TR-88-035

軟狀物體之產生及其動畫效果

TR-88-025

論文名稱：軟狀物體之產生及其動畫效果　　頁數：82

校(院)所組別：淡江　大學　資訊工程　研究所

畢業時間及提要別：七十七學年度第二學期碩士學位論文提要。

研究生：曾念先　　指導教授：鄭國揚博士

論文提要內容：

本論文提出以電腦繪圖產生軟狀物體形狀的一種方法。根據Blinn(1982)及Wyvill et al.(1986)的研究結果顯示，若採用適當範圍的控制點代入一些事先訂定之位能函數，以表現函數等位面之外觀，即可產生軟狀物體之輪廓。這種觀念是用單一的控制點代表一個等位值的球體，若二個控制點相互接近進行結合時，則球體之位能起了變化，於是球體等位能的外形亦產生變化，因此改變控制點間的空間關係，可得到各種不同狀況的軟狀物體的外觀及變化上的控制。

按照他們的研究，當二個控制點互相接近，球體等位能外形的變化，只集中於受另一控制點影響而位能起了變化的部分，其它區域的位能由於沒有改變，仍然維持原來的形狀。等位能外形的改變表示體積的變化，也就是質量的改變，但質量須維持不變，不能任意增加或減少，故等位能外形的體積應維持不變。

軟狀物體受外力作用會產生形狀的變化，變化之際仍然維持體積的不變。但物體以多邊形網孔組成，形狀一旦改變，多邊形亦須修改。因此本論文由使用者定義外力及形狀變化的方向，以產生軟狀物體的輪廓，同時維持它的體積不變，以符合軟狀物體的變形。並提供一些實驗的例證，以顯示此種方法的可行性。

本實驗之環境係在E&S PS300及SUN MICRO設備的主機上發展，以及利用THREE-D和ANIMATOR套裝軟體來顯示物體的多邊形架構及著色後的畫面，進行軟狀物體產生之系統設計。

Title of thesis: Generating Soft Objects with Animation Effect Total pages : 82

Name of Institue: Graduate Institute of Information Engineering, Tamkang University

Graduate Date: 06/89 Degree Conferred: Master

Name of Student: Nien-San Shu 舒念先 Advisor: K.Y. Cheng

Abstract :

In this thesis, a method of generating soft objects in computer graphics is presented. The method is based on the idea of Blinn (1982) and Wyvill et al. (1986). According to their research results, suitable choice of a field function and a set of key points with different radius of influence can be used to simulate soft objects. Soft object shape is the iso-surface, i.e., every point on the surface possesses the same field value. Under this point of view, the shape of an iso-surface constructed by a key point is a sphere. When two key points come closer, each of their shapes will be changed when their radius of influence intersects with each other. Therefore, number of key points and their spatial relations control the shape of the soft objects. That is, when these spatial relations are altered with time, then the shapes are also changed softly from time to time.

However, Blinn and Wyvill considered the change of shpere-form iso-surface only on the area with field values which are affected by another key point when two key points come closer. The shape of other area remains unchanged. But in real situation, the shape of iso-surface changes at all areas, the only thing that remains unchanged is the total volume. Thus we conclude that the genèration of iso-surface should put the condition that the final stablized surface creates no volume change in the soft object representation.

Our method is to let users to alter soft-object shapes in response to external forces, while keep total volume unchanged. We consider polyhedra objects, when they are depressed or streched the shape of polygons will be changed with the condition that the wrapped volume remains unchanged. Some experimental results will be illustrated to show the feaseability of this approach.

Our experiment environment includes E & S PS300 and SUN MICRO systems and two software packages : THREE-D and ANIMATOR.

章節目錄

頁次

圖 表 目 錄

頁 次

第一章　緒論

1.1 介紹軟狀物體

日常生活存在一些與水有關的自然現象，例如昆蟲奔跑於水面，不會下沈。水面原先保持平坦，當昆蟲經過之時，水面不再保持平坦；昆蟲離開後，水面又恢復平坦，水面並非永久地保持固定的形狀，它會受環境的影響而不斷地變化。

因此，軟體物體為受環境影響而產生形狀變化的物體。這類物體經常可見，例如水、蝴蝶幼蟲、坐褥等，它們的形狀會經常地改變。

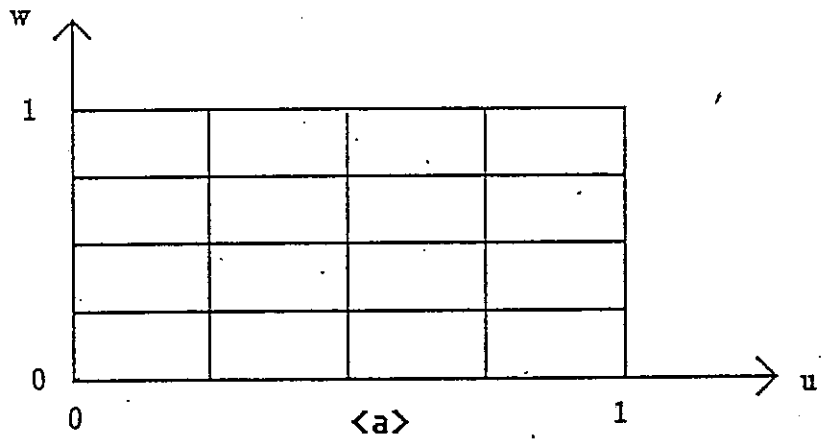
這種形狀變化的效果，可應用於卡通。在卡通影片，經常可看到誇張的畫面，例如鐵鎚敲打釘子，釘子依然不動，鐵鎚的握柄反而彎曲。這類的形狀變化，只是模擬誇張的動作，達到娛樂的目的而已。

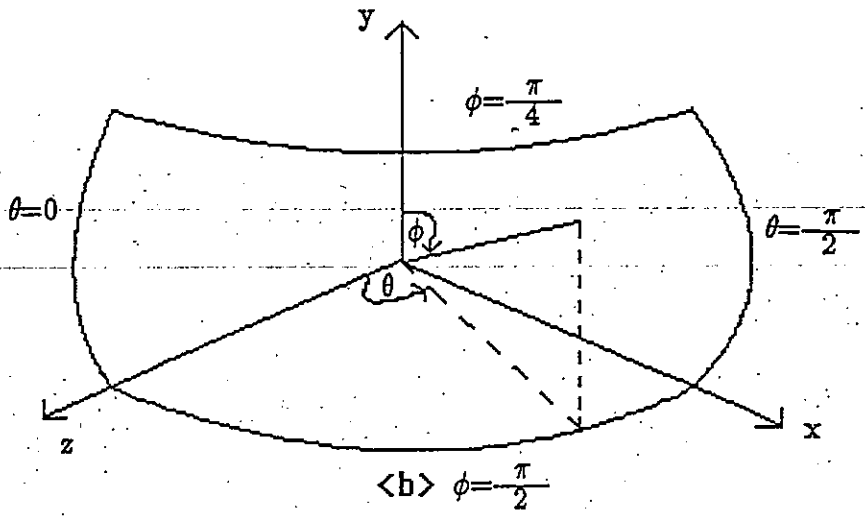
1.2 在電腦繪圖及電腦動畫的意義

本節討論軟狀物體在電腦繪圖及電腦動畫上的應用。電腦繪圖方面，面述紋理 (Texture)和視線探索法 (Ray

Tracing)的含意，接著說明軟體物體在它們的應用；動畫方面，先說明'動畫'這個名詞的含意，再強調軟狀物體的形狀改變。

由於移去觀察者看不見的線段和曲面，計算觀察者看見區域的明暗程度，電腦產生的畫像(image) 與實際的狀況，非常類似。然而在許多情況下，可立即判斷畫像是否由電腦產生，主要的原因在於用電腦產生的物體，每個面都非常光滑，真實生活的例子並非如此。例如用金屬或塑膠製造的杯子，表面非常光滑，故電腦產生的金屬或塑膠杯子的畫像，非常接近真實的杯子；至於橘子和人類皮膚的電腦畫像，看起來並非很真實，因為它們的表面有許多紋路。幾乎所有物體的表面，佈滿著眼睛能夠看得清楚的細小結構，這些結構稱為紋理，它提供物體表面的自然性質。





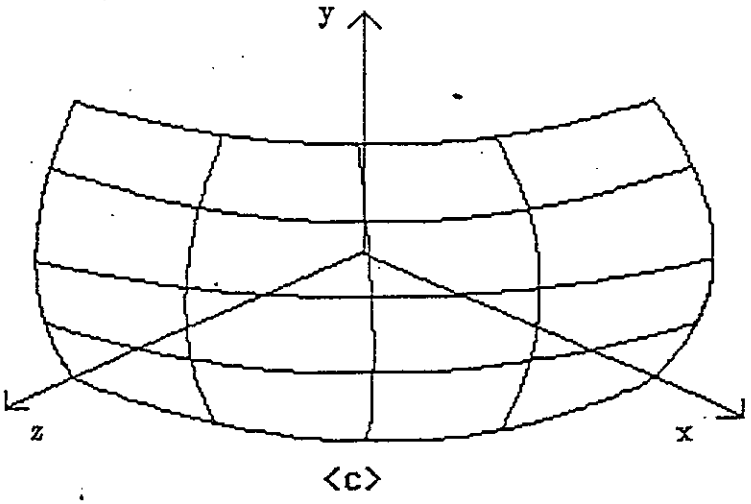


圖 1-1 Catmull 的 mapping

為了表現物體表面的自然特性，Catmull (1975)將圖案貼在曲面上，如圖 1-1，(a)為圖案，(b)為物體的表面，圖案貼上後成為 (c)。圖案及曲面均有各自的座標系統。圖案貼至曲面的紋理，基本上是屬於座標轉換。但此種方法合成後的曲面仍是光滑。

另一類紋理是藉著干擾法向量，達到表面凹凸不平的效果。規劃軟狀物體，要用位能函數(在第三章詳細討論)，藉著它的特性，能做流質的紋理。球體投影至平面，可視為圓，圓內的任一點A與圓心，再建立一個圓。由於A點位於新的圓，容易求得A點的單位切向量$\vec{D}$及單位法向量$\vec{N}$。再求得A點的位能 c(r)，r為A點至圓心的距離。利用位能 c(r) 改變$\vec{N}$及$\vec{D}$成為

$$\vec{N'} = \vec{N} * c(r) \qquad 0 \le c(r) \le 1$$

$$\vec{D'} = \vec{D} * (1 - c(r)) \qquad 0 \le c(r) \le 1$$

再由$\vec{N'}$與$\vec{D'}$的合向量做為A點的法向量。這種方式的干擾法向量，王興華(1989)設計冰淇淋的紋理。

視線探索法是模擬光線前進的方式，以產生反射、折射和陰影等效果。直覺上，光線從光源射出，依直線方式

前進，照射至物體，分成折射及反射繼續前進，直到觀察者為止。若要模擬上述的過程，會非常不恰當，主要的原因在於光線從光源出發，只有少部份的光線到達觀察者的眼睛，對每一個發射的光線計算它的反射，折射效果，卻往往發現觀察者不會看到它。Appel(1968) 採取相反的過程，光線由觀察者的眼睛出發，沿著使用者與物體之間的直線前進，照至物體再分成反射及折射二部分，繼續前進直至光源為止，如圖 1-2。

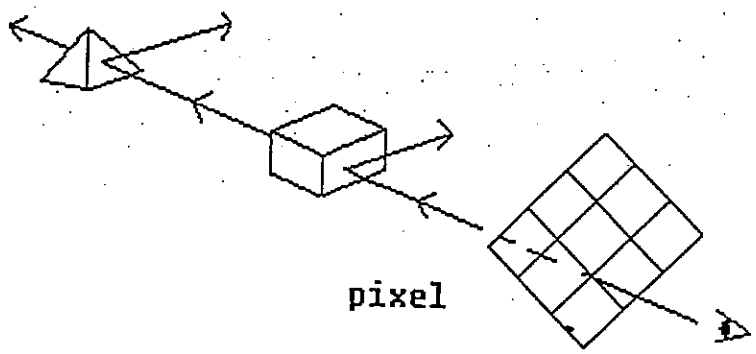


圖 1-2　視線探索法

視線探索法雖可產生包含反射、折射和陰影的高品質畫像，但它主要的一個缺點在於對每個位置檢查光線變化

的情形，耗費許多的時間才能產生畫像。因此有許多的改進技巧，其中一類著重於減少不必要的計算，先將物體完全地置於範圍空間 (bounding volume)。若光線並未通過範圍空間，明顯地，光線不會照在物體的表面；否則檢驗光線是否會照在物體表面。這種技巧只適用於檢查光線是否會通過範圍空間花費的計算，遠小於檢查光線是否會照在物體所花費的計算，故範圍空間通常都是簡單的幾何物體，如球體、正方體、長方體等。軟狀物體的外形，是由球體結合而成，可以找到一個簡單的幾何物體充當範圍空間，它不僅能完整地包含整個軟狀物體，同時範圍空間內大部份的區域都被軟狀物體佔據，造成進入範圍空間的大部分光線都要計算它的反射、折射效果。故軟狀物體非常適合採用範圍空間的方法，減少視線探索法花費的時間。圖1-3 為採用矩形的範圍空間。

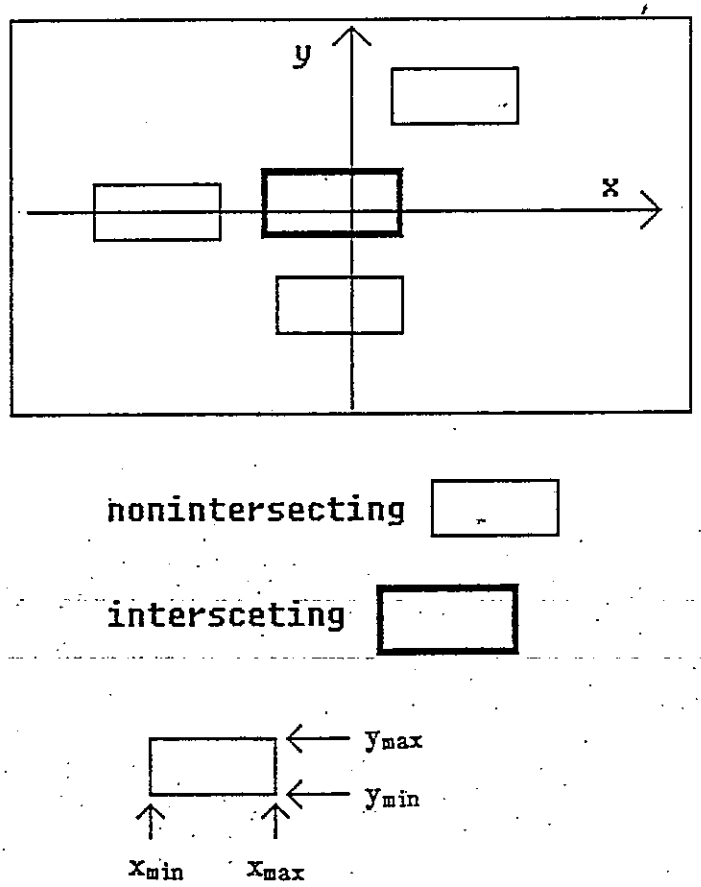


圖1-3　矩形的範圍空間

'動畫'這個名詞有許多不同的解釋，Halas和Manvell (1968)認為動畫的本質在於移動 ('Movement is the essence of animation')，比較詳盡的解釋為：

1.一連串獨立的圖形(drawing)拍攝於電影膠片(film)，膠片的每一個畫面(frame) 只容納一個圖形，再以一定的速率(通常每秒24張畫面)放映膠片，會產生物體

移動的感覺(illusion of movement)，故動畫是種技巧，它會產生移動的感覺。

2.動畫是指將物體拍攝至電影膠片的畫面之過程，每個畫面與它之前的畫面，物體會稍微不同。

Magnenat-Thalmann (1985)指出將動畫描述成移動是不恰當，因為在動畫中，物體不一定要移動，例如顏色和光線的變化，會產生動畫的效果，但物體的位置並未移動，故認為傳統的動畫，主要是為了產生二維的卡通。在使用電腦後，電腦動畫這個名詞變得更不準確，因為電腦可扮演不同的角色，包括

1.圖形的製作

2.產生動作

3.著色

4.模擬攝影機的動作

5.圖形拍攝至膠片

電腦動畫利用中插法(in-between)產生物體的動作。描述二張主要的畫面(key-frame)，藉著計算相關點間的線性關係，如圖 1-4，求得這二張主要畫面之間的畫面，可達到運動的效果。但中插法的主要問題在於

1.每點都以直線的方式移動，二張主畫面所含的點數須

相同，才能做如圖1-4 的內差。

2.若動作不只由二張的主要畫面構成，在這些主畫面會有不連續的情形發生。

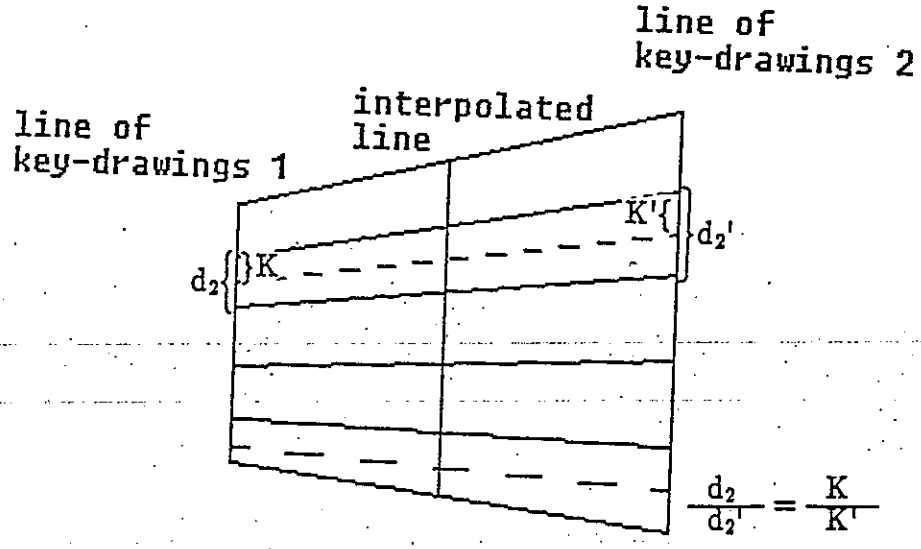


圖 1-4　中插法

雖然中插法有一些問題，有的動作它仍然無法達成，例如水滴的聚合過程，外形會隨水滴之間的距離而有不同的變化，使用中插法很難做到；再從圖1-4 分析，若有一物體由二部分組成，在第一張的主畫面，這二個部分是相連的，在第二張主畫面，這二個部分是分開的，中插法就無法處理這種情形。軟狀物體包含許多控制點，每個控制點會產生一個球體。軟狀物體的外形，是由球體結合而成

。對軟狀物體而言，只要將控制點放得近一點，即可產生這二部分是相連的，控制點的距離遠一點，則此二部分是分開的。

若軟狀物體的控制點，加上適當的限制，可避免中插法的缺點。選擇一個曲線，如B-軟樑曲線，由它上面的點表示軟狀物體的控制點，造成每點並非直線運動。假如曲線本身已滿足連續的條件，在不同時間取出適當的控制點，則不會有畫面不連續的情形。

1.3 各章節的內容簡介

第二章：規範本論文的重點

第三章：位能函數的決定方式及維持體積不變的方法。

第四章：物體外形的建立，以便顯示於螢幕。

第五章：說明實驗的結果。

第六章：結論與展望。

第二章　背景分析

軟狀物體的外形，會受環境影響而產生變化。2.1 節分析二顆水滴相互接近時，外形的變化過程，得到水滴的總體積應維持不變。2.2 節使用位能函數表示水滴的外形變化。2.3 節節述作畫的考慮。2.4 節節述系統的架構。2.5 節節述本論文的研究範圍。

2.1 水滴形狀的改變

物質由分子組成，分子間除了彼此碰撞外，還有吸引力，此吸引力只有在相當的範圍才變得顯著。若以一分子為中心，吸引力所能作用之最大距離為半徑，做一球體，則球體內的分子與位於球心的分子，均有分子的作用力存在，故稱此球體為作用範圍。

在液體裡，分子之間的距離短，必須考慮分子之間的作用力，它對液體內部分子及表面分子有不同程度的影響。液體內部的分子，受到來自四面八方的吸引力，而分子吸引力平均是無方向，因此淨力為零，所以液體內部的分子，其動能不受分子吸引力的影響。但在液體表面的分子，它在液面上下兩方所受的力並不相等，在液面上的部分

受已氣化的分子吸引，在液面下的部分仍受液內分子的吸引，因為氣化的分子密度較小而且與液面分子的距離較大，故在液面的分子受一個往液內拉的力量，如圖2-1 所示，分子A為內部分子，作用於分子A的合力為零，分子B為液面分子，作用於分子B的合力為T。

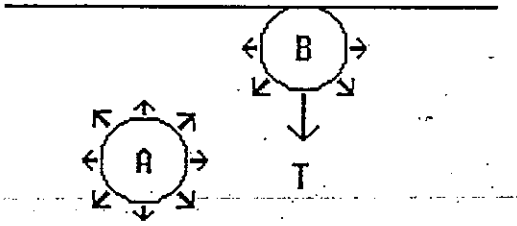


圖2-1　液體內部分子及表面分子受力情形

由於液面分子受到一個往液面內拉的力量，可以認為液內分子在液面造成一個引力場。因此當液內分子到達表面時，受此拉力影響，其速率降低，動能減少，以能量的觀點來看，其位能增加。因此，若液體表面愈大，則在表面的分子愈多，故其總位能愈大。但在力學裡，質點系統在穩定平衡時，一定維持最小的位能。所以液體若在平衡狀態時，則必使其面積最小，以保持最少的位能，也就是說，它有一種縮小表面積的傾向，以保持其平衡狀態。這就是表面張力形成的原因。

由於表面張力的作用，水滴會形成最小面積的球體。當二顆球體狀的水滴相互靠近時，由於分子間的作用力影響，它的形狀會變化。圖2-2 為水滴的結合過程。考慮不同水滴上的A，B二點，圖2-2(a)，當水滴相距甚遠，A和B只各自受向水滴內部吸引力的。縮短水滴之間距離，此時A和B二點不只受向水滴內部的吸引力，它們彼此間的吸引力亦須考慮。A，B之間的吸引力會使水滴的外形做細微的變化，如圖2-2(b)。若繼續縮短水滴間距離，外形變化更大，如圖2-2(c)，再繼續靠近，則會重合在一起。

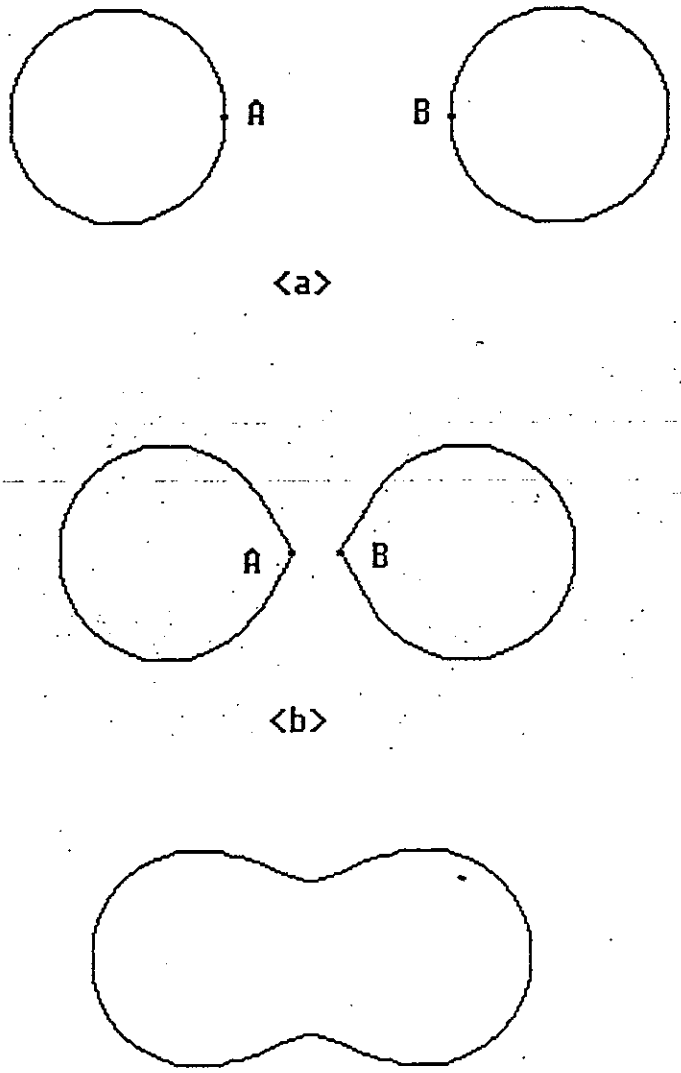


圖 2-2 水滴結合過程的側面圖

再從能量的觀點，分析水滴的結合過程。水滴表面的分子具有相同的位能，圖2-2(b)的凸起部分及圖2-2(c)的凹下部分，它們的表面分子亦具有相同的位能。當另一水滴接近時，改變水滴原先的位能分佈，造成水滴形狀的變化。

綜合上述的分析過程，仍然沒有周詳。當水滴的形狀呈球體時，表面分子均受一個朝球心的作用力吸引；當水滴外形不再呈球體時，表面分子仍然受一個向內吸引的力，但它指的位置，不再是球心。這樣所造成的形狀變化，不只包括受其它水滴影響的部分，還包含未受其它水滴影響的部分。當前者凸起時，合力的位置會偏向凸起的區域，後者會稍微凹下。二者變化的範圍亦不同，前者窄而尖，後者寬而扁，雖然變化的範圍有差別，它們的體積應該相同，因為水滴的體積正比於它的質量，體積的改變表示質量的變化，質量不能任意的增加或消失，以符合物質不滅定律，故體積應維持不變。

2.2 數學背景

本節分別站在表面張力及能量的觀點，討論單一水滴的形狀及水滴相互接近時，形狀的變化。雖然二者都可表

示變化的過程，但能量的觀點會比較容易。

表面張力使液面趨向最小的面積。在液面上任取 S 長度之線段，則表面張力垂直於 S 線段。在 S 上取微小長度 ds，且設作用於 ds 之力為 dF，則表面張力為

$$\sigma = \frac{dF}{ds}$$

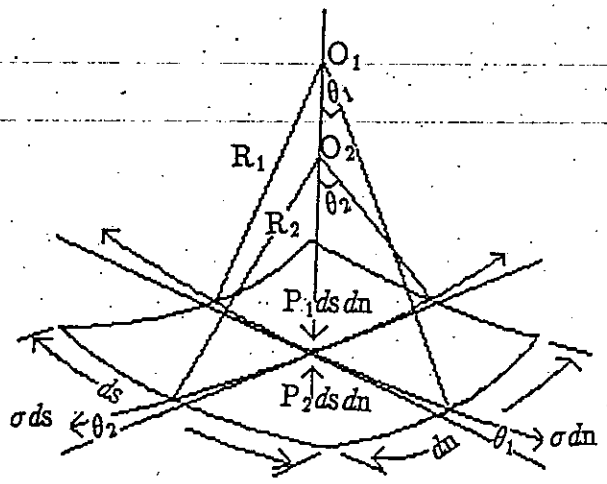


圖 2-3 兩向曲率之曲面

圖 2-3 為球體表面一塊微小面積受表面張力作用的曲面。由於表張力的結果，曲面內外存在壓力差，依力學的平衡原則：合力為零，得 $P_1 ds dn - P_2 ds dn - 2\sigma dn \sin\theta_1 - 2\sigma ds \sin\theta_2 = 0$

除以 $ds dn$，$$P_1 - P_2 = 2\sigma\left(\frac{\sin\theta_1}{ds} + \frac{\sin\theta_2}{dn}\right)$$

因 θ_1 及 θ_2 甚小， $\sin\theta_1 \cong \theta_1 = \frac{1}{2}\frac{ds}{R_1}$ $\sin\theta_2 \cong \theta_2 = \frac{1}{2}\frac{dn}{R_2}$

得 $$\Delta P = P_1 - P_2 = \sigma(\frac{1}{R_1} + \frac{1}{R_2})$$

其中： σ ＝接觸面之表面張力；

R_1 , R_2 ＝接觸面之曲率半徑；

P_1 , P_2 ＝接觸面之內外側的壓力強度，

ΔP 為二者的差值；

由於壓力差只作用於液體的表面分子，這一層非常薄，故可視為

$$R = R_1 \cong R_2$$

得

$$\Delta P = \frac{2\sigma}{R}$$

$$R = \frac{2\sigma}{\Delta P}$$

站在表面張力的觀點，球體狀的水滴，它的半徑為表面張力與球體內外壓力差的比值。

站在能量的觀點，水滴內部的分子對表面的分子形成一個引力場，因此考慮行星或衛星的運動。

質量為m之物體(如行星或衛星)繞質量為M之巨大物

體(如太陽或地球)的運動。設M靜止於慣性參考系中，物體m在圓形軌道上繞M運動，此系統的位能為$\acute{U}(r) = -\frac{GMm}{r}$ r為圓形軌道半徑，G為萬有引力係數。

