TR--88--087

產生動態紋理方法之研究

TR-88-027

No.

這本經過整理的Document是有關於Texturing的研究，
希望日後的學員，略讀此文後能有所助益。

此Document概分為：

。學生的論文

。程式說明

誌　謝

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

同學史國賢、林俊仁、陳茂盛、許丕中、舒念先等，在課業上相互砥礪，增廣見聞。研究所就讀期間，家人的鼓勵與支持，使我專心於學業，無後顧之憂。在論文撰寫期間，中央研究院資訊科學研究所提供的優良的研究環境，使得論文順利進行。

最後，謹將本論文獻給我敬愛的父母。

論文名稱：產生動態紋理方法之研究　　頁數：74

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：王興華　　指導教授：鄭國揚博士

論文提要內容：

近來，在電腦繪圖的研究領域中，除了隱藏面消去法及塗影上色的技術外，紋理在表現物體、使物體呈現更眞實逼眞的效果上，佔了相當重要的地位。本論文的研究目的，首先是對紋理加以分類，以求得每一特定紋理的產生方法。但這些方法均有複雜的數學公式，使用上不甚方便，經此分類整理後，我們抽取出了一些可調的參數。在適當地調整這些參數後，同一種紋理（如雲）便可呈現出不同的外貌（如積雲、卷雲等）。

其次，我們的研究中亦包括了一種稱爲"擾亂法向量"的紋理方法。本技巧是藉由一個連續函數來控制法向量及切向量方向的改變，以使物體表面產生粗糙的效果。同時，我們在此方法中亦提供了可調的參數（如形狀、顏色、透明度及凹或凸等）。並經此法，我們製造了如冰淇淋溶化的紋理。除此之外，對於已製作完成的每一種紋理亦可再加以複合成更豐富的紋理。

本文所提之紋理法係採用以多邊形表現的物體及掃瞄線演算法爲基礎，當成各項紋理的實驗環境。這種方式的好處是，能提給一般使用者（如畫家）一個方便的創造環境，創造他所喜歡的紋理效果。

Title of thesis: A Study of Dynamic Texturing Total Pages: 74

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

Graduate Date: 06/89 Degree Conferred: Master

Name of Student: Shing-Hwa Wang Advisor: K.Y. Cheng

王興華

Abstract :

Recent reserch in the area of computer graphics shows that in addition to hidden-surface and shading, texturing is important in rendering objets to a more realistic scene. The purpose of this study is to categorize texture types in order to find appropriate texturing for each specific type. Different texture types require different methods for texturing. However, each method has very complicated methematical model of its own, it is rather inconvenient for novice to handle each of them. Therefore, it is important to categorize each type by ranging some adjustable parameters to distiquish differences among them. For instance, by properly adjusting some parameters cloud may presents defferent appearances such as cirrus and cumulus.

In addition to the above mentioned categorization, our study also includes a normal perturbation texturing technique. This technique generates rough surfaces by changing normal and tangent to the surface, and the changed values are determined by a continueous function. Associated with each change, there are operations such as coloring, tansparencing, and bumping or depressing provided for the adjustment of texturing. We have use this technique to simulate liquid textures such as ice-cream melting and water drops. Moreover, each texture that has been thus created can be combined together to form another texture so that a richer texturing can be obtained.

The objects and their display for texturing discussed in this paper is based on polyhedra object represention and scan-line algorithm. The advantage of using this texturing environment is that it can provide artists to creat texture in a manner of what-he-sketches-is-what-he-gets.

目　錄

頁 次

第一章　緒論

近來，在電腦繪圖的研究領域中，由早期的只藉助一些幾何塑型來表現規則的物體，推演至迷人，真實影像的表達。除了隱藏面消去法 (Hidden Surface Removing)及塗影上色 (Shading) 的技術外，紋理的技術 (Texturing) 在呈現物體，使物體更真實逼真的效果上站了相當重要的地位。

追朔以往，研究紋理產生法可推至60年代。當時，紋理可說是影像處理 (Image Processing) 的一支。直至70年後半，才有學者將紋理產生法推展至電腦繪圖的領域。早期電腦繪圖於描摹生活中的物件時，多半是金屬、塑膠類的物品，物品的表面呈現的多是只具立體感的光滑面，以及只有單一的顏色。很明顯的，對於表面有紋路的物體這是不夠的。因此，在電腦繪圖領域中積極地研究紋理產生法的工作便如雨後春筍般的展開了。

早期的紋理的技術，不外乎是取自照片，也就是將物體所需的紋路，如木紋、大理石紋以相機拍下，經影像處理的技術讀入電腦中，再將此取得的紋路佈著於物品上，可能是桌子、椅子等。如此一來，物體的表面不再是單一

的色澤了。可是表面仍是光滑的，對於不規則的表面，如桔子皮、草莓等物體又不夠使用，因此產生表面皺痕的紋理技巧也就因應而生。

至今，強調電腦動畫的同時，所謂動態紋理(Dynamic Texturing)的方法也是各家研究的重點。說物體隨時間運動是很容易察覺的，而舉凡自然界中，因時間而興起變動的紋理有那些呢。大體說來有木紋、水紋、雲、霧、煙、火焰及冰淇淋流動等。它們的變動，有的是顏色上或透明度的改變(如雲、霧、煙)；有的則是形狀的改變（如水紋、冰淇淋)。 在這些紋理的產生過程中加入了時間的因素，無非促進原本佈著靜態紋理的物體有了更活潑、豐富的效果。因此，帶動了更多人從事這方面的研究。而本論文的紋理研究，也就針對動態紋理加以討論。

基本上說來，每一類紋理均由一些簡單的幾何元件組合成，動態紋理也不例外。在產生動態紋理時，便是將這些幾何元件隨時間的改變後再加以而組合，以達動態的效果。

水紋，是由波心 — 點 — 向外延伸的同心圓組合。隨時間的增加，調整同心圓間的距離及對面的影響層度而達動態波動的紋理。

木紋，是線條或同心圓的組合。線與同心圓的增加及互相擠壓而有了變形的效果。

冰淇淋，其實應稱為流質紋理。目前對紋理的研究中，此一環似無人觸及，本文對此紋理有進一步的說明。而它的幾何元件，可說是由大小不同的圓組合而成。每一個圓正確地說是流質中的單一質點所形成，各個質點因彼此的張力之聚合或離散以達成流動的效果。這種流動與水紋不同，水紋雖也有表面張力，但流動是向外擴散式的。不像冰淇淋有散有聚的互動。

雲、霧、煙等的幾何元件比較複雜，先前討論的均是 2-D 的幾何元件，而雲等則是 3-D 的元件（如橢圓球）。雲本身是一種密度函數，但可藉光學上顏色的濃淡及透明層次來代替。因顏色與密度成比例的特性，將描繪成的各橢圓球組合，便創造出如積雲、卷雲等不同雲種。[1985, Geoffrey Y. Gardner]。本文中討論的煙亦類似此方式。

火焰，是類似雲的方法產生紋理，同樣是濃淡及透明度的作用。但它的幾何元件可以是任何立體物體，而每一元件均給與生命函數，元件依其生命是處於中年、老年、幼年而給與不同的濃淡顏色及透明度。[1983, Reeves]

有些紋理不是簡單的幾何元件便可組合而生，仍需複

雜的數學模式。在原本規則的幾何元件中加入不規則的因子，使紋理更真實迷人。這類的數學模式大致上分為：Stochastic Process；其次是Random Function Transformation 以及 Predefined Function。

Stochastic Process一般是用在造山及地形上。而對於火焰的產生得須此數學模式來控制點增加的數目及點的運動速率，而使紋理成不規則（如火焰燃燒）的景象。

Random Function Transformation [1985 Gardner]，藉著光譜函數的特性，擾亂物體上面的顏色，呈濃淡密度的觀感。如先前討論的雲便經此數學模式而生成。

Predefined Function:水紋的波函數，及流質紋理等均須一些簡單而不複雜的函數以達成本身紋理的效果，這些函數稱為 Predefined Function。因為不似前兩項無規則可循，但有了它便可製造不錯的紋理。

經由前述對紋理的介紹，不難發現如果以畫家的眼光來創造紋理，上述的方法對他們並不容易使用。尤其那些複雜的數學模式如何地調配才會達到他們的需求，他們也無一個依循的方向。有鑑於此，我們的目標是希望提供一整合後的紋理系統，在創造紋理的過程中簡化數學的介面，而提出一比較方便操作的實驗環境 。

在軟體上我們將各紋理產生的方法加以分類，並自各類不同的紋理的幾何元件中提供一些可調的參數以達動態效果。並進而提出一似 Object-Oriented理念的使用者介面，一方面方便我們的研究，另一方面亦可提供一般使用者一理想的作畫環境。

本論文研究的重點，針對流質紋理、煙紋理的處理做了深入的研究。並針對一個稱為法向量擾亂 (Normal Perturbation) 的紋理產生法做了多方面的應用。之後，我們設計了一個以IC為導向的 Texturing-IC 的構想及相應的描述語言以供紋理生成的使用。

接下來幾章，於第二章中，我們對各種現有的紋理產生方法作進一步的分析及介紹，並對它們方法上的不同而分為幾何紋理產生法 (Geomtric Texturing) 及色彩紋理產生法 (Color Texturing)。第三章中，對我們的紋理系統及現階段處理的紋理產生法做進一步介紹。第四章所述為整個系統的流程及架構。第五章為一些實驗的成果。第六章為我們對於紋理的應用及未來研究的方向。

第二章　背景

2.1 物體的表現

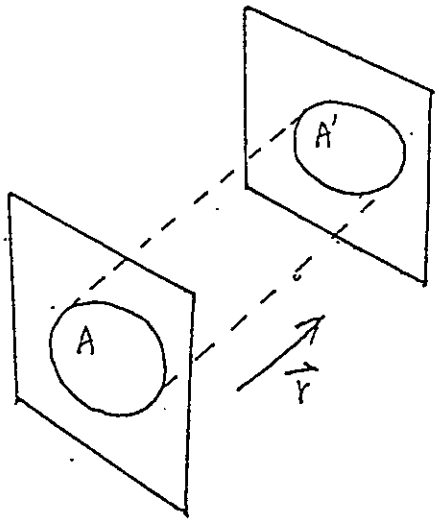
物體塑型 (Object Modeling)雖不是本論文的討論主題，但對於物體的表現仍與紋理的方法有關，因此於此概略述說 Modeling 的方法。

2.1.1 Sweep Representation

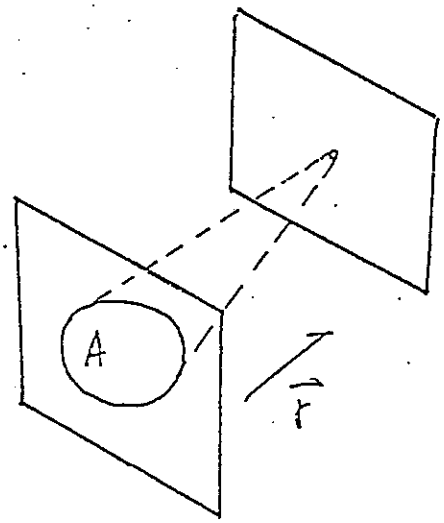
Sweep Representation 在 Modeling的領域中是最常被採用的方法，而約略又分為兩類：一為 Translational sweeping 及 Rotational Sweeping。

2.1.1.1 Translation Sweeping

以例子說明之。Translation Sweeping法產生的物體最典型的就是圓柱體。如圖所示使用者給與一個平移的面 A，並指定平移軸$\vec{T}$，此法便是沿著$\vec{T}$平移出欲製造的物體 (圓柱體)。

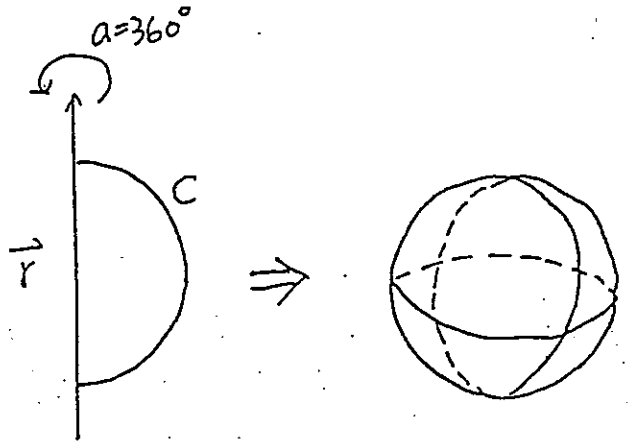


這個方法亦可加入一些改變，例如當給與面A及$\vec{r}$時，可設定面A沿$\vec{r}$移動時以等速度的改變面A大小（面A的半徑等速減小），如此製造的物體，便由圓柱體變成圓椎體了。



2.1.1.2 Rotational Sweeping

也以例子說明之。如果欲製造一圓球，首先給與一半圓的曲線 C 、一個旋轉軸 $\vec{r}$ 及旋轉角度 a 角度後掃出的面，便是欲得的物體。此例中，旋轉角度為 360° 。

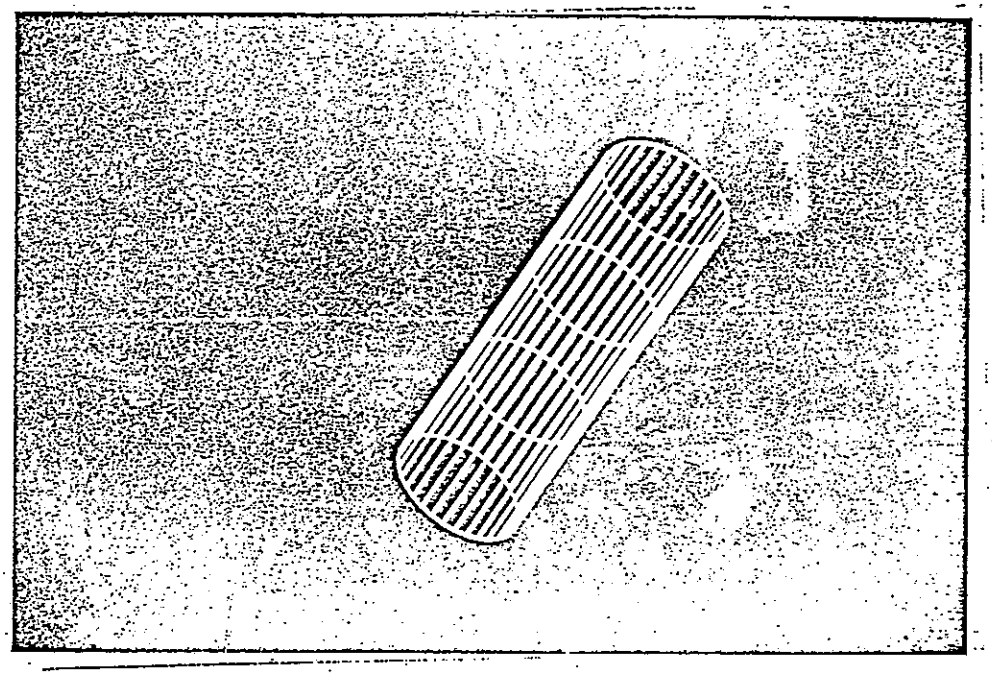


2.1.2 Polyhedral Representation

經 Sweeping Representation求得物體，為了符合本系統的使用，故得將其 Polygonized。而在切成 Polygon 的動作，是與做 Sweeping時一起完成。

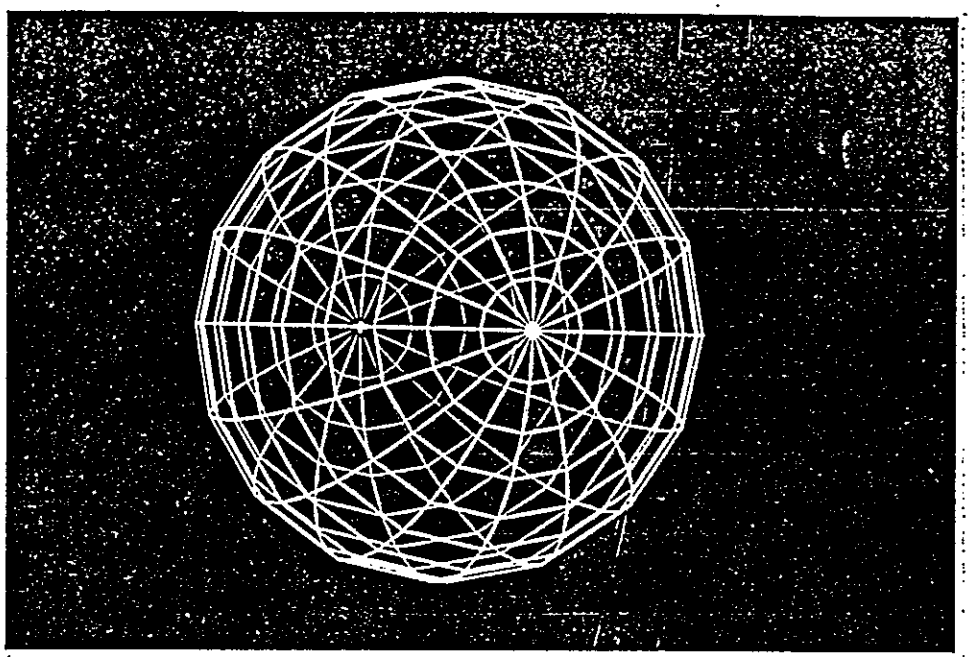
2.1.2.1 Polygonized Translation Sweeping

我們將前例中面A以線段來趨近，如欲愈逼近原平滑的面，線段得切得更細。再沿給定的平移軸r移動，依每隔多少間隔切一個 Polygon 出來。如圖便是Polygonized後的圓柱體。



2.1.2.2 Polygonized Rotational Sweeping

與 Polygonized translation Sweeping 大致相同。將原曲線C也以線段逼近，並將這些線段組成的曲線旋轉得一個由 Plogyon 表達的物體了。



2.2 紋理的產生方法

舉凡紋理的產生方法 (Texturing)可分為兩大類：一為 Image Mapping 法；另一為 Perturbation mapping。接下來，我們對此兩項說明。

2.2.1 Image Mapping

Image Mapping 是最原始的紋理產生法。是將取得的紋理樣本 (texture Pattern)，予以數位化，取以參考座標 (u, v)，對應於物體 3-D的座標 (x, y, z)。在做 Shading 時便依循 Mapping的技巧取得紋理樣本上的色澤代入做計算。[1974, Catmull]

$$t = t(u,v)$$

上式中 t為取得的紋理於座標 (u, v) 上的色澤，而 (v, u) 與物體空間座標的對應為

$$(u,v) = N(x,y,z)$$

N 為 Mapping Function, (x, y, z) 為物體上的某一點的座標。因此，歸納得

$$t = t(N(x,y,z))$$

至於紋理樣本的取得，可由照片，作畫者繪出或函數產生不等。這類的紋理大多是木紋、大理石紋等。

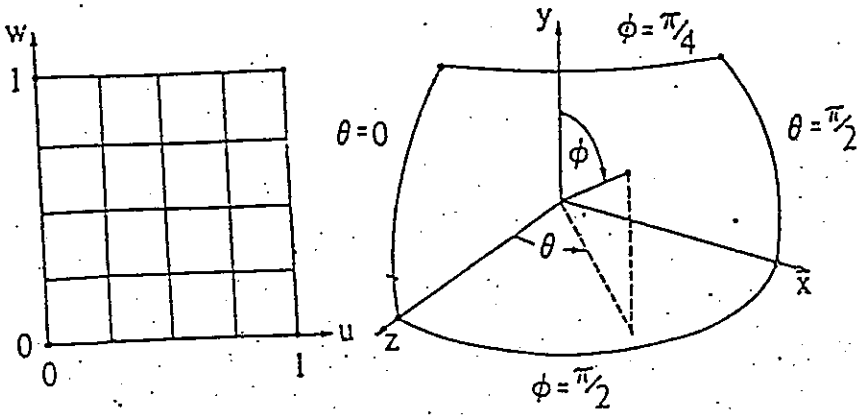


圖 2-1 Image Mapping

2.2.2 Perturbation Mapping

這一類又可分為兩項：一是Color Perturbation：另一是 Normal Perturbation。

2.2.2.1 Color Perturbation

Color Perturbation多半是藉助一個紋理函數以達顏色擾亂的效果。也就是說，將一個 2-D 或 3-D 的物體，在計算完物體的顏色後，將此物體上每一點的顏色與傳回 0 ～ 1之間的擾亂函數依比例相加而得顏色：

$$I_t = T * t + S * (1-t)$$

T：0～1，自紋理函數傳回的值；T＝t(x，y，z)

S：物體上某一點的顏色。

I_t ：經擾亂後的物體顏色。

t： 一加權量，以決定函數傳回值擾亂物體原本顏色的層度。

經此擾亂後的顏色，使物體表面的顏色有了不規則的色彩變化。

而有時除了顏色的擾亂外，學者們又考慮對透明度的擾亂。將前述擾亂過後的顏色與一取定的顏色層比較，並依比較後的值來設定透明度的指數，以示物體的濃密層度。

$$TR = 1 - \frac{(I_t - P)}{D}, \quad 0 <= TR <= 1$$

P : threshold value

D : The range of texturing function values across which translunce varies from 0 to 1

TR : 調整後的透明指數。 1 : 表示物體完全透明 ; 0 物體不透明。

這一類的技巧，大都是運用於雲，火焰等與濃密度有關的紋理上。

至於紋理函數的選擇上，學者們也各自設計了適合的函數，如創造雲紋理的 Poor Man's Fouries Series [1985 Gardner] :

Man's Fourier Series :

$$t(X,Y,Z) = K \sum_{i=1}^{n} [C_i \sin(FX_i X + PX_i) + T_o] \sum_{i=1}^{n} [C_i \sin(FY_i Y + PY_i) + T_o]$$

