;j:=0;sum:=0;
    for i:=1 to n1 do
      begin
        add:=0;
        ok := true;
        while (j < n2) and ( ok ) do
          if (d2[j] < d1[i]) and (d2[j] >= d1[i-1]) then
            begin
              add:=add+1;
              j:=j+1;
            end
          else ok:=false;

        temp[i]:=add-1;
      end;

    for i:=1 to n1 do
      if temp[i] < 0 then
        begin
          ok:=true;
          j:=i-1;
          while (ok) and (j > 0) do
            if temp[j] > 0 then
              begin
                temp[j]:=temp[j]-1;
                ok:=false;
                temp[i]:=0;
              end
```

```
            else j:=j-1;
          if ok then
            begin
              j:=i+1;
              while (ok) and (j<= n1 ) do
                if temp[j] > 0 then
                  begin
                    temp[j]:=temp[j]-1;
                    ok:=false;
                    temp[i]:=0;
                  end
                else j:=j+1;
            end;
          if ok then writeln('It is impossible ');
        end;

    for i:=1 to n1 do
      plus(temp[i],pu[i],pv[i],tu,tv,k);

    for i:=1 to k-1 do
      begin
        pu[i]:=tu[i];
        pv[i]:=tv[i];
      end;
    if n2 <> (k-1) then
      writeln(' error : more2. n2 <> (k-1) ',n2,k-1)
    else writeln(' o.k : equal ');
  end;

procedure expanding2{(int:integer;var obj1,obj2:objectptr)};
  /*
      purpose : let the number of segments of obj1 and obj2 be equal
                by the algorithm msc's paper
  */
  var s1,s2:integer;
  begin
    writeln(debugf,'expanding2');
    saving(obj1,p1u[int],p1v[int]);
    saving(obj2,p2u[int],p2v[int]);
    s1:=obj1^.segment;
    s2:=obj2^.segment;
    getdistance(p1u[int],p1v[int],s1,dist3);
    getdistance(p2u[int],p2v[int],s2,dist4);
    if s1 < s2 then
      begin
        more2(dist3,dist4,s1,s2,p1u[int],p1v[int]);
        obj1^.segment:=s2;
      end
    else
      begin
        more2(dist4,dist3,s2,s1,p2u[int],p2v[int]);
        obj2^.segment:=s1;
      end;
  end;
```

```
program raytrace(input,output,debugf,pixels,fview,filmdata,filmdata2,translationf,

(* this main has:
 *                - YES minmax logic
 *
 *)

#include "pconst.h"
#include "const.h"
#include "ptypes.h"
#include "type.h"
#include "papertype.h"
#include "drttype.h"
#include "pvars.h"
#include "var.h"
#include "drtvar.h"
#include "papervar.h"
i4,i1,i2,i3,quickx,quicky,quickx2,quicky2 : integer;
temp1,temp2: array [1..10] of by4;
back : charcolor;
msc : boolean;
#include "pextprocs.h"
#include "proc.h"
#include "drt.h"
#include "paper.h"


begin
   initialize;
   new(aray);

   getangle(sinz,cosz,sinx,cosx);

   sinx := -sinx;
   sinz := -sinz;           (* for inverse rotations in translookpt *)

   painting := normal1;

   if paper=1 then
     for i2:=1 to papernumber do
       goahead(i2,table[papermatrix[i2].start]^.ptr,table[papermatrix[i2].ending]^.

   if motion=1 then
   for i2:=1 to motionnumber do
       begin
         for i3:=1 to 4 do
           for i4:=1 to 4 do
               with table[motionmatrix[i2]]^ do
                 begin
                   temp1[i2].a[i3,i4]:=ptr^.transfm^[i3,i4];
                   temp2[i2].a[i3,i4]:=ptr^.inverse^[i3,i4];
                 end;
         temp1[i2].sphereo:=table[motionmatrix[i2]]^.ptr^.sphereo;
       end;

      rgbtochars(backgrnd,back);
```

```
      quickx:=0;
      quickx2:=virthighx;
      quicky:=0;
      quicky2:=virthighy;