由於水滴表面的分子具有相同的位能，它們與球心的距離都相同，故認為水滴的位能分佈，類似行星或衛星運動的位能，只與距離有關。

當二顆水滴相互靠近，外形產生變化。若用表面張力描述其過程，要說明表面積的變化，但這是個繁雜的工作。若用位能的觀點，水滴接近時，引起位能分佈的變化，位能只與水滴的距離有關，只要求得個別水滴的位能影響，總合這些位能影響，就可得到真正的位能。再判斷此位能是否為水滴表面分子的位能，即可決定形狀。

因此採用能量的觀點，說明水滴形狀的變化。每個水滴可視為一個球體或一個控制點，它的屬性包括球心位置及作用範圍；水滴的質量集中於球心位置，作用範圍的大小表示水滴質量的大小。水滴表面的分子具有相同的位能，因此要選擇特定的值(設為magic)表示此位能。圖2-4表示控制點的屬性。圖2-5 為相同作用範圍的二個控制點相互接近，以模擬水滴的變形。圖2-6 表示不同作用範圍的形狀變化。

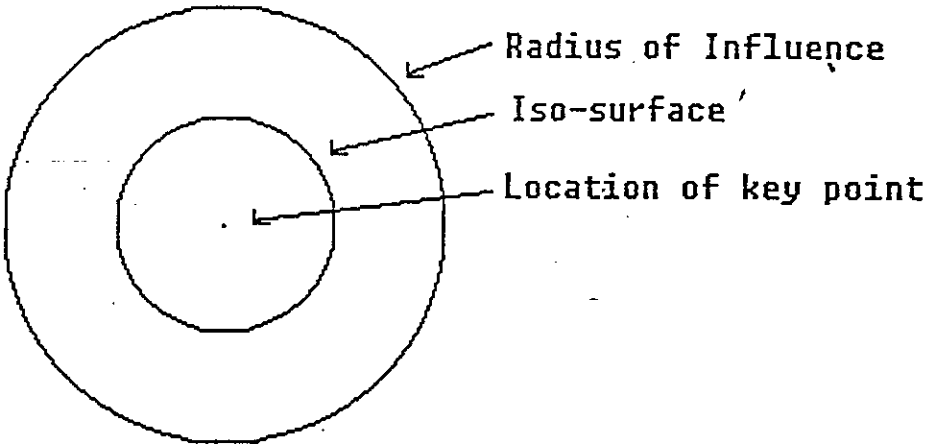
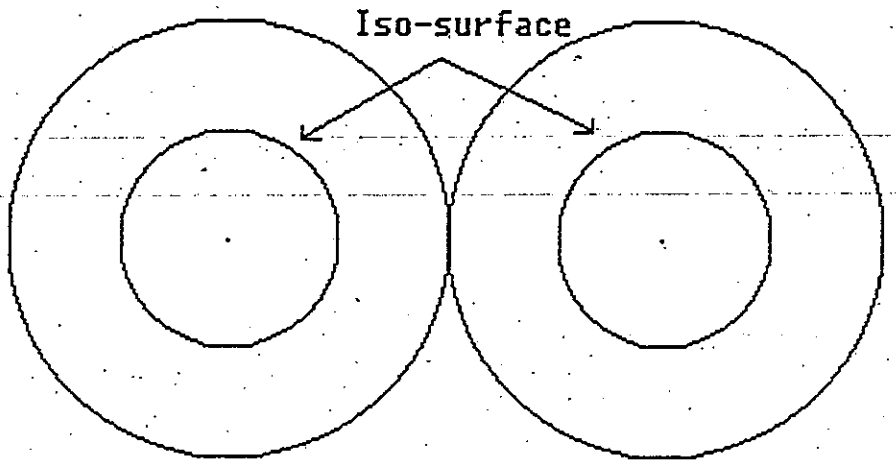
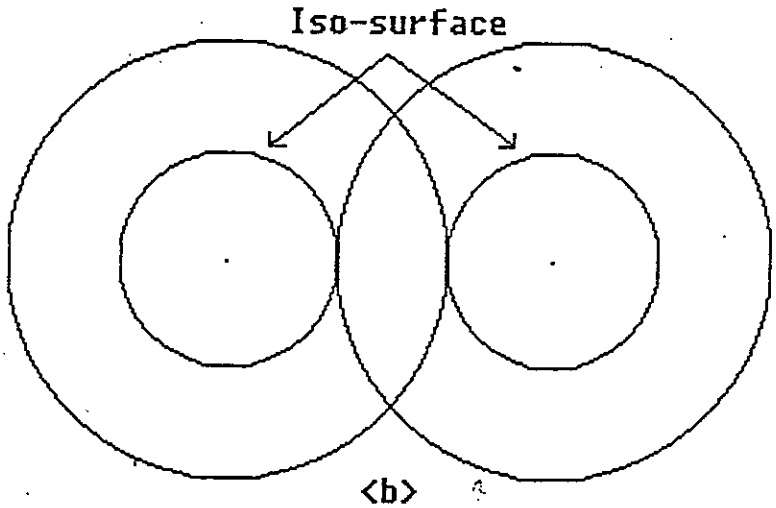


圖 2-4 控制點屬性

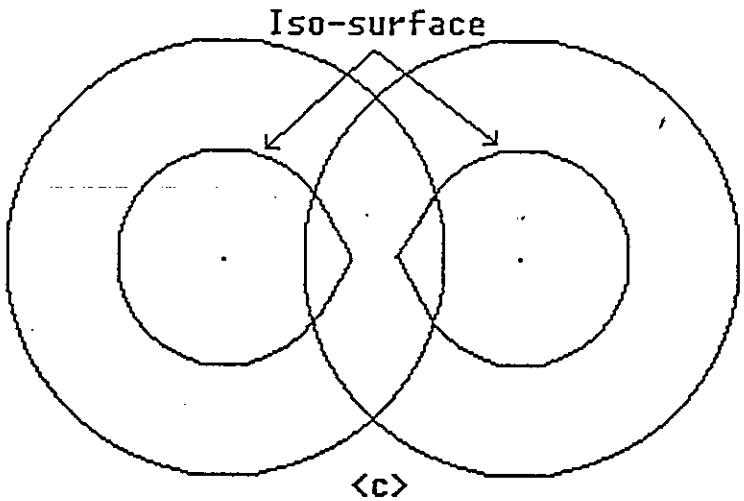


<a>

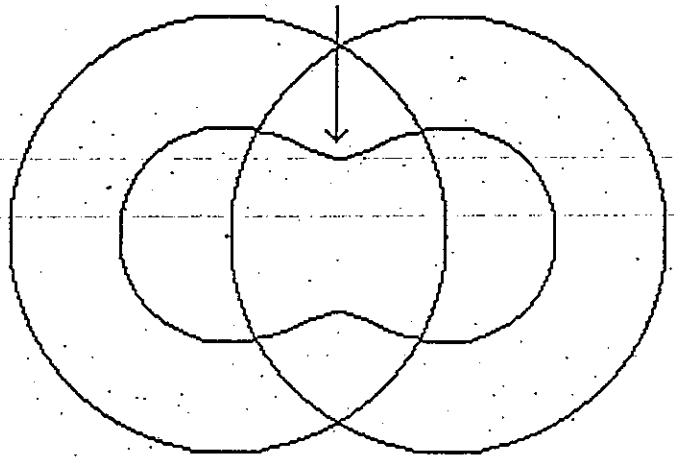


<b>

圖 2-5 模擬水滴變形



<c>



<d>

圖 2-5 模擬水滴變形

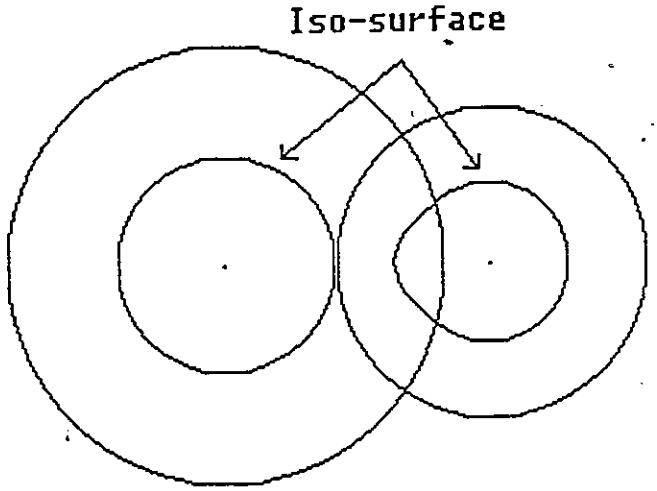


圖 2-6 不同作用範圍的形狀變化

上述的過程，雖然訂定位能的分佈，模擬水滴外形的變化，但它的體積增加，不符合體積保持不變的要求，因此還要維持體積不變，詳細的做法在3.3節說明。

2.3 作畫考慮的因素

畫家作畫時，站在特定的角度觀察物體，先描其外形，再塗上顏色。若用電腦模擬這種過程，首先須建立物體的外形，再從特定的角度處理此物體，最後加上紋理及視線探索法的效果，對於看不到的面，就不必它求的紋理及視線探索法的效果。

模擬畫家作畫的過程，第一步要建立物體的外形，雖然畫家從某個角度觀察物體，有許多面看不見，但使用電腦規劃物體外形，還是要建立物體的完整輪廓，再對物體作平移，旋轉，顯示從某個角度所觀察的輪廓。

由於本論文是研究塑造物體的外形，故作畫方面的考慮，偏重於物體外形的建立。

2.1節及2.2節對球體狀的水滴做討論，基本上，軟狀物體的外形都可用球體拼湊而成，藉著球體間不同的結合程度，如圖2-5及圖2-6，可產生變化多端的外形。若再移動球體(控制點)的位置，即可達到形狀的變化。

建立物體外形的步驟，相當於選擇控制點的位置，為了容易分辨控制點產生的外形，最好將產生相同部分(component)的控制點放在一起。例如物體由A和B二部分組成，A的形狀由C及D二個控制點產生；B的形狀由E、F及G三個控制點組成，整個物體可視為圖2-7 的樹形結構。

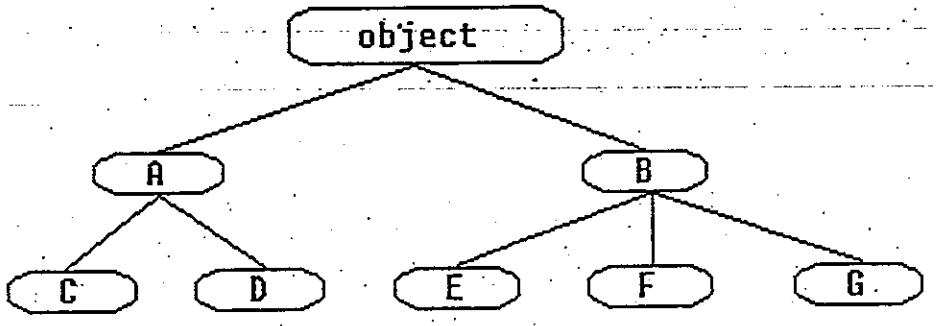


圖2-7 控制點的空間關係

2.4 系統架構

依照畫家做畫的過程，設計系統的架構。第一部分為界面，以畫家能懂的視覺符號(Icon)指定控制點的位置。根據控制點的空間關係，在第二部分產生多邊形網孔，以便在第三部分使用套裝軟體THREE-D,藉著旋鈕(dial)的調

整，從不同的角度觀看物體。若不滿意物體的形狀，跳回至第一部分，重新指定控制的空間關係。若滿意，則至第四部分使用套裝軟體ANIMATOR，產生單一的畫面。重複上述的步驟，產生許多畫面後，就可做成動畫。

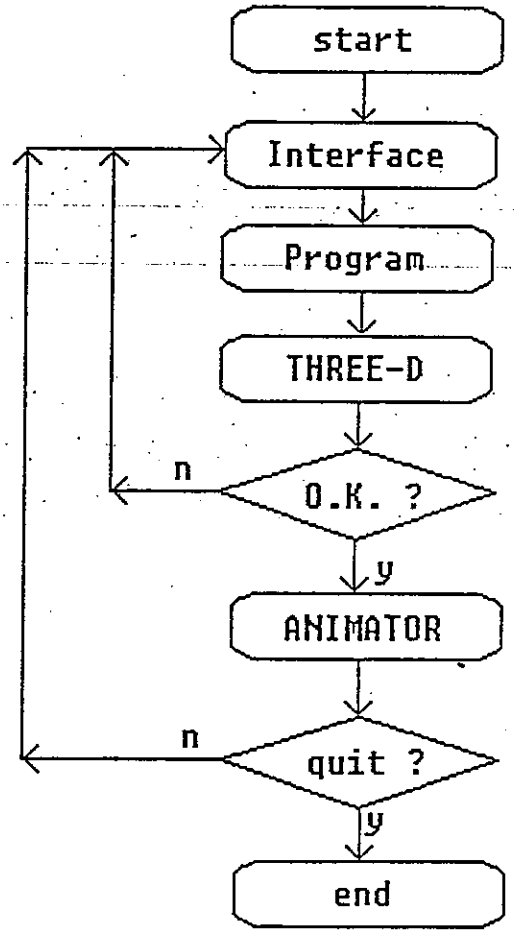


圖2-8 系統架構

2.5 本論文探討的範圍

論文的範圍，在系統架構方面，產生物體外形的多邊形網孔；在形狀變化方面，著重體積的不變；在動畫方面，試著產生誇張的效果，但只限於形狀變化的範圍。

圖2-8 的系統架構包括四部分，第三及第四部分是套裝軟體，本論文著重第二部分的程式設計，產生多邊形網孔，以便使用套裝軟體，將物體顯示於螢幕。第一部份的界面並沒有寫好，由於它會建立控制點，供第二部份使用，故第二部分的程式，暫時從檔案讀入控制點，等界面建立後，形成完整的系統架構。

二顆水滴互相接近而產生形狀的變化，在這過程，只針對體積不變討論。事實上，還受許多物理因素的限制，例如溫度、壓力、附著力、黏度等，它們都未列入考慮的範圍，因此無法模擬許多與這些因素有關的形狀變化。

軟狀物體的形狀會受環境的影響而變化，此種效果若應用於非軟狀物體，例如鐵鎚敲打釘子，釘子不動，鐵鎚握柄彎曲，這雖不符合真實的情形，但可用於卡通，產生誇張的效果。

第三章　位能函數

本章討論位能函數的設定方式，根據位能函數求得位能後，使用第四章的方法，可以決定軟狀物體的輪廓。 3.1節分析 Blinn及 Wyvill et al.採用的方法。根據他們的研究結果，加上軟狀物體的特性，在 3.2節設定本論文採用的位能函數。依照位能函數計算位能時，由於其它控制點的影響，位能的分布發生變化，造成體積的變化，故在 3.3節敘述維持體積不變的方法。

3.1 Blinn及 Wyvill et al.採用的位能函數

Blinn (1982)描述一種顯示分子構造模型的方法，它是個模擬電子密度 (election density) 的數學模式。量子化學 (quantum chemistry)將原子內的電子表示為空間上的密度函數。例如單一氫原子的密度函數為

$$D(x, y, z) = \exp(-ar)$$

$$r = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2}$$

(x_1, y_1, z_1) = location of atom

若是一群原子，它們的密度函數為個別原子對密度函數的貢獻量 (contribution) 的總合，表示成數學式子為：

$$D(x, y, z) = \sum_i b_i \exp(-a_i r_i)$$

r_i = distance between location (x, y, z) and atom i

按照 Blinn的方法，曲面上的點均滿足函數：

$$F(x, y, z) = 0$$

，加上電子的密度函數及曲面上的每點都有某類起始值 (some threshold amount)，故曲面函數為：

$$F(x, y, z) = D(x, y, z) - T$$

，所有在曲面內的點，其電子密度均大於T。為了計算方便，密度函數決定函數改為：

$$D(x, y, z) = \sum_i b_i \exp(-a_i r_i^2)$$

，這不必花時處理根號的根算。上式的指數項為位於 r_i，高度 b_i，標準差 a_i 的 Gaussian bump，藉著調整 a_i 及 b_i，同一個原子可達到許多不同的效果，這些效果會改變物體的細微程度 (blobbiness)。Blinn使用原子的半徑 R_i 及改變物體細微狀況的 B_i，估計單一原子的密度貢獻為

$$D_i(x, y, z) = T \exp\left(\frac{B_i}{R_i^2} r_i^2 - B_i\right)$$

Blinn 的方法值得效法的地方在於 (1)曲面內部的點，其電子密度均大於一個特定的值。(2)曲面的形成是由數值累加而成，原子聚集較密，位置 (x，y，z)的電子密度較大，若它大於 (1)所敘述的特定值，則此位置會在曲面內。若原子較少，電子密度小於 (1)所敘述特定值，則此位置會在曲面外。但他的方法非常耗時，因為採用指數項，對每個位置的密度函數的貢獻量都是大等或等於零，在計算電子的密度時，須找出所有原子的貢獻量，這樣需要的時間較長。

Blinn 的方法與以位能函數 (field function) 規劃物體外形的方法，基本上是相同的。位能函數是一個定義於空間的數學函數，在 Blinn的方法，位能函能是電子的

密度函數。Wyvill et al. (1986b)擴張 Blinn的做法，視每個原子為一個控制點，同時設定一個位能函數取代電子的密度函數。

Wyvill et al.希望位能函數是一個連續函數，每個控制點都有各自的作用範圍，作用範圍內的點，其位能不為零，作用範圍外的點，其位能為零。位置 (x，y，z)的位能，等於所有控制點對此位置位能貢獻量的總合。控制點對其所在位置的位能貢獻最大，設為 1 。在作用範圍內，離控制點愈遠，位能的貢獻愈小，當距離為作用範圍時，位能的貢獻量為零。位能貢獻量從 1 開始，逐漸減少至零，由於位能函數是連續函數，位能貢獻量減少的過程會非常平滑 (smooth) 。

按照上列要求，若用 c(r) 表示位能函數， r為欲計算位能的位置至控制點的距離， R為控制點的作用範圍，位能函數在 r=0及 r=R須滿足

$$c(0.0) = 1.0 \qquad c'(0.0) = 0.0$$
$$c(R) = 0.0 \qquad c'(R) = 0.0$$

c' 為 c對 r的微分，根據上述條件可定義唯一的函數

$$c(r) = 2(\frac{r}{R})^3 - 3(\frac{r}{R})^2 + 1$$

，但計算 r時，要處理根號運算，比較耗時，故改成

$$c(r) = a(\frac{r}{R})^6 + b(\frac{r}{R})^4 + c(\frac{r}{R})^2 + 1$$

，再由

$$c'(0.0) = 0.0$$

$$c'(1.0) = 0.0$$

$$c(0.5) = 0.5$$

，代入上式求得

$$a = -0.444444$$

$$b = 1.888889$$

$$c = -2.444444$$

參考 Blinn及 Wyvill et al. 的做法，設定位能函數時，每個控制點都有作用範圍，控制點對作用範圍外的位置，位能的貢獻量為零，作用範圍內的位置，位能的貢獻量不為零；位能函數須是連續函數，而且沒有根號的運算。至於產生的曲面，曲面內部的位置，其位能均須大於一個特定的值；曲面表面位置的位能，均等於此特定的值；

曲面外部位置的位能，小於此特定的值。

3.2 本論文使用的位能函數

在介紹軟狀物體時，根據液體分子的受力情形，認為液體內部分子對液面造成一個引力場，液面分子受到一個朝液體內部吸引的力，這種情形非常類似行星或衛星的運動，故依照此類運動來設定位能函數。

質量為m之物體（如行星或衛星）繞質量M之巨大物體（如太陽或地球）的運動，設M靜止於慣性參考系統中，物體m在圓形軌道上繞M運動，此系統的位能為

$$U(r) = -\frac{GMm}{r}$$

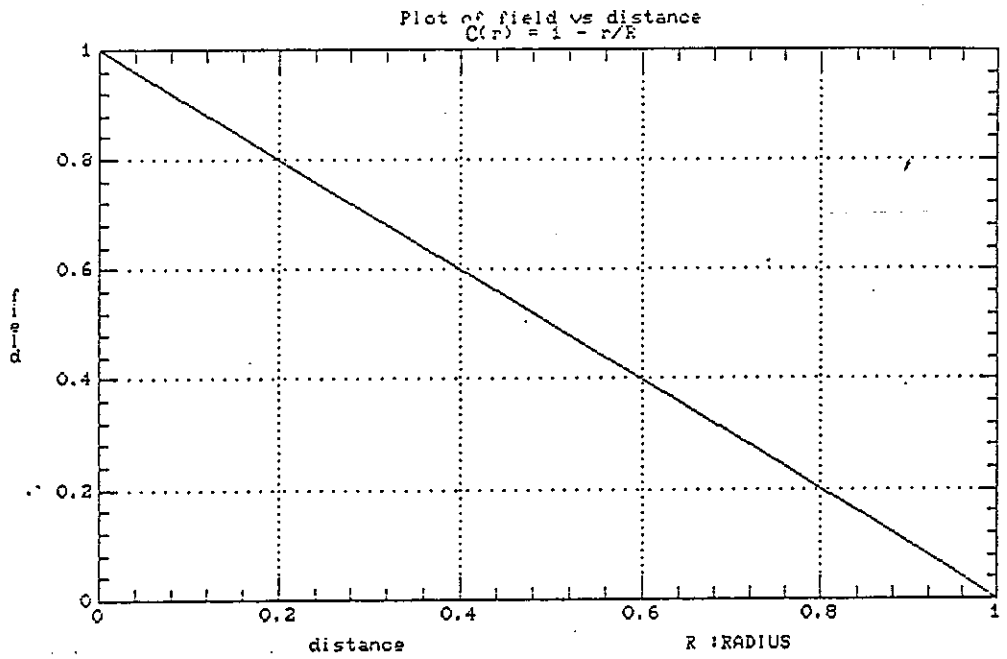
，r為圓形軌道半徑，G為萬有引力係數。由此式知，位能的大小與物體間的距離有關，隨著距離的增加，位能慢慢地變小，並非急速地變小。

軟狀物體的每個控制點都有作用範圍，作用範圍外的位置，控制點對它的位能貢獻為零，作用範圍內的位置，位能貢獻量不為零，位能值均在零與1之間；行星或衛星運動的位能，只與距離有關。根據此二觀點設定的位能函

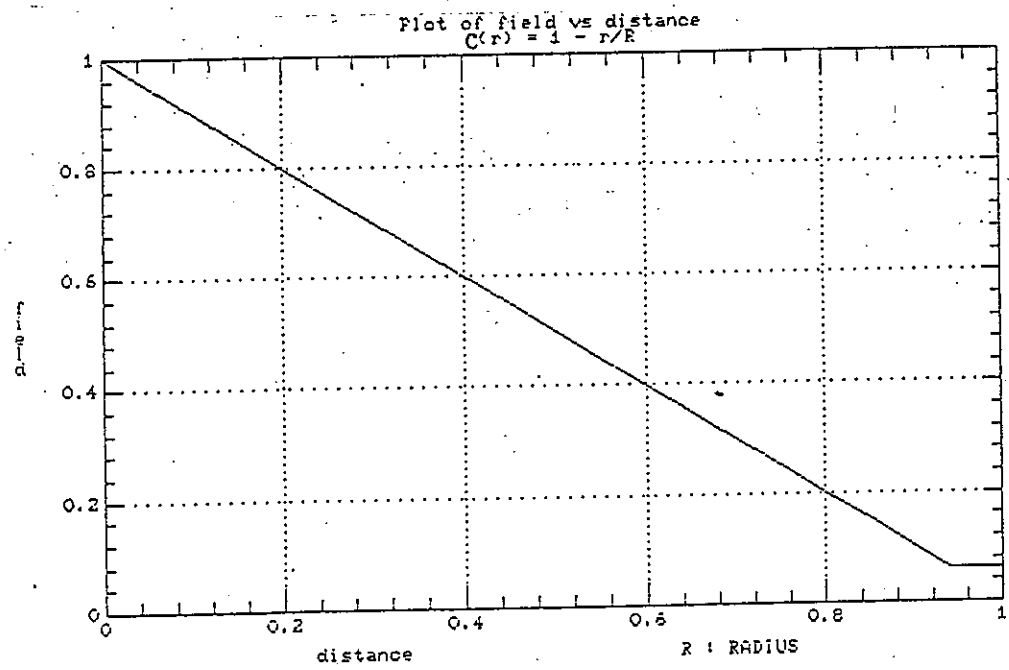
數為

$$c(r) = 1 - \frac{r}{R} \qquad 0 \le r \le R$$

，其中 R為控制點的作用範圍， r為欲計算位能之位置至控制點的距離。若 r>R，則位能為零。此位能函數是直線，如圖 3-1 (a)，水平方向表示控制點至欲計算位能之位置的距離，垂直方向表示距離所對應的位能。



<a>



<b>

圖 3-1 位能函數 $c(r) = 1 - \frac{r}{R}$ $\quad 0 \leq r \leq R$

計算位能時，總合所有控制點非零的位能貢獻量，以求得位能。若控制點的作用範圍為16，當 r=15 ，位能為0.0625，r=15.5時，位能為 0.03125。若二個作用範圍均為16的控制點，它們之間的距離為31，則有一部分的位置，同時受到二個控制點的影響，在計算這些位置的位能時，要總合每個控制點的位能貢獻量，才能得到真正的位能。此二控制點間的中點，同時位於二個控制點的作用範圍內，且與二個控制點的距離分別為15.5，每個控制點對中點的位能貢獻均為 0.03125 ，故中點的位能為0.0625。

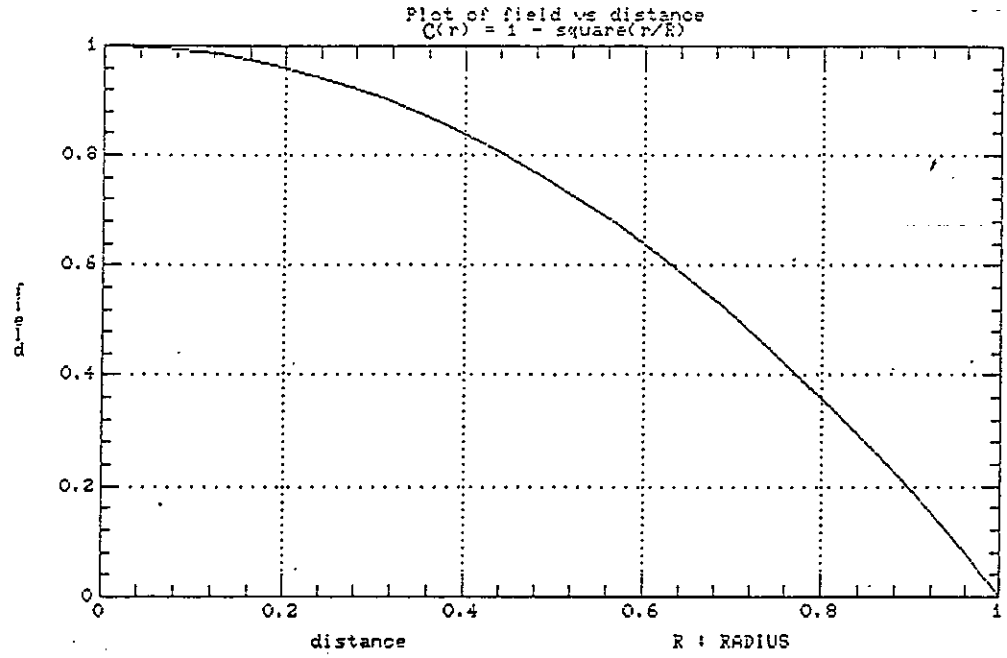
從連接上述二個控制點的直線取一線段，它的二個端點與任一控制點的距離分別為15及16，則此線段上每點的位能值均為0.0625，如圖 3-1 (b)，在此線段上，位能應隨著距離的增加而逐漸變小，雖然受另一控制點的影響，位能會稍微增加，但此增加的量須有一定的限制。對任何控制點，其位能仍應隨著距離的增加而慢慢減少，位能減少至某一值後，由於另一控制點位能貢獻量變得顯著，位能再慢慢地增加，位能應該非常平滑地放變，絕對不可能有如圖 3-1(b)所示，隨著距離增加，位能依舊不變的情形。