其中 X, Y 向的頻率，以下式變化 :

$$FX_{i+1} = 2FX_i$$

$$FY_{i+1} = 2FY_i$$

$$C_{i+1} = 0.707 C_i$$

PX 和PY 是相位移，用來增加紋理的自然感，免得紋理太規則和有格子狀的情形產生。

$$PX_i = \pi / 2 \sin(0.5 FY_i Y)$$

$$PY_i = \pi / 2 \sin(0.5 FX_i X)$$

其它的函數分見於 [1980 Schachter] [1986 Masa Inakage] 等。

2.2.2.2 Normal Perturbation

在計算物體表面某一點的顏色時，必須考慮它的表面法向量 (Normal),計算法向量與光線的角度，再得出顏色，如此 Shading出來的物體，呈現出在光源照射下明暗分佈的情況。如果在每一點的法向量上，予以控制得當的擾亂，則 Shading出來的物體不再是只有均勻地增強或減弱的明暗情況，而是有皺摺紋理的表面。

這種方法又稱為 Bump Mapping。Blinn於1978年提出了如何求取經擾亂後之法向量的方法。

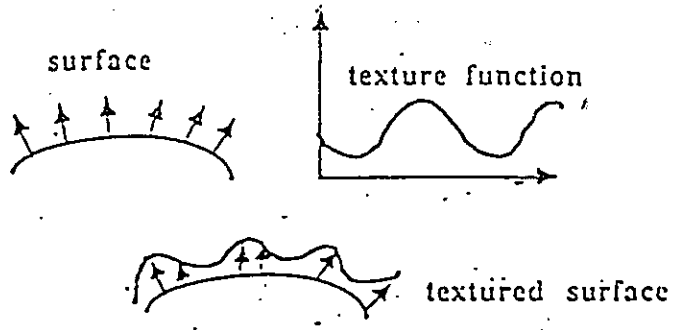


圖 2-2 Bump Mapping

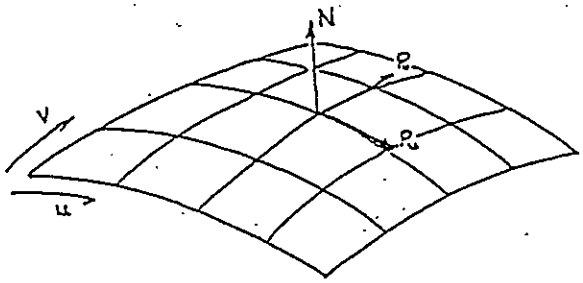
Blinn 假設物體的表面是由參數式表示，對於面上一點 P＝(X, Y, Z) 的參數是為

$$X = x(u,v) \quad Y = y(u,v) \quad Z = z(u,v)$$

而此點的法向量取得為：

$$P'_u = \frac{\partial P}{\partial u} = \left\langle \frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}, \right\rangle$$

$$P'_v = \frac{\partial P}{\partial v} = \left\langle \frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right\rangle$$



$$N = P'_u \times P'_v$$

N 為 P 點的法向量，而 P'_u 與 P'_v 分別為 P 點切平面上的兩個向量，也就是兩個切向量。

而對於 P 點沿法向量拉高某 F 個量後，其法向量的計算又如何呢。首先，P 應拉高成 Q 點

而 Q 點的法向量取得為：

$$Q = P + F\frac{N}{|N|} = P + Fn$$

$$M = Q'_u + Q'_v$$

$$= (P'_u + F'_u n + Fn'_u) \times (P'_v + F'_v n + Fn'_v)$$

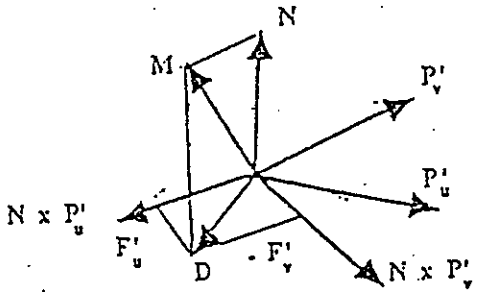
$$= P'_u \times P'_v + F'_u(n \times P'_v) + F'_v(P'_u \times n) + F'_u F'_v(n \times n)$$

因為提高的量 F很小，故最後新的法向量 M 可寫成

$$M = N + D$$

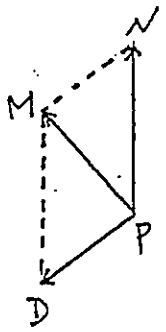
$$D = F'_u(n \times P'_v) - F'_v(n \times P'_u)$$

經此推導過程的結果，我們可深入得知 D在此的意義。以幾何上觀點來說



P'_u，P'_v，$N \times P'_u$ 及 $N \times P'_v$ 均是位於點 P 的切平面上的切向量，而 D 只是 $N \times P'_v$ 與 $N \times P'_u$ 兩向量以 $(F'_u, -F'_v)$ 的量

做向量合而得。因此，D 是個切向量而所謂的擾亂，不外



乎是自 P 點取一切向量 D，而將此向量-D 與原法向量求其二者向量合而得新的法向量M，故歸成

$$M = N + D$$

稍後，於第三章中，我們會對 Normal Pertubation 用於多邊形物體上的應用加以討論，並進而製造出流質紋理。

Blinn 曾用此法設計了橙，草莓表皮等有凹凸的紋理。

2.3 Illumination Model

在紋理的處理中，Shading 是最後的一部份，但也是最重要的。根據 Phong 於1975提出的 Illumination Models 中，它將 Shading 分成了兩項：

$$I = I_d + I_s$$

I_d : diffuse reflection

I_s : specular reflection

2.3.1 Diffuse Reflection

Diffuse Reflection 是根據 Lambert's Law,這個模式的基本精神是以光源與面上所有點的法向量夾角的Cosine值為基楚，強調如果光線的方向與面上法向量的夾角愈小，表示物體直接受光源的照射，没有太大的偏差。因此，其受光直接照射，它的明暗度也就是較亮的一面。反之，若夾角愈大，並非直接照射到的面，則面的明暗度就暗下來。如果是一圓滑的面，它的亮度便會有自亮點（直接受光點）的漸次暗下來的效果，而突顯出成 3-D的圓滑面。

Diffuse Reflection (Lambert's Law):

$$I_d = K_d C_s \sum_j^m I_j (N L_j) \qquad m : \text{light sources}$$

K_d : the diffuse reflection coefficient

C_s : the surface color

N : the surface normal

I_j : the light color L_j : the direction of the j–th ligth

圖 2-3 Diffuse Reflection Model

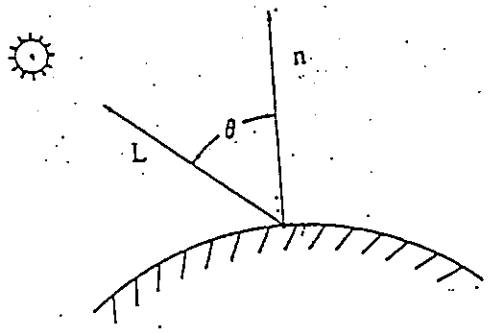


圖 2-4 Diffuse Reflection 解析圖

2.3.1 Speclar Reflection

Specular Reflection 乃是一實驗的模式，目的是為了模擬物表面的光滑質感。Phong 指出，如果只採Diffuse

Reflection 的模式，物體顯得呆滯不生動，但配合上 Specular Reflection 物體的表面便可呈現出光滑與粗糙的層度。例如光滑的撞球面便是由 Specular Reflection 達成。

Specular Reflection (empirical model)

$$I_s = K_s C_r \sum_j^m I_j (N H_j)^n \qquad m : \text{light sources}$$

K_s : the specular reflection coeffiecient

C_r : the reflective color, where we take the defference of I_j and C_s,

$C_r = I_j - C_s$

H_j : the direction halfway between the observer and the the j–th light,

$$H_j = \frac{(L_j + E)}{|L_j + E|}$$

n : 1 —— 200,

Large value of n correspond to metaallic and shinny

圖 2-5 Specular Reflection Model

模式中的 Shinny Coefficient 在我們的紋理系統中已視為一個可調的變數。它利用 Cosine 值的收斂情況，將光源的能量分散至整個面，抑或是集中於面上的某一部分，使此部分受特別大的光源而達光滑的效果。也就是說，Shinny Coefficient 愈大值，物體光滑的效果愈好；而值小時物體顯得較粗糙。

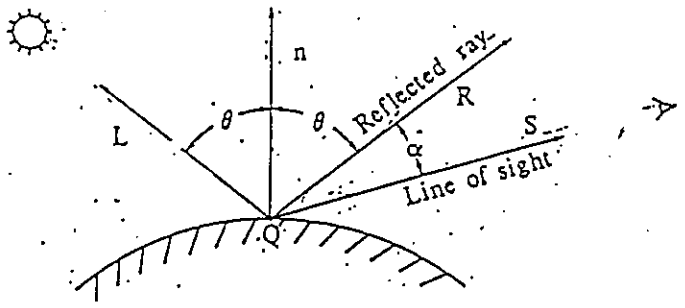


圖 2-6 Specular Reflection

2.4 紋理系統架構

經前幾節的對紋理產生方法的討論，我們將它們分成兩大類：一為 Geometric Texturing；另一為 Color Texturing。

我們將 Normal Perturbation 及Specular Reflection視為 Geometric Texturing的一支，是因為它們均與物體表面的光滑與粗糙有關之故，對物體幾何效果有影響。而 Color Texturing 的 3 類中，Image Mapping, Color Perturbation, 及Transparency均是對已經計算好明暗度的顏色做適當的改變，與物體的幾何結構無關。

經上述的分類，我們的紋理設計便更具結構化了，而於第三章中，我們更將此擴展成 Texturing-IC 的設計。此外，我們也對 Normal Perturbation在多邊形化物體的應用，與流質紋理、煙等的產生方法做深入的討論。

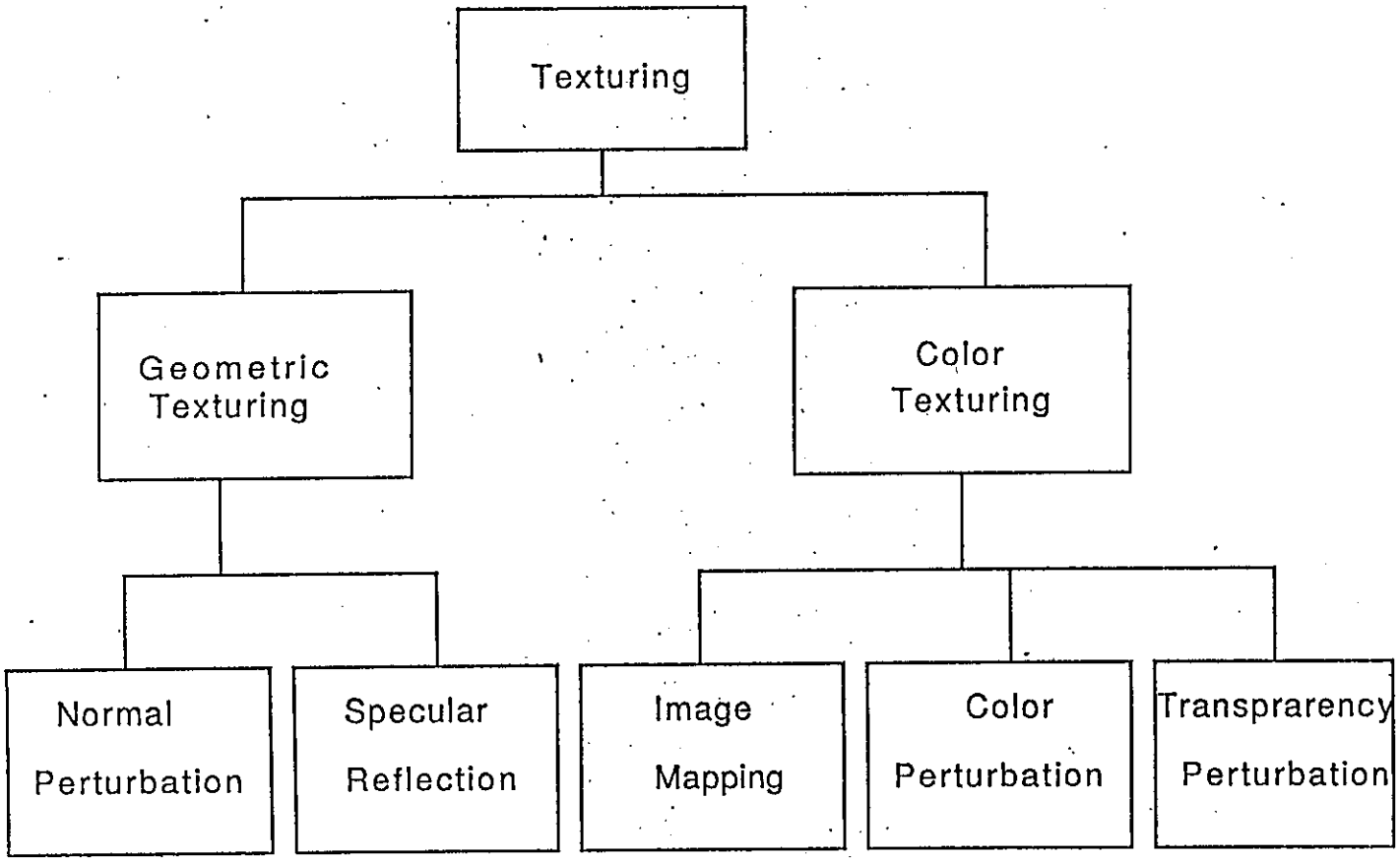
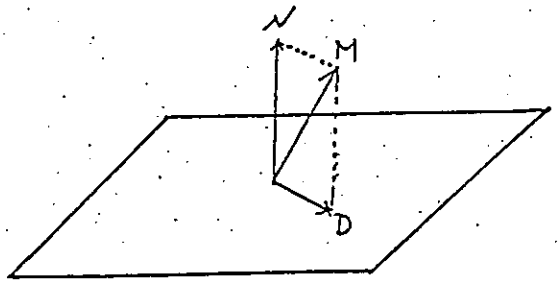

Texturing
Geometric Texturing
Color Texturing
Normal Perturbation
Specular Reflection
Image Mapping
Color Perturbation
Transprarency Perturbation

圖 2-7 紋理生成法分類

第三章　紋理系統

3.1 再談 Normal Perturbation

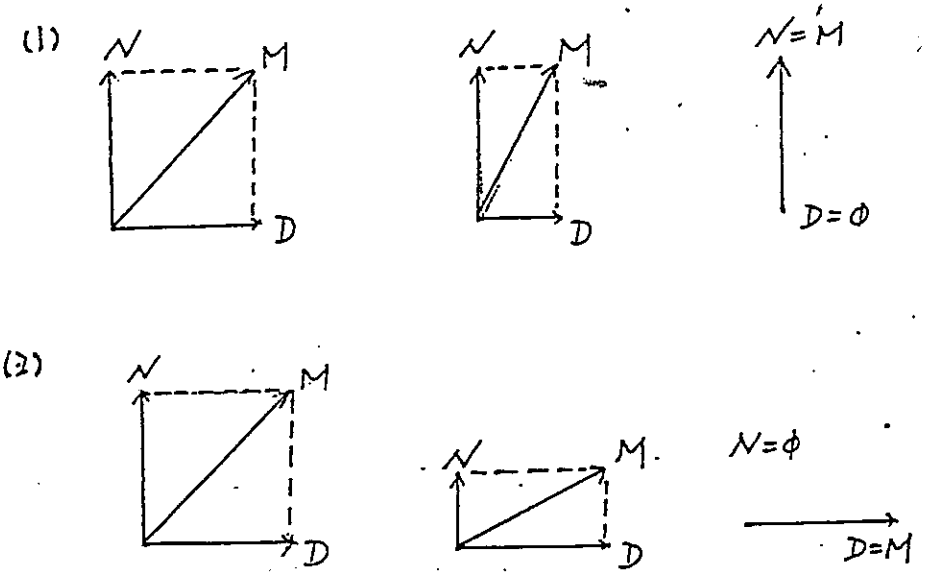
前章曾對 Normal Perturbating的方法大致上的步驟做法細說過。在此我們針對多邊形化物體如何採用此法做更進一步的說明，雖與前述的方法不同（因為先前皆應用於參數化的物體），但對於多邊形化物體其基本目的仍是相同的，也就是得到擾亂後的法向量代入 Illumination Model 做明暗度的計算。



$$M = N + D$$

在前章，曾提及改變法向量 N，得借助一平面上的切向量 D，將 N 與 D 的向量合產生一個新的法向量 M，這 M 便是我們的目標向量。如此一來問題便在如何得取 M，換句話說就是經由 N 及 D 的改變以得M 的組合。以下我

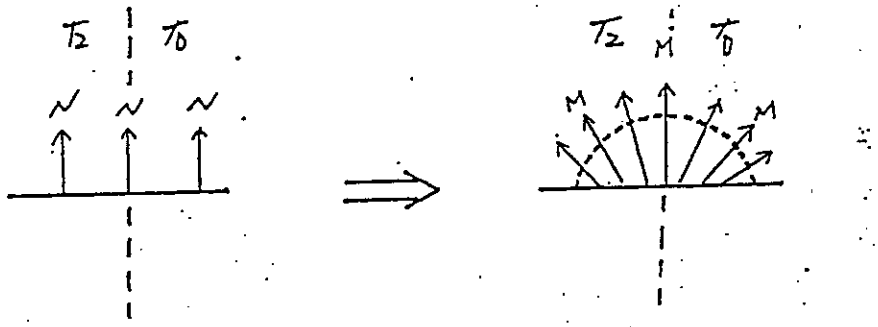
們便對此點加以說明。



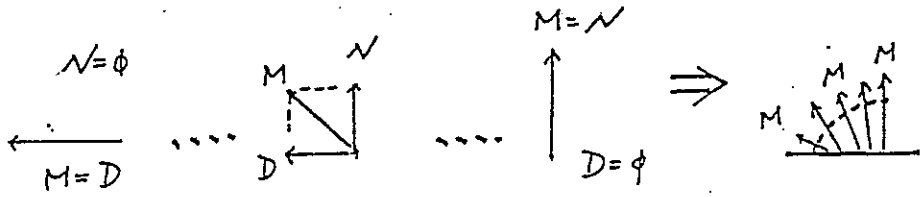
如上圖的說明，(1) 圖中，我們如果只對 D方向的量改變可得一組不同的 M，而 (2) 圖中對 N 方向的量改變又可得另一組的 M，當然了這兩種對 N 及對 D 的量改變是可以同時進行的。例如欲模擬一個平面上的圓形凸起，我們便可以將上述的方式，對 D與 N方向的量調整而組合成。如下圖所示

平面的法向量均朝同一方向或均勻的變化，而沒有特別的突然凸起，如果在這樣的面上，我們模擬一個突然凸起，面上的法向量便有了急遽的改變，對這變化的模擬，我們分述於下：

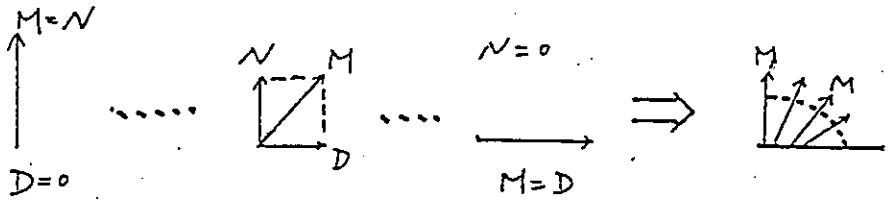
首先我們將上圖分成左右兩邊來考慮



對於左邊的情形，我們便可以將面上原始的法向量 N與切向量 D 做如下的組合：



而右邊的情況亦如上式的組合，只是改變 D的方向及對 D 的量的改變成相反的量，N 則不變。如下：



經由上述對 N 及 D 的組，我們不難發現，對面上凸起模擬簡化許多；至於欲模擬凹的紋理，同樣地可以此相似的方法，我們也圖解的方式以茲了解。

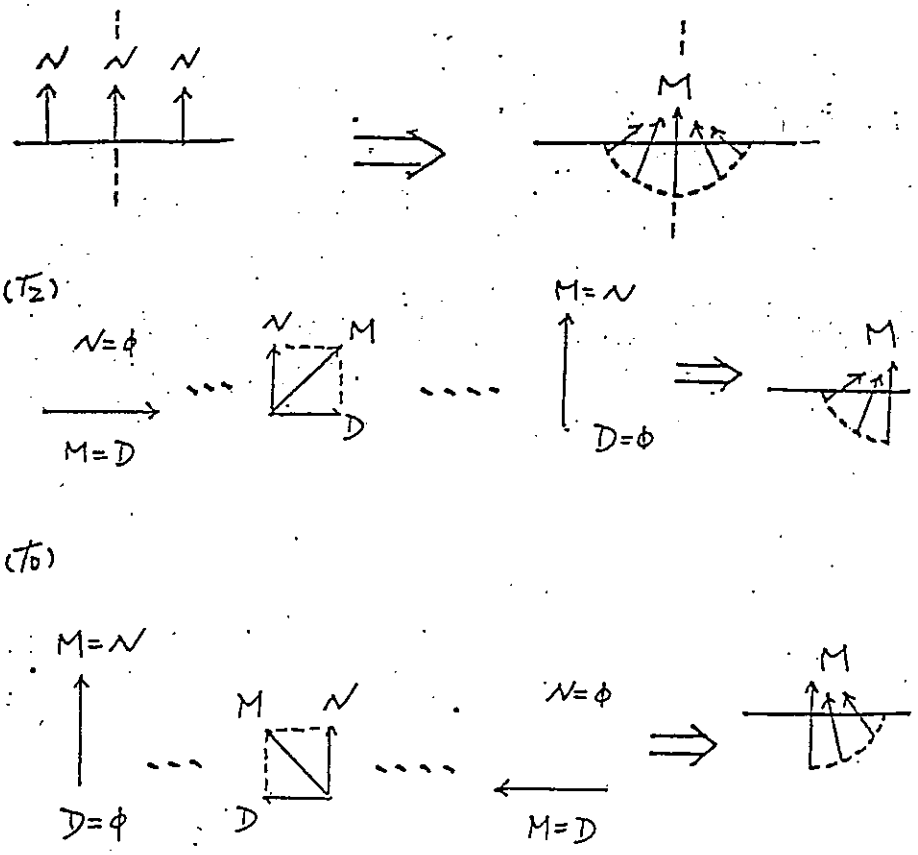
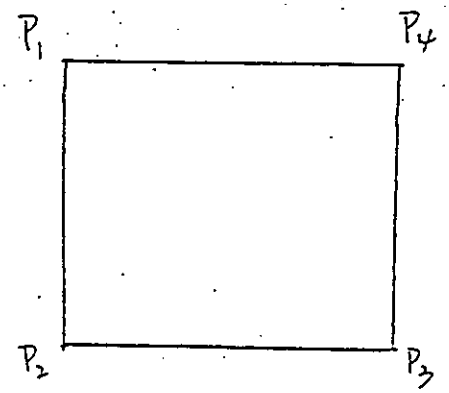