      for scanline := 1 to quicky do
        begin
        writeln('                          scanline=',scanline:5);
        writeln(debugf,' scanline=',scanline:5);
        for scancol := 1 to virthighx do
          if (masksize > 1) then
            virtual[scancol,scanline mod masksize +1]:=backgrnd
          else actual[scancol,scanline]:=back;
        if (masksize > 1) and (scanline mod masksize =0) then
          begin
            i4:=scanline div masksize;
            antialias(1,i4,acthighx,i4);
          end;
        end;

      for scanline := quicky+1 to quicky2 do
       begin
         writeln('                           scanline=',scanline:5);
         writeln(debugf,' scanline=',scanline:5);
         flush(debugf);

        for scancol := 1 to quickx do
          if masksize=1 then
            actual[scancol,scanline]:=back
          else
            virtual[scancol,scanline mod masksize +1]:=backgrnd;

        for scancol := quickx to quickx2 do
          begin
    (*      writeln(debugf,scancol);            *)

            xpt := (scancol+noise2[aindex]*aliasing) / (halfvirtualx*2.0 -1.0) * s
                   screensizex/2;
            zpt := (scanline+noise2[aindex]*aliasing) /(halfvirtualy*2.0 -1.0) * s
                   screensizey/2;
            ypt := - scrndistance;
     (*
            aindex:=aindex + 1;
            if aindex > maxindex then aindex:=1;
     *)
            translookpt(xpt,ypt,zpt,sinz,cosz,sinx,cosx);
            with aray^ do begin
               x0:=eyept.x;
               y0:=eyept.y;
               z0:=eyept.z;
               x1:=xpt;
               y1:=ypt;
               z1:=zpt;
             (*
               if dof=1 then
```

```
                depthoffield(x0,y0,z0,x1,y1,z1);
             *)
            end; (* with *)

            msc:=false;

            if motion=1 then
              begin
                i2:=1;
                while i2 <= motionnumber do
                  begin
motionblur(aray,i2,temp1[i2],temp2[i2],table[motionmatrix[i2]]^.ptr,msc);
                    i2:=i2+1;
                  end;
              end;
          (*
             if (paper=1) and (not msc) then acolor:=backgrnd
             else
           *)
              raytrace(aray,1.0,false,acolor,1);
             if masksize > 1 then
               virtual[scancol,scanline mod masksize +1 ] := acolor
             else
               begin
                 rgbtochars(acolor,threechars);
                 actual[scancol,scanline]:= threechars;
               end;

             painting := normal1;

          end;  (* for scancol *)

         for scancol := quickx2+1 to virthighx do
           if masksize=1 then
             actual[scancol,scanline]:=back
           else
             virtual[scancol,scanline mod masksize +1]:=backgrnd;

          if (masksize > 1) and (scanline mod masksize = 0) then
            begin
              i4:=scanline div masksize;
              antialias(1,i4,acthighx,i4);
            end;

       end;  (* for scanline *)

       for scanline := quicky2+1 to virthighy do
         begin
         writeln('                          scanline=',scanline:5);
         writeln(debugf,' scanline=',scanline:5);
         for scancol := 1 to virthighx do
           if (masksize > 1) then
             virtual[scancol,scanline mod masksize +1]:=backgrnd
           else actual[scancol,scanline]:=back;
         if (masksize > 1) and (scanline mod masksize =0) then
           begin
```

```
            i4:=scanline div masksize;
            antialias(1,i4,acthighx,i4);
          end;
        end;


   totheta := totheta + 2.0*pi / totalframes;
   innertheta := innertheta - 2.0*pi / totalframes;
   totaltime := totaltime + deltatime;

   writeln(debugf,'kfTIRcnt=',kfTIRcnt:9);
   writeln(debugf);
   flush(debugf);


   if not onetoone and
      not onto then
        begin
          outputframe;
        end
   else if onetoone        then
         outputframe
   else begin
        expandframe(virtlow, virtlow, virthighx, virthighy);
        outputframe;
   end;


   writeln(debugf,'FRAME NUMBER',framnum:4,'---DONE');
   writeln('Finis frame #',framnum:4);

end.
```