Blinn 及 Wyvill et al. 在設定位能函數時，都極

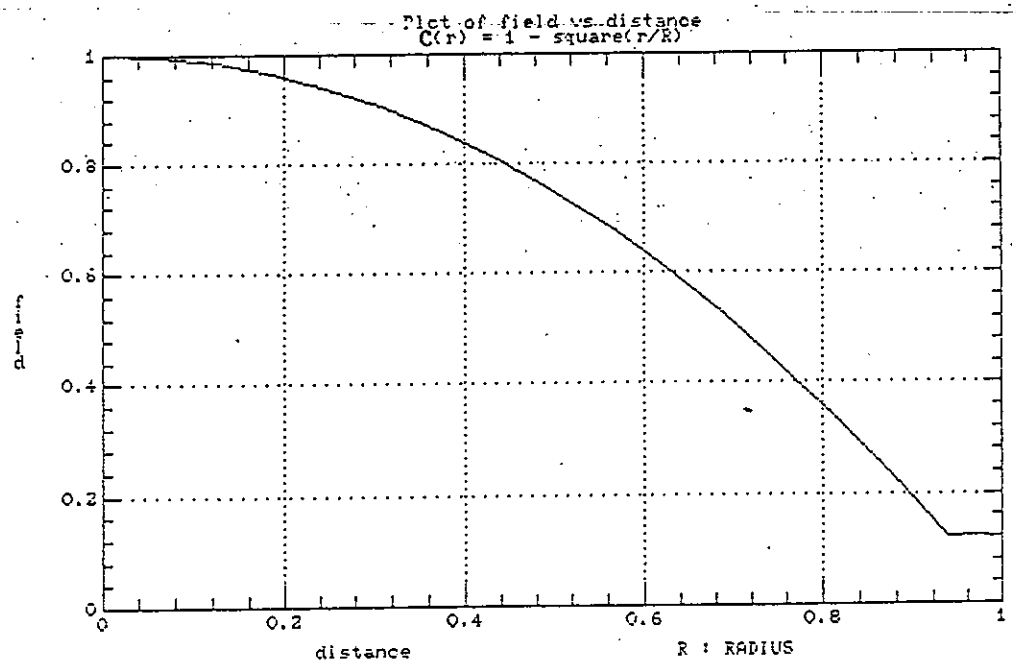
力避免處理根號的運算，上述的位能函數，計算控制點與欲求位能之位置間的距離，要使用根號運算。由於此位能函數需要用到根號運算及位能不是平滑地改變，故它不列入考慮範圍。若將位能函數改為

$$c(r) = 1 - (\frac{r}{R})^2 \qquad 0 \leq r \leq R$$

，此位能函數的形狀如圖 3-2 (a)。雖然這個新設的位能函數不需使用根號運算，但要檢查它的位能是否能平滑地改變。

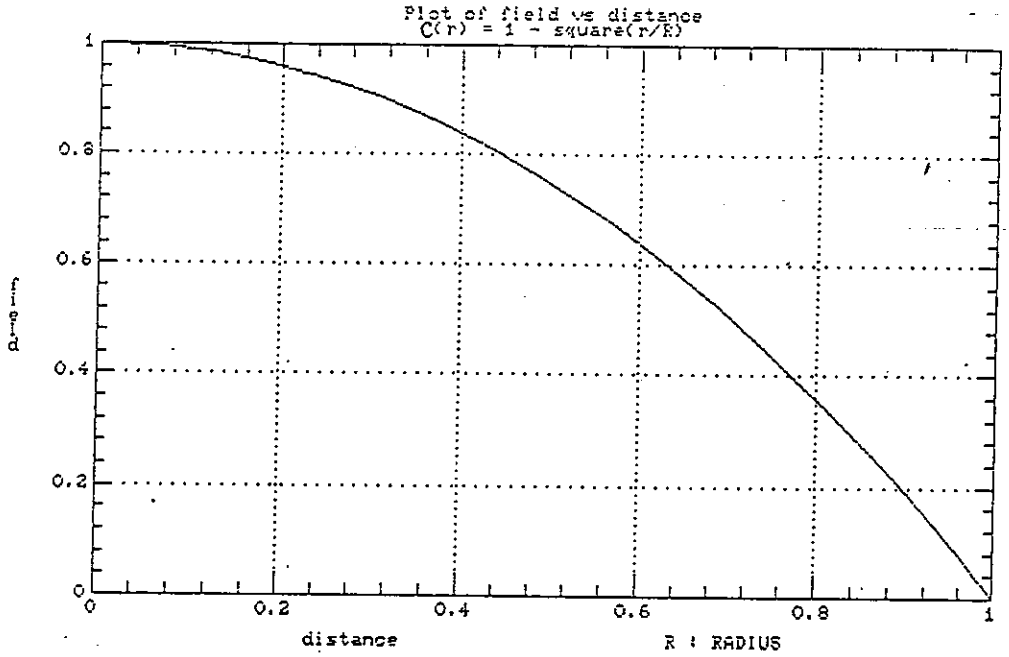


<a>

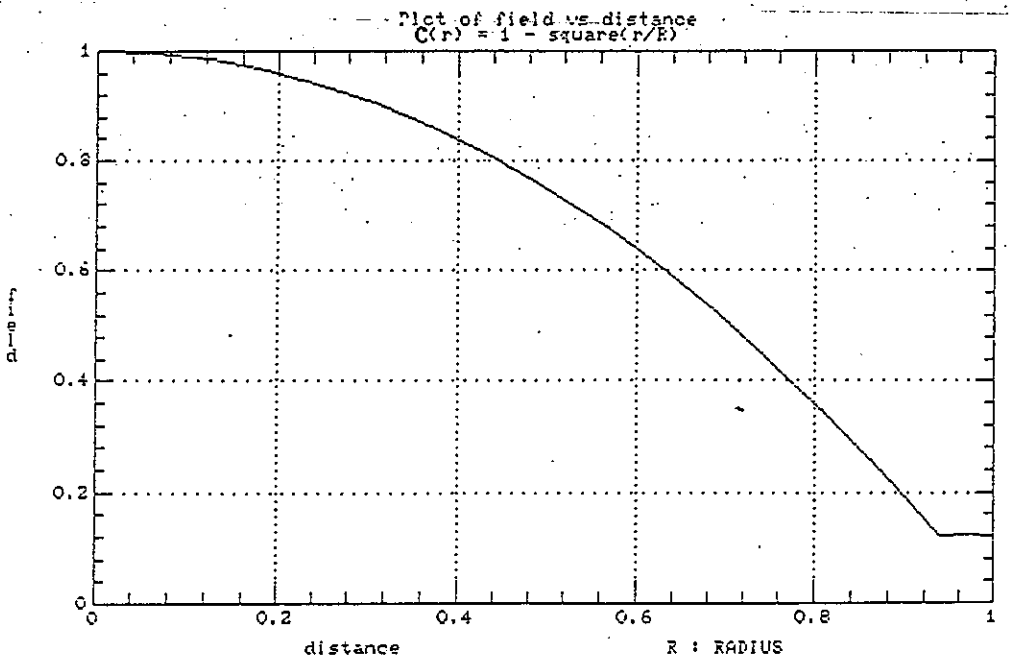


<b>

圖 3-2 位能函数 $c(r) = 1 - (\frac{r}{R})^2$ $\quad 0 \leq r \leq R$

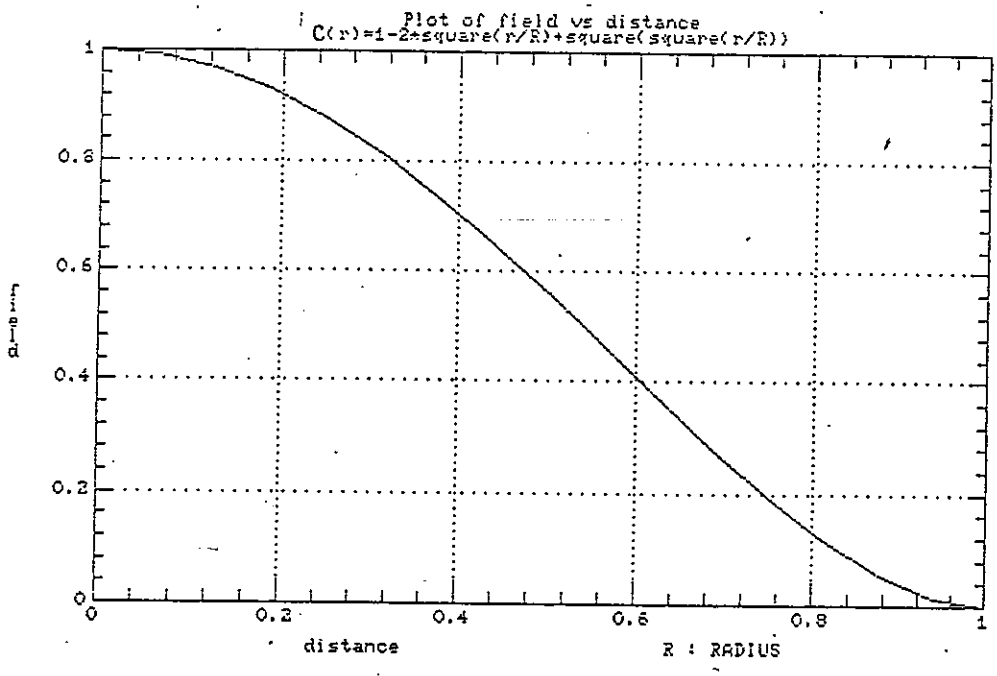


<a>

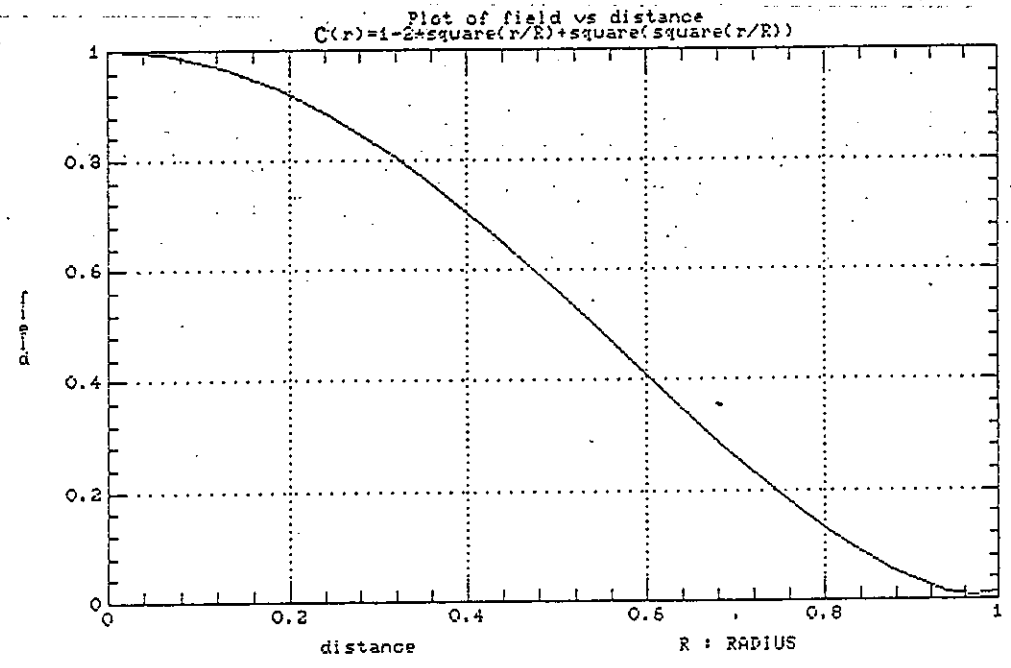


<b>

圖 3-2 位能函數 $c(r) = 1 - (\frac{r}{R})^2$ $\quad 0 \leq r \leq R$



<a>



<b>

圖 3-3 位能函數 $c(r) = 1 - 2(\frac{r}{R})^2 + (\frac{r}{R})^4 \quad 0 \le r \le R$

以前相同，線段中點的位能為0.0075，二個端點的位能為0.0147，如圖 3-3 (b)。由於此位能函數不須使用根號運算，位能的變化非常平滑，符合 Blinn及 Wyvill et al. 的研究結果，故採用它為本論文的位能函數。

3.3 保持體積不變

本節討論維持球體等位能外形體積不變的做法。首先對每個控制點增加一個相當於外力的屬性，在外力的作用下，等位能外形會被位起或壓下，體積隨之變化。當體積發生變化時，求得位能改變的量，再將改變的量平均地反映至其它未發生變化的部分，以維持體積的不變。

每個控制點對其作用範圍內的位置，均產生由0至1的位能貢獻量。若賦予控制點一個屬性 'sign' ，它只有二個值，正 (ON) 或負 (OFF)，當它為正時，位能貢獻量仍為0至1；當它為負時，位能的貢獻量乘以-1，變為-1至0 。位能的計算方法，仍然要總合所有控制點對此位置的位能貢獻量。若位置 (x，y，z)同時位於二個控制點的作用範圍內，假如二個控制點的 'sign' 均為正，對位置(X，Y，Z)的位能貢獻量均為正，相加之後，位置(x，y，z)的位能增加；假如一個控制點的'sign'為正，另一個控制點的'sign'為負，對位置(X，Y，Z) 的位能貢獻量均為一正一負，相加後，位置(x，y，z)的位能減少。

軟狀物體表面的位置具有相同的位能，會形成等位能的外形。單一控制點且'sign'為正時，會產生球體狀的等位能外形。當有另一個控制點接近時，位能發生變化，等

位能外形亦隨之改變。若另一個控制點的'sign'為正，由於二個控制點之位能貢獻量均為正，相加後，球體狀的等位能外形會稍微凸起，減少控制點的距離，發生變化的區域擴大，如同二顆水滴相互接近時的外形。若慢慢接近之控制點，其'sign'為負，一正一負的位能貢獻量相加後變小，球體狀的等位能外形會向內凹下，如同球體被壓下的效果。

賦予控制點屬性'sign'，根據等位能外形的變化，可視為它'作用力'的效果。當'sign'為正時，產生拉力的效果，將其它'sign'為正的控制點所產生的球體等位能外形拉起。當'sign'為負時，會產生類似推力的效果，使得由其它'sign'為正的控制點所產生的球體狀等位能外形凹下。但此種凹下的效果如同外力作用於球體的一點。若外力不是集中作用於一點，它作用的方式呈平面狀，如同手掌伸平壓著球體，則將許多'sign'為負的控制點緊地放在同一平面上，就可產生平面狀的外力。若外力作用的方式彎曲狀，如同手掌彎曲擠壓球體，只要將'sign'為負的控制點適當地排列，即產生所需要的推力。

屬性'sign'為正的單一控制點會產生球體狀的等位能外形，當其它控制點接近時，位能分布起了變化，等位能

外形亦隨之變化，故體積改變。在對軟狀物體做分析時，曾敘述體積應維持不變，因此要描述維持體積不變的做法。

呈球體狀的等位能外形，從正上方看下，可視為由許多的圓組成。當等位能外形發生變化時，圓的形狀亦改變，若能求得每層的變化量，將它反應在外形並未改變的部分。組合各層外形已改變的圓，不僅完整地表示形狀的變化，亦可維持體積的不變。

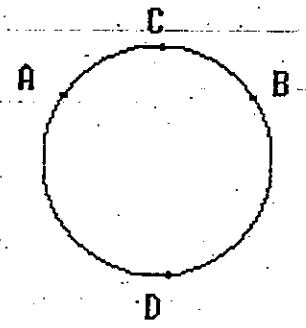


圖 3-4 外形變化的圓

圖 3-4為外形變化的圓，由A經C至B的圓弧，受其它控制點的影響而產生形狀的變化。記錄由A經C至B的位能變化量，將它平均地反應全由A經D至B的圓弧上，以保持體積的不變。受其它'sign'為正的控制點影響，圓弧ACB的位能增加，它會呈凸起狀，此段位能的增加量為另一段位能的減少量，再將應該減少的量平均地反應圓

弧ＡＤＢ上，故圓弧ＡＤＢ會變得比未受影響時還小。若受'sign'為負的控制點影響，圓弧ＡＣＢ的位能減少，它的形狀向內凹，圓弧ＡＤＢ的位能增加，會變得比原先未受影響時還大。

當不只與一個控制點發生作用，按照上述方法，分別計算每個控制點的影響，可求得真正的外形。按照物理的觀念，若物體同時受許多外力的作用，有二種方法可求得物體最終的運動狀況。求得所有合力，可決定物體的運動狀況。分別求得每個力對物體的影響，再總合所有的變化量，決定物體的運動狀態。這二種方式的結果是相同的。當不只與一個控制點發生作用時，每個控制點可視為一個外力，欲求外形的變化時，依序求得每個力對外形所產生的變化，即可求得最終的外形。若求得所有外力的合力，再做形狀的變化，這樣會不好處理，主要的原因在於每個外力的影響範圍不同，位能變化量須反應的範圍亦不同，造成圓弧上每點應該反映的位能不是完全相同。

第四章　塑型之考慮

本章依照 Wyvill et al. (1986b) 的做法，產生軟狀物體的外形。 4.1節介紹塑型的種類及簡略地敘述軟狀物體的產生方法。為了有效率地計算位能及避免重複計算先前已經求過的位能，使用二種資料結構的赫序表 (Hash Table)達到這個目的，4.2及4.3二節分別敘述這二種赫序表。由於赫序表的使用須用到赫序函數 (Hash Function)，4.4節說明赫序函數的設定理由。4.5節根據求得的位能決定多邊形的外形，但在決定多邊形時，須避免產生面積為零的多邊形，4.6節說明此現象的解決方式。4.7節敘述套裝軟體的使用，以便在顯示器上看到軟狀物體的外形。

4.1 塑型的種類

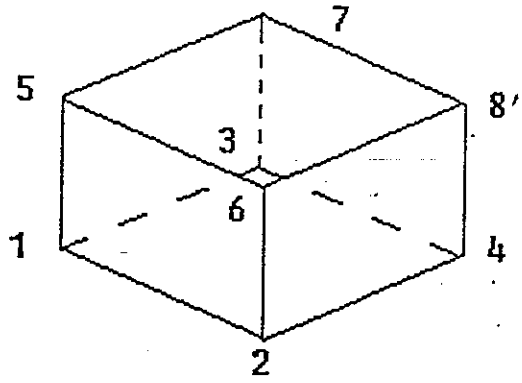
軟狀物體表面是由相同位能的點形成一個等位能外形，選定一個數字代表此外形的位能後，單一控制點會產生一個球體狀的等位能外形。若相同位置存放二個控制點，則球體等位能外形的體積變為原先的二倍。使用一組控制點塑造物體的外形時，由於單一控制點會產生球體狀的等位能外形，因此軟狀物體的輪廓是藉著球體狀的等位能外

形結合而組成。改變控制點間的遠近距離，球體等位能外形結合的程度亦改變，則軟狀物體的外形亦隨之改變。若不斷地更改控制點間的空間關係，軟狀物體的形狀會不斷地改變，這樣可達到動畫的效果。

球體是幾何物體，軟狀物體的外形是藉著結合一種幾何物體塑造而成。Reguichq和Voelcker (1977) 提出的塑形技巧：CSG(Constructive Solid Geometry)，結合球體、圓柱體、長方體等不同的幾何物體塑造物體的外形。雖然 CSG使用不同的幾何物體，但它不易表現形狀的改變。對於軟狀物體，調整控制點之間的距離，球體結合的程度亦改變，可達到形狀的改變。

根據Wyvill et al. (1986b)的研究方法，分割空間成為許多正立方體，物體的輪廓不外乎通過或沒有通過這群正立方體。考慮與物體相交的正立方體，會在正立方體內產生一個多邊形，集合所有的多邊形，即可組成軟狀物體的外形。如同拼圖遊戲，只須將每個多邊形擺在它應該在的位置，就可拼出整個圖案。若物體移至其它位置，重新計算正立方體內的多邊形，又組成物體的外形。當改變控制點間的距離，正立方體內的多邊形亦改變，故軟狀物體的形狀發生變化。

物體的外形由許多面組成，每個面用一個多邊形表示，採用多邊形網孔塑造物體外形的動畫系統，如套裝軟體ANIMATOR及THREE-D；必須詳細敘述組成物體外形的多邊形。由於多邊形的端點和邊可能與其它的多邊形共用，使用者須仔細處理這種情形，要敘述一個正立方體，須做如圖 4-1的工作，明確地指出它所用到的端點及組成每個多邊形所用的端點。對於複雜的物體或物體形狀須改變，這是件令人討厭的工作。



● number of points
● number of polygons

```
data 8 6                    8 6
0.0 0.0 0.0                 0.0,0.0,0.0
1.0 0.0 0.0                 1.0,0.0,0.0
0.0 1.0 0.0                 0.0,1.0,0.0
1.0 1.0 0.0                 1.0,1.0,0.0
0.0 0.0 1.0                 0.0,0.0,1.0
1.0 0.0 1.0                 1.0,0.0,1.0
0.0 1.0 1.0                 0.0,1.0,1.0
1.0 1.0 1.0                 1.0,1.0,1.0
4 1 2 6 5                   4 1 2 6 5
4 2 4 8 6                   4 2 4 8 6
4 4 3 7 8                   4 4 3 7 8
4 3 1 5 7                   4 3 1 5 7
4 3 4 2 1                   4 3 4 2 1
4 5 6 8 7                   4 5 6 8 7
```

<a>　　<b>

圖 4-1 多邊形網孔組合的正立方體

(a) ANIMATOR的規格 (b) THREE-D的規格

若空間由許多的正立方體組成，一旦確定軟狀物體的控制點，即可根據位能函數計算正立方體頂點的位能，產生物體外形的多邊形，再將這些多邊形轉換成ANIMATOR或THREE-D能接受的規格，使用者不必處理如圖 4-1的繁雜工作，他（她）只要指定控制點的座標，就可使用ANIMATOR或THREE-D ，看到物體在三度空間上的輪廓。

以軟樑曲面細片塑造物體外形，必須使用一組控制點，調整細片的形狀，以產生符合使用者想法的形狀，若要改變外形，調整軟樑曲面的控制點即可達成，但控制點座標的選擇，必須對軟樑曲面特性有充分的了解，才可勝任。

規劃軟狀物體的輪廓時，分割空間成為許多的正立方體，根據控制點的位置，計算這些正立方體每個頂點的位能，再由頂點的位能決定物體的輪廓。若改變控制點的位置，正立方體頂點的位能隨著改變，多邊形的外觀也改變。因此正立方體頂點的功能，如同軟樑曲面細片的控制點，具有調整形狀的能力。

軟狀物體的控制點與軟樑曲面的控制點，都可塑造出變化多端的外形，但有不同的實用程度。後者不僅要確定控制點的座標，還得充分了解軟樑曲面的性質，對於一個

沒有電腦背景的使用者，這是一件困難的工作；前者只須確定控制點的座標即可，位能的計算，多邊形的產生，都由程式處理，故比較好使用。

軟狀物體外形的產生，非常類似拼圖遊戲，只要將每個多邊形放在它應該在的位置，即可塑造出物體的外形。Wyvill et al.分二個步驟建立多邊形網孔。首先找出與等位能外形相交的正立方體，接著求出此正立方體內的多邊形。

為了不必查驗所有的正立方體而有效率地找出與等位能外形相交的正立方體，必須利用一個性質：考慮等位能外形通過的某個正立方體，則等位能外形亦可能通過此正立方體周圍的正方體。對於每個控制點，先找出最靠近它的正立方體頂點，再沿著x軸、y軸和z軸，分別計算經過的正立方體頂點的位能，直至找到一對相鄰的頂點，它們位能有一個大於、另一個小於等位能外形的位能。這表示等位能外形會通過這相鄰二頂點間的直線，也就是說，等位面外形通過包含此二頂點的四個正立方體，故稱這些正立方體為種子正立方體 (seed cube)。

對於每個種子正立方體，檢查相鄰的正立方體，若與等位面外形相交，則再檢查其周圍的正立方體。重複此動

作，直至與物體相交的正立方體完全被檢查過，這樣完成了第一步驟。第二步驟要建立正立方體內部的多邊形，根據正立方體頂點的位能，判斷多邊形的形狀，一旦找出所有的多邊形，自然就塑造出物體的外形。

4.2 儲存控制點的赫序表

採用多邊形網孔表現軟狀物體的輪廓，必須先確定所需要的多邊形。在求多邊形時，碰到二個問題，因此使用二種赫序表來處理它。每種赫序表由不同的資料結構組成，只要發揮其特定的功能，即可解決定所面臨的問題。

第一個問題是要有效率地計算位能。位能值的大小係由控制點決定，要計算某位置的位能，先選擇一個控制點，求出此控制點與欲求位能值的位置之間的距離，將此距離帶入位能函數，即求得此控制點對欲求位能值的位置的位能貢獻量。接著重複上述的步驟，總和其它控制點的位能貢獻量，即可求得真正的位能值。但每個控制點對其作用範圍外的位置，賦予的位能貢獻量為零，也就是說，計算位能時，花費時間及資源求得遠處控制點的位能貢獻量若為零，最好能事先避免，以節省時間及資源。這表示必須避免檢查過遠而位能貢獻量為零的控制點。

控制點可能分布於空間中的任一角落，某些區域有較多的控制點，某些區域的控制點較少，因此必須適當地存放它們，以便能很快地決定聚在一起的控制點及離群獨居的控制點。故在計算位能時，能很快地確定對其位能貢獻量為零的大部分控制點。先分割空間成為許多的正立方體，接著確定控制點所處的正立方體。這樣一來，就可決定控制點的疏密程度。計算某位置的位能時，先確定此位置所屬的正立方體，檢查此正立方體周圍的正立方體，即可決定對此位置有非零的位能貢獻量的控制點，至於其它正立方體內的控制點，表示對此位置的位能貢獻量為零，不必花時間及資源去處理這些遠處的控制點。

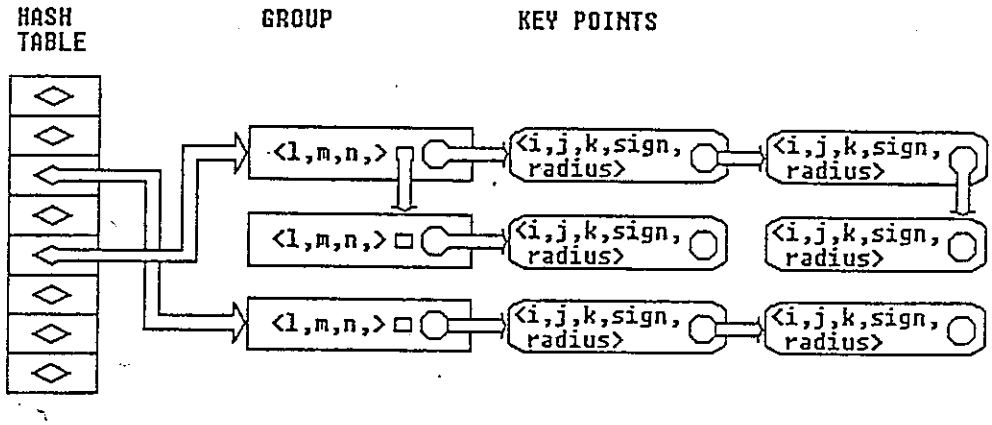


圖 4-2 控制點的赫序表

圖 4-2表示扮演上述功能的資料結構所組成的赫序表(hash table)，它是一個一維的向量。由於使用赫序函數決定正立方體儲存於赫序表內的位置，再加上赫序函數有碰撞 (collision)的特性，故向量的每個元素不只儲存一個正立方體。若向量元素的值為空 (null) ，則表示應存於此元素的正立方體並未儲存任何控制點。若向量元素值為非空，則表示應存於此元素的正立方體包含控制點。也就是說，經由這個表格可以決定控制點的群聚程度。

為了能明確地表示控制點所屬的正立方體，使用聯結串列將同一正立方體內的控制點串在一起。明顯地，代表控制點的聯結串列節點至少需要儲存控制點的座標 (x，y，z)及下一個節點的位址。由於赫序表的每個元素可儲存不只一個的正立方體，為了分辨這些正立方體，在聯結串列的起始處加上一個節點，以達到此目的。每個聯結串列的第一個節點並不代表控制點，節點內原先儲存控制點座標的欄位用 (l，m，n)取代。若它乘以正立方體的邊長R，成為 (l*R，m*R，n*R)，則此座標位置為正立方體所有位置的最小座標值，因為只有一個最小值，比較聯結串列的第一個節點就可分辨不同的正立方體。但是赫序表的每個元素不只儲存一個正立方體，每個正立方體又使用一個

聯結串列儲存位於其內的控制點，也就是要將這些聯結串列綁在一起，所以串列的節點再增加一個欄位，以連接所有的串列。因此，第一個赫序表的資料結構包含處理聯結串列的二個指標欄位，儲存控制點的三個實數欄位，控制點作用範圍的實數欄位，以及此控制點產生正或負位能的整數欄位。

位置 (x，y，z)的位能，由它與控制點之間的距離決定，故在計算位能時，須經常存取赫序表內的控制點。由於控制點均包含在正立方體之內，只要找到正立方體就可存取控制點。每個正立方體的左下角位置： (l*R，m*R，n*R)，R為正立方體邊長，此位置為正立方體內具有座標值最小的點，它是唯一的，因此採用(l，m，n) 為此正立方體的編號。經由赫序表存取控制點時，先找到包含控制點的正立體的編號，將此編號代入赫序函數，計算得到的值做為赫序表的索引，由於一個索引下會有不只一個正立方體的聯結串列，接著比較聯結串列的第一個節點，直到找到正確的正立方體，然後存取控制點計算位能。

每個控制點都有它的作用範圍，在作用範圍之外的位置，控制點對其位能的貢獻量為零。若所有控制點的作用範圍均不超過儲存控制點的正立方體邊長，則不必查驗所

有的控制點就可計算位能。任何正立方體都有26個相鄰的正立方體，計算位置 (x，y，z)的位能，先求出包含此位置的正立方體的編號，確定相鄰的26個正立方體。不在這27個正立方體內的控制點，對位置 (x，y，z)的位能貢獻量為零，計算位能時，不必查驗這些控制點，以增加效率。至於27個正立方體內的控制點，若與位置 (x，y，z)的距離超過它的作用範圍，則不必查驗。上述過程檢查完後，位置 (x，y，z)會在剩餘控制點的作用範圍之內，分別計算位能的貢獻量，即可求得位置 (x，y，z)的位能。若控制點的作用範圍超過正立方體的邊長，則增長正立方體的邊長，再依上述步驟計算位能。