圖 3-1 凹的紋理的模擬解析

由上圖的解說，可發現凹是凸的相反的順序組合的 M 而成。將最後的法向量 M，代入 Illunination Model 便可得凹與凸的紋理，使原本平滑均勻的表面有了不規則的表面，如一顆高爾球，便可以凹下去的紋理附著於球的表面上。

之前，我們是假設 N 與 D 已知的情況，對 N 及 D 改變組合而得 M，至於多邊形化物體的 N 與 D 的獲取與第二章所提的方法又有所不同。

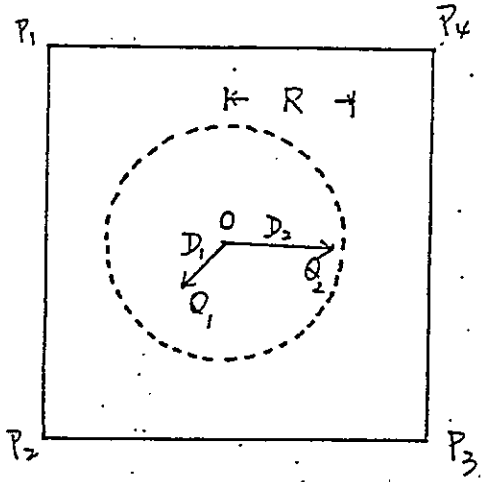
平面的法向量的得到比較容易。以多邊形的物體而言，每一個多邊形均是一個平面，而平面是由幾個頂點組成如圖的四邊形平面所示，由 P1 P2 P3 P4 組成，而平面



的法向量 N 可由下式得之

$$N = \overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$$

計算得 N 後，對於 D 的求取，我們以圓形凸起為例。我



們選定面上一點 O 為圓形凸起的中心，及決定圓形凸起的半徑 R，我們就針對以 R 掃出的圓內做 D 的計算。由圖可知，面上距 O 愈遠的點 Q2 ，它的切向量 D2 就愈大。而距 O 愈近的點 Q1，它的 D1就愈小，這正與我們做凸起紋理時 D 的量組合正好相合。

經上述的整理，我們可得知，圓形凸起的 D向量的量與圓內點至圓心的距離成正比，而 D的方向與OQ相同，如下：

$$|D| \cong |QO| \qquad \vec{D} \cong \vec{OQ}$$

至於凹的圓形紋理，也可由相似的方法求得，只是 D 的改變量與圓內點至圓心的距離仍成正比了，但 D的方向

與 OQ 的方向相反。如下：

$$|D| \cong |QO|$$

$$\vec{D} \cong -\vec{OQ}$$

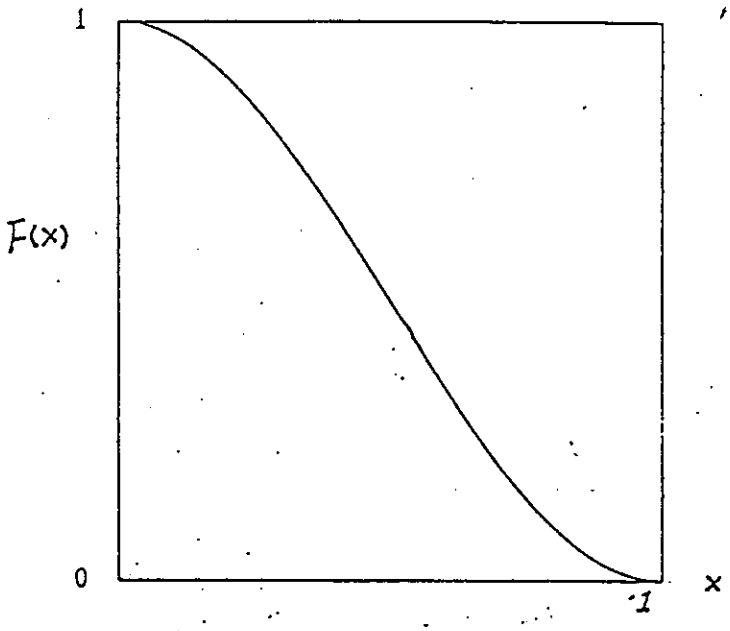
3.2 流質紋理的模擬：

介紹過 Normal Perturbating之後，對於它的應用，我們製作了一種流質似的紋理，於此我們便可以產生動態紋理的效果。接下來便對此討論。

流質紋理的產生，除了藉助前節討論的如何產生圓形凸起的紋理外，仍需藉助一個連續函數來控制流質紋理凸起及流質的外形。這個函數稱為 "場力函數" (Field Function)。我們選擇了在1986年 Geoff Wyvill提出的一做軟性物體時所採用的函數。

$$F(x) = 2x^3 - 3x^2 + 1$$

這個函數的圖形如下所示：

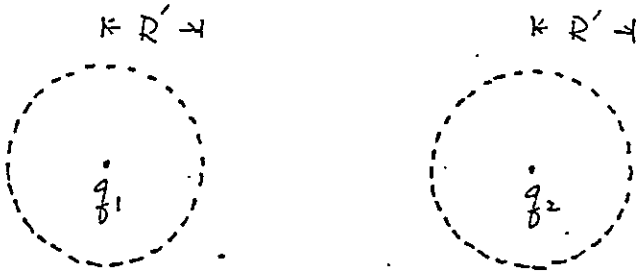


在流質紋理產生時，場力函數對於除了 D 和 N 的量的改變外，還對於流質的外形有控制的作用。我們首先對於外形的控制說明，其次再對 D 和 N 的控制討論。

3.2.1 對於外形的控制：

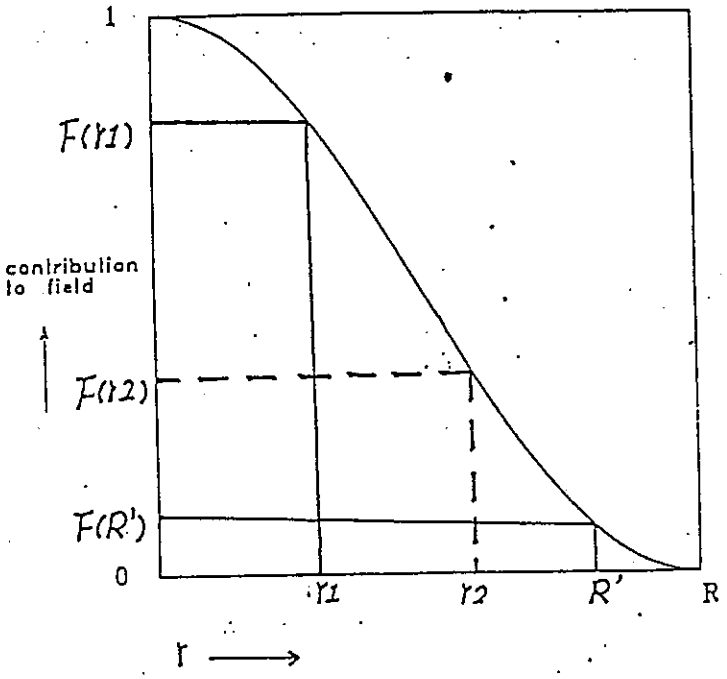
・單一中心質點的作用範圍：

根據 Geoff Wyvill 所提，一圓形的水滴可視為由一中心的引力作用凝聚而成，如下所示：



如以場力函數考慮之，也就是上圖中 q1 與 q2 兩引力場在方圓 R′ 的範圍內對其週圍的質點施引力聚合而成的圓形，方圓之外的質點因引力太小而忽略。而愈進 q1 與 q2 點的質點其受引力內聚的力愈大，愈遠者反之受力愈小，如此正合於前述的場力函數的特性（曲線圖）。

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

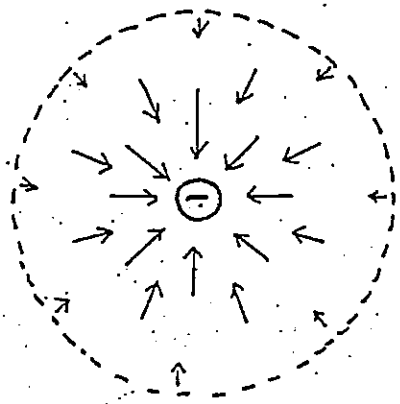


r1 < r2

F(r1) > F(r2)

如圖中，r1 距離 q1 的質點所受的引力 F(r1),與距 q1 有 r2 遠的質點其受引力 F(r2) 相比，便知 F(r1)> F(r2)，正合我們的需求。

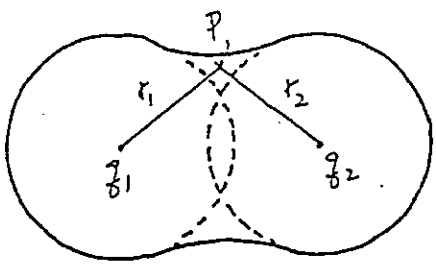
上述的理論,亦可以電子學的理論解釋。亦就是說，q1 是帶負電荷的電子對其周圍的正電荷電子，影響的範圍R內施以引力，聚合成一圓形區域的引力場。而在引力場之外的電子便任其游離。



上述只對單一中心質點形成的影響區域討論，如果兩中心質點相接近時，它們對周圍質點的作用又如何呢 ？

・相互接近的中心質點的作用範圍

依據場力函數的特性，兩相接近的中心質點對周圍質點施引力而在外形上會由原本的圖形而有融合的形狀，如示：



$F(r1) < F(R')$ but $F(r1) + F(r2) > F(R')$

$F(r2) < F(R')$

也就是因為，原受力小於 F(R′) 的質點 (各別對 q1 及 q2 而言)，但因為其受的總力 (Fq1 (r1) 與Fq2 (r2) 相加後所受內聚引力)，足以將其包合起來，故一種溶合的現象便產生了，而這種現象正合於我們設計流質紋理的外形。

在這必須強調的，不同 Geoffy Wyvill的是，Geoffy Wyvill 對 Soft Object的塑型，是對 3-D 的物體建立而設計；而我們只是藉其融合的特性產生流質紋理的外形，而此外形只有 2-D 輪廓，至於要達至 3-D 凸起的效果便由前述的Normal Perturbation 的方法，將此流質的外形，依法向量改變的漸進方式形成拉高的效果。

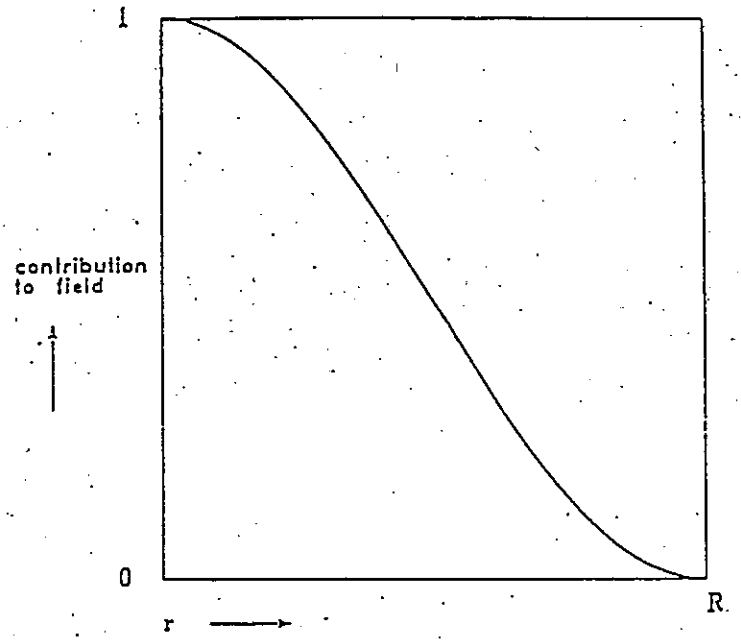
・對於中心質點可控制的參數

我們對於中心質點，提供了 3 項可調的參數分別是，作用範圍 R′，顏色，及Moving Path。依據這些可調的參

數我們製造了如冰淇淋溶化的紋理效果。設計者可經由這3項可調的參數設計出自己的流質紋理。

3.2.2 場力函數用於 D 與 N 的控制

依據 3.1節中對平面上製作圓形凸起紋理的方式，我們擴充它的應用，以場力函數的特性來控制 D 與N。對於單一的水分子的作用範圍（也就是一個圓）的製作方式與前述無異，只是以場力函數使 D 與 N 變化得均勻些而已



$$|N| \cong F(r)$$

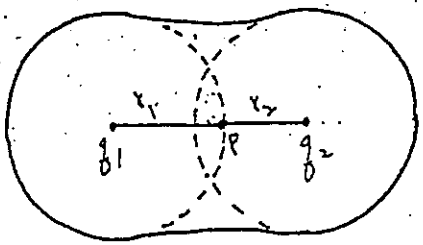
$$|D| \cong 1 - F(r)$$

。N 與 D的方向也與前述無異。N 的量與 F(r) 成正比，而D 的量與 (1－F(r))成正比；也就是說 N的單位向量乘以 F(r) 得一個 N，D 則乘上 1-F(r) 得一個 D′ 的量。

$$N' = \frac{N}{|N|} * F(r)$$

$$D' = \frac{D}{|D|} * (1-F(r))$$

至於結合中的質點，其作用範圍在 D的影響，我們分"方向" 與 "量" 兩項來討論。對於同一點 P；分別受 q1 與 q2 的影響，得 F(r1) 及 F(r2) 的引力，因為 P 受 q2 的引力較大 (r1＞r2)



P 對 q2 求得的 D2 大於 q1 的 D1，採合向量將 D1 與 D2 相加，我們可求得於 P 點的切向量 D3 的方向為：

$$D_1' = \frac{D_1}{|D_1|} * F(r_1)$$

$$D_2' = \frac{D_2}{|D_2|} * F(r_2)$$

$$D_3 = D_1' + D_2'$$

D3 為位在 P 點的切向量 "方向"，至於 D3 的 "量" 得再代一次場力函數而得之。此時，P 點的場力值為 F(r1) 與 F(r2) 之和，故計算為：

$$D_3' = \frac{D_3}{|D_3|} * (1-(F(r1)+F(r2)))$$

我們以兩相結合圓形凸起的側面圖，對 D的計算作一解析會比較清楚。

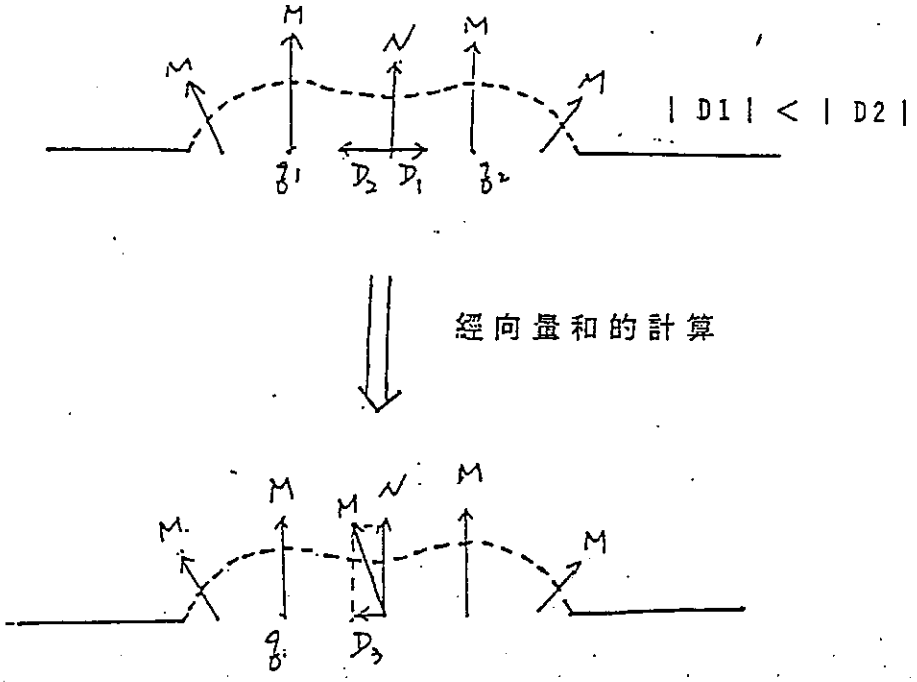


圖 3-2 圓形凸起的側面變化圖

至此，我們均假設 N朝同一方向，而事實上，我們也針對不規則面的法向量作上述流質紋理的模擬，其計算方式與上述大致相似。基於此理念，我們設計了冰淇淋的紋理。以及餅乾殼上的溶化後的冰淇淋紋理，之後，我們對這複合式紋理有進一步說明。

3.3 其它與 Normal Perturbating 有關的紋理

前節曾對流質紋理的產生詳述過了，除了流質紋理外，根據第一章對紋理的幾何元件的討論，我們設計了一些

簡單的幾何元件，經 Normal Perturbation的紋理技術而產不同凹凸效果的紋理。

・線條紋理：

將一些粗細不等，且長度不等的線段，加以組合間插地安置於物體的表面上，施與 Normal Perturbation技術，將其提高使之凸起。如此一來，到有幾分似樹木的樹皮的紋理。

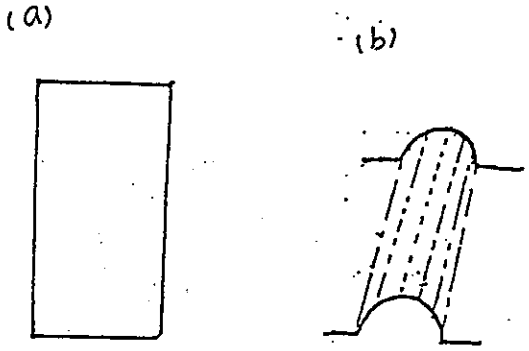


圖 3-3 一條線段的正視圖 (a) 與其凸起的透視圖 (b)

・漣漪紋理

我們是針對一個物體或水點觸擊水面時所形成的向外擴散的漣漪加以模擬。它的基本元件是一個個同心圓組合而成，每個同心圓也給與不等的寬度。再以漸次向外擴展而逐漸增加同心圓的個數而達動態的效果。

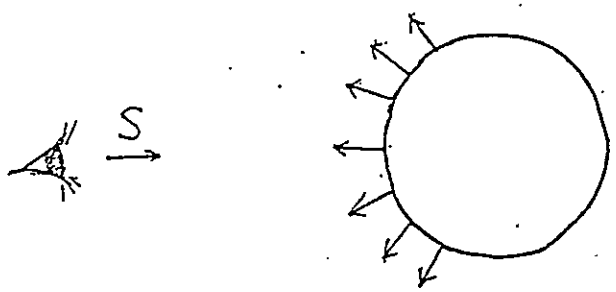
此外，還有高爾夫球紋理、餅乾殼紋理等。可參見第五章的實驗結果。

3.4 與透明度有關的紋理

說與透明度有關的紋理有雲、煙、火焰等。正如第一章所述，對於這些紋理，透明度的作用，不外乎是表示它們濃度及明暗的層度。例如，透明度不大則雲層便顯得厚些，同樣地，透明度弱則火焰的光度就暗些。

因為基本上這些紋理是由微細的粒子（例如球及橢圓球）組合而成，也就是說它們的幾何元件是球或橢圓球，所謂的透明度便是對這些 3-D幾何元件做處理。我們製造的煙是以球為基原件，對球做透明度的控制。接下來是我們的討論。

當觀測一個透明玻璃瓶時，我們發現正視它的部位是最透明的，而與觀測中心愈遠，形成一夾角的部位因為厚度的因素會成半透明的情況。由此得知，透明度與觀測者和物體法向量夾角的 Cosine 值成正比。



t : transparency factor

$$t = \frac{-SN}{|S||N|}$$

圖 3-4 透明度的討論

為了使透明度的變化能收斂地快些，我們對 t 設計了與 Shinny Coefficient 一樣的，透明度系數；

$$t = t^n$$

在 n 愈大時物體就愈不透明，只有中間一點(也就是正視眼睛的點)是透明的。而反之 n 愈小，收斂地慢，物體就愈近透明的層次。

對球物體做了透明度的處理後，我們將數十個球不規則的組合，各個球給與不同的透明層度，以製造似輕煙的紋理。混濁的球組合而成的煙，濃度大；較透明的球組合則濃度小。此外亦可對球的顏色改變而形成不同顏色的煙幕。

3.5 複合式紋理

前幾個章節，我們對各種 Texture的製造法詳加的描述過了。在這節中，我們強調一種新的概念，稱為紋理的複合法。

以往的紋理處理方式，不外乎是一個物體只給其一種紋理。尚未有各個紋理可相互組合形成複合紋理的情況。所謂複合紋理便是將由設計者已經設計好的各項紋理，及處理紋理的方式，經由先後的順序安排，產生一新紋理。我們且以冰淇淋的實物加以說明。

前節我們對於處流質紋理以及餅乾殼的紋理已經介紹過了。對於我們在製作冰淇淋溶化的效果實物時，首先對圓錐體表面施與餅乾殼的紋理，而其上的冰淇淋球正隨時間逐漸變化著，因此圓錐體不只要有餅乾紋理仍得有流質的紋理，由此觀之，必須對已上餅乾紋理的圓錐體再施與流質的紋理。所以餅乾紋理加上流質紋理便形成了新紋理效果，我們稱此效果為複合式紋理(Composited Texture)。