按照這樣的做法，須檢查27個正立方體，它們的總體積為 $27R^3$，但對位能有影響的控制點，只集中於 $\frac{4}{3}\pi R^3$ 的體積，二者之比均為 7:1，也就是花費額外的資源，檢查許多對位能無影響的控制點。

若使用較小的正立方體且搜尋較多的正立方體，則檢驗較少的控制點就可求得位能。但當正立方體含有較少的控制點，會非常浪費時間，因為需要額外的時間搜尋其周圍較多的正立方體。舉例說明這種情形，當正立方體邊長為 R 時，須找尋27個正立方體，總共的空間為 $27R^3$，若正

立方體的邊長為0.5R時，須找尋 125個正立方體，總共的空間為 $(\frac{125}{8})R^3$ ，找尋的空間為原先的0.57倍，找尋的正立方體為原先的 4.6倍。因此仍然維持控制點的作用範圍不能超過正立方體的邊長，在計算位能時，須搜尋27個正立方體。

4-3 儲存位能的赫序表

第二個問題是在找尋等位能外形通過的正立方體時，避免重新計算先前已算過的位能。此處所說的正立方體與儲存控制點的正立方體不同，它的體積較小，等位能外形通過它時，會產生一個多邊形。採用圖 4-3的赫序表處理此問題。我們只計算正立方體 8個頂點的位能。使用 (i，j，k) 表示這些頂點，其中 (i*d，j*d，k*d)為頂點的真正座標，d 為正立方體的邊長。每個頂點由 (down，i，j，k，field，hd_fld，wk_fld，ps_fld，done，)表示，它們都儲存於聯結串列，經由赫序表可取得頂點的資料。'down'為聯結串列的指標，'field' 表示頂點的位能，'hd_fld'、'wk_fld'及'ps_fld'在處理體積不變會用到，'done' 的含意稍後再解釋。

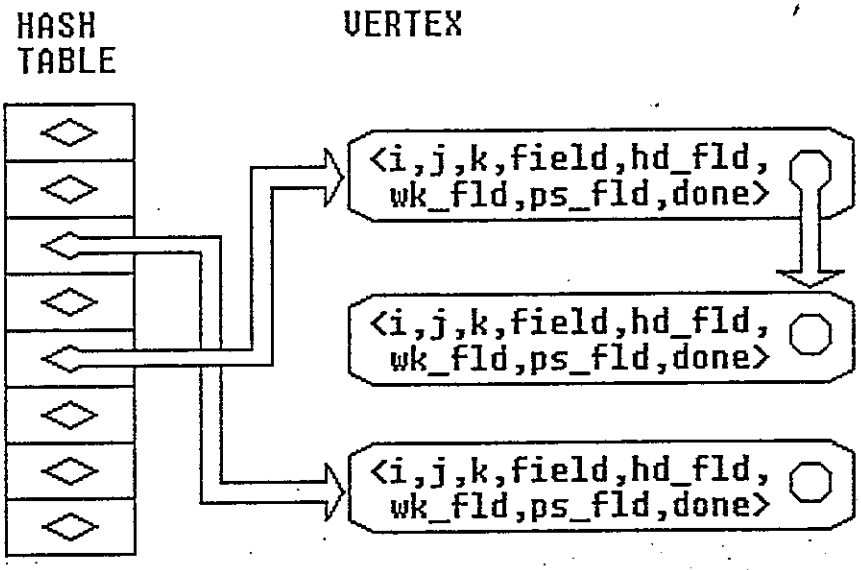


圖4-3 位能的赫序表

存取特定頂點的資料，將它的 (i，j，k)代入赫序函數得到赫序表的索引。此索引包含由頂點組成的聯結串列的指標，搜尋此串列即可找到頂點。當它第一次被引用時，搜尋會失敗，此刻須計算頂點的位能，再將此頂點的相關資料加至聯結串列。每個頂點會被 8個正立方體共同使用，但它的位能只計算一次，往後想存取時，會在赫序表內找到，因此不會重新計算先前已算過的位能。

等位能外形通過的正立方體會產生多邊形，在找尋所有與等位能外形相交的正立方體時，對於已產生多邊形的正立方體必須能辨別，以避免重複計算已經求得的多邊形

。正立方體左下角的頂點，也就是最小 (i，j，k)的頂點，為此正立方體的編號。若此頂點的'done'為真 (True)，則顯示等位能外形通過此正立方體產生的多邊形已經求過，不必再計算一次。

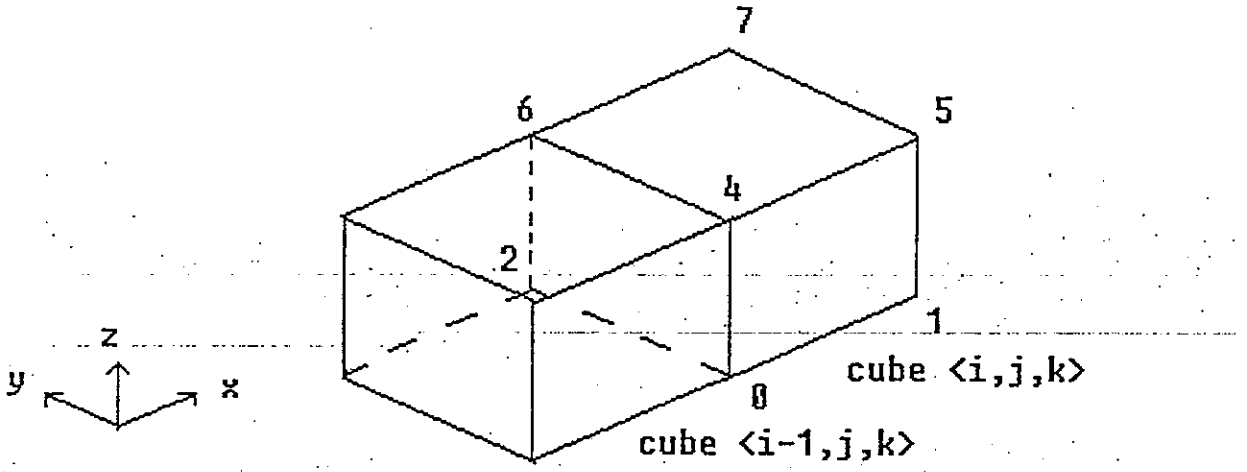


圖 4-4 相鄰的正立方體

對每個控制點沿著 x 軸、y 軸和 z 軸找尋與等位能外形相交的正立方體，稱這些正立方體為種子立方體。由於種子立方體周圍的正立方體亦可能與等位能外形相交，故從種子立方體開始檢查，可找到所有與等位能外形相交的正立方體。觀察圖 4-4的相鄰正立方體，編號為 (i，j，k)的正立方體，它的 8個頂點賦予 0至 7的號碼。考慮號碼為2、0、4、6的四個頂點，它們亦被正立方體 (i-1，j

，k)共用。若此四點的位能完全大於或完全小於等位能外形的位能，則等位能外形並未通過此四點組成的正方形；假如它們的位能有的大於及有的小於等於等位能外形的位能，則等位能外形會通過此四點組成的正方形。求得通過正立方體 (i，j，k)的多邊形後，若正立方體 (i-1，j，k)的多邊形尚未求得，則計算其多邊形。為了處理等位面外形通過的正立方體，先將其'done'設成真，接著把它加入佇列 (queue)。詳細的做法如下：

```
{Set seed cube's done flag to true;
 Add seed cube to the queue;
 While queue is not empty {
    Remove one cube from the queue;
    for each face of cube {
       if surface intersects face {
          select neighbour cube for that face;
          if neighbour's done flag is not true {
             Set neighbour's done flag to true;
             Add neighbour to queue
          }
       }
    }
```

```
    Pass vertices and values for cube  to  second
    stage
  }
}
```

4.4 排序函數的設定

儲存控制點及產生多邊形時，均將空間分割成許多的正立方體，為了分辨及存取正立方體，賦予每個正立方體獨一無二的編號，但是包含控制點的正立方體及等位能外形通過的正立方體，均只佔所有正立方體的一小部分。若對每個正立方體都保留記憶體以儲存相關資訊，則大部分的記憶體都未儲存任何訊息，這是種很大的浪費。最好的方法是只儲存包含控制點的正立方體及等位能外形通過的正立方體，因此使用表格（一維的向量）儲存這些正立方體。

每個正立方體的編號 (i，j，k)都是三維的向量，儲存它的表格是一維的向量。這種情形非常類似程式語言儲存 n 維向量的方式。基本上，記憶體可視為一維的向量，n 維向量儲存於記憶體，相當於將它儲存於一維的表格，儲存的方法又可為以列為主 (row major)及以行為主 (co

lumn major) 二種。因此模擬程式語言儲存 n 維向量的方式，以儲存正立方體。

採用以列為主的儲存方式，設定一個赫序函數，將正立方體的三維編號轉換成一個數值，而此數值表示一維向量的索引。若一維向量共有 l*m*n個元素，則會將 (i、j、k)轉換成

(rem(i、l)*m+rem(j、m))*n+rem(k、n)

其中 rem(x、y)表示 x除 y的餘數。

若控制點可出現的範圍為

{(x，y，z) | 0≦x≦639，0≦y≦479，0≦z≦479}

，而 l=m=n=16，正立方體的邊長亦16。整個空間共有 36000個正立方體，一維的表格共有4096個元素，只佔全部正立方體的九分之一，故使用較少的記憶體，經由赫序函數的處理，可儲存空間的任一正立方體。

由於並非儲存所有的正立方體，有可能二個編號不同的正立方體，經由赫序函數的計算，對映至一維表格的同一個元素，這個情形相當於赫序函數發生碰撞(collision)。處理的方法為使用聯結串列，將對映至相同元素的所有

正立方體串在一起，當要存取資訊時，比較聯結串列的相關節點，直至找到所要的正立方體。對聯結串列做找尋，只能從第一個節點開始，一個接一個找下去，若聯結串列很長，這是件耗時的工作。但由於被使用的正立方體，只佔全部正立方體的一小部分，也就是發生碰撞的機率非常小，故搜尋不是件耗時的工作，因此赫序函數的使用及赫序對資料的儲存方式，都很合理。

4.5 多邊形的產生

在第一步驟，已經求得正立方體頂點的位能；在第二步驟，根據求出位能值的大小，決定等位能外形通過正立方體產生的多邊形。

首先找到等位能外形通過正立方體各邊的交點座標，再連接這些相交的點組成多邊形。正立方體頂點的位能，若大於等位能外形的位能，則顯示此位置在等位能外形的輪廓內，用H表示；若頂點的位能小於或等於等位能外形的位能，則顯示此位置在等位能形成的輪廓外，用C表示。若頂點的位能剛好等於等位能外形的位能，它的複雜情形雖然較低，但要避色產生面積為零的多邊形。

正立方體相鄰的二個頂點 p 和 q，它們的位能分別為

f_p和f_q。若 q位於等位能外形之內， p位於等位能外形之外，也就是 q的位能大於等位能外形的位能， p的位能小於等位能外形的位能，故 q標為H， p標為C。設 magic 為等位能外形的位能，

$$\frac{\text{magic} - f_p}{f_q - f_p}$$

的值再乘以正立方體的邊長，得到一個距離。若從 p點開始，沿著通過 p和 q二點的正立方體的邊，移動上述距離，則此位置表示等位能外形與pq邊的交叉點。以此種線性插補法 (linear interpolation)計算得到的交叉點，雖然不是真正的交叉點，但此種方法容易計算，倘若正立方體很小，以此種方法計算得到的多邊形，來逼近等位能外形通過正立方體的曲面，非常合理且效果不錯。四個彼此相鄰的正立方體，有一個共用的邊，若等位面通過此邊，則在四個正立方體計算此共用邊的交叉點時，須完全相同。

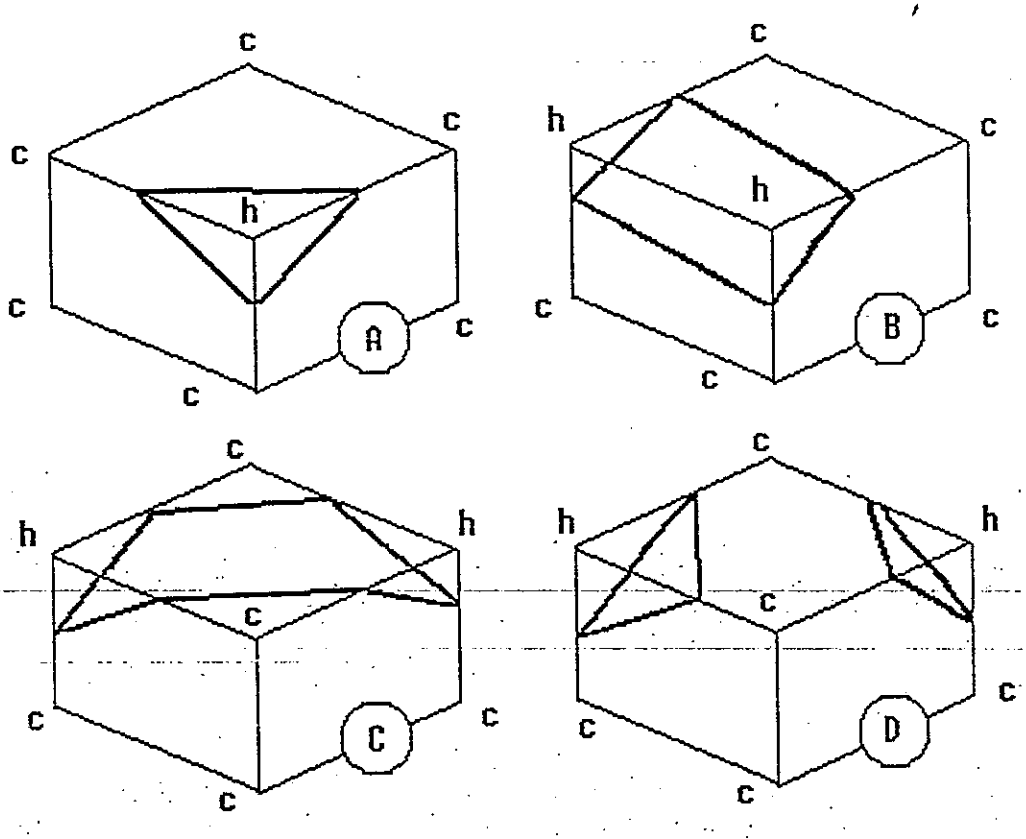


圖 4-5 正立方體內的多邊形

連接等位面外形與正立方體間的交叉點，以形成多邊形，此過程非常複雜，故由範例開始解釋。圖 4-5(A)，若正立方體的頂點，只有一個在等位能外形之內，則通過此正立方體的等位能外形，約略為一個由 3個端點組成的多邊形。圖 4-5(B)，若二個相鄰頂點在等位能外形之內，其它的頂點都在等位能外形之外，則產生約略由四個頂點組成的多邊形。若正立方體只有二個同平面、不相鄰的

頂點位於等位能外形內，則會形成包含此二頂點的多邊形，它是由六個端點組成，如圖4-5(C)。為什麼選擇一個六角形而不採用二個三角形，如圖4-5(D)？事實上，採用圖4-5(C)，主要是為了聯結在等位能外形內部的頂點，分離在等位能外形外部的頂點。至於聯結等位能外形內部的頂點或等位能外形外部的頂點，必須由這些頂點的位能決定。包含此四頂點的正方形，其中心點的位能值，約略等於四個頂點位能的平均值。假如此平均值大於等位能外形的位能，則聯結等位能外形內部的正立方體頂點，若非如此，則聯結等位能外形外部的正立方體頂點。

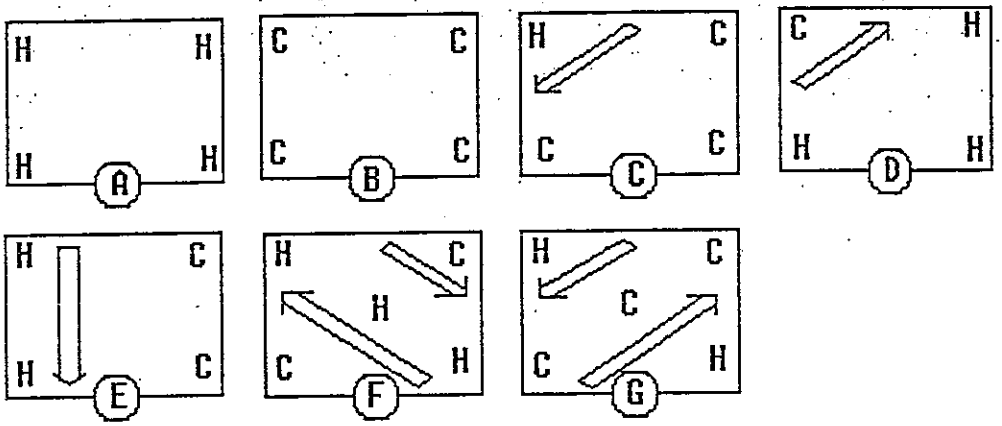


圖4-6 多邊形的端點及邊

多邊形由端點及邊組成，連接端點會形成多邊形的邊。若等位能外形通過的正立方體的邊，則產生的交叉點為

多邊形的一個端點。針對正立方體的每個面，收集等位能外形通過此面的邊所形成的交叉點，可求得多邊形的所有端點，接著再依圖 4-6的方式，決定多邊形的邊，因此確定了多邊形的外觀。圖 4-6共考慮七種情況。若正立方體每個面的四個頂點，全在等位能外形的內部或外部，如 A和 B二種實例，則不會產生多邊形的邊。若只有 3個頂點或只有 1個頂點位於等位能外形之內，如 C和 D二種實例，則會產生一個多邊形的邊。若只有二個相鄰的頂點位於等位能外形之內，則會產生實例 E的多邊的邊。若只有二個頂點在等位能外形的內部，但它們處於對角的位置，則依此面中心點的位能，則連結等位能外形內部的二個頂點，如實例 F，否則連結在等位能外形外部的二個頂點，如實例 G。聯結多邊的端點形成多邊形的邊時，須注意聯結的順序，相反的順序會產生不可預測的結果，故在圖 4-6中，箭號表示多邊形端點連接的順序。

多邊形的每個邊都由首、尾、二個端點組成，這二個端點的順序，決定多邊形邊的方向，同時為了分辨多邊形的端點，賦予每個端點獨一無二的編號。多邊形的各邊收集完後，必須組合這些邊以形成完整的多邊形。由於這些邊會形成一個封閉的區域，任一邊的尾部端點，必定為某

一邊的起始端點，故從任一邊開始，找尋與它相接的邊，只要每一個邊恰好被走一次，就可建立完整的多邊形。根據上述理由，使用一個三維向量儲存多邊形的邊，向量的前二個元素記錄多邊形邊的端點的編號，第三個元素表示此邊是否被走過。詳細的演算法如下：

```
{
    for each edge of cube, <p.g>   {
    if p is 'h' and g is 'c' or p is 'c' and g is
     'h'  {
            create intersection <p,g>;
        }
}
    for each face of cube   {
        create edges according to fig 4-6;
}
    while edges remain   {
       start : =  any edge; polygon  : = start ;
       remove start from edge array;
       next : = successor of start ;
```

```
        while next != start  {
            polygon  : = polygon + next ;
            remove next from edge array;
          }
          output polygon;
      }
```

{

三個點會落在同一個平面上，當組成多邊形的端點超過 3個時，這些端點不一定會在同一個平面上，所以要將它們三角化。首先求得此多邊形的中心點，多邊形每邊的首尾二個端點和中心點相連，即形成三角形。若多邊形端點的順序為

$$P_i = (x_i, y_i, z_i) \qquad 0 \leq i < n$$

其中心點為

$$C = <\frac{1}{n}\sum_{i=0}^{n-1} x_i, \frac{1}{n}\sum_{i=0}^{n-1} y_i, \frac{1}{n}\sum_{i=0}^{n-1} z_i>$$

則三角化為

$<P_{i-1}, P_i, C>$ $\quad 0 < i \leq n-1$

和 $<P_{n-1}, P_0, C>$

4.6 避免面積爲零的多邊形

由端點及連接端點形成的邊所組成的多邊形，經過三角化後，變成許多個三角形，計算個別三角形的面積，則可求得多邊形的近似面積，故等位能外形通過的正立方體產生的多邊形，其面積應不為零，但在某些情況下，會產生面積為零的多邊形，造成這種現象的主要原因在於數值經過除法運算後，計算的結果非常小，列印時會發生截位的誤差 (Truncation Error)。

設正立方體的二個相鄰頂點 p 和 q，各自的位能為 f_p 及 f_q，q在等位能外形之內，p在等位能外形之外，等位能外形的位能值為 magic，則由

$$\frac{\text{magic} - f_p}{f_q - f_p}$$

可求得等位能外形與pq邊的交點，若 magic = 0.5 、$f_p = 0.25$ 、$f_q = 0.75$，求得 0.5，表示交點位於pq邊的中點；f_p= 0.49983，$f_q = 0.75$，求得0.00067935，正立方體的邊長為

2，則等位能外形會通過pq邊上離 p點 0.0013782的位置，此位置非常靠近 p點。

使用套裝軟體時，必須將描述物體的資料事先存於檔案，如圖 4-1的規格，再從檔案讀取資料，產生需要的畫像。列印上述交叉點及 p點的座標時，由於點可位於

$\{(x、y、z) \mid 0 \leqq x \leqq 639，0 \leqq y \leqq 479，0 \leqq z \leqq 479\}$

的範圍，小數點後二位數字保留，其餘捨去，則交叉和 p點會變為同一點，對圖 4-5(A) 的三角形，由於交叉點均和頂點相同，故會產生面積為零的多邊形；對於圖 4-5(B)的四邊形，退化成一條直線，它的面積亦為零。

等位能外形與正立方體邊的交點，太靠近正立方體之頂點，易產生面積為零的多邊形。為避免此問題，凡是頂點的位能與等位能外形的位能，它們之間的差值小於一定的範圍，則視此頂點的位能為位能外形的位能，故 f_q 原先為 0.49983，可視為 0.5。計算交叉點時，若發生此種情形，則不產生多邊形；這種處理方式不會產生破洞的情形，因為周圍的多邊形會擴張，把這破洞補好。

位能值的四捨五入，以避免產生面積為零的多邊形，此種做法在對多邊形三角化時，會有邊際效用。建立多邊形時，採用的特性為多邊形的各邊恰好使用一次，每個端

點恰好連接二個邊。但正立方體頂點的位能值經過四捨五入後，會產生有趣的特例，如圖 4-7，有個端點連接四個邊的情形，也就是說，同一個正立方體內存在二個多邊形，這二個多邊形有個共同的交點。放在建立多邊形時，若端點同時為二個多邊形的交點，則表示要從此位置開始，找尋相連的邊，建立多邊形。

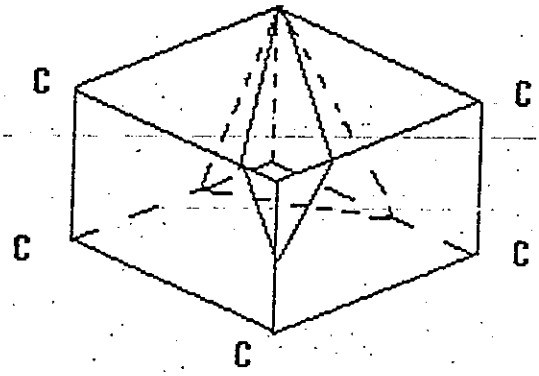


圖4-7 共用端點的的多邊形

4.7 套裝軟體的使用

多邊形組成後，仍是一大堆的數字，由這些數字很難看出軟狀物體的形狀，故使用套裝軟體THREE-D 及ANIMATOR 顯示物體的輪廓。前者是以線構架方式 (wire-frame)，將物體呈現在向量顯示器上，再經由轉鈕的調整，可從不同的角度看到物體的外形，它還可將物體放大、縮小及

調整明暗程度。ANIMATOR則使用一群參數描述光源、顏色、角度....等屬性，將物體呈現於掃描顯示器上，以模擬真實的影像。

使用套裝軟體時，必須將資料轉換成它們能接受的形式，首先要敘述組成物體輪廓所需的點及多邊形的數目，接著以一列一列的方式，敘述點及多邊形，如圖 4-1所使用的點、記錄其座標、套裝軟體會依照點出現的先後順序，賦予編號，這些編號由 1 開始，依序增加。敘述完點之後，接著說明多邊形。每列代表一個多邊形，它的第一個數字表示此多邊形包含的點數，再依序指出組成多邊形各點的編號。由於每一點及每一個多邊形都用一列表示，點的列數與多邊形的列數，須與最先敘述組成物體的點與多邊形的數目符合一致。在說明這些事情時，對不同的套裝軟體，格式要稍微調整，以符合各自的需求。根據套裝軟體對資料的規格，確定軟狀物體外形的多邊形的儲存方式。

軟狀物體的外形是由許多的多邊形組成，規劃軟狀物體外形時，採用類似拼圖遊戲的方法，只要每個多邊形在其應該在的位置即可，至於先後順序沒什麼關係。軟狀物體的等位能外形，通過正立方體時，會形成一個多邊形。

每次針對這樣的正立方體找到的多邊形，都要將它儲存起來，一旦找到所有的多邊形，就塑造出物體的輪廓。

儲存多邊形，也就是儲存它的端點及邊；邊的形成又由端點的連結順序決定，故對每一個端點，除了儲存它的座標之外，還要賦予一個編號，所有的編號均為大於零的整數，從1開始，依序增加。賦予編號後，不僅能很順利地表示多邊形的邊，還可由最大的編號顯示所使用的端點數目。由於每個端點不只被一個多邊形使用，賦予端點編號後，對於產生的多邊形的端點加以檢查，可決定此端點是否被儲存，因此端點在不同的多邊形不會被重複儲存。

儲存多邊形的端點時，採用雙向的聯結串列，串列的每個節點由6個欄位組成，包括二個指標，端點的座標及編號，聯結串列的詳細構造如圖 4-8。至於多邊形，因為它的產生與正立方體頂點的位能有關，故採用類似位能的儲存方式，記錄多邊形。設定一個赫序表，此表的每個索引可不只儲存一個正立方體的多邊形，但在記錄多邊形時，不需驗明此多邊形所屬的正立方體，而且多邊形經過三角化後，只要儲存三個頂點的編號即可，故赫序表的每個索引指著一個聯結串列，串列的每個節點表示一個多邊形，如圖 4-9。

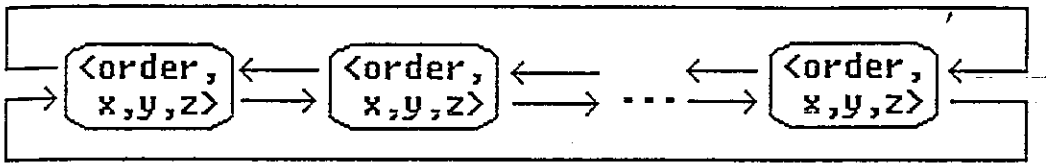


圖 4-8 多邊形端點的聯結串列

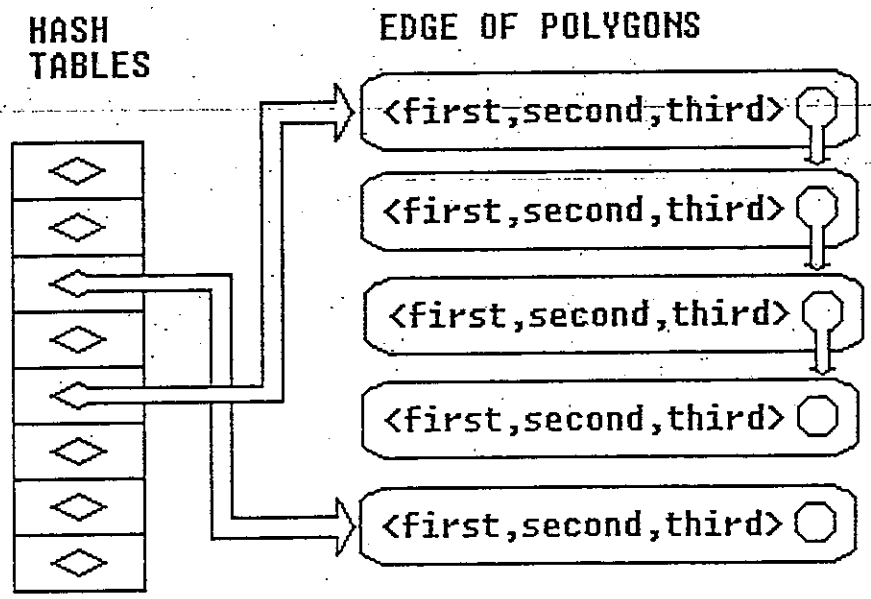


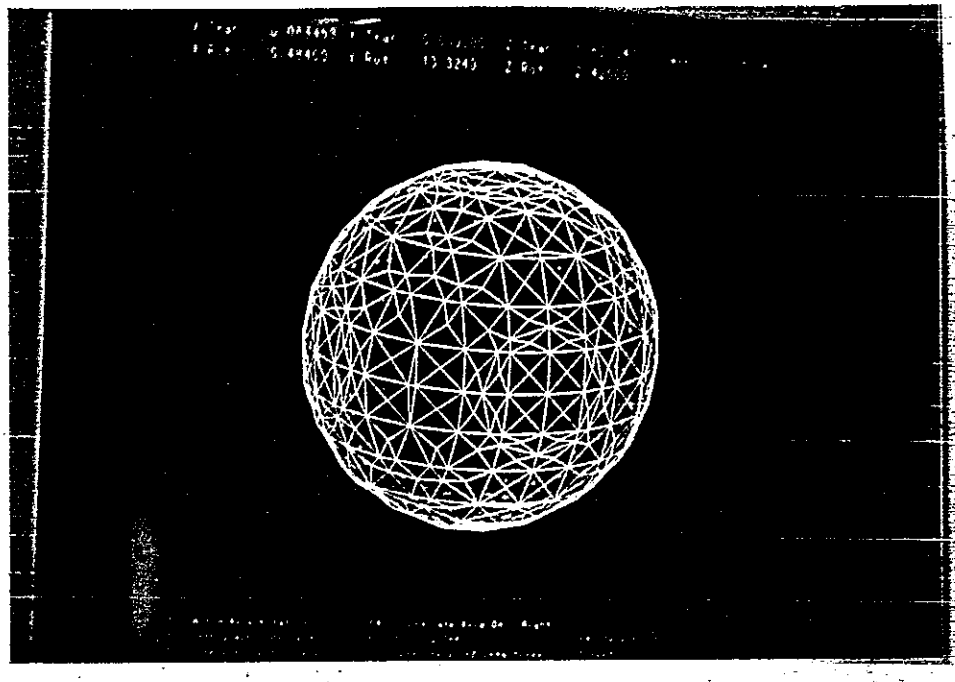
圖 4-9 多邊形各邊的赫序表

軟狀物體輪廓的所有多邊形產生後，多邊形的端點及邊分別儲存於如圖 4-8和圖 4-9的聯結串列及赫序表內。由前者的最後一個節點，得到總共使用的端點數目，再計

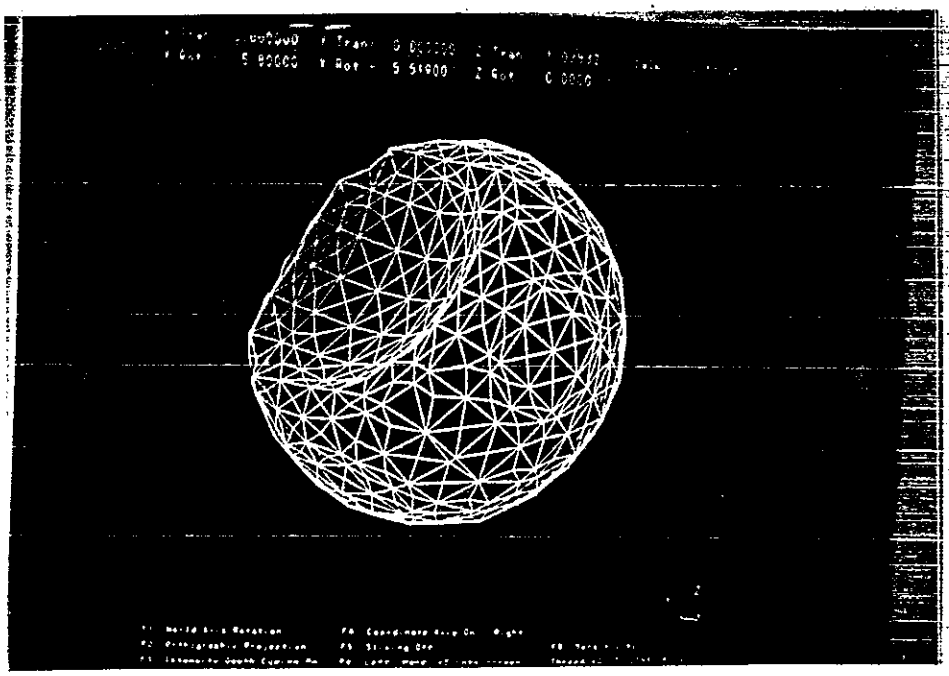
算赫序表內的多邊形，可得到多邊形的總數。確定這四類訊息後，產生符合套裝軟體 THREE-D 及 ANIMATOR需求的資料，接著將物體呈現於向量顯示器或掃瞄顯示器上，觀看它的輪廓。

第五章　實驗結果

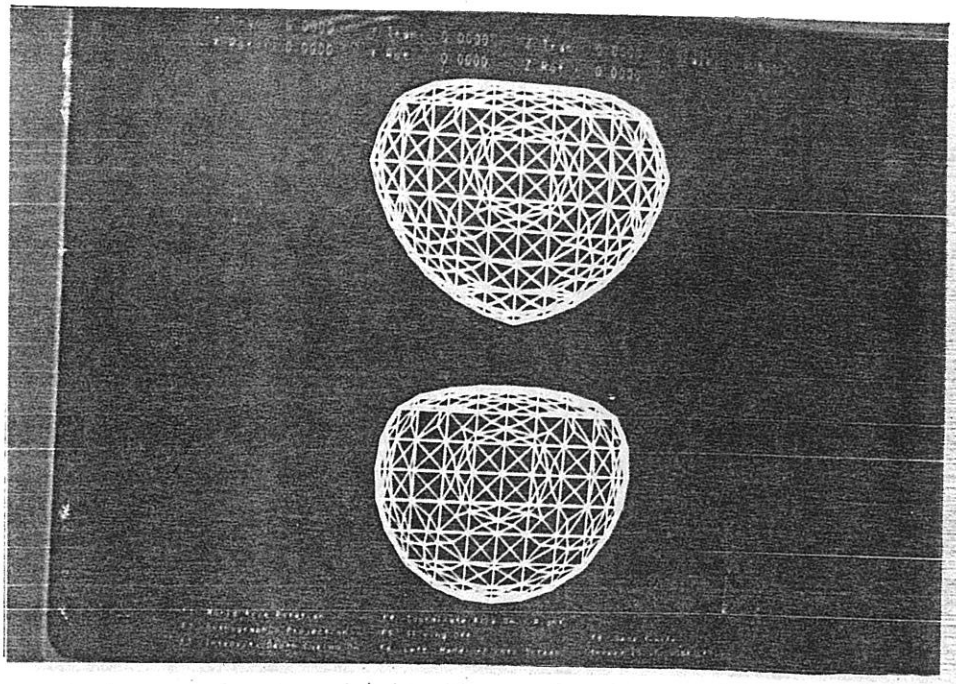
本章以照片說明代表軟狀物體外形的多邊形網孔。照片 1 為單一控制點產生的球體。照片 2 為球體受到外力作用，有部分區域凹陷。照片 3 及照片 4 均維持球體的體積不變。照片 3 由二部分組成，下端為球體被壓的形狀，上端為保持球體的體積不變，照片 4 模擬二顆水滴接近時，形狀的變化。它由二部分組成，左邊為保持體積不變，右邊的體積增加。照片 5 照片 6 模擬誇張的動作，鐵鎚敲打釘子，釘子不動，鐵鎚的握柄反而彎曲。照片 5 為鐵鎚的形狀，照片 6 為彎曲的鐵鎚。蝴蝶幼蟲為環節動物，全身由許多的環組成，照片 7 為單一的環，連接許多的環會形成蝴蝶幼蟲，如照片 8 。



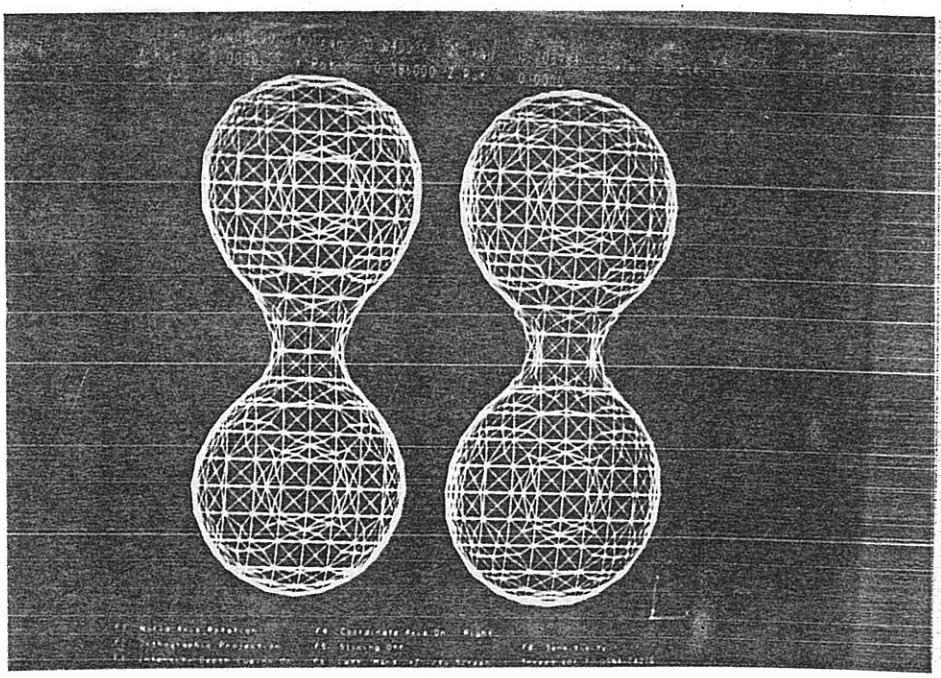
照片 1　球體



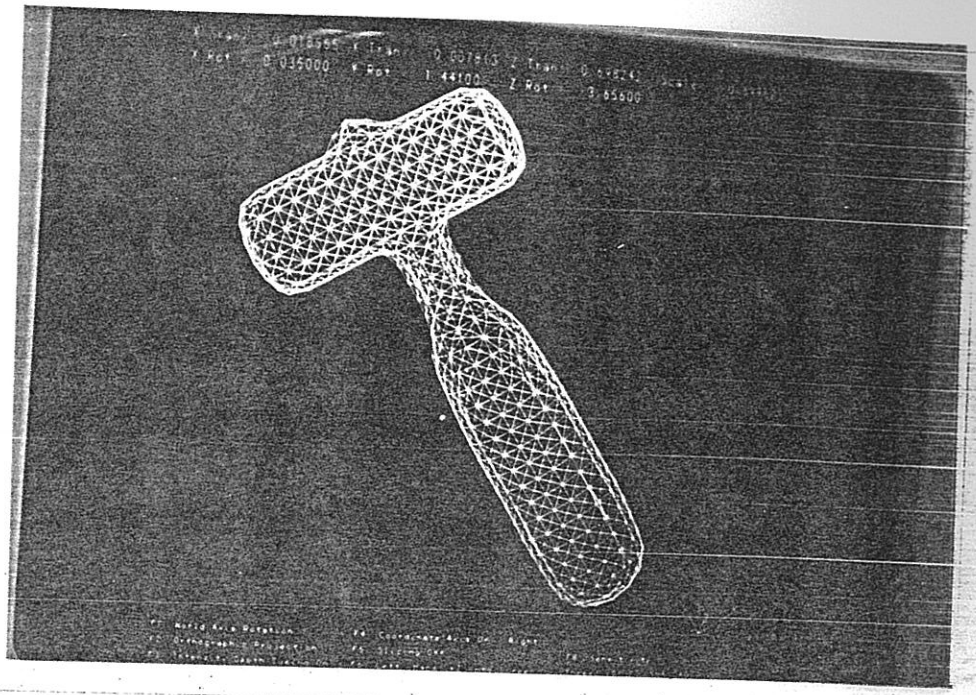
照片 2　凹陷的球體



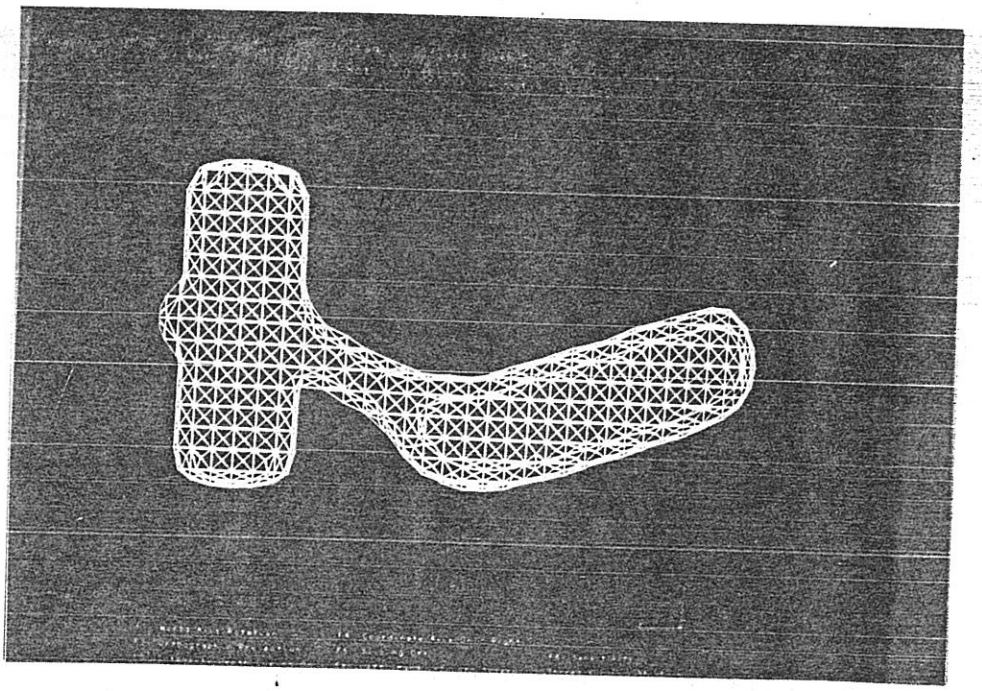
照片 3　體積改變及體積不變的球體(1)



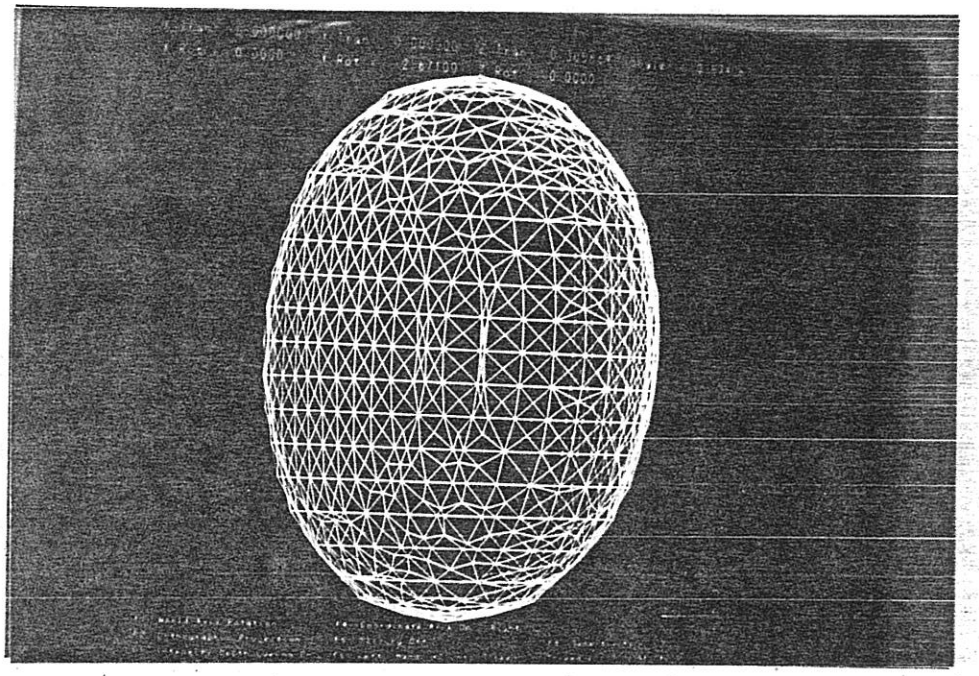
照片 4　體積改變及體積不變的球體(2)



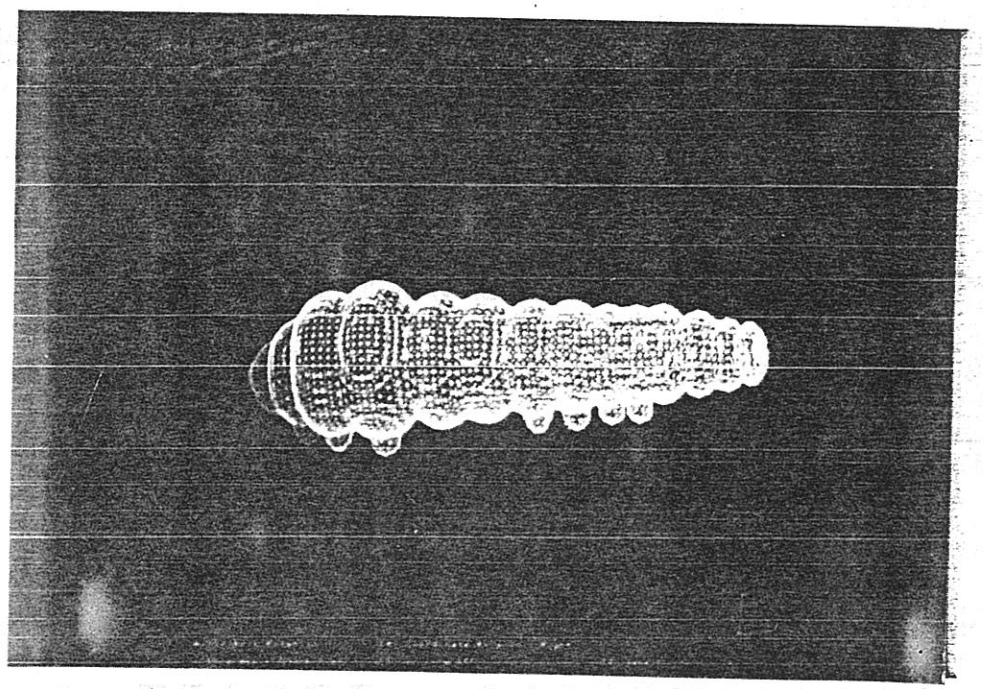
照片 5　鐵鎚



照片 6　彎曲的鐵鎚



照片 7　環



照片 8　蝴蝶幼蟲

第六章　結論與展望

水滴表面的水分子，具有相同的位能，形成球體狀的等位能外形。當二顆水滴互相接近時，位能分佈發生變化，等位能外形亦隨之變化。等位能外形的改變表示體積的變化，也就是質量的改變，但質量須維持不變，不能任意增加或減少，故等位能外形的體積應維持不變。這樣的做法，比較符合自然界的現象。

軟狀物體由許多控制點組成，每個控制點會產生球體狀的等位能外形，藉著球體的結合，形成物體的輪廓。再由不同大小的球體，成功地塑造鐵鎚及蝴蝶幼蟲的外形；調整控制點間的空間關係，模擬鐵鎚握柄彎曲的情形。

在塑造物體外形時，要指定控制點的位置，由於使用者的界面未完成，目前均從檔案讀入控制點。調整軟狀物體控點的位置，才能達到形狀的改變，若軟狀物體由許多控制點組成，選出相關的控制點並調整它們的位置，這是件繁雜的工作。由於每個控制點並沒有指定顏色，它所產生的球體狀等位能外形是相同的顏色，造成物體都是相同的顏色，顯得非常單調。故使用者的界面及控制點的屬性，都是值得繼續研究的主題。

未來研究的方向包括：

1.建立界面：

本論文的系統架構分成四部分，後二部分為套裝軟體，第二部分為產生軟狀物體外形的多邊形網孔，已經完成，只剩第一部分的界面未完成。在設計界面時，最好採用Macintosh 的方法，使用許多視覺符號，能讓不懂電腦的畫家，畫出他所需要的圖案。界面還可以定義物體的彎曲程度，產生誇張的效果，以製作卡通動畫。

2.增加控制點的屬性：

目前控制點的屬性包括：位置、作用範圍、外力。至於物體的顏色，紋理等性質，都可加入，若是液體，不妨考慮它的附著力，黏度等性質。

3.增加位能函數：

目前只使用一種位能函數，產生球體狀的等位能外形，藉著球體的結合形成物體的外形。不妨考慮多種的位能函數，每種位能函數會產生不同形狀的等位能外形，再由這些等位能外形規劃物體的外觀。

參 考 文 獻

Appel A (1968) Some Techniques for Shading Machine Rendering of Solids. Proc. SJCC, AFIPS, Reston, Va. 32:37-45

Blinn J (1982) A Generation of Algebraic Surface Drawing. ACM Transaction on Graphics 1(3):235-256

Catmull E (1975) Computer Display of Curved Surface. Proc. IEEE Conf, on Computer Graphics Pattern Recognition and Data Structure, pp11-17

Halas J, Manvell R (1968) The Technique of Film Animation. Hostings House, New York

Magnenat-Thalmann N, Thalmann D (1985) Computer Animation: Theory and Practice. Springer, Tokyo

Magnenat-Thalmann N, Thalmann D (1987) Image

Synthesis:Theory and Practice. Springer, Tokyo

Requicha AAG, Voelcker HB(1977) Boolean operations in solid modeling. Tech. Memo No26, Production Automation Project, University of Rochester.

Wyvill B, McPheeters C, Wyvill G (1986a) Animating Soft Object. The Visual Computer 2:235-242

Wyvill B, Wyvill G (1989) Field Functions for Implicit Surfaces. The Visual Computer 5:75-82

Wyvill B, McPheeters C, Wyvill G (1986b) Soft Objects. Advanced Computer Graphics, Proc CG Tokyo 1986, pp113-128

王興華 (1989)，"產生動態紋理方法之研究"
碩士論文，淡江大學，78年6月

李怡嚴 (1968)，物理學，東華書局，台北

盧衍祺 (1972)，流體力學，東華書局，台北

第 5 章 程式清單

程式共分為 6 個檔案，各個檔案的名稱及功能分別為：

Part_2.mak	:	make 檔，編譯及聯結其它檔案。
header.h	:	定義常數，結構表及聯結串列的資料結構；宣告副程式的型態（type）。
object.c	:	讀入控制點及呼叫其它的副程式，以產生多邊形網孔。
seed.c	:	使能動態的設定及多邊形的產生。
table.c	:	處理四種結構，三個結構表，一個聯結串列之資料的加入、找尋等的工作。
volume.c	:	維持體積的不變。

5.1 敘述 header.h 及 object.c 檔的內容，5.2、5.3、5.4 分別敘述 seed.c、table.c、volume.c 檔的內容。在各節中，首先敘述宣告於相關檔案內的副程式名稱及功能，再用圖表說明這些宣告的副程式所

引用(reference)的副程式，若沒有引用其它的任何副程式，則圖表內宣告副程式的右側為空白。詳細列出副程式之間的引用關係，以達到易於閱讀、維護的目的。

說明這些副程式時，使用一些符號。當英文名稱的尾端附上（），表示英文名稱為副程式的名稱；若英文名稱完全在「」或（）之內，表示英文名稱為副程式輸入引數(parameter)的名稱；若英文名稱的尾端附上[]，表示它為一維的向量，通常是陣列表；若英文名稱尾端附上[][]，表示它為二維的向量。

header.h 會告訴所有副程式的型態。C 語言在編譯時，遇到沒有宣告過的副程式，都將它的型態認成整數。若此副程式的型態並非整數，編譯此副程式的主體（body）時，會產生重複宣告的錯誤，現在 header.h 檔說明所有副程式的型態，並在其它的 .c 檔包含 header.h 檔，以避免上述的錯誤。

表 5-1 為宣告於 object.c 檔的副程式，及它們所引用的副程式，宣告之副程式的名稱及功能分別為：

main()：呼叫其它副程式，以產生多邊形網孔。

data_read()：讀入型狀物件控制點的座標、作用範圍、作用力，存入結構表 CTRLpoint[] 及聯結串列 CTRLlist

表5-1 →

控制點的值存在於資料檔內；若有一界面能讓使用者非常方便地指定控制點的屬性，使著可利用本程式產生數狀物體。main() 內，詢問控制點產生的球形狀等位能外形是否維持體積不變，更好的做法是把它設成程式的選擇項(option)。

5.2 seed.c 檔的副程式

seed.c 檔案內的副程式，主要是達成二個功能，計算位能及產生多邊形，宣告於 seed.c 檔的副程式及引用之副程式列於表5-2，的名稱及功能分述如下：（副程式）

fieldvalue()：　輸入欲計算位能之位置(x,y,z)，輸

出此位置的位能。

field_func() : 欲計算位能之位置至控制點的距離，除以控制點的作用範圍，可得 'value'，輸入 'value'，計算此控制點的位能貢獻量，使用此副程式時，須滿足 0 ≤ 'value' ≤ 1 的條件。

hash_func() : 輸入正立方體的編號 (g-i, g-j, g-k)，求得雜湊表的索引。

poly-edge() : 當位能外形通過正立方體會產生多邊形，多邊形的各邊都經過正立方體的面。輸入正立方体的編號 (g-i, g-j, g-k) 及指定正立方体的一個面 'face'，求得此面的四個頂點及落在此面的多邊形邊的總數，以供 poly-utxp() 使用。

poly-head() : 多邊形包含許多端點，從任一端點開始，可經過其它的端點而走回至出發的端點，位於編號 (g-i, g-j, g-k) 的正立方体的多邊形，從此多邊形挑一個端點，視之為出發的端點，以便在 stage_two() 建立多邊形。

poly-utxp() : 副程式 poly_head() 的輸出，做為 poly-utxp()

的輸入，計算多面形端點的座標，呼叫 point_add()，將端點加至聯結串列，同時取得此端點的編號，存入 edge_vtxp[][]，以便在 stage_two() 產生多面形。

seed_cube() : 搜尋探到點的所在位置，決定種子正立方體，再將它們的編號儲存於佇列 'Queue'。

stage_one() : 從佇列 'Queue' 內取得一個正立方體，檢驗它周圍的六個正立方體，以便產生位於編號 $(i-1, j-1, k-1)$ 正立方體內的多面形。以正立方體的各面，使用 poly_edge() 求得多面形的端點反迴，並對多面形進行三角化，以保持多面形是位於同一平面。

standard() : 多面形的端點反迴分別儲存於聯結串列 POINTlist 及稀疏表 POLYgons[]，從這二類的資料結構產生符合真善美 ANIMATOR 及 THREE-D 的資料格式。

stripe_write() : 在 stage_two() 產生多面形時，若遇到無法產生的情況，通常是有些情況未列入考慮，當發生這種情形時，

使用此副程式列印出 poly_edge() 求得的多邊形的邊，以供檢查

test_seed()： 輸入包含控制點的正立方体三編號 (g-i, g-j, g-k)，沿著 'drt' 所指的方向，找尋種子正立方体。

value_trnct()： 對位能值加以調整，以避免產生面積為零的多邊形。

加入表5-2

table.c 檔的副程式

table.c 檔內的副程式，主要是處理四種資料結構的插入、找尋或去除資料，表5-3為包含於 table.c 的副程式及其引用的副程式，副程式名稱分成二部分，每部分的名稱合起來，即成為此副程式的功能。每部分均有各自的含意，分述如下：

ctrl_…… ()： 處理控制点的資料結構。
plgn_…… ()： 處理多邊形邊的資料結構。
pord_…… ()： 處理多邊形端點的資料結構。
vtxp_…… ()： 處理正立方体頂点位能的資料結構。
……_add ()： 加入資料至相關的資料結構。
……_alloc ()： 向系統要求記憶体以儲存資料。
……_chock ()： 檢查聯結串列每個節點的欄位 fingerprint

，以便持續檢的完整。

...insert()：加節點至聯結串列。

...release()：歸還特定表及聯結串列使用的記憶體。

...write()：資料寫至檔案以供檢查。

若副程式的名稱為 ctrl_alloc()，則此副程式的功能為向系統要求一個記憶體，儲存控制點的資料。

table.c 檔還包含其它的副程式，它們的名稱及功能分別為：

ctrlt_add()：控制点同時加入聯結串列 CTRList，以便檢查輸入之控制點是否正確及方便後續的處理步驟。

ctrlt_write()：列印上述聯結串列之資料。

creatsfnam()：提供一個名字，做為檔案的名稱，這些檔案通常儲存相關的資料，以便確定未列入考慮的情形。

point_add()：輸入多面形之端点(x,y,z)，若它已經存在聯結串列 POINTlist，輸出它的編號'order'，若它不在聯結串列內，將它加入聯結串列，賦予一個不重複的編號，再輸出它的編號。

point_find()：輸入一個號碼'order'，找尋此號碼的多面形端點，若成功，輸出含此編號之節点的位置，若不成功，輸出'NULLPOINT'。

table_initial()：給定標存表及聯結串列的初值。

vtxp_corner()：輸入正立方體的編號(g-i, g-j, g-k)，計算此正立方体 8 個頂點的位能，輸出位於等位能外形內部的頂點數目。由它的輸出，可判斷等位能外形是否會通過正立方体(g-i, g-j, g-k)。

找尋等位能外形的演算法需使用佇列，處理佇列的副程式為：

vtxp_q_a()：編号為(g-i, g-j, g-k)的正立方體加至佇列的尾端。

vtxp_g_d()： 從佇列的前端取出資料，並將它佔用的記憶體歸還給系統。

vtxp_g_w()： 列印佇列的內容至檔案，以檢查資料是否正確。

vtxp_w_check()： 輸入正立方體的編號(g-i, g-j, g-k)，列印此正立方體頂點的位能至檔案，以檢查資料是否正確。

(加入表 5-3)