餅乾紋理＋流質紋理＝有溶化冰淇淋的餅乾殼

冰淇淋是逐漸溶化的，因此設計者可以依時間的變化，產生冰淇淋溶化的各個變化過程。

時間 1：餅乾殼紋理＋流質紋理 1＝時間 1的餅乾殼

時間 2：餅乾殼紋理＋流質紋理 2＝時間 2的餅乾殼

時間 3：餅乾殼紋理＋流質紋理 3＝時間 3的餅乾殼

•

•

•

這便是一動畫設計的目的。如此隨時間的改變而設計的紋理更可達到真實逼真的效果。

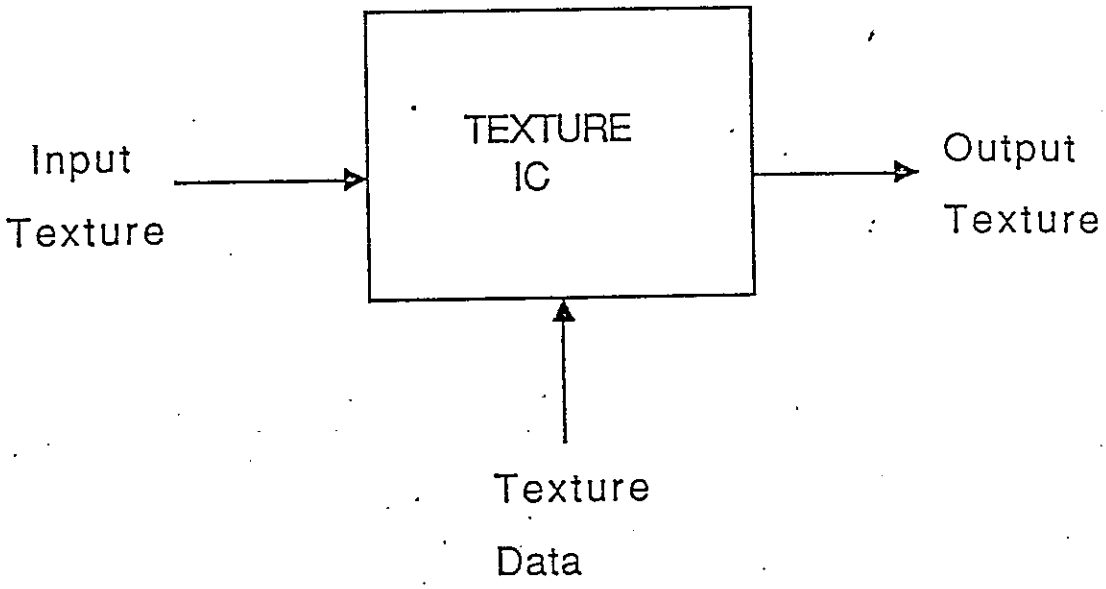
基於對複合式紋理的操作，為了方便之故，我們又提出了一種類似IC元件的 Texture-IC,使用者及設計者可方便地將各個已經設計好功能的紋理處理方法以循序地方式相互連接而製出新的紋理。下一節我們便對此方法加以說明及目前使用中對於 Texture-IC 使用上的描述語言。

3.6 紋理-IC

討論過複合紋理的製作後，我們參考了邏輯電路設計及物體導向語言的概念，設計了以紋理-IC 為基楚的紋理系統，提供我們對紋理設計的介面，進一步地能供設計者、作畫者一個方便的設計環境。

我們對於每一個紋理處理方法均給與一個IC編號，而

其IC的圖為：



每顆IC各有一個輸入、輸出及紋理資料的接腳。對於每個接腳的使用方式，我們說明如下：

• Input Testure 接腳

自其它紋理-IC 的 Output Texture 接入。依據複合式紋理的概念，第一顆紋理-IC 的此接腳為接地。

• Output Texture 接腳

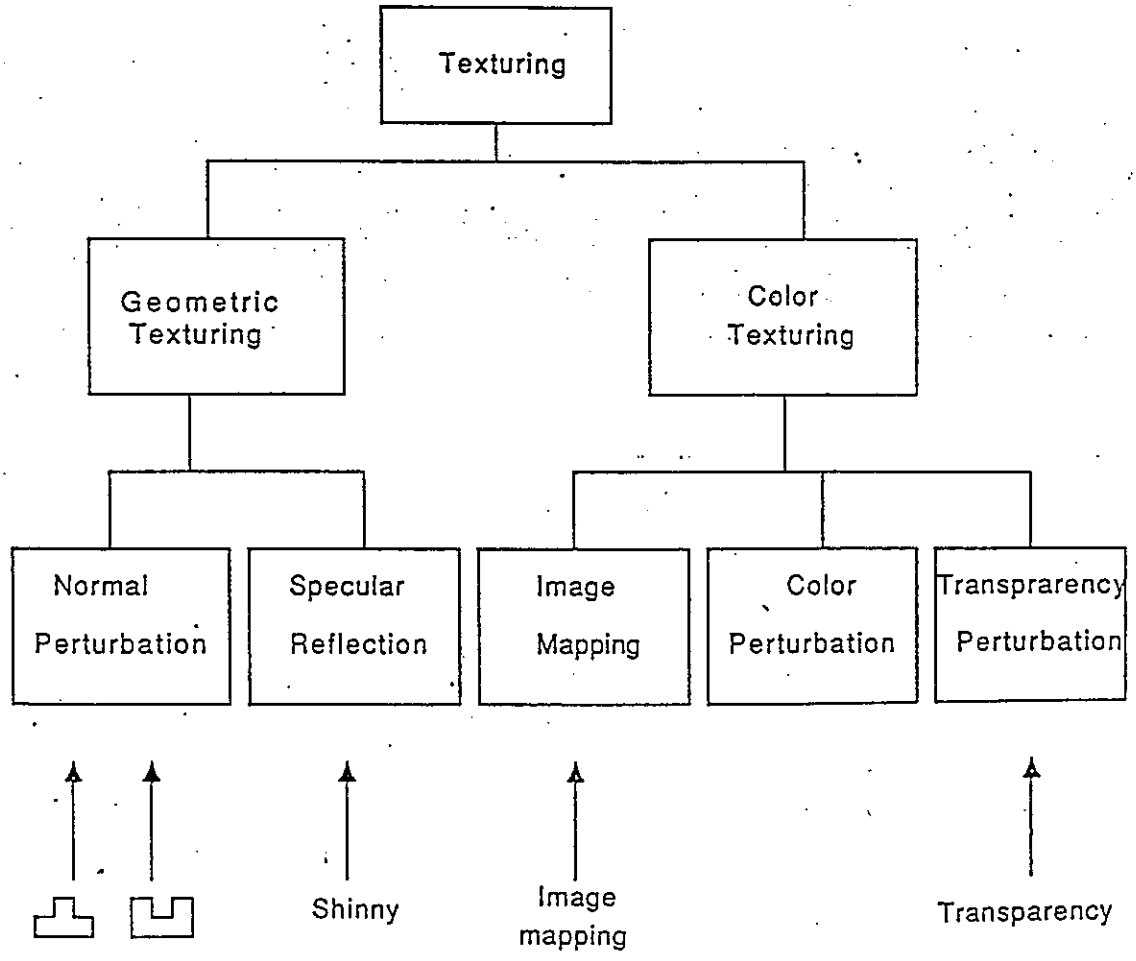
此為經過紋理-IC 處理後的紋理效果的輸出接腳。此接腳可接至下一顆紋理-IC 的輸入接腳做更進一步的紋理處理。

• Texture Date 接腳

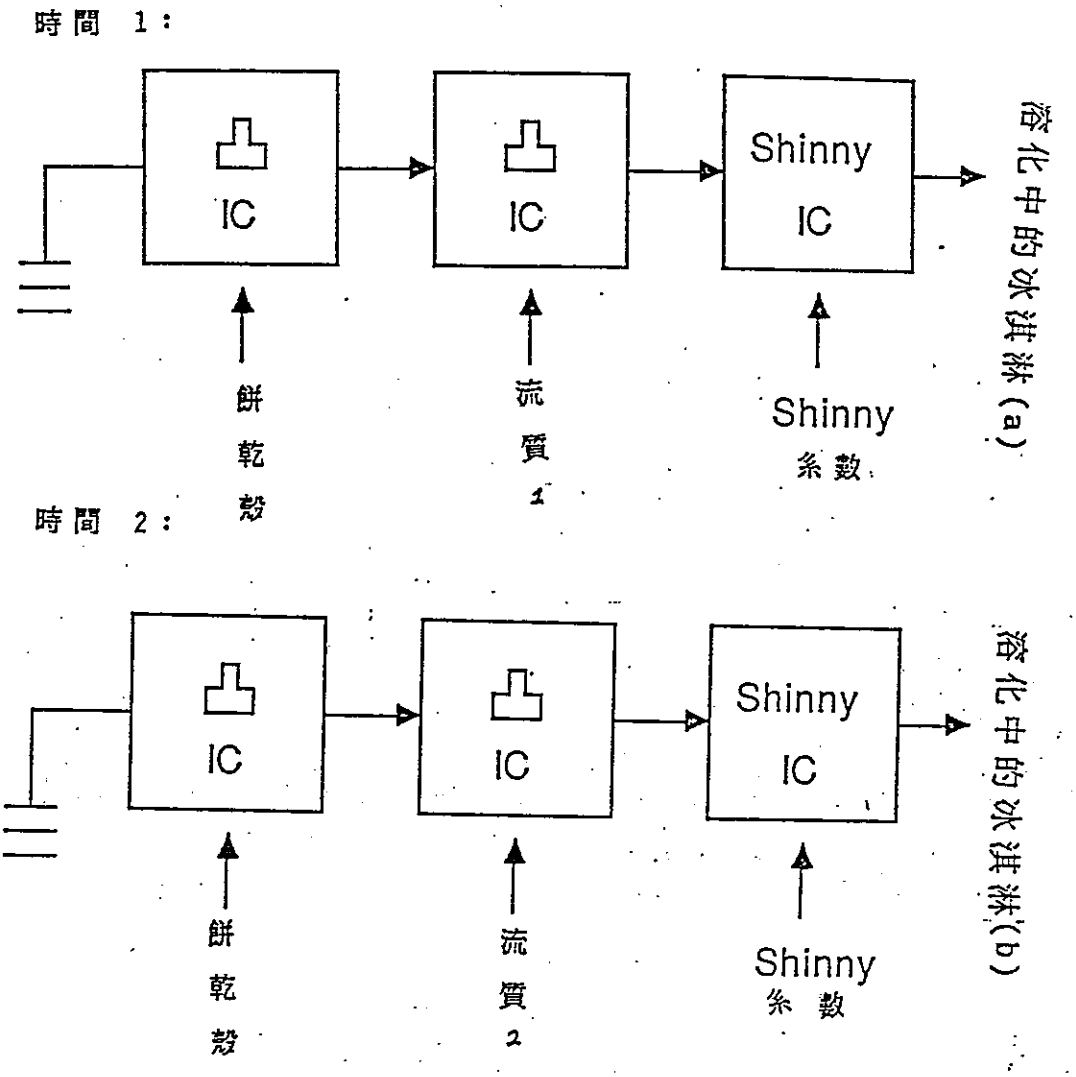
這支接腳是紋理產生的命脈。將先前討論過的例如流

質紋理或餅乾殼紋理的資料自此腳輸入。也就是說流質是由多少質點組成，將這些質點的位置，顏色等相關資料，以檔案的形式輸入此腳。

目前，我們根據設計中的紋理技術，依次定出了一組紋理-IC。它們分別為:凸起-IC，凹-IC，Shinny-IC，Image Mapping-IC 及透明度-IC。



接下來，我們將紋理-IC 與複合式紋理的使用方式以圖析說明之。首先以冰淇淋說明之：

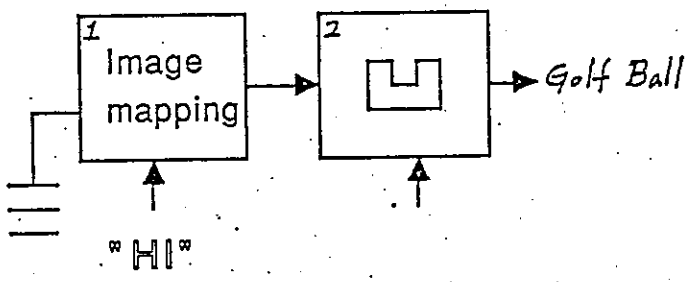


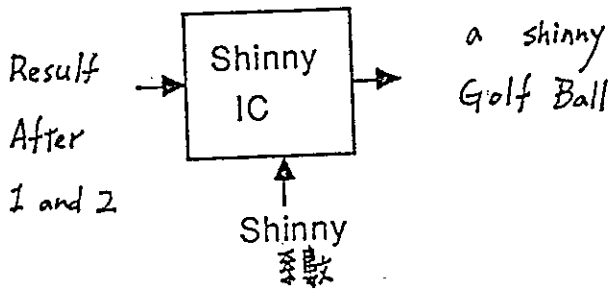
我們很明顯看出來，以IC為設計觀點的好處，使用者只須拆下中間那顆紋理 -IC 換以新的紋理-IC，可得下一個時的冰淇淋溶化的紋理。

對於紋理-IC 在使上是相當方便，設計者不必知道 Normal Periurbation 的技巧為何，只要知道這方法使物

體凸起或凹下的效果。如果欲使物體看起來光亮些，又可再將已經設計安裝好的IC線路圖加上一個光亮的IC，使物體的效果更好。

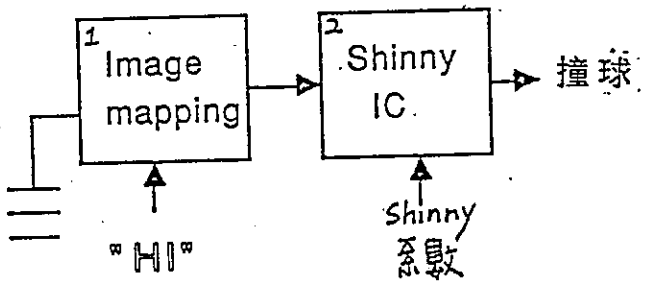
設計者只需裝拆IC便可設計不同的物體表面效果。如欲設計一個有 "Hi" Mark 的 Golf Ball 的IC設計可如下安排：





如果使用者想改變設計，只希望設計一顆撞球，則只須拆

下中間的IC就可以了。



目前紋理-IC 的 2-D Graphic介面仍在未來研究的範疇，目前我們的系統是採一種描述語言 (Discription Langnage) 的使用介面。也就是說，對於每個物體均由一組描述語言來描述，而將這一組描述語言以檔案的方式輸整個作畫系統。接下來我們對描述語言各條加以說明：

```
title       cone for ice-cream
detail      cone.det
type        polygon
texture-begin
    bumping    cookie
    bumping    melting1
    shinny     150
texture-end
```

• Title：只是個註解，說明物體的用處。

• Type：表示物體是由多邊形組成。

• Detail：指示物體的多邊形檔案的名稱。

• Texture-begin 與 Texture-end：

表示於此之間的，為對於物體欲做的紋理處理。這是依循紋理-IC 的構架而設計的。

我們希望能完全地將目前這個紋理系統擴展成以物體導向語言，使其操作上更方便宜人。

在本節中，我們提出了紋理-IC 及描述語言中的有關紋理的指令。也就是說我們設計了一個簡單的直譯器，將這些描述語言代入相應的處理單元，對物體作適當地紋理處理。接下來是我們整個紋理的系統架構。

由圖中不難發現，我們只針對了描述語言的介面設計；此外，物體塑型的介面我們採用美國海軍大學設計的一個簡單的塑型介面。對於紋理資料的設計介面乃是未來努力的方向，促使設計者能有效且方便地製造出自己的紋理幾何外形及元件。換句話說，我們希望這整個架構均能以似 MS-Windows 的介面提供給設計者使用。

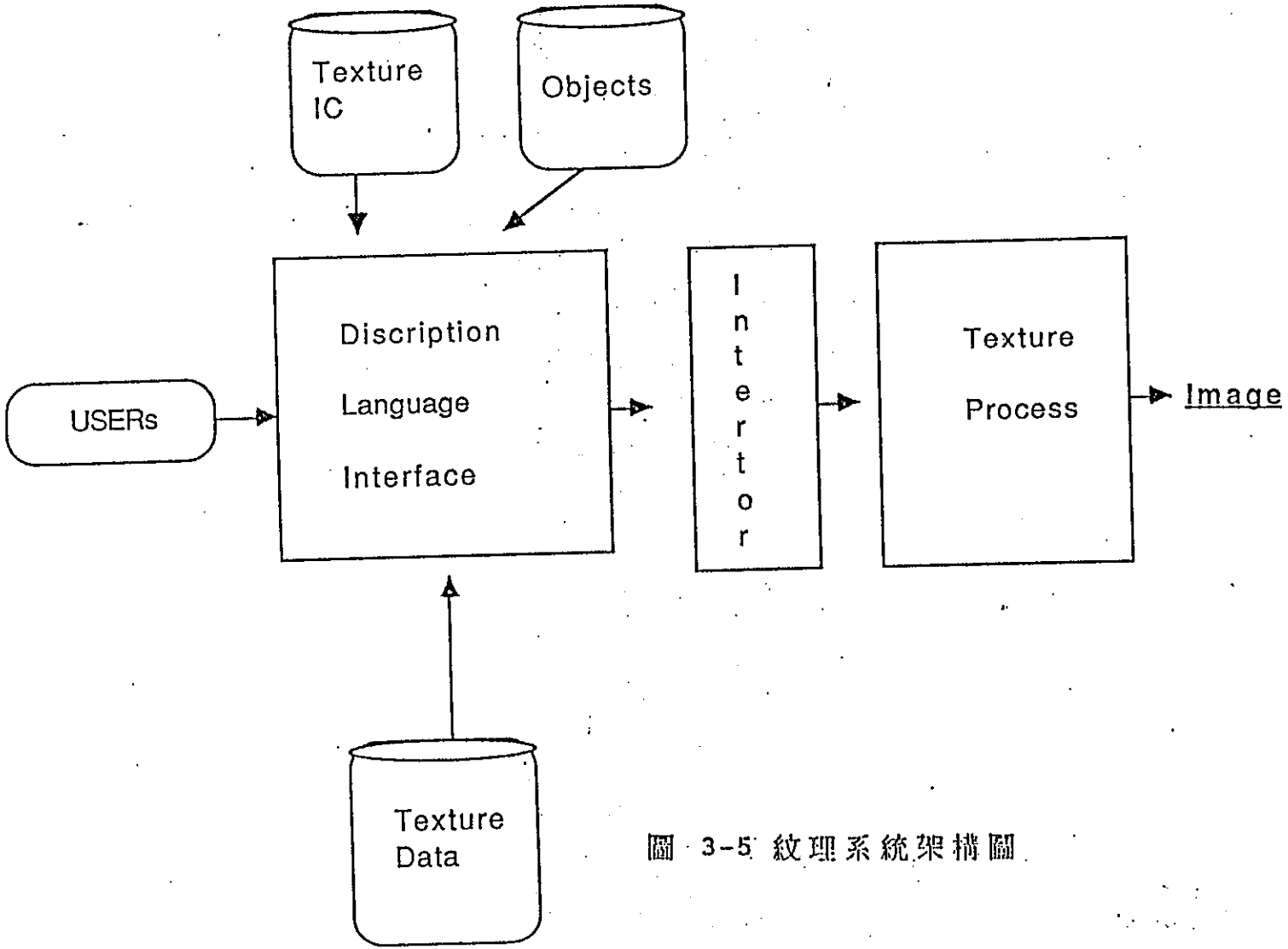

Texture
IC
Objects
Discription
Language
Interface
USERs
Interator
Texture
Process
Image
Texture
Data

圖 3-5 紋理系統架構圖

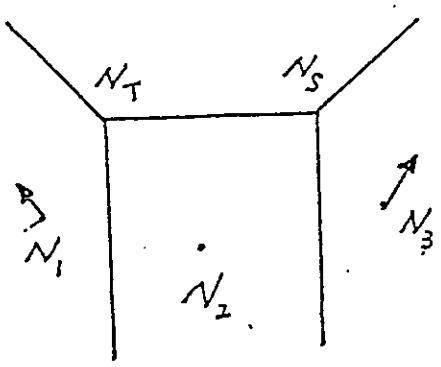
第四章　系統流程

在第三章中，我們對紋理系統的內部演算法則及採用的介面（描述語言）做了深入的討論。在本章中，我們針對整個系統流程加以說明。這其中包含了隱藏面消除法：掃描線演算法（Scan-Line Algorithm）及紋理座標的對應方式等。

4.1 Scan-Line Algorithm

在做紋理時，我們一定得求得多邊形面上每一點的3度空間座標及每一點的法向量。求3度空間的座標是因為計算點的切向量(在做 Normal Perturbation 時）或是代入求取紋理的顏色等，而法向量在透明度，明暗度及 Normal Perturbation 時均用得上。在計算點座標及法向量上，我們是根據 Scan-Line Algorithm。