5.4 volume.c 檔的副程式

volume.c 檔內的副程式，主要是維持體積的不變。表 5-4 為宣告 volume.c 檔的副程式及它們引用的副程式，宣告之副程式的名稱及功能分別為：

examine_bound()： 確定作用範圍會相互影響的控制點，找出其中的二個 'p_head' 及 'p_work'，供 get_bound() 及 pass_edge() 使用。

get_bound()： 輸入 examine_bound() 求得的 'p_head' 及 'p_work'，求得相互影響區域的範圍，存入 range[][]，此區域可視為由許多層的 x-y 平面組成。

pass_edge()： 計算並記錄相互影響區域內，控制點 'p_head' 及 'p_work' 的位能貢獻量，並求

得在相互影響區域內，每個 x-y 平面上，等位能外形通過的範圍 'min_i'、'max_i'，'min_j'，'max_j'，再使用 spread_field() 以達到體積的叉差流經的 x-y 平面。

spread_field()： 求在相互影響範圍內，確定控制點 'ctrl_work' 產生的等位能外形通過的所有正方形的面，每面的一個頂點位於等位能外形之內，求得這頂点的位能變化量，總合所有的位能變化量，再將它反映至等位能外形通未改變的區域。'ctrl_work' 表示上面的控制點，'key' 表示 'ctrl_work' 為 examine_bound() 求得的 'p_head' 或 'p_work'。'level' 表示特定的 x-y 平面，'min_i'，'min_j'，'max_i'，'max_j' 的意義，如 pass_edge() 所述。

spread_error()： 相互影響範圍內的第 'level' 層上的正方形頂點，判斷它們是否會落於 'ctrl_work' 產生的等位能外形之內。輸入參數的含意與 spread_field() 相同，將頂點的資料寫至檔案以便檢查。

true_field()：結構 UTXPdata 的 field 欄位，儲存三立方體頂點的位能。為維持體積之差而調整的位能值存於 field 欄位，但它是儲存真正的位能，故需使用 fieldvalue() 求得的位能，加至 field 欄位，才是真正的位能。

加入表5-6

宣告的副程式	引用的副程式
main()	Table_initial()
	data_read()
	examine_bound()
	True_field()
	stage_one()
	standard()
data_read()	ctrl上_add()
	ctrl_add()

表 5-1 object.c 檔宣告及引用的副程式

호출하는 함수	호출되는 함수
field value ( )	hash_func ( )
	field_func ( )
	value_struct ( )
field_func ( )	
hash_func ( )	
poly_edge ( )	vtxp_find ( )
	poly_vtxp ( )
poly_head ( )	stripe_write ( )
	port_write ( )
	vtxp_w_check ( )
	plgn_write ( )
poly_vtxp ( )	vtxp_find ( )
	port_add ( )
seed_cube ( )	test_seed ( )
stage_one ( )	seed_cube ( )
	vtxp_g_w ( )
	vtxp_find ( )
	vtxp_corner ( )
	stage_two
	vtxp_g_d ( )

stage_two	poly_edge()
	poly_head()
	stripe_write()
	point_write()
	utxp_w_check()
	plan_write()
	point_find()
	utxp_find()
	plan_add()
standard()	
stripe_write()	
test_seed()	utxp_find()
	fieldvalue()
	utxp_add()
value_trnct()	

表 5-2　seed.c 檔宣告及引用的副程式

主程序的副程式	副程式自己的副程式
ctrlIf_add()	ctrl_alloc()
ctrlIf_write()	
ctrl_add()	hash_func()
	ctrl_alloc()
	ctrl_insert()
ctrl_alloc()	
ctrl_check()	
ctrl_insert()	ctrl_check()
ctrl_release()	
ctrl_write()	createfname()
createfname()	
plgn_add()	plgn_alloc()
	plgn_check()
	hash_func()
plgn_alloc()	
plgn_release()	
plgn_write()	createfname()
pont_add()	pont_alloc()
	pont_insert()
pont_alloc()	
pont_check()	

cont_find()	
cont_insert()	cont_check()
cont_release()	
cont_write()	creatfname()
table_inital()	utxp_alloc()
	cont_alloc()
utxp_add()	utxp_alloc()
	hash_func()
utxp_alloc()	
utxp_check()	
utxp_corner()	utxp_find()
	utxp_add()
utxp_find()	hash_func()
utxp_insert()	utxp_check()
utxp_g_a()	utxp_alloc()
	utxp_insert()
utxp_g_d()	
utxp_g_w()	
utxp_release()	
utxp_write()	creatfname()
utxp_w_check()	utxp_find()

表 5 - 3　table.c 檔宣告及引用的副程式

宣告的副程式	引用的副程式
examine_bound()	get_bound()
	pass_edge()
get_bound()	
pass_edge()	vtxp_find()
	vtxp_add()
	spread_field()
spread_field()	vtxp_find()
	vtxp_add()
spread_error()	create_fnalue()
	vtxp_find()
True_field()	fieldvalue()

表 5 - 2　volume.c 檔宣告及引用的副程式

宣告的副程式	引用的副程式
examine_bound()	set_bound()
	pass_edge()
get_bound()	
pass_edge()	vtxp_find()
	vtxp_add()
	spread_field()
spread_field()	vtxp_find()
	vtxp_add()
spread_error()	create_frame()
	vtxp_find()
True_field()	fieldvalue()

表 5 - 4　volume.c 檔宣告及引用的副程式

```
CFLAGS = -O -w
BIN    = ../bin
BACKUP = ../soft_backup
SNH    = SINICA:/usr/users/ps300/student/snh/soft

HDRS = header.h

OBJ1 = object.o
OBJ2 = seed.o
OBJ3 = table.o
OBJ4 = volume.o
OBJS = $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4)

animator:  $(HDRS) $(OBJS)
        cc $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4) -lm -o part_2
        mv part_2  $(BIN)
        cp *.mak *.h *.c $(BACKUP)
        rcp *.mak *.h *.c $(SNH)
```

```
/* subprogram : HEADER.H - header file for soft object ,          */

#include <math.h>
#include <stdio.h>

#define RADIUS          16
#define GRID            2
#define LENGTH          80
#define STACK_LIMIT     16
#define NULLCHAR        '\0'
#define X_SIZE          16
#define Y_SIZE          16
#define Z_SIZE          16
#define X_L             0
#define X_R             639
#define Y_I             479
#define Y_O             0
#define Z_U             479
#define Z_B             0
#define ON              1
#define OFF             0
#define PI              3.1415962
#define CIRCLE          (RADIUS/GRID)*4
#define ESC             0x1b
#define ESELON          0.01
#define CTRLsignature   0x0000
#define PLGNsignature   0x0002
#define PONTsignature   0x0003
#define VTXPsignature   0x0004
#define NULLCTRL        (struct CTRLdata *)0
#define NULLPLGN        (struct PLGNdata *)0
#define NULLPONT        (struct PONTdata *)0
#define NULLVTXP        (struct VTXPdata *)0
#define SQUARE(x)       ((x)*(x))

/* structure declaration for control point.                       */
struct CTRLdata {
                struct CTRLdata *down, *next;
                unsigned fingerprint;
                double x, y, z, radius;
                int sign;
                };

/* structure declaration for polygon.                             */
struct PLGNdata {
                struct PLGNdata *down;
                unsigned fingerprint;
                int first, second, third;
                };

/* structure declaration for point of polygon.                    */
struct PONTdata {
                struct PONTdata *from, *next;
                unsigned fingerprint;
                int order; double x, y, z;
                };
```

```
/* structure declaration for vertex.                                  */
struct VTXPdata {
                struct VTXPdata *down;
                unsigned fingerprint;
                int i, j, k, done;
                double field, hd_fld, wk_fld, ps_fld;
                };

/* declare in table.c                                                 */

void ctrllt_add ();
void ctrllt_write ();
void ctrl_add ();
struct CTRLdata *ctrl_alloc ();
int  ctrl_check ();
int  ctrl_insert ();
void ctrl_release ();
void ctrl_write ();

void createfname ();

void plgn_add ();
struct PLGNdata *plgn_alloc ();
int  plgn_check ();
void plgn_release ();
void plgn_write ();

int  pont_add ();
struct PONTdata *pont_alloc ();
int  pont_check ();
struct PONTdata *pont_find ();
int  pont_insert ();
void pont_release ();
void pont_write ();

void table_initial ();

void vtxp_add ();
struct VTXPdata *vtxp_alloc ();
int  vtxp_check ();
int  vtxp_corner ();
struct VTXPdata *vtxp_find ();
int  vtxp_insert ();
void vtxp_q_a ();
void vtxp_q_d ();
void vtxp_q_w ();
void vtxp_release ();
void vtxp_write ();
void vtxp_w_check ();

/* declare in seed.c                                                  */
double fieldvalue ();
double field_func ();
int hash_func ();
```

```
void poly_edge ();
int  poly_head ();
void poly_vtxp ();
void seed_cube ();
void stage_one ();
void stage_two ();
void standard ();
void stripe_write ();
void test_seed ();
double value_trnct ();

/* declare in volume.c                                              */
void examine_field ();
void get_bound ();
void pass_edge ();
void spread_field ();
void spread_error ();
void true_field ();

/* declare in object.c                                              */
void data_read ();

/*345678901234567890123456789012345678901234567890123456789012345678*/
/*        1       2       3       4       5       6                */
```

```
/* subprogram name : object.c                                          */
#include "header.h"

struct CTRLdata *CTRLpont[X_SIZE*Y_SIZE*Z_SIZE], *CTRLlist, *RayTrace;
struct PLGNdata *POLYgons[X_SIZE*Y_SIZE*Z_SIZE];
struct PONTdata *PONTlist;
struct VTXPdata *VTXPpont[X_SIZE*Y_SIZE*Z_SIZE], *Queue, *Q_Rear;

double Magic; int volume, grid_snh[CIRCLE][2];

/* MAIN -                                                              */
main ()
{
    char fname[LENGTH];

    table_initial ();
    data_read ();
    printf ("volume unchange ? (Y/N) : ");
    fgets (fname, LENGTH, stdin);
    volume = fname[0];
    if (volume == 'n') {
        printf ("Please wait\n");
        examine_bound ();
        true_field ();
        printf ("volume change\n");
    }
    stage_one ();
    standard ();
}

/*DATA_READ - read control point from file - 'ctrllist.snh'           */
/*
void data_read ()
{
    FILE *input; char fname[LENGTH], strg[LENGTH]; int sign;
    double x, y, z, radius;

    strcpy (fname, "ctrllist.snh");
    if ((input = fopen (fname, "r")) == NULL) {
        printf ("\ndata_read () : unable to open %s\n", fname);
        abort ();
    }

    CTRLlist = ctrl_alloc ();
    CTRLlist->next = CTRLlist->down = CTRLlist;
    fgets (strg, LENGTH, input); Magic = atof (strg);
    while (fgets (strg, LENGTH, input) != NULL) {
        if (strg[0] == '#') {
            printf ("%s", strg); continue;
        }
        sscanf (strg, "%10lf %10lf %10lf %8lf %2d",
                &x, &y, &z, &radius, &sign);
        ctrllt_add (x, y, z, radius, sign, OFF);
        ctrl_add (x, y, z, radius, sign);
        printf ("x=%12.8g y=%12.8g z=%12.8g r=%8.4g s=%2d\n",
                x, y, z, radius, sign);
```

```
    }
    fclose (input);
}

/*3456789012345678901234567890123456789012345678901234567890123456789012345678*/
/*       1         2         3         4         5         6         */
```

```
/* subprogram : SEED.C construct polygon mesh                          */

#include "header.h"
#define EDGE_NUM        10

extern struct CTRLdata *CTRLpont[X_SIZE*Y_SIZE*Z_SIZE], *CTRLlist;
extern struct PLGNdata *POLYgons[X_SIZE*Y_SIZE*Z_SIZE];
extern struct PONTdata *PONTlist;
extern struct VTXPdata *VTXPpont[X_SIZE*Y_SIZE*Z_SIZE], *Queue;
extern double Magic;
extern int volume;

int edge_vtx[EDGE_NUM+EDGE_NUM][3], edge_cnt, stripe[EDGE_NUM];

/* FIELDVALUE - calculate field value, (x,y,z) is real coordinate   */
double fieldvalue (x, y, z)
    double x, y, z;
{
    double value=0.0, dist, tempx, tempy, tempz, dtemp, ftemp;
    int r, s, t, sx, ex ,sy, ey, sz, ez, i, j, k, entry, itemp;
    struct CTRLdata *p_down, *p_next;

    r = (int)(x) / RADIUS;
    sx = r - 1, ex = r + 2; itemp = abs (X_R-X_L+1) / RADIUS;
    if (sx < 0) sx = 0; if (ex > itemp) ex = itemp;
    s = (int)(y) / RADIUS;
    sy = s - 1, ey = s + 2; itemp = abs (Y_I-Y_O+1) / RADIUS;
    if (sy < 0) sy = 0; if (ey > itemp) ey = itemp;
    t = (int)(z) / RADIUS;
    sz = t - 1, ez = t + 2; itemp = abs (Z_U-Z_B+1) / RADIUS;
    if (sz < 0) sz = 0; if (ez > itemp) ez = itemp;

    for (i=sx; i<ex; i++) {
        for (j=sy; j<ey; j++) {
            for (k=sz; k<ez; k++) {
                entry = hash_func (i, j, k); itemp = ON;
                p_down = CTRLpont[entry];
                while (p_down != NULLCTRL) {
                    if (p_down->x == i && p_down->y == j &&
                        p_down->z == k) {
                        itemp = OFF; break;
                    }
                    p_down = p_down->down;
                }
                if (itemp == ON) continue;
                p_next = p_down->next;
                while (p_next != NULLCTRL) {
                    tempx = (p_next->x - x) * (p_next->x - x);
                    tempy = (p_next->y - y) * (p_next->y - y);
                    tempz = (p_next->z - z) * (p_next->z - z);
                    dist = tempx + tempy +tempz;
                    dtemp = p_next->radius * p_next->radius;
                    if (dist > dtemp) {
                        p_next = p_next->next; continue;
```

```
                        }
                        ftemp = dist / dtemp;
                        dist = field_func (ftemp);
                        if (p_next->sign == ON) value += dist;
                        else value -= dist;
                        p_next = p_next->next;
                    }                       /* while (p_next != NULLCTRL) */
                }                               /* for (k=sz; k<ez; k++) */
            }                                   /* for (j=sy; j<ey; j++) */
        }                                       /* for (i=sx; i<ex; i++) */
    }

    ftemp = value;
    value = value_trnct (ftemp);
    ftemp = Magic - value;
    dtemp = value_trnct (ftemp);
    if (dtemp <= 0.01 && dtemp > 0.0)
        value = Magic;

    return value;
}

/* FIELD_FUNC : field function                                              */
/*                                                                          */
double field_func (value)
    double value;
{
    double work, temp;

    work = value;
    temp = (work - 2.0) * work + 1;
    return temp;
}

/* HASH_FUNC - calculate entry of hash table, (g_i, g_j, g_k)               */
/*             is grid point, not real coordinate.                          */
/*                                                                          */
int  hash_func (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    return (((g_i%X_SIZE) * Y_SIZE+g_j%Y_SIZE) * Z_SIZE+g_k%Z_SIZE);
}

/* POLY_EDGE - compute polygon edge on specific face of cube                */
/*                                                                          */
void poly_edge (g_i, g_j, g_k, face)
    int g_i, g_j, g_k, face;
{
    struct VTXPdata *p_work; double d_c_f;
    int gd[4][3], cn[4][3], count, corner, i, j, k, flag[8], snh[2];

    switch (face) {
      case 0 :                                              /* 0, 1, 5, 4 */
        gd[0][0]=g_i  ,gd[1][0]=g_i+1,gd[2][0]=g_i+1,gd[3][0]=g_i  ;
        gd[0][1]=g_j  ,gd[1][1]=g_j  ,gd[2][1]=g_j  ,gd[3][1]=g_j  ;
        gd[0][2]=g_k  ,gd[1][2]=g_k  ,gd[2][2]=g_k+1,gd[3][2]=g_k+1;
        break;
```

```
      case 1 :                                                  /* 1, 3, 7, 5 */
         gd[0][0]=g_i+1,gd[1][0]=g_i+1,gd[2][0]=g_i+1,gd[3][0]=g_i+1;
         gd[0][1]=g_j  ,gd[1][1]=g_j+1,gd[2][1]=g_j+1,gd[3][1]=g_j  ;
         gd[0][2]=g_k  ,gd[1][2]=g_k  ,gd[2][2]=g_k+1,gd[3][2]=g_k+1;
         break;
      case 2 :                                                  /* 3, 2, 6, 7 */
         gd[0][0]=g_i+1,gd[1][0]=g_i  ,gd[2][0]=g_i  ,gd[3][0]=g_i+1;
         gd[0][1]=g_j+1,gd[1][1]=g_j+1,gd[2][1]=g_j+1,gd[3][1]=g_j+1;
         gd[0][2]=g_k  ,gd[1][2]=g_k  ,gd[2][2]=g_k+1,gd[3][2]=g_k+1;
         break;
      case 3 :                                                  /* 2, 0, 4, 6 */
         gd[0][0]=g_i  ,gd[1][0]=g_i  ,gd[2][0]=g_i  ,gd[3][0]=g_i  ;
         gd[0][1]=g_j+1,gd[1][1]=g_j  ,gd[2][1]=g_j  ,gd[3][1]=g_j+1;
         gd[0][2]=g_k  ,gd[1][2]=g_k  ,gd[2][2]=g_k+1,gd[3][2]=g_k+1;
         break;
      case 4 :                                                  /* 2, 3, 1, 0 */
         gd[0][0]=g_i  ,gd[1][0]=g_i+1,gd[2][0]=g_i+1,gd[3][0]=g_i  ;
         gd[0][1]=g_j+1,gd[1][1]=g_j+1,gd[2][1]=g_j  ,gd[3][1]=g_j  ;
         gd[0][2]=g_k  ,gd[1][2]=g_k  ,gd[2][2]=g_k  ,gd[3][2]=g_k  ;
         break;
      case 5 :                                                  /* 4, 5, 7, 6 */
         gd[0][0]=g_i  ,gd[1][0]=g_i+1,gd[2][0]=g_i+1,gd[3][0]=g_i  ;
         gd[0][1]=g_j  ,gd[1][1]=g_j  ,gd[2][1]=g_j+1,gd[3][1]=g_j+1;
         gd[0][2]=g_k+1,gd[1][2]=g_k+1,gd[2][2]=g_k+1,gd[3][2]=g_k+1;
         break;
      default : return;
   }                                                        /* switch (face) */

   corner = count = 0; flag[0] = 1; d_c_f = 0.0;
   for (i=0; i<4; i++) {
        p_work = vtxp_find (gd[i][0], gd[i][1], gd[i][2]);
        d_c_f += p_work->field;
        if (p_work->field > Magic) { corner += flag[0]; count++; }
        flag[0] *= 2;
   }                                                    /* for (i=0; i<4; i++) */

   if (count == 0 || count == 4) return;

   if (count == 1 || count == 3 || (count == 2 && corner != 5 &&
       corner != 10)) {
        switch (corner) {
          case 1 : case 14 :
             flag[0]=3, flag[1]=0, flag[2]=1, flag[3]=0, snh[0]=0;
             break;
          case 2 : case 13 :
             flag[0]=0, flag[1]=1, flag[2]=2, flag[3]=1, snh[0]=1;
             break;
          case 4 : case 11 :
             flag[0]=1, flag[1]=2, flag[2]=3, flag[3]=2, snh[0]=2;
             break;
          case 8 : case  7 :
             flag[0]=2, flag[1]=3, flag[2]=0, flag[3]=3, snh[0]=3;
             break;
          case 3 :
             flag[0]=3, flag[1]=0, flag[2]=2, flag[3]=1, snh[0]=4;
             break;
```

```
            case 6 :
               flag[0]=0, flag[1]=1, flag[2]=3, flag[3]=2, snh[0]=5;
               break;
            case 12:
               flag[0]=1, flag[1]=2, flag[2]=0, flag[3]=3, snh[0]=6;
               break;
            case 9 :
               flag[0]=2, flag[1]=3, flag[2]=1, flag[3]=0, snh[0]=7;
               break;
         }                                          /* switch (corner) */

         for (i=0; i<4; i++)
              for (j=0; j<3; j++)
                   cn[i][j] = gd[flag[i]][j];

         count = face * 10 + snh[0];
         poly_vtxp (cn, count);
    }          /* count == 1 or 3, count == 2 && corner != 5 and 10 */

    if (count == 2 && (corner == 5 || corner == 10)) {
         if (d_c_f > Magic && corner == 5) i = 1;
         else if (d_c_f <= Magic && corner == 5) i = 2;
         else if (d_c_f > Magic && corner == 10) i = 3;
         else i = 4;

         switch (i) {
           case 1 : case 4 :
              flag[0]=0, flag[1]=1, flag[2]=2, flag[3]=1, snh[0]= 1;
              flag[4]=2, flag[5]=3, flag[6]=0, flag[7]=3, snh[1]= 3;
              break;
           case 2 : case 3 :
              flag[0]=3, flag[1]=0, flag[2]=1, flag[3]=0, snh[0]= 0;
              flag[4]=1, flag[5]=2, flag[6]=3, flag[7]=2, snh[1]= 2;
              break;
         }                                          /* switch (i) */

         for (k=0; k<2; k++) {
              count = k * 4;
              for (i=0; i<4; i++)
                   for (j=0; j<3; j++)
                        cn[i][j] = gd[flag[count + i]][j];
              count = face * 10 + snh[k];
              poly_vtxp (cn, count);
         }                                  /* for (k=0; k<2; k++) */
    }          /* if (count == 2 && (corner == 5 || corner == 10)) */
}

/* POLY_HEAD : get start point of stripe[]                          */
/*                                                                  */
int  poly_head (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    int i, j, flag, head, count;

    flag = OFF;
    for (i=0; i<edge_cnt; i++) {
```

```
        if (edge_vtx[i][2] == ON) continue;
        head = edge_vtx [i][0]; count = 0;
        for (j=0; j<edge_cnt; j++) {
             if (edge_vtx[j][2] == ON) continue;
             if (edge_vtx[j][0] == head) count++;
        }
        switch (count) {
             case 1 : flag = ON; break;
             case 2 : return i; break;
             default: stripe_write (g_i, g_j, g_k, 0);
                      pont_write (999); vtxp_w_check (g_i, g_j, g_k);
                      plgn_write (999);
                      printf ("Error in finding polygon head\n");
                      abort ();
                                                  /* switch (count) */
        }
                                          /* for (i=0; i<edge_cnt; i++) */
    }
    if (flag == ON) {
        i = 0;
        while (i < edge_cnt && edge_vtx[i][2] == ON) i++;
        return i;
    } else
        return -1;

}

/* POLY_VTXP : compute vertex of polygon on specific position        */
/*
void poly_vtxp (cn, post)
    int cn[][3], post;
{
    double fd[3], vt[2][3]; struct VTXPdata *p_work; int i, j, k;

    for (i=0; i<2; i++) {
        j = i * 2;
        p_work = vtxp_find (cn[j  ][0], cn[j  ][1], cn[j  ][2]);
        fd[i] = p_work->field;
        p_work = vtxp_find (cn[j+1][0], cn[j+1][1], cn[j+1][2]);
        fd[2] = p_work->field;
        if (fd[i] > Magic && fd[2] <= Magic) {
            fd[i] = (Magic - fd[2]) / (fd[i] - fd[2]);
            fd[i] = 1 - fd[i];
        } else
            fd[i] = (Magic - fd[i]) / (fd[2] - fd[i]);
        fd[i] *= (double)(GRID);
                                          /* for (i=0; i<2; i++) */
    }
    for (i=0; i<2; i++) {
        k = i * 2;
        for (j=0; j<3; j++)
            vt[i][j] = (double)(cn[k][j] * GRID);
    }

    switch (post) {
        case  0 : vt[0][2] -= fd[0]; vt[1][0] -= fd[1]; break;
        case  1 : vt[0][0] += fd[0]; vt[1][2] -= fd[1]; break;
        case  2 : vt[0][2] += fd[0]; vt[1][0] += fd[1]; break;
        case  3 : vt[0][0] -= fd[0]; vt[1][2] += fd[1]; break;
```

```
        case  4 : vt[0][2] -= fd[0]; vt[1][2] -= fd[1]; break;
        case  5 : vt[0][0] += fd[0]; vt[1][0] += fd[1]; break;
        case  6 : vt[0][2] += fd[0]; vt[1][2] += fd[1]; break;
        case  7 : vt[0][0] -= fd[0]; vt[1][0] -= fd[1]; break;

        case 10 : vt[0][2] -= fd[0]; vt[1][1] -= fd[1]; break;
        case 11 : vt[0][1] += fd[0]; vt[1][2] -= fd[1]; break;
        case 12 : vt[0][2] += fd[0]; vt[1][1] += fd[1]; break;
        case 13 : vt[0][1] -= fd[0]; vt[1][2] += fd[1]; break;
        case 14 : vt[0][2] -= fd[0]; vt[1][2] -= fd[1]; break;
        case 15 : vt[0][1] += fd[0]; vt[1][1] += fd[1]; break;
        case 16 : vt[0][2] += fd[0]; vt[1][2] += fd[1]; break;
        case 17 : vt[0][1] -= fd[0]; vt[1][1] -= fd[1]; break;

        case 20 : vt[0][2] -= fd[0]; vt[1][0] += fd[1]; break;
        case 21 : vt[0][0] -= fd[0]; vt[1][2] -= fd[1]; break;
        case 22 : vt[0][2] += fd[0]; vt[1][0] -= fd[1]; break;
        case 23 : vt[0][0] += fd[0]; vt[1][2] += fd[1]; break;
        case 24 : vt[0][2] -= fd[0]; vt[1][2] -= fd[1]; break;
        case 25 : vt[0][0] -= fd[0]; vt[1][0] -= fd[1]; break;
        case 26 : vt[0][2] += fd[0]; vt[1][2] += fd[1]; break;
        case 27 : vt[0][0] += fd[0]; vt[1][0] += fd[1]; break;

        case 30 : vt[0][2] -= fd[0]; vt[1][1] += fd[1]; break;
        case 31 : vt[0][1] -= fd[0]; vt[1][2] -= fd[1]; break;
        case 32 : vt[0][2] += fd[0]; vt[1][1] -= fd[1]; break;
        case 33 : vt[0][1] += fd[0]; vt[1][2] += fd[1]; break;
        case 34 : vt[0][2] -= fd[0]; vt[1][2] -= fd[1]; break;
        case 35 : vt[0][1] -= fd[0]; vt[1][1] -= fd[1]; break;
        case 36 : vt[0][2] += fd[0]; vt[1][2] += fd[1]; break;
        case 37 : vt[0][1] += fd[0]; vt[1][1] += fd[1]; break;

        case 40 : vt[0][1] += fd[0]; vt[1][0] -= fd[1]; break;
        case 41 : vt[0][0] += fd[0]; vt[1][1] += fd[1]; break;
        case 42 : vt[0][1] -= fd[0]; vt[1][0] += fd[1]; break;
        case 43 : vt[0][0] -= fd[0]; vt[1][1] -= fd[1]; break;
        case 44 : vt[0][1] += fd[0]; vt[1][1] += fd[1]; break;
        case 45 : vt[0][0] += fd[0]; vt[1][0] += fd[1]; break;
        case 46 : vt[0][1] -= fd[0]; vt[1][1] -= fd[1]; break;
        case 47 : vt[0][0] -= fd[0]; vt[1][0] -= fd[1]; break;

        case 50 : vt[0][1] -= fd[0]; vt[1][0] -= fd[1]; break;
        case 51 : vt[0][0] += fd[0]; vt[1][1] -= fd[1]; break;
        case 52 : vt[0][1] += fd[0]; vt[1][0] += fd[1]; break;
        case 53 : vt[0][0] -= fd[0]; vt[1][1] += fd[1]; break;
        case 54 : vt[0][1] -= fd[0]; vt[1][1] -= fd[1]; break;
        case 55 : vt[0][0] += fd[0]; vt[1][0] += fd[1]; break;
        case 56 : vt[0][1] += fd[0]; vt[1][1] += fd[1]; break;
        case 57 : vt[0][0] -= fd[0]; vt[1][0] -= fd[1]; break;

    }                                                   /* switch (post) */

    p_work = vtxp_find (cn[1][0], cn[1][1], cn[1][2]);
    if (cn[1][0] == cn[3][0] && cn[1][1] == cn[3][1] &&
        cn[1][2] == cn[3][2] && p_work->field <= Magic)
         for (i=0; i<3; i++) {
```

```
                fd[2] = vt[0][i], vt[0][i] = vt[1][i], vt[1][i] = fd[2];
        }

    if (vt[0][0] == vt[1][0] && vt[0][1] == vt[1][1] &&
        vt[0][2] == vt[1][2]) return;
    for (i=0; i<2; i++)
         for (j=0; j<3; j++)
              vt[i][j] = value_trnct (vt[i][j]);
    i = pont_add (vt[0][0], vt[0][1], vt[0][2]);
    j = pont_add (vt[1][0], vt[1][1], vt[1][2]);
    edge_vtx[edge_cnt][0] = i, edge_vtx[edge_cnt][1] = j;
    edge_cnt++;
}

/* SEED_CUBE : insert all seed cube to VTXPdata-type Queue          */
/*                                                                  */
void seed_cube ()
{
    struct CTRLdata *ctrl_head, *ctrl_tail;
    int g_i, g_j, g_k, i;

    ctrl_tail = CTRLlist; ctrl_head = ctrl_tail->next;
    while (ctrl_head != ctrl_tail) {
         g_i = (int)(ctrl_head->x) / GRID;
         g_j = (int)(ctrl_head->y) / GRID;
         g_k = (int)(ctrl_head->z) / GRID;
         for (i=0; i<6; i++) test_seed (g_i, g_j, g_k, i);
         ctrl_head = ctrl_head->next;
    }                              /* while (ctrl_head != ctrl_tail) */
}

/* STAGE_ONE : find all cube intersected with iso-surface, then     */
/*             call STAGE_TWO generating standard file              */
void stage_one ()
{
    struct VTXPdata *p_work;
    int i, j, k, sx, ex, sy, ey, sz, ez, itemp, count;

    seed_cube ();
    if (Queue->down == Queue) {
         printf ("stage_one () : queue is empty\n");
         abort ();
    }

    vtxp_q_w ();
    while (Queue->down != Queue) {
         p_work = Queue->down;
         sx = p_work->i - 1, ex = p_work->i + 2;
         itemp = abs (X_R - X_L + 1) / GRID;
         if (sx < 0) sx = 0; if (ex > itemp) ex = itemp;
         sy = p_work->j - 1, ey = p_work->j + 2;
         itemp = abs (Y_I - Y_O + 1) / GRID;
         if (sy < 0) sy = 0; if (ey > itemp) ey = itemp;
         sz = p_work->k - 1, ez = p_work->k + 2;
         itemp = abs (Z_U - Z_B + 1) / GRID;
         if (sz < 0) sz = 0; if (ez > itemp) ez = itemp;
```

```
        for (i=sx; i<ex; i++)
             for (j=sy; j<ey; j++)
                  for (k=sz; k<ez; k++) {
                       p_work = vtxp_find (i, j, k);
                       if (p_work != NULLVTXP && p_work->done == ON)
                            continue;
                       count = vtxp_corner (i, j, k);
                       if (count == 0 || count == 8) continue;
                       vtxp_q_a (i, j, k);
                       p_work = vtxp_find (i, j, k);
                       p_work->done = ON;
                  }
        p_work = Queue->down;
        stage_two (p_work->i, p_work->j, p_work->k);
        vtxp_q_d ();
     }                                  /* while (Queue->down != Queue) */
}

/* STAGE_TWO : generating polygon                                        */
/*                                                                       */
void stage_two (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    int i, j, count, start, next, center, i_temp;
    double d_c_x, d_c_y, d_c_z, d_temp; struct PONTdata *p_work;
    struct VTXPdata *vtxp_work; double d_f[2]; int v_i,v_j,v_k;

    for (i=0; i<EDGE_NUM+EDGE_NUM; i++)
         for (j=0; j<3; j++) edge_vtx[i][j] = 0;
    edge_cnt = 0;
    for (i=0; i<6; i++) poly_edge (g_i, g_j, g_k, i);
    count = edge_cnt;

    for (i=0; i<edge_cnt; i++) {
         p_work = pont_find (edge_vtx[i][0]);
         if (p_work == NULLPONT) {
              stripe_write (g_i, g_j, g_k, 4); pont_write (999);
              vtxp_w_check (g_i, g_j, g_k); plgn_write (999);
              printf("stage_two():Error in deciding point number/n");
              printf("Press any key to continue\n"); getchar();
              return ;
         }                                  /* if (p_work == NULLPONT) */
    }                                   /* for (i=0; i<edge_cnt; i++) */

    while (count > 0) {
         for (i=0; i<EDGE_NUM; i++) stripe[i] = 0;
         i = poly_head (g_i, g_j, g_k);
         if (i == -1) {
              stripe_write (g_i, g_j, g_k, 1); pont_write (999);
              vtxp_w_check (g_i, g_j, g_k); plgn_write (999);
              printf ("Error in construction polygon, 1\n");
              abort ();
         }
         start = stripe[0] = edge_vtx[i][0];
         next  = stripe[1] = edge_vtx[i][1];
```

```
        j = 2, count--,     edge_vtx[i][2] = ON;
        while (start != next && j < 10) {
             i = 0;
             while (i < edge_cnt && (edge_vtx[i][2] == ON ||
                    edge_vtx[i][0] != next)) i++;
             if (i == edge_cnt) {
                  stripe_write (g_i, g_j, g_k, 2); pont_write (999);
                  vtxp_w_check (g_i, g_j, g_k); plgn_write (999);
                  printf ("Error in construction polygon, 2\n");
                  abort ();
             }
             next = stripe[j] = edge_vtx[i][1];
             edge_vtx[i][2] = ON; count--; j++;
                                  /* while (start != next && j < 10) */
        }

        switch (j) {
           case 3 :
             for (i=0; i<2; i++) {
                  p_work = pont_find (stripe[i]);
                  if (p_work == NULLPONT) {
                       stripe_write (g_i, g_j, g_k, i+1);
                       vtxp_w_check (g_i, g_j, g_k);
                       pont_write (999), plgn_write (999);
                       printf ("Error in construction polygon, 3\n");
                       abort ();
                  }
                  v_i = (int)(p_work->x) / GRID;
                  v_j = (int)(p_work->y) / GRID;
                  v_k = (int)(p_work->z) / GRID;
                  vtxp_work = vtxp_find (v_i, v_j, v_k);
                  if (vtxp_work == NULLVTXP) {
                       stripe_write (g_i, g_j, g_k, i+1);
                       vtxp_w_check (g_i, g_j, g_k);
                       pont_write (999), plgn_write (999);
                       printf ("v_i = %4d, v_j = %4d, v_k = %4d\n",
                               v_i, v_j, v_k);
                       printf ("Error in finding vertex, 4\n");
                       abort ();
                  }
                  d_f[i] = vtxp_work->field;
                                       /* for (i=0; i<2, i++) */
             }
             if (d_f[0] == Magic && d_f[1] == Magic) ;
             else {
                  stripe_write (g_i, g_j, g_k, i+1);
                  vtxp_w_check (g_i, g_j, g_k);
                  pont_write (999), plgn_write (999);
                  printf ("Error in construction polygon, 5\n");
                  abort ();
             }
             break;
           case 4 :
             plgn_add (g_i,g_j,g_k,stripe[0],stripe[1],stripe[2]);
             break;
           case 5 : case 6 : case 7 : case 8:
             d_c_x = d_c_y = d_c_z = 0.0; i_temp = j - 1;
             for (i=0; i<i_temp; i++) {
```

```
                    p_work = pont_find (stripe[i]);
                    if (p_work == NULLPONT) {
                         stripe_write (g_i, g_j, g_k, i+1);
                         vtxp_w_check (g_i, g_j, g_k);
                         pont_write (999), plgn_write (999);
                         printf ("Error in construction polygon, 6\n");
                         abort ();
                    }
                    d_c_x += p_work->x, d_c_y += p_work->y,
                    d_c_z += p_work->z;
                                        /* for (i=0; i<i_temp, i++) */
               }
               d_temp = (double)(i_temp);
               d_c_x /= d_temp, d_c_y /= d_temp, d_c_z /= d_temp;
               center = pont_add (d_c_x, d_c_y, d_c_z);
               for (i=0; i<i_temp; i++)
                  plgn_add (g_i,g_j,g_k,stripe[i],stripe[i+1],center);
               break;
            default :
               stripe_write (g_i, g_j, g_k, 3); pont_write (999);
               vtxp_w_check (g_i, g_j, g_k); plgn_write (999);
               printf ("Error in construction polygon, 7\n");
               abort ();
                                                  /* switch (j) */
         }
                                                /* while (count > 0) */
      }
   }

/* STANDARD : generate 'standard' file suited animator                */
/*                                                                    */
void standard ()
{
     struct PONTdata *pont_work; struct PLGNdata *plgn_work;
     FILE *output; char fname[LENGTH], fwork[LENGTH];
     int i, check = 0, pont_cnt, plgn_cnt, count;
     double d_x, d_y, d_z, t_x, t_y, t_z;
     struct CTRLdata *ctrl_head, *ctrl_tail;

     ctrl_tail = CTRLlist, ctrl_head = CTRLlist->next;
     d_x = d_y = d_z = 0.0; count = 0;
     while (ctrl_head != ctrl_tail) {
          if (ctrl_head->sign != ON) {
               ctrl_head = ctrl_head->next; continue;
          }
          d_x += ctrl_head->x, d_y += ctrl_head->y,
          d_z += ctrl_head->z, count++,
          ctrl_head = ctrl_head->next;
                                   /* while (ctrl_head != ctrl_tail) */
     }
     if (count == 0) {
          printf ("At least one key point should be positive\n");
          abort ();
                                                  /* if (count == 0) */
     }
     d_x /= ((double)(count)), d_y /= ((double)(count)),
     d_z /= ((double)(count));

     printf ("File name for 'standard' file : ");
     fgets (fname, LENGTH, stdin);
     while (fname[check] != '\n' && check < LENGTH) check++;
```

```
    fname[check] = NULLCHAR;

    if (volume == 'n')
         strcat (fname, "_c");
    else
         strcat (fname, "_n");

    pont_work = PONTlist->from; pont_cnt = pont_work->order;
    plgn_cnt = 0;
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
         plgn_work = POLYgons[i];
         while (plgn_work != NULLPLGN) {
              plgn_cnt++; plgn_work = plgn_work->down;
         }
    }

    for (count=0; count<2; count++) {
         strcpy (fwork, fname);
         if (count == 0)
              strcat (fwork, ".std");
         else
              strcat (fwork, ".pf");

         if ((output = fopen(fwork, "w")) == NULL) {
              printf ("\nstandard () : unable to open %s\n", fwork);
              abort ();
         }

         if (count == 0)
              fprintf (output, "data %6d %6d\n", pont_cnt, plgn_cnt);
         else {
              fprintf (output, "ball\n");
              fprintf (output, "%6d, %6d\n", pont_cnt, plgn_cnt);
         }

         pont_work = PONTlist->next;
         while (pont_work != PONTlist) {
             t_x = pont_work->x - d_x,
             t_y = pont_work->y - d_y,
             t_z = pont_work->z - d_z;
             if (count == 0)
                  fprintf(output,"%12.6g %12.6g %12.6g\n",t_x,t_y,t_z);
             else
                  fprintf(output,"%12.6g, %12.6g, %12.6g\n",t_x,t_y,t_z)
             pont_work = pont_work->next;
         }
         for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
           plgn_work = POLYgons[i];
           while (plgn_work != NULLPLGN) {
             fprintf (output, "3 %5d %6d %5d\n", plgn_work->first,
                      plgn_work->second, plgn_work->third);
             plgn_work = plgn_work->down;
           }
         }
         fclose (output);
    }                                   /* for (count=0; count<2; count++) */
```

```
}

/* STRIPE_WRITE : write edge_vtx[][] for examining error                */
/*                                                                      */
void stripe_write (g_i, g_j, g_k, flag)
    int g_i, g_j, g_k, flag;
{
    struct PONTdata *p_work; FILE *output; char fname[LENGTH];
    int i;

    strcpy (fname, "stripe.999");
    if ((output = fopen (fname, "w")) == NULL) {
         printf ("\nstripe_write () : unable to open %s\n", fname);
         abort ();
    }

    fprintf (output, "POINT ON POLYGON and its ORDER\n");
    switch (flag) {
         case 0 :
              fprintf (output, "Error in poly_head ()\n");
              break;
         case 1 :
              fprintf (output, "Error in first 'i == edge_cnt'\n");
              break;
         case 2 :
              fprintf (output, "Error in second 'i == edge_cnt'\n");
              break;
         case 3 :
              fprintf (output, "point out of concern\n");
              break;
         default :
              fprintf (output, "point not in PONTlist, %d\n", flag);
    }                                                  /* switch (flag) */
    fprintf (output, " small cube (%4d, %4d, %4d)\n", g_i, g_j, g_k);
    fprintf (output, "\n     start  next    flag\n");
    for (i=0; i<edge_cnt; i++)
         fprintf (output, "     %5d %5d %5d\n", edge_vtx[i][0],
                  edge_vtx[i][1], edge_vtx[i][2]);
    fprintf (output, "\npoint order\n");
    for (i=0; i<EDGE_NUM; i++)
         fprintf (output, " %3d ", stripe[i]);

    fclose (output);
}

/* TEST_SEED : find seed cube along x-, x+, y-, y+, z-, z+              */
/*                                                                      */
void test_seed (g_i, g_j, g_k, drt)
    int g_i, g_j, g_k, drt;
{
    struct VTXPdata *vtxp_work;
    int lf, rt, up, dn, in, ot, o_i, o_j, o_k, snh, flag;
    double d_x, d_y, d_z, field;

    o_i = g_i, o_j = g_j, o_k = g_k;
    lf = dn = in = 0;
```

```
    rt = abs (X_R - X_L + 1) / GRID;
    ot = abs (Y_I - Y_O + 1) / GRID;
    up = abs (Z_U - Z_B + 1) / GRID;

    vtxp_work = vtxp_find (g_i, g_j, g_k);
    if (vtxp_work != NULLVTXP && vtxp_work->done == ON) return;
    if (vtxp_work == NULLVTXP) {
         d_x = (double)(g_i * GRID);
         d_y = (double)(g_j * GRID);
         d_z = (double)(g_k * GRID);
         field = fieldvalue (d_x, d_y, d_z);
         vtxp_add (g_i, g_j, g_k, field);
    } else {
         field = vtxp_work->field;
    }
    if (field > Magic) {
         flag = ON; snh = field > Magic;
    } else {
         flag = OFF; snh = field <= Magic;
    }
    while (snh) {
         switch (drt) {
              case 0 : g_i--; break;
              case 1 : g_i++; break;
              case 2 : g_j--; break;
              case 3 : g_j++; break;
              case 4 : g_k--; break;
              case 5 : g_k++; break;
         }
         if (g_i < lf || g_i > rt || g_j < in || g_j > ot ||
             g_k < dn || g_k > up) {
              printf ("center (%4d, %4d, %4d), board (%4d, %4d, %4d)\n",
                       o_i, o_j, o_k, g_i, g_j, g_k);
              return;
         }
         vtxp_work = vtxp_find (g_i, g_j, g_k);
         if (vtxp_work == NULLVTXP) {
              d_x = (double)(g_i * GRID);
              d_y = (double)(g_j * GRID);
              d_z = (double)(g_k * GRID);
              field = fieldvalue (d_x, d_y, d_z);
              vtxp_add (g_i, g_j, g_k, field);
         } else
              field = vtxp_work->field;
         if (flag == ON) snh = field > Magic;
         if (flag == OFF) snh = field <= Magic;
    }                                              /* while (snh) */

    switch (drt) {
         case 0 : break;
         case 1 : g_i--; break;
         case 2 : break;
         case 3 : g_j--; break;
         case 4 : break;
         case 5 : g_k--; break;
    }
```

```
/* subprogram : TABLE.C - auxilary routine managing table operation*/

#include "header.h"

extern struct CTRLdata *CTRLpont[X_SIZE*Y_SIZE*Z_SIZE], *CTRLlist;
extern struct CTRLdata *RayTrace;
extern struct PLGNdata *POLYgons[X_SIZE*Y_SIZE*Z_SIZE];
extern struct PONTdata *PONTlist;
extern struct VTXPdata *VTXPpont[X_SIZE*Y_SIZE*Z_SIZE],*Queue,*Q_Rear;
extern double Magic;

/* CTRLLT_ADD: add item to CTRLlist. In CTRLlist, '->dwon' means     */
/*           from, not down. For making CTRL double_linked list.     */
/*                                                                   */
void ctrllt_add (x, y, z, radius, sign, flag)
    double x, y, z, radius; int sign, flag;
{
    struct CTRLdata *p_tail, *p_head, *p_work;
    p_work = ctrl_alloc ();
    if (p_work != NULLCTRL) {
        p_work->x = x, p_work->y = y, p_work->z = z;
        p_work->sign = sign, p_work->radius = radius;
    } else {
        printf ("ctrllt_add () : Sorry. No room for data\n");
        abort ();
    }
    if (flag == OFF) p_tail = CTRLlist;
    else p_tail = RayTrace;
    p_head = p_tail->down;
    p_work->down = p_head; p_work->next = p_tail;
    p_head->next = p_tail->down = p_work;

}

/* CTRLLT_WRITE - write content of *CTRLlist to file.                */
/*                                                                   */
void ctrllt_write (sign)
    int sign;
{
    struct CTRLdata *p_work, *p_exit; FILE *out; char fname[LENGTH];

    if (sign == OFF) strcpy (fname, "CTRLlist.snh");
    else strcpy (fname, "RayTrace.snh");
    if ((out = fopen (fname, "w")) == NULL) {
        printf ("\nctrllt_write () : unable to open %s\n", fname);
        abort();
    }

    if (sign == OFF) fprintf (out, "CONTROL POINTS in CTRLlist\n");
    else fprintf (out, "CONTROL POINTS in RayTrace\n");
    if (sign == OFF) {
        p_work = CTRLlist->next; p_exit = CTRLlist;
    } else {
        p_work = RayTrace->next; p_exit = RayTrace;
    }
    while (p_work != p_exit) {
```

```
        fprintf (out, "%8.6g %8.6g %8.6g %8.6g %2d\n", p_work->x,
                 p_work->y, p_work->z, p_work->radius, p_work->sign);
        p_work = p_work->next;
    }
    fclose (out);
}

/* CTRL_ADD - add an item to CTRLdata-type hash table                    */
/*                                                                       */
void ctrl_add (x, y, z, radius, sign)
    double x, y, z, radius; int sign;
{
    struct CTRLdata *p_temp, *p_work;
    int entry, r, s, t, flag;

    r = (int)(x)/RADIUS; s = (int)(y)/RADIUS; t = (int)(z)/RADIUS;
    entry = hash_func (r, s, t); p_work = CTRLpont[entry]; flag = ON;
    while (p_work != NULLCTRL) {
        if (p_work->x == r && p_work->y == s && p_work->z == t) {
            flag = OFF; break;
        }
        p_work = p_work->down;
    }
    if (flag == ON) {                    /* not found, (r, s, t) not in */
        p_temp = ctrl_alloc ();
        if (p_temp != NULLCTRL) {
            p_temp->x = r, p_temp->y = s, p_temp->z = t;
        } else {
            printf ("ctrl_add () : no room for data\n");
            abort();
        }
        p_temp->down = CTRLpont[entry]; CTRLpont[entry] = p_temp;
        p_work = p_temp;
    }
    p_temp = ctrl_alloc ();
    if (p_temp != NULLCTRL) {
        p_temp->x=x, p_temp->y=y, p_temp->z=z, p_temp->sign=sign;
        p_temp->radius = radius, p_temp->down = NULLCTRL;
    } else {
        printf ("ctrl_add () : no room for data\n");
    }
    if (ctrl_insert (p_work, p_temp)) {
        printf ("c_a () :   x = %8.4d,   y = %8.4d,   z = %8.4d\n",
                 p_work->x, p_work->y, p_work->z);
        printf ("c_a () :   x = %8.4d,   y = %8.4d,   z = %8.4d\n",