首先，進入 Scan-Line Algorithm之前，我們得先求得多邊形上每一頂點的法向量，這個法向量的求得，以下圖觀之為：

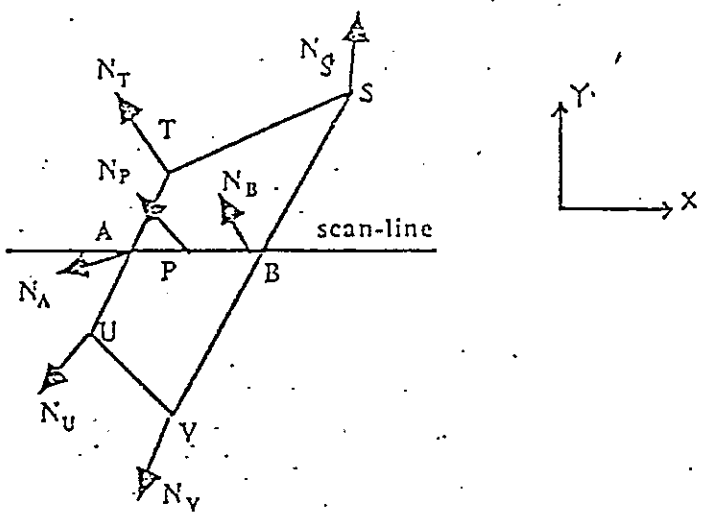


$$N_t = \frac{N_1 + N_2}{2}$$

$$N_s = \frac{N_2 + N_3}{2}$$

為了使物體看起來圓滑，我們對多邊形頂點法向量的計算是與共此頂點（如 t 點）的所有面的法向量之合平均。如此物體在上色時才不至於顯得一塊塊的而不連續。

求得每個頂點的法向量後，每次將一塊多邊形代入 Scan-Line Algorithm，以求得每一點的法向量。此時，雖然每一個多邊形已經過透視投影至螢幕座標(x, y)上，但在這之前，對於其相應的 3 度空間的資料，例如頂點的座標及法向量等均已保留，至此時再度取出來計算。至於每一點的法向量計算方式，我們以下圖來解式求取面上一點 P 的法向量。



$$N_A \doteq \frac{N_t(Y_A - Y_u) + N_u(Y_t - Y_A)}{Y_t - Y_u}$$

$$N = \frac{N_s(Y_B - Y_v) + N_v(Y_s - Y_B)}{Y_s - Y_v}$$

$$N_P = \frac{N_A(X_B - X_P) + N_B(X_P - X_A)}{X_B - X_A}$$

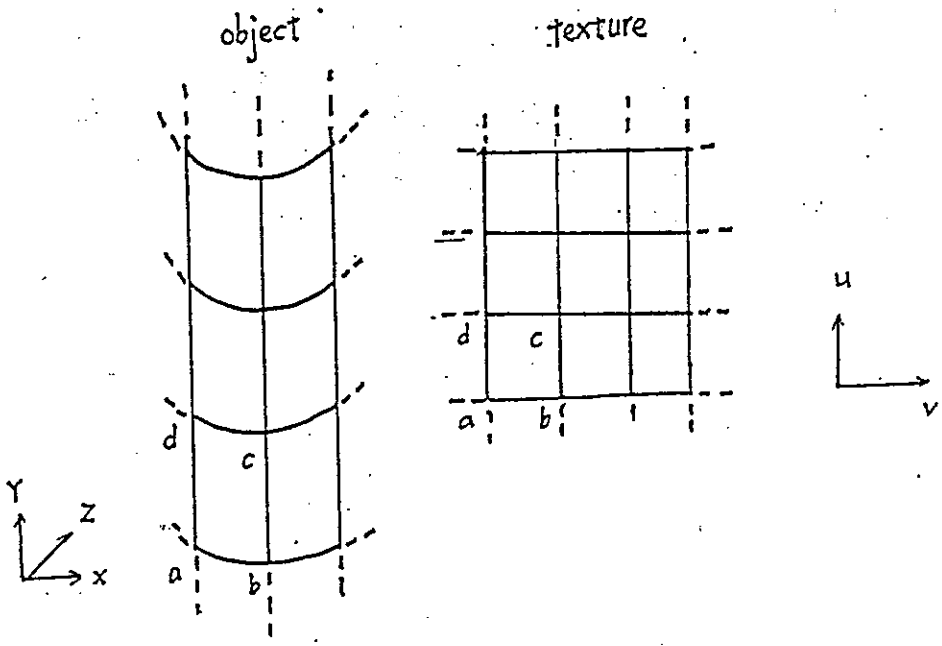
圖 4-1 Scan-Line Algorithm for normal calculation

由圖觀之，我們是先以線性內插方式求得 A點的法向量，同樣地方法求得 B點的法向量，最後再以 A，B 兩點求得 P 點上的法向量。

在計算法向量的同時，我們也計算了點 P的 3 度空間座標及相應的紋理座標。因為方法相似便不再贅述。只須

將法向量改成 3-D座標即可。

至於紋理座標的計算，我們做進一步的說明。系統中物體均切成以四邊形組成的物體。而我們的紋理資料，現階段是以2-D的一張紋理平面為設計者畫布(如果能進一步地採 3-D作畫空間會更好，這是未來努力的方向）。因此，我們也將紋理的平面切成一片片四邊形，與物體的四邊形一一對應。而物體四邊形頂點座標與紋理的四邊形頂點座標也相互對應，在求取物體某一面上點 P的紋理座標便可以求得了。



如此規則地切割與對應，我們可保証物體多邊形在相鄰邊界上的紋理不會發生不連續的現象。(這是一般紋理

處理中最常發生的現象。)

由上對紋理資料的介紹，我們不難發現，設計紋理資料的介面現階段是以 2-D的方式。例如設計一流質紋理，設計者可面對一張紋理資料畫布點上質點及設定質點的作用範圍和顏色，便可以先在 2-D平面上看見預計的流動效果而進一步地修改或送入紋理系統中產生 3-D的紋理效果。但如果發展成 3-D的紋理資料及設計介面時，就無法有這般方便的操作了。這是互有利弊的選擇。

4.2 Scan-Line Algorithm for Hidden-surface Removal

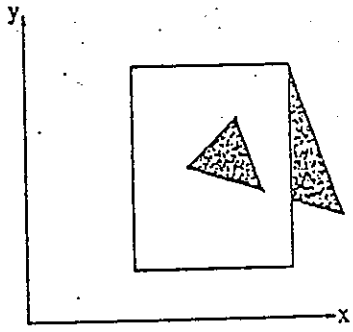


圖 4-2 Hidden-surface Removal with Z-buffer.

當遇螢幕上同一個 pixel中有兩個多邊形的點座標時，便利用在做 Scan-Line時也計算了兩多邊形此點上的 Z座標，我們依其 Z值來決定誰在前誰在後。而 pixel顯示

在前的顏色。如此一來，在後面的多邊形便被居前者的多邊形的顏色遮住而達到隱藏面消去的目的。同樣地，具有Z-buffer的 Scan-Line Algorithm，亦可做物體相互刺穿的影像。如圖：

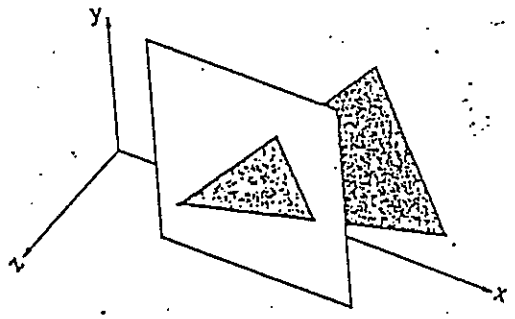


圖 4-3 物體刺穿的影像

也是決定經透視影後的物體，於同一 pixel上的點，決定那一個 Z值較小而顯示在前者的顏色。

雖然 Scan-Line Algorithm一次只針對一塊多邊形處理，但具有 Z-buffer 記憶的演算法，可解決了隱藏面及穿透式影像上的問題，是個簡易的好演算法。

4.3 Anti-aliasing

Aliasing 是電腦繪圖不可避免的問題。舉凡，階梯式的邊，微小物無法顯現及線段會有中間斷裂的現象均是

Aliasing。這些是空間中的 Aliasing 的情況，如果將時間的 aliasing 亦考慮，便是另一門的學問，我們有同學做此方面的研究。

有了 Aliasing 問題，就有了解決 aliasing 的方法，稱為 Anti-aliasing。一般說來，anti-aliasing 的技巧有 3 類，分別為增加螢幕的解析度，Super-Sampling及prefiltering。在我們的系統中，Anti-aliasing 是採 Super-Sampling 的方法。

所謂 Super-sampling便是將原本螢幕上的一個pixel對應的多邊形上的點，不再是一對一，而是一對多。

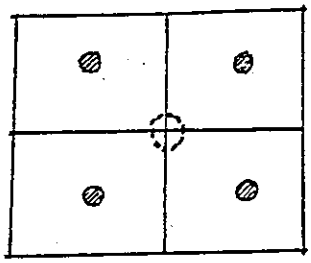


圖 4-4 Super-sampling for one pixel

如上圖所示，原本一對一的情況是只取中間的點，但super-Sampling 處理後，同一個 pixel經取樣 4 個點，將這 4

個點計算後的結果平均之後才為這個 pixel的值。不必多說，這麼一來，一定比原本只取一點來得客觀。

目前我們只取 4 點對一個 pixel，當然多取幾個取樣點一定會好，但時間是另一個考慮的因素。經4個點Super-Sampling 的結果大致尚稱良好。

4.4 系統描述

經前幾節的演算法介紹，整個系統的處理方法就很清楚了。接下來是幾個概略的系統流程，希望能對了解先前介紹過的演算法的使用及系統操作有所助益。

值得一提的，本系統是架在我們自美國柯蘿拉多州大學發展的 Animator 系統下設計的。在他們的系統中，我們擴展了紋理系統及紋理描述語言，以助往後研究的學員能有很方便接手之處。

在第 5 章中，我們展現了一些以本系統製造的紋理結果及對這些結果做進一步解說及評析。

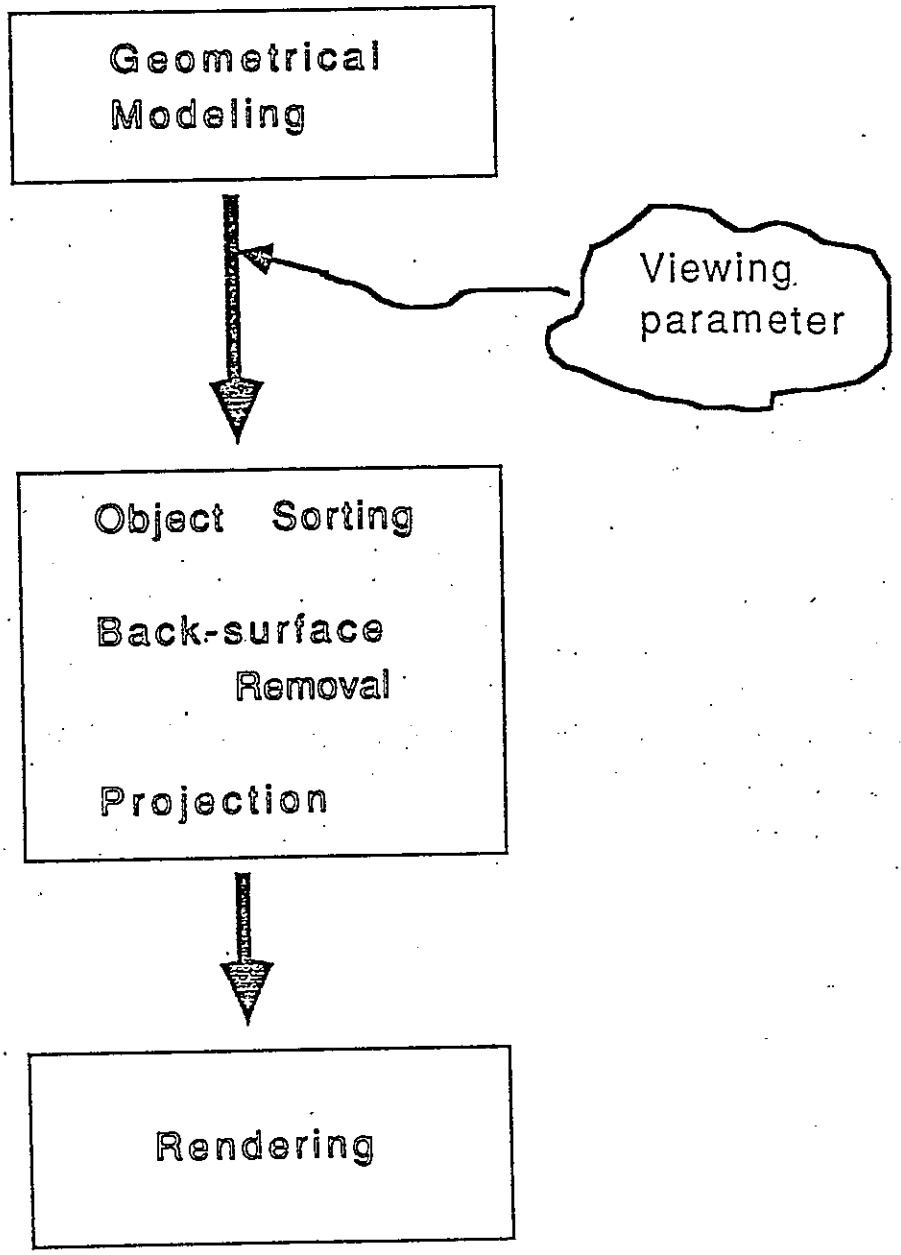

Geometrical
Modeling
Viewing
parameter
Object Sorting
Back-surface
Removal
Projection
Rendering

圖 4-5 系統主流程

Rendering

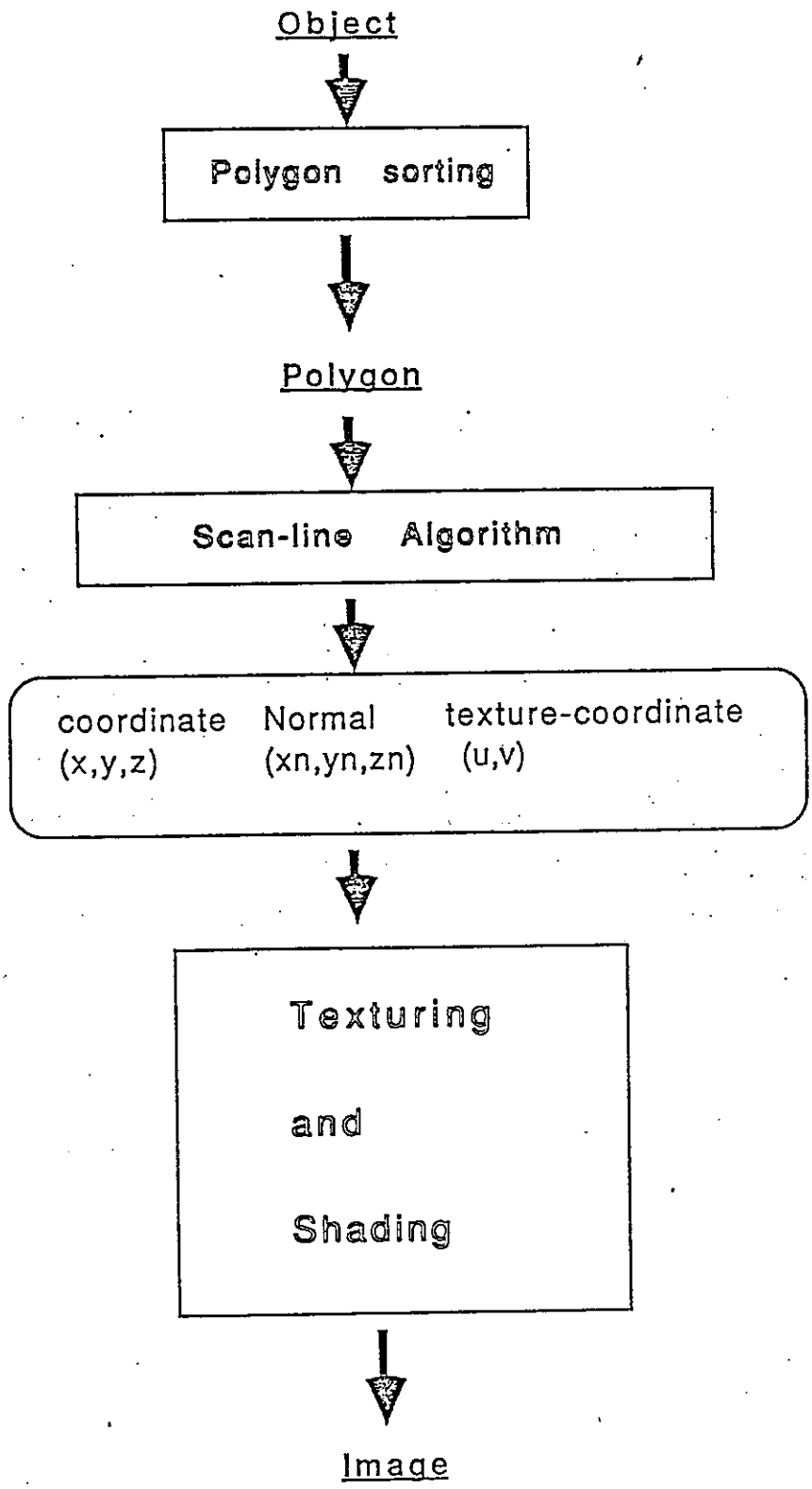


圖 4-6 Rendering

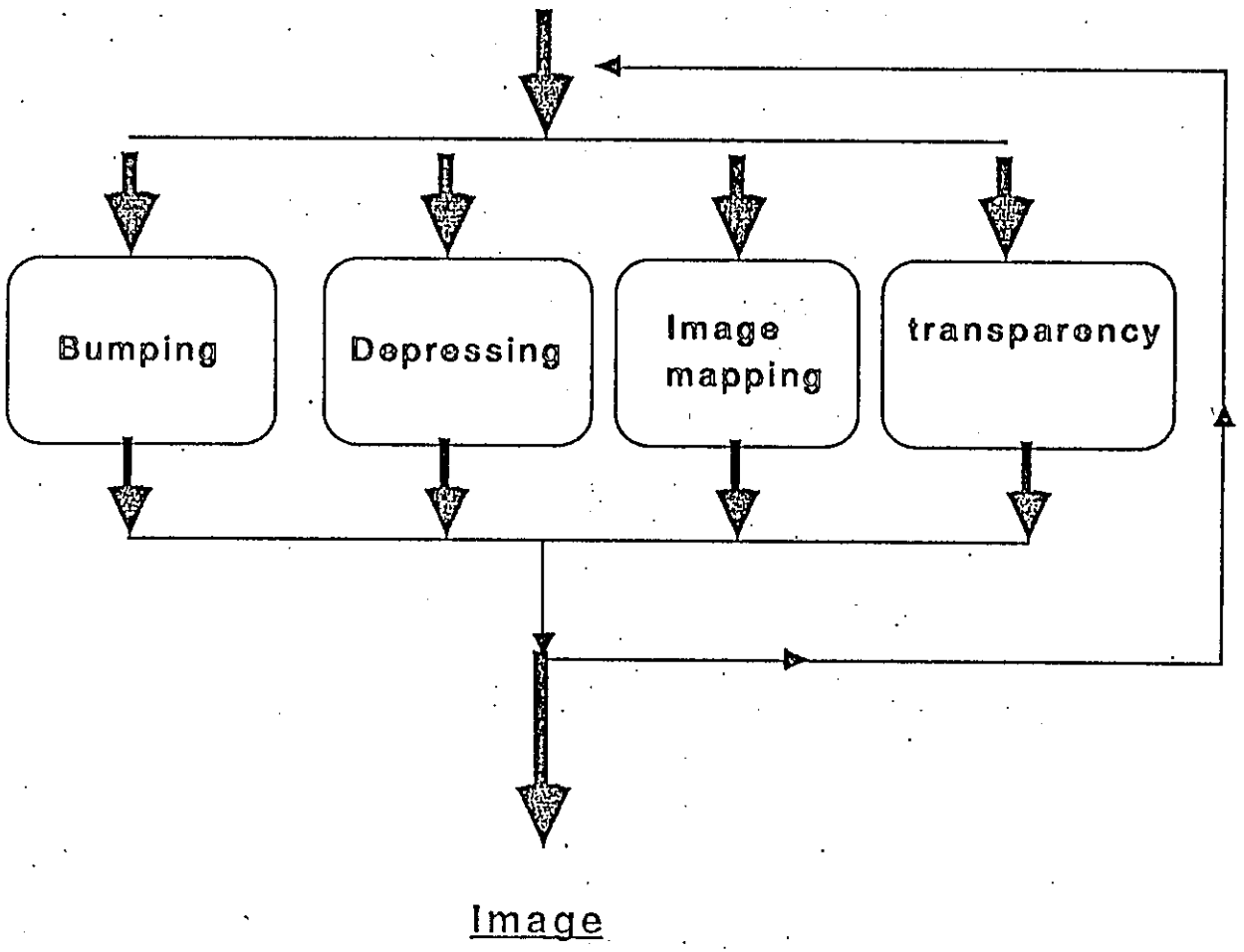
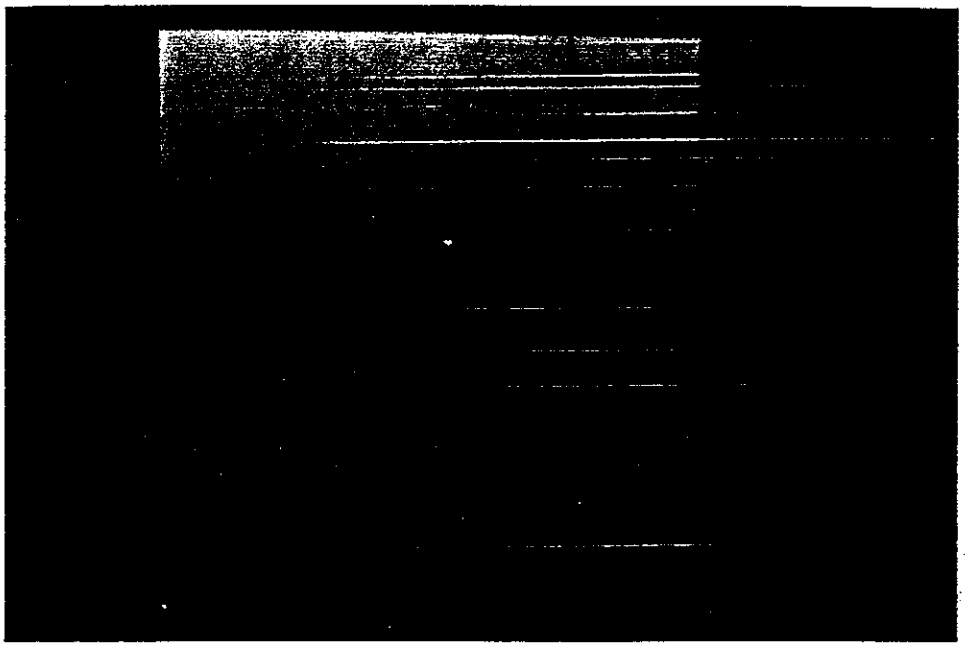


圖 4-7 Texturing 及 Shading

第五章　實驗結果與評析

根據第三章及第四章對於紋理系統及系統架構，我們是在 VAX 11/750 及 SUN MICRO下設計我們的實驗程式。而實驗的結果則是顯示在 E&S 的 PS系列的工作站上。

發展此系統時，我們以美國柯羅拉多州大學的Animator軟體為基礎，於其下加入我們的紋理實驗環境。我們擴充了 Animator 中 Scan-Line的程式，也就是加入Z-buffer以處理各種隱藏面的問題；之外，便是紋理系統了。為了速度上的問題，我們的實驗結果是 280×280 的解析度，如果放大觀之難免有 aliasing的現象。但這些結果只花了不過 1 分多鐘的時間(在 SUN MICRO 下)，有的不至1分鐘。但在 VAX 11/750 下時，時間就多了10倍左右。



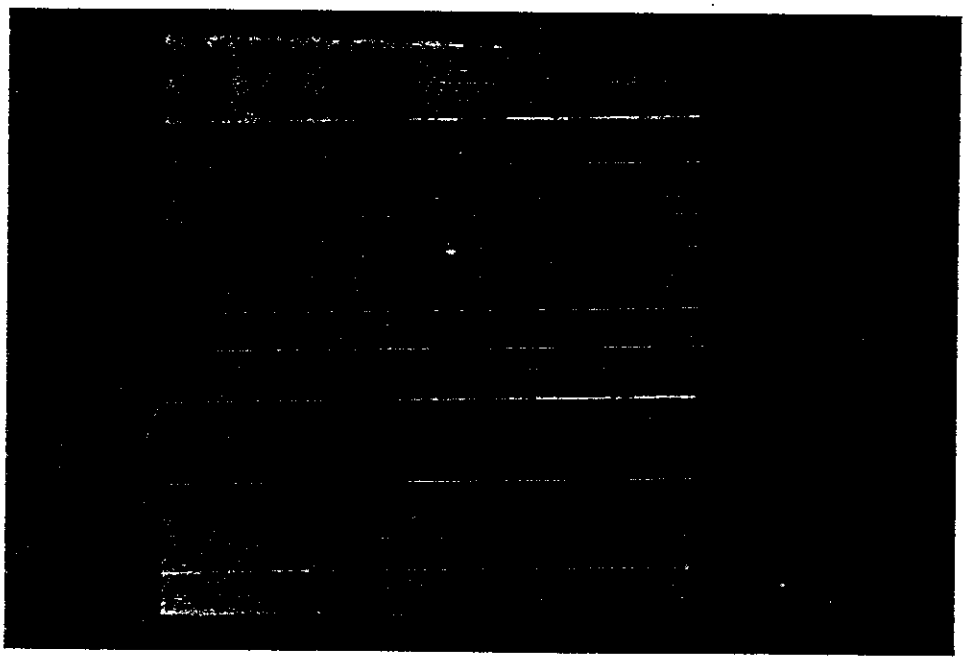


圖 5-1 溶化中的冰淇淋(a), (b)

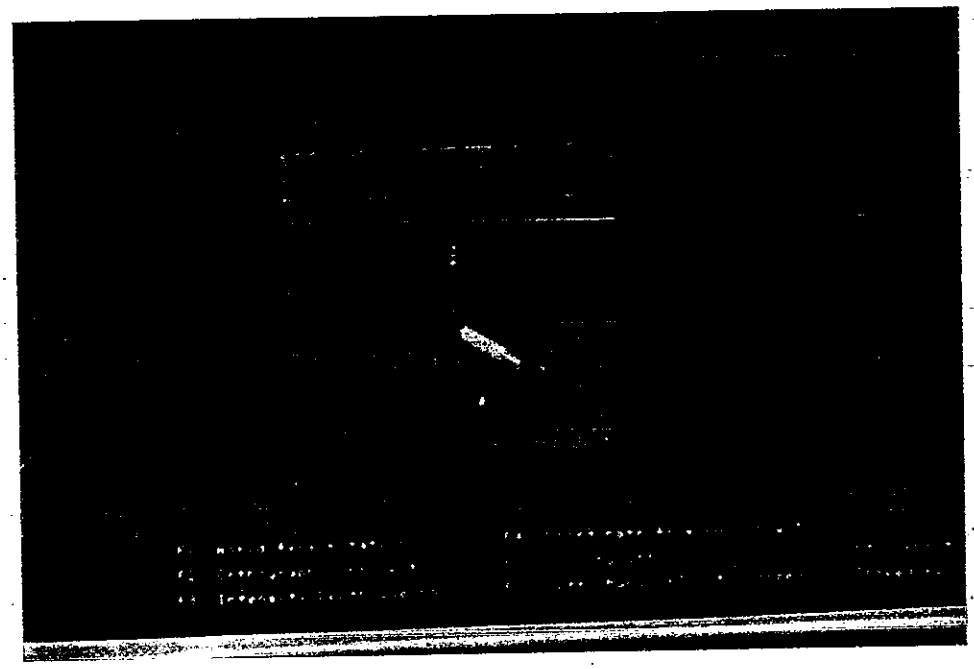


圖 5-2 煙幕與香菸

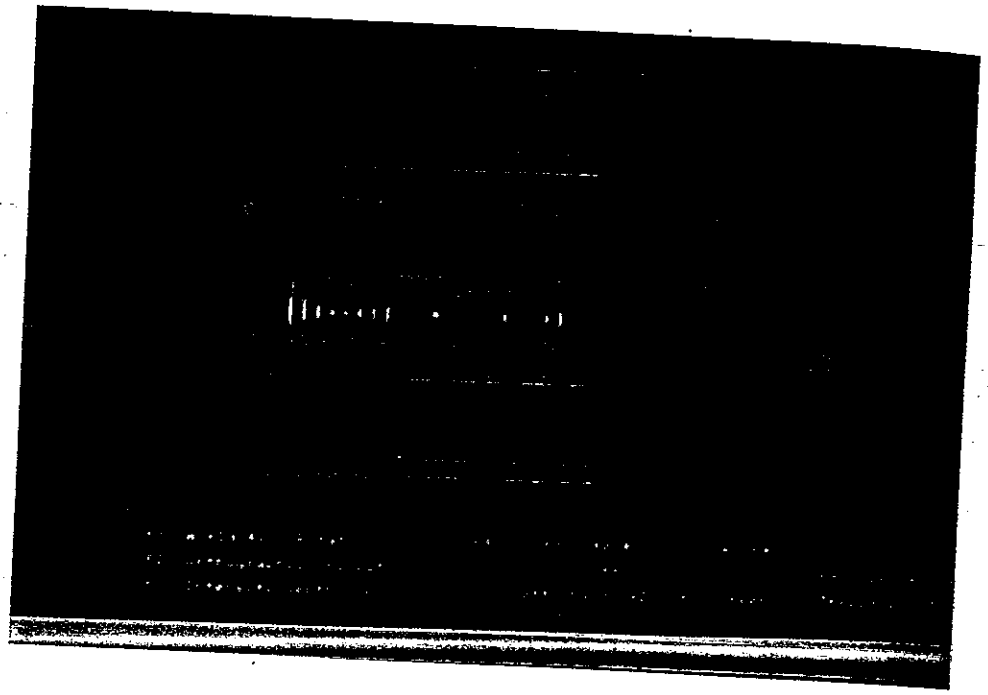


圖 5-3 圓形水紋

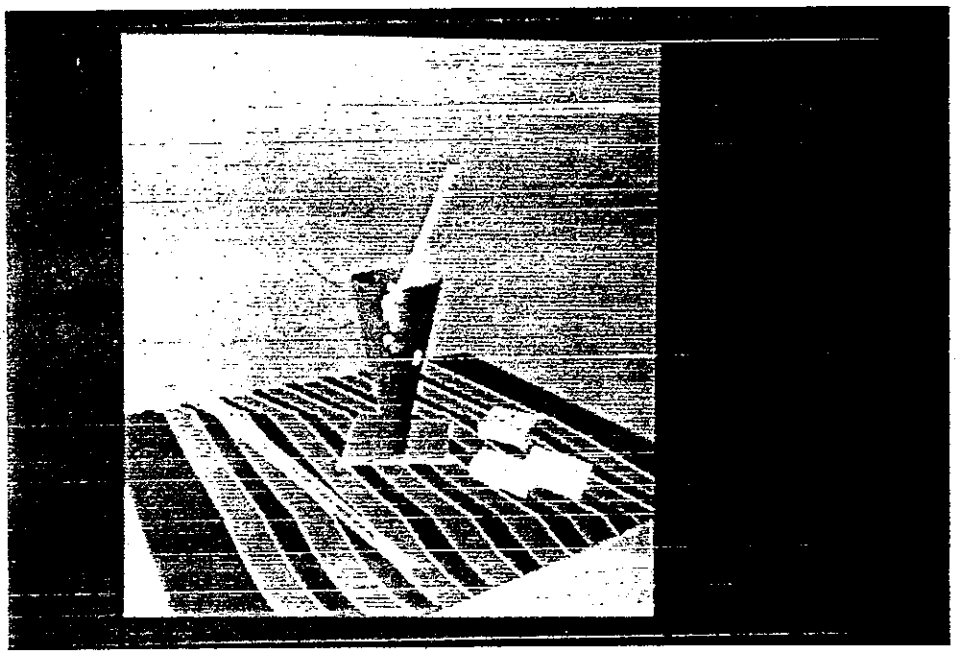


圖 5-4 飲料與餅乾

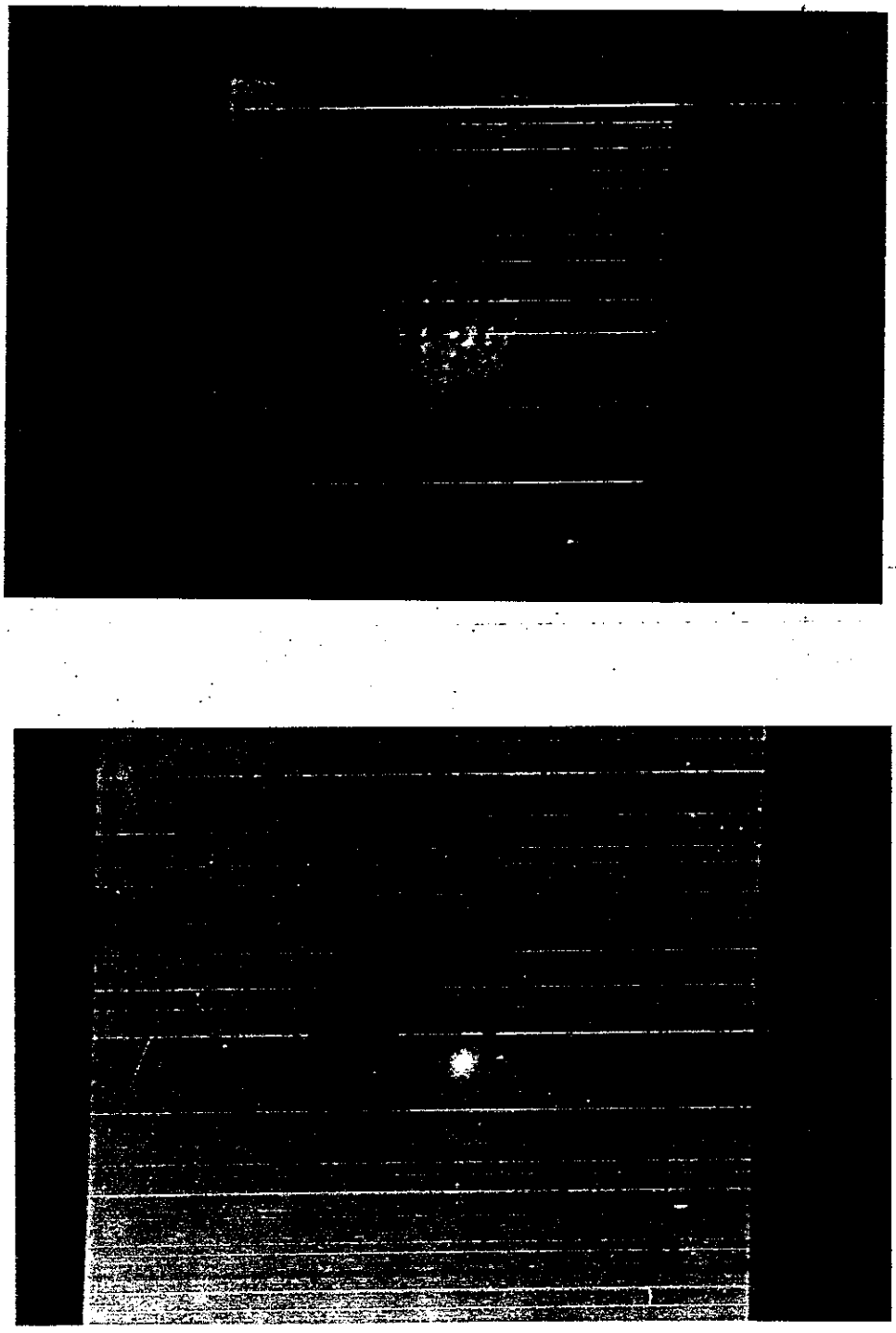


圖 5-5 高爾夫球與撞球

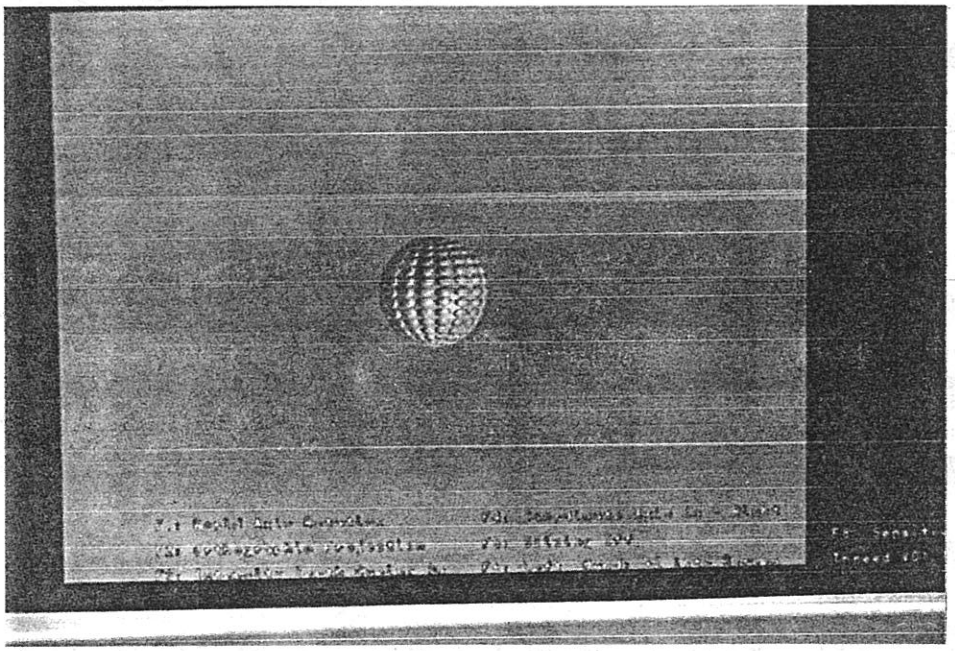


圖 5-6 圓形凸起

第六章　結　論

經本論文的討論，不難發現紋理在電腦繪圖中生動了物體的呈現。電腦繪圖最主的目的便是希望使製造的景象更真實迷人，紋理可說是這方面的一大功臣。再加上動畫的製作之需，動態紋理更是不可缺的一門技術。但是有鑑於以往學者將紋理的製造複雜化了，不是一般非專業的新手或設計動畫的使用者能順手地操作。但經本論文的研究，發現這些紋理產生法中仍有簡單易用之處。

首先，我們發現紋理均是由一組簡單的幾何元件組合而成，設計者可依循一些已經預先製造好的元件，適度地調整它們及組合，並設定時間的軌跡以求其隨時間變動而達動態紋理的效果。如本文中提及的流質紋理及及幕便是個簡化的紋理設計。其次，通常一種紋理只佈於一個物體面上，但我們卻允許二種或三種的紋理佈於一個物體面上。而為了設計上的便利，採用了IC介面的紋理系統，主要的目的是希望縮短使用者作畫的時間，避免太複雜的數學介入；另一方面，對於往後系統地擴充也有了整體的架構而有所依循。

當然，目前的系統仍是個雛型階段。不論是介面或紋

理的產生方法仍有待改良。一種以Object-oriented language 為主體的介面是我們的目標；而更Generalized 的紋理產生法也是有待努力的一環；增加更多更豐富的紋理更是當務之急。但不論是動態或靜態的紋理，我們提供往後有志於此的學者們一個參考。

在愈來愈多廣告用電腦繪圖的技術來製造短片（如新聞節目的片頭）及Visual Simulation 上的需求，一個不是專業電腦繪圖人員的使用者也能使用的環境是迫切的。我們設計本系統的主旨正是 "what-he-sketchs-is-what-he-gets"，而紋理系統的介入，更能滿足設計師們充滿多采多姿的想象空間。我們冀望不只是對學術界有所助益，對於商業，軍事訓練上也能有所貢獻。

參考文獻

- Blinn JF(1977) "Simulation of wrinkled surfaces." Proc. SIGGRAPH '78, Computer Graphics 12(3):286-292

- Blinn JF, Newell ME(1976) "Texture and reflection in computer generated images" Comm. ACM 19(10): 362-367

- Blinn J(1982) "Light reflection functions for simulation of clouds and dusty surfaces." Proc. SIGGRAPH '82 Computer Graphics 16(3): 21-29

- Blinn J(1982) "A generalization of algebraic surface drawing". ACM Transaction on Graphics 1(3):235-256

- Schachter BJ(1980) "Long created wave models" Computer graphics and Image Processing 12:187-201

• Reeves WT (1983) "Particle System-A technique for modeling a class of fuzzy objects" Computer Graphics 17(3):359-375

• Carpenter L, Fournier A and Fussell D (1982) " Computer rendering of stochastic models" Comm. ACM 25(6):371-384

• Rogers DF(1985), Procedural elements for computer Graphics, McGraw-Hill

• Peachey DR(1985), "Solid texturing of complex surfaces" Computer Graphics 19(3):279-286

• Gardner GY(1985) "Visual simulation of clouds", Computer Graphics 19(3):297-303

• Barsky BA and Harutama S(1984) "Using stochastic modeling for texturing generation" IEEE CG&A, March 1984

- Cook RL(1984) "Shade tree" Computer Graphics 18(3):223-231

- Wyvill G, McPheeters C, Wyvill B(1986) "Soft objects" In:Kunii TL (ed) Advanced Computer Graphics PP113-127

- Wyvill G, McPheeters C, Wyvill B(1986) "Data structare for soft objects" The Visual Computer 2(4):227-234

- Wyvill B, McPheeters C, Wyvill G(1986) "Animating soft objects", The Visual Computer 2(4):235-242

- Whitted T(1980) "An improved illumination model for shaded display" Comm. ACM 23(6):343-349

- Perlin K(1985) "An image syuthesizer" Computer Graphics 19(3):287-296

• Phong BT(1975) "Illumination for computer-generated pictures" Comm. ACM 18(6):311-317

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

• Magnenat-Thalmann N, Thalmann D(1985) Image Synthesis: Theory and Pratice. Springer, Tokyo

• Max NL(1986) "shadows for Bump-Mapped Surfaces" In: Kunii TL(ed) Advanced Computer Graphics. Springer. Tokyo. PP145-156

No.

程式

整个實驗環境是架在柯羅拉多州大學發生的 Computer Graphic 系統 — AnimaTek 下，於 AnimaTek 中，學生修改了自 pixels.c 以下的程式，便主能從事 Texturing 的實驗。

以下是 AnimaTek 的各程式的流程。

animator.c ⇄ animator.c

↓

obj_sort.c

↓

task_master.c

↓

render.c

↓

tiler.c

↓

error.c　pixels.c　driver.c

有關於 AnimaTek 的討論可參見 SIGGRAPH 85' 9-18, Cloud FC

(1982) "A more flexible image generation environment".

25×20=500

No.

因為學生是自 pixel.c 的程做了修改，其餘的程式大致維持原貌。對於其它的程式於此概述之：

。animator.c：處理最主要的 command language，對各个 object 做預先的準備。

。Obj-sort.c：決定 object 的先後，及各 object 是否相交等

。Task_master.c：將 sort 过的 object 做更進一步的準備，稍後做 rendering 的工作。

。render.c：是原 Animator 的核心之一，因為原 Animator，採 Groud Shading，故其計算 intensity 是於此計算，當然還有 Normal 的計算。學生於此中，也改了許多。加入了一些實驗的參数。而原本的 Animator 參数却不受影响。

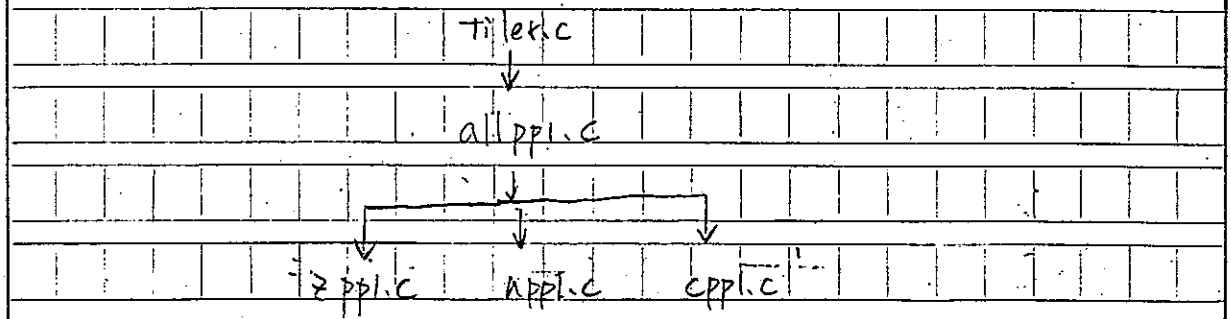
。Tiler.c：是 Scan-line Algorithm，可參見 CACM 20(11)：799-805 中的 Crow FC (1977) "The aliasing problem in computer-generated shaded image"。其中的流程图对於此程式有所了解。

25×20=500

No.