                 p_temp->x, p_temp->y, p_temp->z);
        abort ();
    }
}

/* CTRL_ALLOC - allocate a CTRLdata structure and sign a                 */
/*              CTRLsignature.                                           */
/*                                                                       */
struct CTRLdata *ctrl_alloc ()
{
    struct CTRLdata *ptr;
```

```
    ptr = (struct CTRLdata *) malloc (sizeof (struct CTRLdata) );
    if (ptr != NULLCTRL) {
         ptr->fingerprint = CTRLsignature;
         ptr->down = ptr->next = NULLCTRL;
         ptr->sign = ON;
         ptr->x = ptr->y = ptr->z = ptr->radius = 0.0;
    }
    return ptr;
}

/* CTRL_CHECK- check integrity of a CTRLdata pointer. If OK, return */
/*             a 0, else print message and return a -1.             */
/*                                                                  */
int  ctrl_check (ptr, msg)
    struct CTRLdata *ptr; char *msg;
{
    if (ptr->fingerprint == CTRLsignature)
         return 0;
    printf ("Error.\n Pointer failure on %s.\n", msg);
    return -1;
}

/* CTRL_INSERT - insert a CTRLdata structure to *CTRLpont[].        */
/*               Return a 0 if successful, and a nonzero if not.    */
/*                                                                  */
int  ctrl_insert (position, current)
    struct CTRLdata *position, *current;
{
    if (ctrl_check (position, "arg 'position' to ctrl_insert ()"))
         return -1;
    if (ctrl_check (current , "arg 'current'  to ctrl_insert ()"))
         return -1;

    current->next = position->next; position->next = current;
    return 0;
}

/* CTRL_RELEASE - release content of hash table to system           */
/*                                                                  */
void ctrl_release ()
{
    struct CTRLdata *p_down, *p_next, *p_work;
    int i;

    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
         p_down = CTRLpont[i]; CTRLpont[i] = NULLCTRL;
         while (p_down != NULLCTRL) {
              p_next = p_down->next;
              while (p_next != NULLCTRL) {
                   p_work = p_next; p_next = p_next->next;
                   free (p_work);
              }
              p_work = p_down; p_down = p_down->down;
              free (p_work);
         }
```

```
    }                                   /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
}

/* CTRL_WRITE - write content of *CTRLpont[] hash table to file      */
/*                                                                   */
void ctrl_write (count)
    int count;
{
    struct CTRLdata *p_down, *p_next;
    FILE *output; char fname[LENGTH];
    int i;

    strcpy (fname, "ctrldt");
    createfname (fname, count);

    if ((output = fopen (fname, "w")) == NULL) {
        printf ("\nctrl_write () : unable to open %s\n", fname);
        abort();
    }

    fprintf (output, "HASH TABLE OF CONTROL POINT\n");
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
        p_down = CTRLpont[i];
        while (p_down != NULLCTRL) {
            fprintf (output, "CTRLpont[%4d]\n", i);
            fprintf (output, "    group. r = %8.6g, s = %8.6g",
                     p_down->x, p_down->y);
            fprintf (output, ", t = %8.6g\n", p_down->z);
            p_next = p_down->next;
            while (p_next != NULLCTRL) {
                fprintf (output, "      x = %8.6g, y = %8.6g",
                         p_next->x, p_next->y);
                fprintf (output, ", z = %8.6g ", p_next->z);
                if (p_next->sign == ON)
                    fprintf (output, ", sign = ON\n");
                else
                    fprintf (output, ", sign = OFF\n");
                p_next = p_next->next;
            }
            p_down = p_down->down;
        }
    }                                   /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
    fclose (output);
}

/* CREATEFNAME - create file name.                                   */
/*                                                                   */
void createfname (fname, count)
    char fname[LENGTH]; int count;
{
    int i = 0, temp1, temp2, temp3;

    while (i < LENGTH && fname[i] != '\0') i++;
    fname[i] = '.'; i++;

    temp1 = count; temp2 = temp1 / 100; temp3 = temp1 %100;
```

```
    fname[i] = (char)(48 + temp2); i++;
    temp1 = temp3; temp2 = temp1 / 10 ; temp3 = temp1 %10 ;
    fname[i] = (char)(48 + temp2); i++;
    fname[i] = (char)(48 + temp3); i++;
    fname[i] = NULLCHAR;
}

/* PLGN_ADD - add an item to PLGNdata-type to *POLYgons[].          */
/*                                                                  */
void plgn_add (g_i, g_j, g_k, first, second, third)
    int g_i, g_j, g_k, first, second, third;
{
    struct PLGNdata *p_work; int entry;

    p_work = plgn_alloc ();
    if (p_work != NULLPLGN) {
         p_work->first = first, p_work->second = second,
         p_work->third = third;
    } else {
         printf ("plgn_add () : no room for data\n");
    }
    if (plgn_check (p_work, "arg 'p_work' to plgn_add ()"))
         abort ();
    entry = hash_func (g_i, g_j, g_k);
    p_work->down = POLYgons[entry];
    POLYgons[entry] = p_work;
}

/* PLGN_ALLOC - allocate a PLGNdata structure and sign a            */
/*              PLGNsignature.                                      */
/*                                                                  */
struct PLGNdata *plgn_alloc ()
{
    struct PLGNdata *ptr;

    ptr = (struct PLGNdata *) malloc (sizeof (struct PLGNdata) );
    if (ptr != NULLPLGN) {
         ptr->fingerprint = PLGNsignature;
         ptr->first = ptr->second = ptr->third = 0;
         ptr->down = NULLPLGN;
    }
    return ptr;
}

/* PLGN_CHECK- check integrity of a PLGNdata pointer. If OK, return */
/*             a 0, else print message and return a -1.             */
/*                                                                  */
int  plgn_check (ptr, msg)
    struct PLGNdata *ptr; char *msg;
{
    if (ptr->fingerprint == PLGNsignature)
         return 0;
    printf ("Error.\n Pointer failure on %s.\n", msg);
    return -1;
}
```

```
/* PLGN_RELEASE - release content of hash table to system           */
/*                                                                  */
void plgn_release ()
{
    struct PLGNdata *p_temp, *p_work;
    int i;

    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
        p_work = POLYgons[i]; POLYgons[i] = NULLPLGN;
        while (p_work != NULLPLGN) {
            p_temp = p_work, p_work = p_work->down;
            free (p_temp);
        }
    }                           /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
}

/* PLGN_WRITE - write content of *POLYgons[] hash table to file     */
/*                                                                  */
void plgn_write (count)
    int count;
{
    struct PLGNdata *p_work; FILE *output; char fname[LENGTH];
    int i, flag;

    strcpy (fname, "plgndt");
    createfname (fname, count);

    if ((output = fopen (fname, "w")) == NULL) {
        printf ("\nplgn_write () : unable to open %s\n", fname);
        abort();
    }

    fprintf (output, "HASH TABLE OF POLYGON EDGE\n");
    fprintf (output, "     FRIST SECOND THIRD\n");
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
        p_work = POLYgons[i]; flag = ON;
        while (p_work != NULLPLGN) {
            if (flag == ON) {
                fprintf (output, "POLYgons[%4d]\n", i);
                flag = OFF;
            }
            fprintf (output, "     %5d %6d %5d\n", p_work->first,
                    p_work->second, p_work->third);
            p_work = p_work->down;
        }
    }                           /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
    fclose (output);
}

/* PONT_ADD - whether (x, y, z) is in PONTlist. If OK, return its   */
/*            order. If not, add (x, y, z) at the end of PONTlist   */
/*            and assign an 'order', then return order.             */
/*                                                                  */
int  pont_add (x, y, z)
    double x, y, z;
{
```

```
    struct PONTdata *p_init, *p_work;

    p_work = PONTlist->next;
    while (p_work != PONTlist) {
         if (p_work->x == x && p_work->y == y && p_work->z == z)
              return p_work->order;
         p_work = p_work->next;
    }
    p_work = PONTlist->from;
    p_init = pont_alloc ();
    if (p_init != NULLPONT) {
         p_init->x = x, p_init->y = y, p_init->z = z;
         p_init->order = p_work->order + 1;
    } else {
         printf ("pont_add () : no room for data\n");
    }
    if (pont_insert (p_work, p_init)) abort ();
    return p_init->order;
}

/* PONT_ALLOC - allocate a PONTdata structure and sign a          */
/*              PONTsignature.                                    */
/*                                                                */
struct PONTdata *pont_alloc ()
{
    struct PONTdata *ptr;

    ptr = (struct PONTdata *) malloc (sizeof (struct PONTdata) );
    if (ptr != NULLPONT) {
         ptr->fingerprint = PONTsignature;
         ptr->x = ptr->y = ptr->z = 0.0; ptr->order =0;
         ptr->from = ptr->next = NULLPONT;
    }
    return ptr;
}

/* PONT_CHECK- check integrity of a PONTdata pointer. If OK, return */
/*             a 0, else print message and return a -1.           */
/*                                                                */
int  pont_check (ptr, msg)
    struct PONTdata *ptr; char *msg;
{
    if (ptr->fingerprint == PONTsignature)
         return 0;
    printf ("Error.\n Pointer failure on %s.\n", msg);
    return -1;
}

/* PONT_FIND - find a point which 'order' is specified. If OK,    */
/*             return its address. If not, return NULLPONT.       */
/*                                                                */
struct PONTdata *pont_find (order)
    int order;
{
    struct PONTdata *p_work;
```

```
    p_work = PONTlist->next;
    while (p_work != PONTlist) {
         if (p_work->order == order) return p_work;
         p_work = p_work->next;
    }
    return NULLPONT;
}

/* PONT_INSERT - insert a PONTdata structure to *PONTlist.          */
/*               Return a 0 if successful, and a nonzero if not.    */
/*                                                                  */
int  pont_insert (position, current)
    struct PONTdata *position, *current;
{
    struct PONTdata *p_work;

    if (pont_check (position, "arg 'position' to pont_insert ()"))
         return -1;
    if (pont_check (current , "arg 'current'  to pont_insert ()"))
         return -1;

    p_work = position->next;
    position->next = current; current->from = position;
    current->next = p_work; p_work->from = current;
    return 0;
}

/* PONT_RELEASE - release content of hash table to system           */
/*                                                                  */
void pont_release ()
{
    struct PONTdata *p_temp, *p_work;

    p_work = PONTlist->next; PONTlist->next=PONTlist->from=PONTlist;
    while (p_work != PONTlist) {
         p_temp = p_work; p_work = p_work->next;
         free (p_temp);
    }
}

/* PONT_WRITE - write content of *CTRLpont[] hash table to file     */
/*                                                                  */
void pont_write (count)
    int count;
{
    struct PONTdata *p_work; FILE *output; char fname[LENGTH];
    int i, j, k;

    strcpy (fname, "pontdt");
    createfname (fname, count);

    if ((output = fopen (fname, "w")) == NULL) {
         printf ("\npont_write () : unable to open %s\n", fname);
         abort();
    }
```

```
    fprintf (output, "LINKED-LIST OF POINT\n");
    p_work = PONTlist->next;
    while (p_work != PONTlist) {
         i = (int)(p_work->x) / GRID, j = (int)(p_work->y) / GRID,
         k = (int)(p_work->z) / GRID,
         fprintf (output, "%4d %8.6g %8.6g %8.6g", p_work->order,
                  p_work->x, p_work->y, p_work->z);
         fprintf (output, "  (%4d, %4d, %4d)\n", i, j, k);
         p_work = p_work->next;
    }
    fclose (output);
}

/* TABLE_INITIAL : allocate memory to pointer-tyoe hast table        */
/*                                                                   */
/*
void table_initial ()
{
    int i;

    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) CTRLpont[i] = NULLCTRL;
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) POLYgons[i] = NULLPLGN;
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) VTXPpont[i] = NULLVTXP;

    Queue = vtxp_alloc (); Queue->down = Q_Rear = Queue;
    PONTlist = pont_alloc ();
    PONTlist->next = PONTlist->from = PONTlist;
}

/* VTXP_ADD - add an item to *VTXPpont[]                             */
/*                                                                   */
/*
void vtxp_add (g_i, g_j, g_k, field)
    int g_i, g_j, g_k; double field;
{
    struct VTXPdata *v_curr; int entry;

    v_curr = vtxp_alloc ();
    if (v_curr != NULLVTXP) {
         v_curr->i = g_i, v_curr->j = g_j, v_curr->k = g_k,
         v_curr->field = field, v_curr->done = OFF;
    } else {
         printf ("vtxp_add () : no room for data\n");
         abort ();
    }
    entry = hash_func (g_i, g_j, g_k);
    v_curr->down = VTXPpont[entry], VTXPpont[entry] = v_curr;
}

/* VTXP_ALLOC - allocate a VTXPdata structure and sign a             */
/*              VTXPsignature.                                       */
/*
struct VTXPdata *vtxp_alloc ()
{
    struct VTXPdata *ptr;

    ptr = (struct VTXPdata *) malloc (sizeof (struct VTXPdata) );
    if (ptr != NULLVTXP) {
```

```
        ptr->fingerprint = VTXPsignature;
        ptr->i = ptr->j = ptr->k = ptr->done = 0;
        ptr->field = ptr->hd_fld = ptr->wk_fld = 0;
        ptr->down = NULLVTXP;
    }
    return ptr;
}

/* VTXP_CHECK- check integrity of a VTXPdata pointer. If OK, return */
/*             a 0, else print message and return a -1.             */
/*
int  vtxp_check (ptr, msg)
    struct VTXPdata *ptr; char *msg;
{
    if (ptr->fingerprint == VTXPsignature)
        return 0;
    printf ("Error.\n Pointer failure on %s.\n", msg);
    return -1;
}

/* VTXP_CORNER - input a grid point. Return number of 'hold' points */
/*               at corner.                                         */
int  vtxp_corner (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    struct VTXPdata *p_work; double field, d_x, d_y, d_z;
    int i, j, k, i__, j__, k__, count = 0;

    i__ = g_i + 2, j__ = g_j + 2, k__ = g_k + 2;
    for (i=g_i; i<i__; i++)
     for (j=g_j; j<j__; j++)
      for (k=g_k; k<k__; k++) {
         p_work = vtxp_find (i, j, k);
         if (p_work == NULLVTXP) {
              d_x = (double)(i * GRID);
              d_y = (double)(j * GRID);
              d_z = (double)(k * GRID);
              field = fieldvalue (d_x, d_y, d_z);
              vtxp_add (i, j, k, field);
         } else
              field = p_work->field;
         if (field > Magic) count++;
      }
    return count;
}

/* VTXP_FIND - find grid point in *VTXPpont[]. If successful,       */
/*             return its address. If not, return NULLVTXP.         */
struct VTXPdata *vtxp_find (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    int entry; struct VTXPdata *v_down;

    entry = hash_func (g_i, g_j, g_k); v_down = VTXPpont[entry];
    while (v_down != NULLVTXP) {
        if (v_down->i == g_i && v_down->j == g_j && v_down->k == g_k)
```

```
            return v_down;
        v_down = v_down->down;
    }
    return NULLVTXP;
}

/* VTXP_INSERT - insert a VTXPdata structure to *VTXPpont[].          */
/*               Return a 0 if successful, and a nonzero if not.      */
/*                                                                    */
int  vtxp_insert (position, current)
    struct VTXPdata *position, *current;
{
    if (vtxp_check (position, "arg 'position' to vtxp_insert ()"))
        return -1;
    if (vtxp_check (current , "arg 'current'  to vtxp_insert ()"))
        return -1;

    current->down = position->down; position->down = current;
    return 0;
}

/* VTXP_Q_A : add item to end of queue                                */
/*                                                                    */
void vtxp_q_a (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    struct VTXPdata *p_cur;

    p_cur = vtxp_alloc ();
    if (p_cur != NULLVTXP) {
        p_cur->i = g_i, p_cur->j = g_j, p_cur->k = g_k;
    } else
        printf ("vtxp_q_a () : Sorry. No room for data\n");
    if (vtxp_insert (Q_Rear, p_cur)) {
        printf ("q_a () :  i = %4d,   j = %4d,   k = %4d\n",
                Q_Rear->i, Q_Rear->j, Q_Rear->k);
        printf ("q_a () : g_i = %4d, g_j = %4d, g_k = %4d\n",
                g_i, g_j, g_k);
        abort ();
    }
    Q_Rear = p_cur;
}

/* VTXP_Q_D : add item to end of queue                                */
/*                                                                    */
void vtxp_q_d ()
{
    struct VTXPdata *p_work, *p_temp;

    p_work = Queue->down, p_temp = p_work->down;
    if (p_work != Queue) {
        Queue->down = p_temp;
        if (p_work == Q_Rear) Q_Rear = Queue;
        free (p_work);
    } else {
        printf ("vtxp_q_d () : Illegal Queue\n");
```

```
            abort ();
        }
    }

/* VTXP_Q_W : write content of queue which store seed cube          */
/*                                                                  */
void vtxp_q_w ()
{
    struct VTXPdata *v_down; FILE *output; char fname[LENGTH];

    strcpy (fname, "queues.999");
    if ((output = fopen (fname, "w")) == NULL) {
        printf ("\nvtxp_q_w () : unable to open %s.\n", fname);
        abort ();
    }

    fprintf (output, "QUEUE OF SEED CUBE\n");
    v_down = Queue->down;
    while (v_down != Queue) {
        fprintf (output, " %3d %3d %3d\n", v_down->i, v_down->j,
                 v_down->k);
        v_down = v_down->down;
    }
    fclose (output);
}

/* VTXP_RELEASE - release content of hash table to system           */
/*                                                                  */
void vtxp_release ()
{
    struct VTXPdata *v_down, *v_work; int i;

    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
        v_down = VTXPpont[i]; VTXPpont[i] = NULLVTXP;
        while (v_down != NULLVTXP) {
            v_work = v_down; v_down = v_down->down;
            free (v_work);
        }
    }                           /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
}

/* VTXP_WRITE - write content of *VTXPpont[] hash table to file     */
/*                                                                  */
void vtxp_write (count, flag)
    int count, flag;
{
    struct VTXPdata *v_down; FILE *output; char fname[LENGTH];
    int i, flagv;

    if (flag == ON) strcpy (fname, "vtxpdt");
    else            strcpy (fname, "griddt");
    createfname (fname, count);

    if ((output = fopen (fname, "w")) == NULL) {
        printf ("\nvtxp_write () : unable to open %s\n", fname);
        abort();
```

```
    }

    if (flag == ON) fprintf (output, "HASH TABLE OF VERTEX POINT\n");
    else            fprintf (output, "HASH TABLE OF GRID   POINT\N");
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) {
         v_down = VTXPpont[i]; flagv = ON;
         while (v_down != NULLVTXP) {
              if (flagv == ON) {
                   fprintf (output, "VTXPpont[%4d]\n", i);
                   flagv = OFF;
              }
              fprintf (output, "            ");
              fprintf (output, " i = %3d,", v_down->i);
              fprintf (output, " j = %3d,", v_down->j);
              fprintf (output, " k = %3d,", v_down->k);
              fprintf (output, " field = %8.6g ", v_down->field);
              if (flag == ON)
                   fprintf (output, "done = %2d ", v_down->done);
              if (v_down->field > Magic)
                   fprintf (output, " HOT ");
              fprintf (output, "\n");
              v_down = v_down->down;
         }
    }                                   /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
    fclose (output);
}

/* VTXP_W_CHECK - check field value at the corner of specific cube   */
/*                                                                   */
void vtxp_w_check (g_i, g_j, g_k)
    int g_i, g_j, g_k;
{
    struct VTXPdata *v_down; FILE *output; char fname[LENGTH];
    int i, j, k, i__, j__, k__;

    strcpy (fname, "cubeck.999");

    if ((output = fopen (fname, "w")) == NULL) {
         printf ("\nvtxp_write () : unable to open %s\n", fname);
         abort();
    }

    i__ = g_i +2, j__ = g_j + 2, k__ = g_k +2;
    fprintf (output, "FIELD VALUE AT SPECIFIC CUBE\n");
    fprintf (output, " small cube (%3d %3d %3d)\n\n", g_i, g_j, g_k);
    for (k=g_k; k<k__; k++) {
         for (j=g_j; j<j__; j++)
           for (i=g_i; i<i__; i++) {
              v_down = vtxp_find (i, j, k);
              fprintf (output, "          ");
              fprintf (output, "(%3d, %3d, %3d)", v_down->i, v_down->j,
                       v_down->k);
              fprintf (output, " field = %8.6g ", v_down->field);
              if (v_down->field > Magic)
                   fprintf (output, " HOT ");
              fprintf (output, "\n");
```

```
            }

        fprintf (output, "\n");
    }                           /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE) */
    fclose (output);
}

/*3456789012345678901234567890123456789012345678901234567890123456789012345678*/
/*       1         2         3         4         5         6         */
```

```
#include "header.h"

extern struct CTRLdata *CTRLpont[X_SIZE*Y_SIZE*Z_SIZE], *CTRLlist;
extern struct VTXPdata *VTXPpont[X_SIZE*Y_SIZE*Z_SIZE];
extern double Magic;
extern int grid_snh[CIRCLE][2];

/* EXAMINE_BOUND: examine the intersected area                          */
/*                                                                      */
void examine_bound ()
{
    struct CTRLdata *p_head, *p_tail, *p_work;
    double dist_p, dist_r, dist_g; int range[3][2];

    p_tail = CTRLlist, p_head = p_tail->next;
    while (p_head != p_tail) {
        if (p_head->sign != ON) {
            p_head = p_head->next; continue;
        }

        p_work = p_head->next;
        while (p_work != p_tail) {
            if (p_work == p_head) {
                p_work = p_work->next; continue;
            }

            dist_p  = SQUARE(p_head->x - p_work->x),
            dist_p += SQUARE(p_head->y - p_work->y),
            dist_p += SQUARE(p_head->z - p_work->z),
            dist_r  = SQUARE(p_head->radius + p_work->radius),
            dist_g  = (double)(SQUARE(2*GRID));

            if (dist_p >= dist_r || dist_g >= dist_p) {
                p_work = p_work->next; continue;
            }

            get_bound (p_head, p_work, range);

            pass_edge (p_head, p_work, range);

            p_work = p_work->next;
        }                                   /* while (p_work != p_tail) */
        p_head = p_head->next;
    }                                       /* while (p_head != p_tail) */
}

/* GET_BOUND : get the boundary of intersected area                     */
/*                                                                      */
void get_bound (p_head, p_work, range)
    struct CTRLdata *p_head, *p_work; int range[3][2];
{
    int i, i_temp; double work[2][4];

    work[0][0] = p_head->x, work[0][1] = p_head->y,
    work[0][2] = p_head->z, work[0][3] = p_head->radius,
```

```
    work[1][0] = p_work->x, work[1][1] = p_work->y,
    work[1][2] = p_work->z, work[1][3] = p_work->radius;

    for (i=0; i<3; i++) {
        if (work[0][i] > work[1][i]) {
            range[i][0] = (int)(work[0][i] - work[0][3]) / GRID;
            range[i][1] = (int)(work[1][i] + work[1][3]) / GRID;
        } else if (work[0][i] < work[1][i]) {
            range[i][0] = (int)(work[1][i] - work[1][3]) / GRID;
            range[i][1] = (int)(work[0][i] + work[0][3]) / GRID;
        } else if (work[0][3] > work[1][3]) {
            range[i][0] = (int)(work[1][i] - work[1][3]) / GRID;
            range[i][1] = (int)(work[1][i] + work[1][3]) / GRID;
        } else {
            range[i][0] = (int)(work[0][i] - work[0][3]) / GRID;
            range[i][1] = (int)(work[0][i] + work[0][3]) / GRID;
        }                          /* if (work[0][i] > work[1][1]) */

        switch (i) {
            case 0 : i_temp = abs(X_R-X_L+1) / GRID; break;
            case 1 : i_temp = abs(Y_I-Y_O+1) / GRID; break;
            case 2 : i_temp = abs(Z_U-Z_B+1) / GRID; break;
        }                                          /* switch (i) */

        if (range[i][0] < 0) range[i][0] = 0;
        range[i][1]++; if (range[i][1] > i_temp) range[i][1]=i_temp;
    }                                          /* for (i=0; i<3; i++) */
}

/* PASS_EDGE : edge intersected with iso-surface                    */
/*                                                                  */
void pass_edge (p_head, p_work, range)
    struct CTRLdata *p_head, *p_work; int range[3][2];
{
    int g_i, g_j, g_k, i, min_i, max_i, min_j, max_j, flag;
    double d_x, d_y, d_z, d_hd, d_wk, r_hd, r_wk, d_sgl, d_sum;
    struct VTXPdata *vtxp_work; struct CTRLdata *ctrl_work;

    r_hd = SQUARE(p_head->radius), r_wk = SQUARE(p_work->radius);
    for (g_k=range[2][0]; g_k<range[2][1]; g_k++) {
        d_z = (double)(g_k * GRID);
        for (g_j= range[1][0]-1; g_j<=range[1][1]; g_j++)
          for (g_i= range[0][0]-1; g_i<=range[0][1]; g_i++) {
            d_x = (double)(g_i * GRID),
            d_y = (double)(g_j * GRID);
            d_hd  = SQUARE(p_head->x - d_x),
            d_hd += SQUARE(p_head->y - d_y),
            d_hd += SQUARE(p_head->z - d_z);
            d_wk  = SQUARE(p_work->x - d_x),
            d_wk += SQUARE(p_work->y - d_y),
            d_wk += SQUARE(p_work->z - d_z);

            if (d_hd < r_hd) {
                d_hd = d_hd / r_hd; d_hd = field_func (d_hd);
            } else d_hd = 0.0;
            if (p_head->sign == ON) ;
```

```
                else d_hd *= (-1.0);

                if (d_wk < r_wk) {
                     d_wk = d_wk / r_wk; d_wk = field_func (d_wk);
                } else d_wk = 0.0;
                if (p_work->sign == ON) ;
                else d_wk *= (-1.0);

                vtxp_work = vtxp_find (g_i, g_j, g_k);
                if (vtxp_work == NULLVTXP) {
                     vtxp_add (g_i, g_j, g_k, 0.0);
                     vtxp_work = vtxp_find (g_i, g_j, g_k);
                }                          /* if (vtxp_work == NULLVTXP) */

                vtxp_work->hd_fld = d_hd,
                vtxp_work->wk_fld = d_wk,
                vtxp_work->ps_fld = d_hd + d_wk;
           }                               /* for (g_j ; ; ) for (g_i ; ; ) */

           for (i=0; i<2; i++) {
                if (i == 0) ctrl_work = p_head;
                else        ctrl_work = p_work;
                if (ctrl_work->sign != ON) continue;

                min_i = range[0][1], max_i = range[0][0],
                min_j = range[1][1], max_j = range[1][0];
                flag = OFF;

                for (g_j=range[1][0]; g_j<range[1][1]; g_j++)
                  for (g_i=range[0][0]; g_i<range[0][1]; g_i++) {
                     vtxp_work = vtxp_find (g_i, g_j, g_k);
                     if (i == 0) d_sgl = vtxp_work->hd_fld;
                     else        d_sgl = vtxp_work->wk_fld;
                     d_sum = vtxp_work->ps_fld;

                     if ((d_sgl  > Magic && d_sum  > Magic) ||
                         (d_sgl <= Magic && d_sum <= Magic))
                          continue;

                     if (g_i-1 < min_i) min_i = g_i - 1;
                     if (g_i+1 > max_i) max_i = g_i + 1;
                     if (g_j-1 < min_j) min_j = g_j - 1;
                     if (g_j+1 > max_j) max_j = g_j + 1;
                     flag = ON;
                }                          /* for (g_j ; ; ) for (g_i ; ; ) */

                if (flag == OFF) continue;

                spread_field (ctrl_work,i,min_i,min_j,max_i,max_j,g_k);
                                           /* for (i=0; i<2; i++) */
           }
                /* for (g_k=range[2][0]; g_k<range[2][1]; g_k++) */
      }
 }

 /* SPREAD_FIELD : achieve total volumn unchanged                    */
 /*
 void spread_field (ctrl_work, key, min_i, min_j, max_i, max_j, level)
```

```
    struct CTRLdata *ctrl_work;
    int key, min_i, max_i, min_j, max_j, level;
{
    int w_i, w_j, w_k, count, c_i, c_j, s_i, s_j, s_k, e_i, e_j, e_k;
    int i_snh, i_temp, i_work, i, j, u_i, u_j, u_k, t_i, t_j;
    struct VTXPdata *vtxp_work; double d_sgl, d_sum, amount;

    w_k = level,
    c_i = (int)(ctrl_work->x) / GRID,
    c_j = (int)(ctrl_work->y) / GRID;
 /*
    spread_error (ctrl_work, key, min_i, min_j, max_i, max_j, level);
 */
    for (w_j=min_j+1; w_j<max_j-1; w_j++)
      for (w_i=min_i+1; w_i<max_i-1; w_i++) {
        vtxp_work = vtxp_find (w_i, w_j, w_k);

        if (vtxp_work->done == ON) continue;

        if (key == 0) d_sgl = vtxp_work->hd_fld;
        else          d_sgl = vtxp_work->wk_fld;
        d_sum = vtxp_work->ps_fld;

        if ((d_sgl > Magic && d_sum > Magic) ||
            (d_sgl <= Magic && d_sum <= Magic))
             continue;

        if (d_sgl > Magic) count = 1;
        else               count = 0;
        for (i=0; i<4; i++) {
             switch (i) {
                  case 0 : t_i = w_i - 1; t_j = w_j    ; break;
                  case 1 : t_i = w_i    ; t_j = w_j + 1; break;
                  case 2 : t_i = w_i + 1; t_j = w_j    ; break;
                  case 3 : t_i = w_i    ; t_j = w_j - 1; break;
                                                  /* switch (i) */
             }
             vtxp_work = vtxp_find (t_i, t_j, w_k);
             if (vtxp_work == NULLVTXP) {
                  printf ("spread_field():This should not happen\n");
                  printf (" %4d (%4d, %4d) --- (%4d, %4d, %4d)\n",
                          i, w_i, w_j, t_i, t_j, w_k);
                  printf ("min_i = %4d, max_i = %4d\n",min_i, max_i);
                  printf ("min_j = %4d, max_j = %4d\n",min_j, max_j);
                  abort ();
             }
             if (key == 0) d_sgl = vtxp_work->hd_fld;
             else          d_sgl = vtxp_work->wk_fld;
             if (d_sgl > Magic) count++;
                                          /* for (i=0; i<4; i++) */
        }
        if (count == 0 || count == 5) continue;

        count = abs (w_i - c_i) + abs (w_j - c_j);

        if (count == 0) {
             printf ("spread_field () : count = 0\n");
             printf ("this should not happen\n");
```

```
            continue;
        }                                       /* if (count ==0) */

        i_temp = count * 2; i_snh = count * 4;
        for (i=0; i<=i_temp; i++) {
            if (i == 0) {
                grid_snh[i][0] = c_i - count;
                grid_snh[i][1] = c_j;
            } else if (i == i_temp) {
                grid_snh[i][0] = c_i + count;
                grid_snh[i][1] = c_j;
            } else {
                i_work = i_snh - i;
                t_i = count - i; t_j = count - abs (count - i);
                grid_snh[i][0] = c_i - t_i;
                grid_snh[i][1] = c_j - t_j;
                grid_snh[i_work][0] = c_i - t_i;
                grid_snh[i_work][1] = c_j + t_j;
            }
        }                                       /* for (i=0; i<=i_temp; i++) */

        amount = 0.0, i_snh = 0, i_work = count * 4;
        for (i=0; i<= i_work; i++) {
            t_i = grid_snh[i][0], t_j = grid_snh[i][1];
            vtxp_work = vtxp_find (t_i, t_j, w_k);
            if (vtxp_work == NULLVTXP) {
                vtxp_add (t_i, t_j, w_k, 0.0);
                continue;
            }                                   /* if (vtxp_work == NULLVTXP) */

            if (t_i<min_i || t_i>max_i || t_j<min_j || t_j>max_j)
                continue;

            vtxp_work->done = ON;
            if (key == 0) d_sgl = vtxp_work->hd_fld;
            else          d_sgl = vtxp_work->wk_fld;
            d_sum = d_sgl - vtxp_work->ps_fld;
            if (d_sum == 0) continue;
            amount += d_sum, i_snh++;
        }                                       /* for (i=0; i<=i_work; i++) */

        if (i_snh == 0 || i_snh == i_work) {
            printf ("spread_field () : i_snh = %4d\n", i_snh);
            printf ("(%4d, %4d, %4d) --- (%4d, %4d, %4d)\n",
                    w_i, w_j, w_k, c_i, c_j, w_k);
            continue;
        }                           /* if (i_snh == 0 || i_snh == i_temp) */

        amount /= ((double)(i_snh * 28));

        for (i=0; i<i_work; i++) {
            t_i = grid_snh[i][0], t_j = grid_snh[i][1];

            if (t_i<min_i || t_i>max_i || t_j<min_j || t_j>max_j) {
                s_i=t_i-1, e_i=t_i+1, s_j=t_j-1, e_j=t_j+1;
                s_k=w_k-1, e_k=w_k+1;
```

```
                    for (u_i=s_i; u_i<=e_i; u_i++)
                     for (u_j=s_j; u_j<=e_j; u_j++)
                      for (u_k=s_k; u_k<=e_k; u_k++) {
                         vtxp_work = vtxp_find (u_i, u_j, u_k);
                         if (vtxp_work == NULLVTXP) {
                              vtxp_add (u_i, u_j, u_k, 0.0);
                              vtxp_work = vtxp_find (u_i, u_j, u_k);
                         }              /* if (vtxp_work == NULLVTXP) */
                         vtxp_work->field += amount;
                         if (u_i == t_i && u_j == t_j && u_k == w_k)
                              vtxp_work->field += amount;
                    }   /* for (u_i; ; ) for (u_j; ; ) for (u_k; ; ) */
                    continue;
               }  /* if (t_i<min_i||t_i>max_i||t_j<min_j||t_j>max_j) */

               vtxp_work = vtxp_find (t_i, t_j, w_k);
               if (key == 0) d_sgl = vtxp_work->hd_fld;
               else          d_sgl = vtxp_work->wk_fld;
               d_sum = vtxp_work->ps_fld;

               if ((d_sgl  > Magic && d_sum <= Magic) ||
                   (d_sgl <= Magic && d_sum  > Magic))
                    continue;

               s_i=t_i-1, e_i=t_i+1, s_j=t_j-1, e_j=t_j+1;
               s_k=w_k-1, e_k=w_k+1;
               for (u_i=s_i; u_i<=e_i; u_i++)
                for (u_j=s_j; u_j<=e_j; u_j++)
                 for (u_k=s_k; u_k<=e_k; u_k++) {
                    vtxp_work = vtxp_find (u_i, u_j, u_k);
                    if (vtxp_work == NULLVTXP) {
                         vtxp_add (u_i, u_j, u_k, 0.0);
                         vtxp_work = vtxp_find (u_i, u_j, u_k);
                    }                   /* if (vtxp_work == NULLVTXP) */
                    vtxp_work->field += amount;
                    if (u_i == t_i && u_j == t_j && u_k == w_k)
                         vtxp_work->field += amount;
               }        /* for (u_i; ; ) for (u_j; ; ) for (u_k; ; ) */
          }                              /* for (i=0; i<i_work; i++) */

     }                                   /* for (w_j ; ; ) for (w_i ; ; ) */

     for (w_i=min_i; w_i<=max_i; w_i++)
       for (w_j=min_j; w_j<=max_j; w_j++) {
          vtxp_work = vtxp_find (w_i, w_j, w_k);
          vtxp_work->done = OFF;
     }                                   /* for (w_i ; ; ) for (w_i ; ; ) */
}

/* SPREAD_ERROR : achieve total volumn unchanged                          */
/*                                                                        */
void spread_error (ctrl_work, key, min_i, min_j, max_i, max_j, level)
     struct CTRLdata *ctrl_work;
     int key, min_i, max_i, min_j, max_j, level;
{
     FILE *output; char fname[LENGTH];int i, j, c_i, c_j;
```

```
    struct CTRLdata *vtxp_work; double d_work;

    strcpy (fname, "spread");
    createfname (fname, level);
    if ((output = fopen(fname,"w")) == NULL) {
         printf ("spread_error () : unable to open file %s\n", fname);
         abort ();
    }

    c_i = (int)(ctrl_work->x)/GRID, c_j = (int)(ctrl_work->y)/GRID;
    fprintf (output, "lower left  corner : (%4d, %4d)\n",min_i,min_j);
    fprintf (output, "upper right corner : (%4d, %4d)\n",max_i,max_j);
    fprintf (output, "key point (%8.4g, %8.4g, %8.4g) - (%4d, %4d)\n",
             ctrl_work->x, ctrl_work->y, ctrl_work->z, c_i, c_j);

    fprintf (output, "\nfield value only by an key point\n");
    for (j=max_j; j>=min_j; j--) {
         for (i=min_i; i<= max_i; i++) {
              vtxp_work = vtxp_find (i, j, level);
              if (vtxp_work == NULLVTXP)
                   fprintf (output, " * ");
              else {
                   if (key == 0) d_work = vtxp_work->hd_fld;
                   else          d_work = vtxp_work->wk_fld;
                   if (d_work > Magic) fprintf (output, " H ");
                   else                fprintf (output, " C ");
              }                       /* if (vtxp_work == NULLVTXP) */
         }                            /* for (i=min_i; i<max_i; i++) */
         fprintf (output, "\n");
    }                                 /* for (j=max_j; j>= min_j; j--) */

    fprintf (output, "\nfield value by two key point\n");
    for (j=max_j; j>=min_j; j--) {
         for (i=min_i; i<= max_i; i++) {
              vtxp_work = vtxp_find (i, j, level);
              if (vtxp_work == NULLVTXP)
                   fprintf (output, " * ");
              else {
                   d_work = vtxp_work->ps_fld;
                   if (d_work > Magic) fprintf (output, " H ");
                   else                fprintf (output, " C ");
              }                       /* if (vtxp_work == NULLVTXP) */
         }                            /* for (i=min_i; i<max_i; i++) */
         fprintf (output, "\n");
    }                                 /* for (j=max_j; j>= min_j; j--) */

    fclose (output);
}

/* TRUE_FIELD : compute real field value for each grid point in      */
/*              *VTXPpont[]                                          */
void true_field ()
{
    int i; double field, d_x, d_y, d_z, f_temp, d_temp;
    struct VTXPdata *v_down;
```

```
    for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE;i++) {
         v_down = VTXPpont[i];
         while (v_down != NULLVTXP) {
              d_x = (double)(v_down->i * GRID);
              d_y = (double)(v_down->j * GRID);
              d_z = (double)(v_down->k * GRID);
              field = fieldvalue (d_x, d_y, d_z);
              field += v_down->field;
              d_temp = v_down->field = value_trnct (field);
              f_temp = Magic - d_temp; d_temp = value_trnct (f_temp);
              if (d_temp <= 0.01 && d_temp > 0.0)
                   v_down->field = Magic;
              v_down->done = OFF;
              v_down = v_down->down;
         }                                  /* while (v_down != NULLVTXP) */
    }                          /* for (i=0; i<X_SIZE*Y_SIZE*Z_SIZE; i++) */
}

/*345678901234567890123456789012345678901234567890123456789012345678*/
/*       1         2         3         4         5         6         */
```