• pixels.c：原本的 pixels.c 只是用来計算 lightlight 及透明物体的处理。但修改的 pixels.c 已不只如此。

接下来，我会针对修改後的 pixels.c 的程式加以說明。首先是大流程图：



tiler.c 之後，为了 Texturing 的實驗，又保留原 Animator 的功能，於此之間加入了一个 pipe 程式，有了个選擇。其中 zppl.c 是原 pixels.c 程式，但加入了 z-buffer 的功能，故能处理物体之相穿透的效果。而 nppl.c 是做 Normal perturbation 的主程式，而 cppl.c 是做 color perturbation 的主程式。以下，分别对 Normal perturbation 及 color perturbation 的部分加以說明。但得先提及的，原本的 intensity 計算，遮了 Normal perturbation 而改在各个 pixel 計算，故非 ground shading 了。也就是 render.c 中的 intensity 計算，对 Normal perturbation 無效。

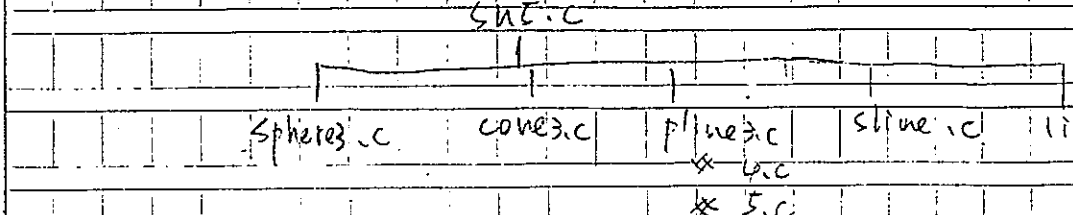
```
/* PIXELS.C */

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   pixels.c - writes pixels for front-to-back algorithms using coverage
              bits.  Uses four corner points of pixel to properly average
              over texture and small highlights.
        Entries:
        - pixels(X_pos,Z_pos,coverage,
                 tp_lft_pxl,bt_lft_pxl,tp_rgt_pxl,bt_rgt_pxl)

        Z_pos is used only for haze calculations in texture mapped objects
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
extern short mys_flag,norp_flag,clop_flag;

/* ++++++++++++++++++++++++++++ PIXELS ++++++++++++++++++++++++++++++++++++*/
                                        /* blend pixels using coverage b
pixels(X_pos,Z_pos,covrge,tp_lft_pxl,bt_lft_pxl,tp_rgt_pxl,bt_rgt_pxl)
    short X_pos; double Z_pos;
    double covrge, tp_lft_pxl[],bt_lft_pxl[], tp_rgt_pxl[],bt_rgt_pxl[];
{
   if (!mys_flag)
    zppixels(X_pos,Z_pos,covrge,tp_lft_pxl,bt_lft_pxl,tp_rgt_pxl,bt_rgt_px
   else if (norp_flag && mys_flag)
    nppixels(X_pos,Z_pos,covrge,tp_lft_pxl,bt_lft_pxl,tp_rgt_pxl,bt_rgt_px
   else if (clop_flag)
    cppixels(X_pos,Z_pos,covrge,tp_lft_pxl,bt_lft_pxl,tp_rgt_pxl,bt_rgt_px
}
```

oo Normal perturbation

Normal perturbation 分了兩大類，一是為了冰淇淋實驗設計的程式；另一為其它相關設計的程式。事實上這是可以合併的，但學生為了趕論文及投ICIP合併後修改不及而做罷。合併的工作不會很麻煩的。

為了維持原 Animator，不論是 Normal perturbation 或 Color perturbation 的程式，在執行前均 save all parameters 待 ~~xxxx~~ perturbation 之後才 restore all parameters。至於 Normal perturbation，因為不採原 Animator 的 intensity 計算，學生自己寫了一individual intensity 計算程式，其副程式之名為 "my_shades"。

對於冰淇淋的程式為：



流質紋理的計算請參見論文及 sphere3.c, cone3.c 的程式。而餅乾類紋理請見 line1.c 及 sline.c。而圓形冰紋的程式是 pline3.c, pline4.c, pline5.c

No.

這裡存主兩組 Animator 接受的 xxx.scm 及 xxx.obj。
這些可在 ~~~student/wsh/icmaker~~ student/wsh/icmaker 的 directory 下可找到。
及程式。

```
place lnd2 at 0 0 1.0

bounding_box    -5 5 -5 5 -5 5
#color          .7 .5 0.4
color           1. .5 0.5
my_shading      1
nor_perting     1
shininess       0
transmittance   0 0
```

metallic1

~/icmaker/

```
title           digitized face
detail          cone.det
type            polygon
#poly_colors     cone.pcl
texture         cone38.txc
bounding_box    -5 5 -5 5 -5 5
#color          .7 .5 0.4
color           1. .5 0.5  ⇒ 0.8, 0.3, 0.3
my_shading      1
nor_perting     1
shininess       0
transmittance   0 0
```

cone.obj | conegc
cone.dir

```
title           polygon format sphere
#detail         s0610.det
detail          s0616.det
type            polygon
#poly_colors     cone.pcl
#texture         s910_910.txc
#texture         s1516_11.txc
texture         s1516_1516.txc
bounding_box    -6 6 -6 6 -6 6
color            1  0  0
my_shading       1
nor_perting      1
shininess        0
transmittance    0. 0.0
```

sphere.obj | spheregc
spdir

```
title           digitized face
detail          face.det
type            polygon  faceted
bounding_box    -1 1 -1 1 -1 1
texture         face11.txc
color           1. 1. 1.
nor_perting     1
my_shading      1
shininess       0
transmittance   0 0
```

paper.obj | plane.c

```
call sphere.obj by sphere
call cone.obj by cone
#call paper.obj by paper
#
paint background with 0.3 0.4 0.75
place center_of_interest at 0 0 0
place eyepoint at 0 75 0
place light at 2000 2000 1600
paint light with 1 1 1
scale light by 4.9104e+11
#
place sphere at 0.0 0 6.2
#place sphere at -1.2 0 5.8
scale sphere by 1.15 1.80 1.50
rotate sphere about 0 0 0 0 0 1 by 240 then
               about 0 0 0 1 0 0 by -30
#
place cone at 0 0 0
scale cone by 1.4 1.4 1.4
#scale cone by 1.2 1.4 1.4
rotate cone about 0 0 0 0 1 0 by 15
#rotate cone about 0 0 0 0 1 0 by 20
#
#paint paper with 1 1 0
#place paper at 0 -9 0
##place paper at 0 -2 -10
#scale paper by 29 26 26
#rotate paper about 0 0 0 0 1 0 by -30
##               about 0 0 0 1 0 0 by -80
##             about 0 0 0 1 0 0 by 30
#
render_on /usr/local/SUN/ic44221.ras

9
0.95  0.65  0.13
0.90  0.77  0.13
0.85  0.65  0.13
0.85  0.55  0.13
0.73  0.65  0.13
0.95  0.88  0.13
0.96  0.55  0.13
0.97  0.37  0.13
0.97  0.27  0.13
```

```
rm: golfcat nonexistent
rm: override protection 644 for paper.obj?

title           digitized face
detail          face.det
type            polygon  faceted
bounding_box    -1 1 -1 1 -1 1
texture         face11.txc
color           1. 1. 1.
nor_perting     1
my_shading      1
shininess       0
transmittance   0 0

title           polygon format sphere
detail          ../cloud/e44610.det
type            polygon
texture         s1516_1516.txc
bounding_box    -6 6 -6 6 -6 6
color           1  0  0
my_shading      1
nor_perting     1
shininess       0
transmittance   0. 0.0

call ball.obj by ball1
call ball.obj by ball2
call paper.obj by water1
call paper.obj by water2
#
paint background with 0.3 0.4 0.75
place center_of_interest at 0 0 0
place eyepoint at 0 75 0
#place light at 2000 2000 1600
place light at 0 2000 0
paint light with 1 1 1
scale light by 4.9104e+11
#
paint ball1 with 1 0 0
scale ball1 by 1.0 1.0 1.0
rotate ball1 about 0 0 0 0 1 0 by 90 then
              about 0 0 0 0 0 1 by 60
place ball1 at -15.5 2 0.
#
paint ball2 with 1 1 0
scale ball2 by 0.5 0.5 0.5
rotate ball2 about 0 0 0 0 1 0 by 90
place ball2 at 0.0 0 0.
#
paint water1 with 0 1 1
scale water1 by 19 16 16
rotate water1 about 0 0 0 0 0 1 by -60
place water1 at 15.0 0 0.
#
paint water2 with 0 1 1
scale water2 by 19 16 16
rotate water2 about 0 0 0 0 0 1 by 60
place water2 at -15.0 0 0.
#
render_on /usr/local/SUN/water4251.ras
```

```
CFLAGS = -O -w

HDRS = animator.h

OBJ1 = animator.o task_master.o obj_sort.o
OBJ2 = render.o ztiler.o allpp1.o zpp1.o npp1.o cpp1.o
OBJ3 = zdriver.o error.o

OBJ4 = ../icmaker/sn5.o ../icmaker/cone3.o ../icmaker/sphere3.o ../icmaker/linel
OBJ5 =  ../icmaker/pline4.o ../icmaker/calxyz.o ../icmaker/fieldf.o ../icmaker/p
OBJ6 =  ../cloud/cloud.o ../cloud/nscene.o
OBJS = $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4) $(OBJ5) $(OBJ6)

animator:  $(HDRS) $(OBJS)
        cc $(OBJ1) $(OBJ2) $(OBJ4) $(OBJ5) $(OBJ6) -lm $(OBJ3) -o ani
```

```
CFLAGS = -O -w

HDRS = animator.h

OBJ1 = animator.o task_master.o obj_sort.o
OBJ2 = render.o ztiler.o allppl.o zppl.o nppl.o cppl.o
OBJ3 = zdriver.o error.o

OBJ4 = ../icmaker/sn5.o ../icmaker/cone3.o ../icmaker/sphere3.o ../icmaker/linel.

OBJ5 =  ../icmaker/pline4.o ../icmaker/calxyz.o ../icmaker/fieldf.o ../icmaker/pl
OBJ6 =  ../cloud/cloud.o ../cloud/nscene.o
OBJS = $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4) $(OBJ5) $(OBJ6)

animator:  $(HDRS) $(OBJS)
        cc $(OBJ1) $(OBJ2) $(OBJ4) $(OBJ5) $(OBJ6) -lm $(OBJ3) -o ani
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
short  pert_flag;

purter_nor(pnx,tx)
double pnx[],tx[];
{
  if ( strcmp("cone",my_obj_name)==0 )
    { save_nor(pnx,tx); line(pnx,tx); cone_purt(pnx,tx);
      my_shades(tx);}

  if ( strcmp("sphere",my_obj_name)==0 )
    { save_nor(pnx,tx); sline(pnx,tx); sphere_purt(pnx,tx); my_shades(tx);}

  if ( strcmp("water1",my_obj_name)==0 )
    { save_nor(pnx,tx); p_line4(pnx,tx); my_shades(tx);}

  if ( strcmp("water2",my_obj_name)==0 )
    { save_nor(pnx,tx); p_line5(pnx,tx); my_shades(tx);}

  if ( strncmp("ball",my_obj_name,4)==0 )
```

```
    { save_nor(pnx,tx); my_shades(tx);}
}

save_nor(pnx,tx)
double pnx[],tx[];
{
short i;

/*printf("save_nor.................\n");
*/
       for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          pnx[i]=tx[i];
/*printf("end save_nor.............\n");
*/}

reset_nor(pnx,tx)
double pnx[],tx[];
{
short i;

/*printf("reset_nor.................\n");
*/
       for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          tx[i]=pnx[i];
/*printf("end reset_nor.............\n");
*/}

my_shades(tx)
double tx[];
{
short  i;
double sqrt(),pow(),fabs();
double norm_mag,dot_prod,spe_prod,red,grn,blu;
double lx,ly,lz,lit_dst,rflx,rfly,rflz,rfl_dst,eye_mag,atten;
struct { double xn,yn,zn,r,g,b;} hi_normal[MAXLTS];

/*printf("my_shades.............\n");
*/
       norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
       if (norm_mag<=0.)
        { printf("less zero\n");
          norm_mag=1.;  }

       red=grn=blu=0.;
    for(i=0;i<nlights;i++) {
       lx=lights[i].x-tx[TX];
       ly=lights[i].y-tx[TY];
       lz=lights[i].z-tx[TZ];

       lit_dst=sqrt(sqr(lx)+sqr(ly)+sqr(lz));

       atten=1.0-lit_dst/lights[i].range;
       atten=(atten>0.)? sqr(atten): 0.;

       if (norm_mag*lit_dst == 0.) dot_prod=0.0;
       else
```

```
          dot_prod=(tx[XN]*lx+tx[YN]*ly+tx[ZN]*lz)/(norm_mag*lit_dst);
       if (dot_prod<0.) dot_prod=0.0;

       eye_mag=sqrt(sqr(tx[TX])+sqr(tx[TY])+sqr(tx[TZ]));
       rflx=(lx/lit_dst-tx[TX]/eye_mag)/2.;
       rfly=(ly/lit_dst-tx[TY]/eye_mag)/2.;
       rflz=(lz/lit_dst-tx[TZ]/eye_mag)/2.;
       rfl_dst=sqrt(sqr(rflx)+sqr(rfly)+sqr(rflz));
/*
       rflx=rflx/rfl_dst;
       rfly=rfly/rfl_dst;
       rflz=rflz/rfl_dst;
*/
/*my_hilit=150;*/

       if (norm_mag*rfl_dst == 0.) spe_prod=0.;
       else
          spe_prod=(tx[XN]*rflx+tx[YN]*rfly+tx[ZN]*rflz)/(norm_mag*rfl_dst);
       if (spe_prod<0.) spe_prod=0.;
       else spe_prod=pow(spe_prod,(double)my_hilit);

/*       spe_prod=atten*(1-ambnt)*spe_prod*(1-ambntcomp);*/
       spe_prod=spe_prod;
       dot_prod=dot_prod*ambntcomp+ambnt;

       if (my_hilit ==0) spe_prod=0.;

       red=red+tx[TR]*dot_prod*lights[i].r*atten
             +spe_prod*(lights[i].r-tx[TR]*255.);
       grn=grn+tx[TG]*dot_prod*lights[i].g*atten
             +spe_prod*(lights[i].g-tx[TG]*255.);
       blu=blu+tx[TB]*dot_prod*lights[i].b*atten
             +spe_prod*(lights[i].b-tx[TB]*255.);

    }/*for nlights*/

 /*      red=red*tx[TR];
       grn=grn*tx[TG];
       blu=blu*tx[TB];
*/
    { double fac;
       fac=255.;
       if (red > fac) fac=red;
       if (grn > fac) fac=grn;
       if (blu > fac) fac=blu;
       fac = 255./fac;
       tx[R]=red; tx[G]=grn; tx[B]=blu;
    }
/*printf("end my_shades.....................\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
extern short    pert_flag;

int n_cir,sp_read_flag=0;
double cru[13],crw[13],Rr[13];

sphere_purt(pnx,tx)
double pnx[],tx[];
{
FILE  *ip_f;
char  string[512];
int i;
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double all_fuv,fu;
double fuv[13];
double cx[13],cy[13],cz[13];
double xx,yy,zz;
double snx1,sny1,snz1,sx1,sy1,sz1;
double r1,r2,g1,g2,b1,b2;
```

```
double w,u;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*       for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          pnx[i]=tx[i];
*/
     if (sp_read_flag==0) {
       ip_f=fopen("sp.cir","r");
       if (ip_f==NULL) printf("error sp\n");
       fgets(string,512,ip_f);
       sscanf(string,"%d",&n_cir);
printf("n_cir=%d\n",n_cir);
     }
       pert_flag=0; all_fuv=0.;

      for (i=0;i<n_cir;i++) {
       if (sp_read_flag==0) {
         fgets(string,512,ip_f);
         sscanf(string,"%F %F %F",&cru[i],&crw[i],&Rr[i]);
       }
       fuv[i]=0.;
       my_field_f(cru[i],crw[i],Rr[i],&fuv[i],tx);
       all_fuv=all_fuv+fuv[i];
      }
      if (sp_read_flag==0) fclose(ip_f);
      sp_read_flag=1;

/*       my_field_f(cru1,crw1,Rr,&fuv1,tx);*/

   if (all_fuv>=0.15) {

       if (all_fuv>1.) all_fuv=1.;

       pert_flag=1;
       snx1=sny1=snz1=0.;

/*       tx[TR]=1.; tx[TG]=0.; tx[TB]=1.;*/

       for (i=0;i<n_cir;i++) {

          cx[i]=cy[i]=cz[i]=0.;
          my_cal_xyz(cru[i],crw[i],&xx,&yy,&zz,tx);
          cx[i]=tx[TX]-xx; cy[i]=tx[TY]-yy; cz[i]=tx[TZ]-zz;
       /**/
       /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
          xx=tx[TX]-xx; yy=tx[TY]-yy; zz=tx[TZ]-zz;*/
       /**/
       }

       xx=yy=zz=0.;
       for ( i=0;i<n_cir;i++) {
          xx=xx+fuv[i]*cx[i];
          yy=yy+fuv[i]*cy[i];
          zz=zz+fuv[i]*cz[i];
```

```
        }

/*        fuv=.25/fuv; fuv=1./exp(fuv*1.5);
        fuv=fuv*fuv*fuv*fuv;
*/      all_fuv=(1.-all_fuv)/(1.-0.15);
        all_fuv=2.*all_fuv*all_fuv*all_fuv-3.*all_fuv*all_fuv+1.;
        norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
/*        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));*/

        fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
        if (fxyz_mag==0.) fxyz_mag=1.;

        snx1=all_fuv*(tx[XN]/norm_mag)+xx/fxyz_mag;/*per NOR*/
        sny1=all_fuv*(tx[YN]/norm_mag)+yy/fxyz_mag;
        snz1=all_fuv*(tx[ZN]/norm_mag)+zz/fxyz_mag;
/*
        snx1=all_fuv*(pnx[XN]/norm_mag)+xx/fxyz_mag;
        sny1=all_fuv*(pnx[YN]/norm_mag)+yy/fxyz_mag;
        snz1=all_fuv*(pnx[ZN]/norm_mag)+zz/fxyz_mag;
*/        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<0.2*/

/*   my_shades(tx);
printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)    ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
extern short    pert_flag;

int n_cir,cone_read_flag=0;
double cru[20],crw[20],Rr[20];

cone_purt(pnx,tx)
double pnx[],tx[];
{
FILE  *ip_f;
char  string[512];
int i;
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double all_fuv,fu;
double fuv[20];
double cx[20],cy[20],cz[20];
double xx,yy,zz;
double snx1,sny1,snz1,sx1,sy1,sz1;
double r1,r2,g1,g2,b1,b2;
```

```
double w,u;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*       for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          pnx[i]=tx[i];
*/
     if (cone_read_flag==0) {
       ip_f=fopen("cone.cir","r");
       if (ip_f==NULL) printf("error==\n");
       fgets(string,512,ip_f);
       sscanf(string,"%d",&n_cir);
printf("n_cir=%d\n",n_cir);
     }
       pert_flag=0; all_fuv=0.;

      for (i=0;i<n_cir;i++) {
       if (cone_read_flag==0) {
         fgets(string,512,ip_f);
         sscanf(string,"%F %F %F",&cru[i],&crw[i],&Rr[i]);
/*printf("cru1=%f\n",cru[i]);*/
       }
       fuv[i]=0.;
       my_field_f(cru[i],crw[i],Rr[i],&fuv[i],tx);
       all_fuv=all_fuv+fuv[i];
      }

      if (cone_read_flag==0) fclose(ip_f);
      cone_read_flag=1;

/*       my_field_f(cru1,crw1,Rr,&fuv1,tx);*/

   if (all_fuv>=0.25) {

       if (all_fuv>1.) all_fuv=1.;

       pert_flag=1;
       snx1=sny1=snz1=0.;

       tx[TR]=0.8; tx[TG]=0.2; tx[TB]=0.0;/**/
/*       tx[TR]=0.3; tx[TG]=0.7; tx[TB]=0.3;/**/
/*       tx[TR]=1.; tx[TG]=0.; tx[TB]=0.;/**/

       for (i=0;i<n_cir;i++) {
          cx[i]=cy[i]=cz[i]=0.;
          my_cal_xyz(cru[i],crw[i],&xx,&yy,&zz,tx);
          cx[i]=tx[TX]-xx; cy[i]=tx[TY]-yy; cz[i]=tx[TZ]-zz;
       /**/
       /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
          xx=tx[TX]-xx; yy=tx[TY]-yy; zz=tx[TZ]-zz;*/
       /**/
       }

       xx=yy=zz=0.;
       for (i=0;i<n_cir;i++) {
```

```
            xx=xx+fuv[i]*cx[i];
            yy=yy+fuv[i]*cy[i];
            zz=zz+fuv[i]*cz[i];
          }

/*          fuv=.25/fuv; fuv=1./exp(fuv*1.5);
          fuv=fuv*fuv*fuv*fuv;
*/        all_fuv=(1.-all_fuv)/(1.-0.25);
          all_fuv=2.*all_fuv*all_fuv*all_fuv-3.*all_fuv*all_fuv+1.;

          norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
/*          norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));*/

          fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
          if (fxyz_mag==0.) fxyz_mag=1.;

          snx1=all_fuv*(tx[XN]/norm_mag)+xx/fxyz_mag;/*per NOR*/
          sny1=all_fuv*(tx[YN]/norm_mag)+yy/fxyz_mag;
          snz1=all_fuv*(tx[ZN]/norm_mag)+zz/fxyz_mag;
/*
          snx1=all_fuv*(pnx[XN]/norm_mag)+xx/fxyz_mag;
          sny1=all_fuv*(pnx[YN]/norm_mag)+yy/fxyz_mag;
          snz1=all_fuv*(pnx[ZN]/norm_mag)+zz/fxyz_mag;
*/          tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

      }/*if fu<0.2*/

/*    my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

line(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx1,cy1,cz1,xx,yy,zz;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru1,crw1,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
       Rr=.030;
```

```
       fu=fuv=0.;
  for (i=0;i<12;i++) {
     if (i==0){ cru1=.9; crw1=tx[TX_Y];}
     if (i==1){ cru1=.8; crw1=tx[TX_Y];}
     if (i==2){ cru1=.7; crw1=tx[TX_Y];}
     if (i==3){ cru1=.6; crw1=tx[TX_Y];}
     if (i==4){ cru1=.5; crw1=tx[TX_Y];}
     if (i==5){ cru1=.4; crw1=tx[TX_Y];}
     if (i==6){ cru1=tx[TX_X]; crw1=.95;}
     if (i==7){ cru1=tx[TX_X]; crw1=.75;}
     if (i==8){ cru1=tx[TX_X]; crw1=.55;}
     if (i==9){ cru1=tx[TX_X]; crw1=.35;}
     if (i==10){ cru1=tx[TX_X]; crw1=.15;}
     if (i==11){ cru1=.3; crw1=tx[TX_Y];}

     fu=sqrt(sqr(tx[TX_X]-cru1)+sqr(tx[TX_Y]-crw1));

    if (fu<=Rr) {

       snx1=sny1=snz1=0.;
       cx1=cy1=cz1=0.;

/*       tx[TR]=0.; tx[TG]=1.; tx[TB]=0.;*/
/**/
       u=(cru1-tquv[X])/(tquv[2+X]-tquv[X]);
       w=(crw1-tquv[Y])/(tquv[3*2+Y]-tquv[Y]);
       xx=(1.-w)*(1.-u)*quv[X]+(1.-u)*w*quv[3*3+X]
                              +u*(1.-w)*quv[3+X]+u*w*quv[2*3+X];
       yy=(1.-w)*(1.-u)*quv[Y]+(1.-u)*w*quv[3*3+Y]
                              +u*(1.-w)*quv[3+Y]+u*w*quv[2*3+Y];
       zz=(1.-w)*(1.-u)*quv[Z]+(1.-u)*w*quv[3*3+Z]
                              +u*(1.-w)*quv[3+Z]+u*w*quv[2*3+Z];
/*       cx1=xx-tx[TX]; cy1=yy-tx[TY]; cz1=zz-tx[TZ];*/
/*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
*/       cx1=tx[TX]-xx; cy1=tx[TY]-yy; cz1=tx[TZ]-zz;

       xx=cx1;
       yy=cy1;
       zz=cz1;

/*       fuv=fuv*fuv*fuv*fuv;*/
/*       fuv=0.5*fuv/Rr;    fuv=1./exp(fuv*8.5);*/
      fuv=fuv/Rr; fuv=2*fuv*fuv*fuv-3*fuv*fuv+1;

       fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
       if (fxyz_mag==0.) fxyz_mag=1.;
       norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));

       snx1=fuv*(pnx[XN]/norm_mag)+(xx/fxyz_mag);/*per NOR*/
       sny1=fuv*(pnx[YN]/norm_mag)+(yy/fxyz_mag);
       snz1=fuv*(pnx[ZN]/norm_mag)+(zz/fxyz_mag);
       tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<=Rr*/
  } /* for cru crw */
/*my_shades(tx);*/
```

```
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY      10
#define TZ      11
#define TR      12
#define TG      13
#define TB      14

#define NORMS           15
#define NORM_PARMS      6
#define MAXLTS          16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

sline(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx1,cy1,cz1,xx,yy,zz;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru1,crw1,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
        Rr=.20;
```

```
        fu=fuv=0.;
  for (i=0;i<28;i++) {
     if (i==10){ crul=5.5; crwl=tx[TX_Y];}
     if (i==11){ crul=6.5; crwl=tx[TX_Y];}
     if (i==12){ crul=7.5; crwl=tx[TX_Y];}
     if (i==13){ crul=1.5; crwl=tx[TX_Y];}
     if (i==0){ crwl=0.0; crul=tx[TX_X];}
     if (i==1){ crwl=1.0; crul=tx[TX_X];}
     if (i==2){ crwl=2.0; crul=tx[TX_X];}
     if (i==3){ crwl=3.0; crul=tx[TX_X];}
     if (i==4){ crwl=13.0; crul=tx[TX_X];}
     if (i==5){ crwl=14.0; crul=tx[TX_X];}
     if (i==6){ crwl=15.0; crul=tx[TX_X];}
     if (i==7){ crwl=0.5; crul=tx[TX_X];}
     if (i==8){ crwl=1.5; crul=tx[TX_X];}
     if (i==9){ crwl=1.5; crul=tx[TX_X];}
     if (i==10){ crwl=3.5; crul=tx[TX_X];}
     if (i==11){ crwl=13.5; crul=tx[TX_X];}
     if (i==12){ crwl=14.5; crul=tx[TX_X];}
     if (i==13){ crwl=15.5; crul=tx[TX_X];}
     if (i==14){ crul=2.0; crwl=tx[TX_Y];}
     if (i==15){ crul=3.0; crwl=tx[TX_Y];}
     if (i==16){ crul=4.0; crwl=tx[TX_Y];}
     if (i==17){ crul=5.0; crwl=tx[TX_Y];}
     if (i==18){ crul=6.0; crwl=tx[TX_Y];}
     if (i==19){ crul=7.0; crwl=tx[TX_Y];}
     if (i==20){ crul=1.0; crwl=tx[TX_Y];}
     if (i==21){ crul=2.5; crwl=tx[TX_Y];}
     if (i==22){ crul=3.5; crwl=tx[TX_Y];}
     if (i==23){ crul=4.5; crwl=tx[TX_Y];}
     if (i==24){ crul=5.5; crwl=tx[TX_Y];}
     if (i==25){ crul=6.5; crwl=tx[TX_Y];}
     if (i==26){ crul=7.5; crwl=tx[TX_Y];}
     if (i==27){ crul=1.5; crwl=tx[TX_Y];}

     fu=sqrt(sqr(tx[TX_X]-crul)+sqr(tx[TX_Y]-crwl));

   if (fu<=Rr) {

        snxl=snyl=snzl=0.;
        cxl=cyl=czl=0.;

/*        tx[TR]=0.; tx[TG]=1.; tx[TB]=0.;*/
/**/
        u=(crul-tquv[X])/(tquv[2+X]-tquv[X]);
        w=(crwl-tquv[Y])/(tquv[3*2+Y]-tquv[Y]);
        xx=(1.-w)*(1.-u)*quv[X]+(1.-u)*w*quv[3*3+X]
                              +u*(1.-w)*quv[3+X]+u*w*quv[2*3+X];
        yy=(1.-w)*(1.-u)*quv[Y]+(1.-u)*w*quv[3*3+Y]
                              +u*(1.-w)*quv[3+Y]+u*w*quv[2*3+Y];
        zz=(1.-w)*(1.-u)*quv[Z]+(1.-u)*w*quv[3*3+Z]
                              +u*(1.-w)*quv[3+Z]+u*w*quv[2*3+Z];
        cxl=xx-tx[TX]; cyl=yy-tx[TY]; czl=zz-tx[TZ];/**/
/*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
        cxl=tx[TX]-xx; cyl=tx[TY]-yy; czl=tx[TZ]-zz;
*/
```

```
        xx=cx1;
        yy=cy1;
        zz=cz1;

/*        fuv=fuv*fuv*fuv*fuv;*/
/*        fuv=0.5*fuv/Rr;   fuv=1./exp(fuv*8.5);*/
       fuv=fuv/Rr; fuv=2*fuv*fuv*fuv-3*fuv*fuv+1;

        fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
        if (fxyz_mag==0.) fxyz_mag=1.;
        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));

        snx1=fuv*(pnx[XN]/norm_mag)+(xx/fxyz_mag);/*per NOR*/
        sny1=fuv*(pnx[YN]/norm_mag)+(yy/fxyz_mag);
        snz1=fuv*(pnx[ZN]/norm_mag)+(zz/fxyz_mag);
        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

     }/*if fu<=Rr*/
   } /* for cru crw */
 /*my_shades(tx);*/
 /*printf("end pertur.............\n");
 */}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

p_line4(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos(),fabs();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx1,cy1,cz1,xx,yy,zz;
double R1,sign_flag;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru1,crw1,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
```

```
        Rr=.055; cru1=.5; crw1=.5;
        fu=fuv=0.;
   for (i=0;i<4;i++) {

      if (i==0){ R1=.15;}
      if (i==1){ R1=.05;}
      if (i==2){ R1=.25;}
      if (i==3){ R1=.35;}

      fu=sqrt(sqr(tx[TX_X]-cru1)+sqr(tx[TX_Y]-crw1));
      fuv=fabs(fu-R1);

    if (fuv<=Rr) {

        if ((fu-R1)>=0.) sign_flag=1.; else sign_flag=0.-1.;

        snx1=sny1=snz1=0.;
        cx1=cy1=cz1=0.;

/*        tx[TR]=0.; tx[TG]=1.; tx[TB]=0.;*/

        my_cal_xyz(cru1,crw1,&xx,&yy,&zz,tx);
/*        cx1=xx-tx[TX]; cy1=yy-tx[TY]; cz1=zz-tx[TZ];*/
/*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
*/        cx1=tx[TX]-xx; cy1=tx[TY]-yy; cz1=tx[TZ]-zz;

        xx=sign_flag*cx1;
        yy=sign_flag*cy1;
        zz=sign_flag*cz1;

/*        fuv=fuv*fuv*fuv*fuv;*/
/*        fuv=0.5*fuv/Rr;    fuv=1./exp(fuv*8.5);*/
       fuv=fuv/Rr; fuv=2*fuv*fuv*fuv-3*fuv*fuv+1;

        fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
        if (fxyz_mag==0.) fxyz_mag=1.;
        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));

        snx1=fuv*(pnx[XN]/norm_mag)+(xx/fxyz_mag);/*per NOR*/
        sny1=fuv*(pnx[YN]/norm_mag)+(yy/fxyz_mag);
        snz1=fuv*(pnx[ZN]/norm_mag)+(zz/fxyz_mag);
        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

     }/*if fu<=Rr*/
   } /* for cru crw */

/*    my_shades(tx);*/

/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)    ((x)*(x))

/* global variables for pixel output routine */

extern double   quv[12],tquv[8];

my_cal_xyz(cru,crw,xx,yy,zz,tx)
double cru,crw,*xx,*yy,*zz,tx[];
{
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double u,w;

/*my_cal_xyz(crul,crwl,&xx,&yy,&zz,tx);*/
       u=(cru-tquv[X])/(tquv[2+X]-tquv[X]);
       w=(crw-tquv[Y])/(tquv[3*2+Y]-tquv[Y]);
       *xx=(1.-w)*(1.-u)*quv[X]+(1.-u)*w*quv[3*3+X]
                               +u*(1.-w)*quv[3+X]+u*w*quv[2*3+X];
       *yy=(1.-w)*(1.-u)*quv[Y]+(1.-u)*w*quv[3*3+Y]
                               +u*(1.-w)*quv[3+Y]+u*w*quv[2*3+Y];
       *zz=(1.-w)*(1.-u)*quv[Z]+(1.-u)*w*quv[3*3+Z]
                               +u*(1.-w)*quv[3+Z]+u*w*quv[2*3+Z];
}
```

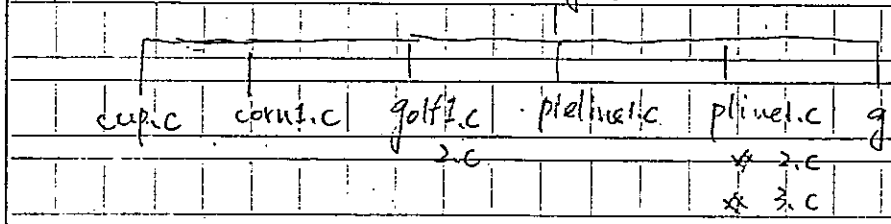
```
#include <stdio.h>
#include <math.h>

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS             15
#define NORM_PARMS         6
#define MAXLTS            16

#define sqr(x)    ((x)*(x))

my_field_f(cru,crw,Rr,fuv,tx)
double cru,crw,Rr,*fuv,tx[];
{
double fu;

/*       my_field_f(cru1,crw1,Rr,&fuv1,tx);*/
       fu=sqrt(sqr(tx[TX_X]-cru)+sqr(tx[TX_Y]-crw));
       if (fu<=Rr) {
         fu=fu/Rr;
         *fuv=2.*fu*fu*fu-3.*fu*fu+1.;
       }
}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

p_line5(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos(),fabs();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx1,cy1,cz1,xx,yy,zz;
double R1,sign_flag;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru1,crw1,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
```

```
        Rr=.055; cru1=.5; crw1=.5;
        fu=fuv=0.;
  for (i=0;i<2;i++) {

      if (i==0){ R1=.2;}
      if (i==1){ R1=.1;}

      fu=sqrt(sqr(tx[TX_X]-cru1)+sqr(tx[TX_Y]-crw1));
      fuv=fabs(fu-R1);

    if (fuv<=Rr) {

        if ((fu-R1)>=0.) sign_flag=1.; else sign_flag=0.-1.;

        snx1=sny1=snz1=0.;
        cx1=cy1=cz1=0.;

/*        tx[TR]=0.; tx[TG]=1.; tx[TB]=0.;*/

        my_cal_xyz(cru1,crw1,&xx,&yy,&zz,tx);
/*        cx1=xx-tx[TX]; cy1=yy-tx[TY]; cz1=zz-tx[TZ];*/
/*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
*/        cx1=tx[TX]-xx; cy1=tx[TY]-yy; cz1=tx[TZ]-zz;

        xx=sign_flag*cx1;
        yy=sign_flag*cy1;
        zz=sign_flag*cz1;

/*        fuv=fuv*fuv*fuv*fuv;*/
/*        fuv=0.5*fuv/Rr;   fuv=1./exp(fuv*8.5);*/
       fuv=fuv/Rr; fuv=2*fuv*fuv*fuv-3*fuv*fuv+1;

        fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
        if (fxyz_mag==0.) fxyz_mag=1.;
        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));

        snx1=fuv*(pnx[XN]/norm_mag)+(xx/fxyz_mag);/*per NOR*/
        sny1=fuv*(pnx[YN]/norm_mag)+(yy/fxyz_mag);
        snz1=fuv*(pnx[ZN]/norm_mag)+(zz/fxyz_mag);
        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<=Rr*/
  } /* for cru crw */

/*    my_shades(tx);*/

/*printf("end pertur.............\n");
*/}
```

No.

至於更多和圖 normal perturbation 的格式以下表：



同樣地，必須上一組 xxx.obj 及 xxx.scn。這些 obj 及 scn 檔於 student/wgh/golf 下可找到。

```
title           polygon format sphere
detail          cup.det
type            polygon open
texture         cup3132.txc
#texture        cup116116.txc
bounding_box    -16 16 -16 16 -16 16
#color           .5 .5 1
color            0. 1.0 1.
my_shading       1
nor_perting      1
shininess        0
transmittance    0. 0.0
```

cup.obj. 在"golf"下 同 cup.obj

```
0
transmittance    0.85 1.0

title           polygon format sphere
detail          mike.det
type            polygon
#texture         cig36431.txc
bounding_box    -16 16 -16 16 -16 16
color            1.  1.  1.
my_shading       0
nor_perting      0
shininess        0
transmittance    0.85 1.0
```

mike.obj

```
title           digitized face
detail          box1616.det
type            polygon  faceted
bounding_box    -4 4 -4 4 -4 4
texture         box1616.txc
#color          0.7  0.2 0.2
color           0.9  0.4 0.4
nor_perting     1
my_shading      1
shininess       0
transmittance   0 0
```

pie.obj 在"golf"下 用 glass2.0

```
title           digitized face
detail          face.det
type            polygon  faceted
bounding_box    -1 1 -1 1 -1 1
texture         face11.txc
color           1. 1. 1.
nor_perting     1
my_shading      1
shininess       0
transmittance   0 0
```

paper.obj 用 (cig.中)

```
title           polygon format sphere
detail          sock8.det
type            polygon
texture          glass1516.txc
bounding_box    -6 6 -6 6 -6 6
color            .5 .5 1
my_shading       1
nor_perting      1
shininess        0
transmittance    0. 0.0
```

golf.obj (用)

```
call cup.obj by cup
call mike.obj by mike
call cupleg.obj by legcup
call pie.obj by piel
call pie.obj by pie2
call pie.obj by pie3
call paper.obj by paper
call golf.obj by golfl
#
#paint background with 1.0 1.0 0.75 0.0
paint background with 0.2 0.3 0.65 0.0
place center_of_interest at 0 0 0
place eyepoint at 0 75 1.-
place light at 2000 2000 1600
paint light with 1 1 .41
scale light by 4.9104e+11
#
scale cup by  2.6 2.6 3.0
rotate cup about 0 0 0 0 0 1 by -30 then
           about 0 0 0 1 0 0 by 10
place cup at  0 0 -5.0
#
scale legcup by  6.0 6.0 6.0
rotate legcup about 0 0 0 0 1 0 by 10 then
              about 0 0 0 1 0 0 by -70
place legcup at  0 0 -10
#
scale mike by  0.3 0.3 2.0
rotate mike about 0 0 0 0 1 0 by 30
place mike at  -2 0 5
#
scale piel by  2.5 2.5 2.5
rotate piel about 0 0 0 1 0 0 by 270 then
             about 0 0 0 0 1 0 by 30  then
             about 0 0 0 1 0 0 by -80
place piel at  -12 8 -7.0
#
scale pie2 by  2.8 1.8 1.8
rotate pie2 about 0 0 0 1 0 0 by 270 then
             about 0 0 0 0 1 0 by 30  then
             about 0 0 0 1 0 0 by -70
place pie2 at  -15 16 -9.0
#
scale pie3 by  2.8 1.5 1.7
rotate pie3 about 0 0 0 1 0 0 by 270 then
             about 0 0 0 0 1 0 by 30  then
             about 0 0 0 1 0 0 by -70
place pie3 at  -10 16 -9.0
#
paint paper with 0 .5 0
place paper at 0 -2 -10
scale paper by 29 26 26
rotate paper about 0 0 0 0 1 0 by -30 then
              about 0 0 0 1 0 0 by -80
##            about 0 0 0 1 0 0 by 30
#
paint golfl with 0.5 1.0 .5
scale golfl by .6 .6 .6
#rotate golfl about 0 0 0 0 1 0 by 90
#             about 0 0 0 1 0 0 by 45
place golfl at 8 19 -9

render_on /usr/local/SUN/cup4241.ras
#render_on smalk4191.ras
```

```
CFLAGS = -O -w

HDRS = animator.h

OBJ1 = animator.o task_master.o obj_sort.o
OBJ2 = render.o ztiler.o allppl.o zppl.o nppl.o cppl.o
OBJ3 = zdriver.o error.o

OBJ4 =  ../golf/maingolf1.o  ../golf/corn1.o ../golf/golf1.o ../golf/golf2.o ../go

OBJ5 =  ../cloud/cloud.o ../cloud/nscene.o
OBJ6 = ../golf/cup.o ../golf/pieline1.o ../icmaker/sphere3.o ../golf/gline2.o

OBJS = $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4) $(OBJ5) $(OBJ6)

animator:  $(HDRS) $(OBJS)
        cc $(OBJ1) $(OBJ2) $(OBJ4) $(OBJ5) $(OBJ6) -lm $(OBJ3) -o ani
```

```
 1  #include <stdio.h>
 2  #include <math.h>
 3
 4  #define X       0
 5  #define Y       1
 6  #define Z       2
 7
 8  #define R       0
 9  #define G       1
10  #define B       2
11  #define T       3
12  #define XN      4
13  #define YN      5
14  #define ZN      6
15  #define TX_X    7
16  #define TX_Y    8
17  #define TX      9
18  #define TY     10
19  #define TZ     11
20  #define TR     12
21  #define TG     13
22  #define TB     14
23
24  #define NORMS            15
25  #define NORM_PARMS        6
26  #define MAXLTS           16
27
28  #define sqr(x)   ((x)*(x))
29
30  /* global variables for pixel output routine */
31  extern short    num_lights,hilit_power,txtr,txtr_obj;
32  extern short    nlights,my_hilit;
33  extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
34  extern double   ambnt,ambntcomp;                    /* lighting specificati
35
36  extern double   quv[12],tquv[8];
37  extern char     my_obj_name[255];
38  short  pert_flag;
39
40  purter_nor(pnx,tx)
41  double pnx[],tx[];
42  {
43    if ( strcmp("cup",my_obj_name)==0 )/*cup==cone*/
44      { save_nor(pnx,tx);  /*gline2(pnx,tx);*/ cup_purt(pnx,tx); my_shades(t
45
46    if ( strncmp("pie",my_obj_name,3)==0 )
47      { save_nor(pnx,tx); pielinel(pnx,tx); my_shades(tx);}
48
49    if ( strncmp("chaco",my_obj_name,5)==0 )
50      { save_nor(pnx,tx); sphere_purt(pnx,tx); my_shades(tx);}
51
52    if ( strncmp("golf",my_obj_name,4)==0 )
53      { save_nor(pnx,tx); a_paint(pnx,tx);/* golf1(pnx,tx);*/ my_shades(tx);
54
55     if ( strncmp("corn",my_obj_name,4)==0 )
56      { save_nor(pnx,tx); a_paint(pnx,tx); corn1(pnx,tx); my_shades(tx);}
```

```
 57
 58      if ( strncmp("paper",my_obj_name,5)==0 )
 59       { save_nor(pnx,tx); p_line3(pnx,tx); my_shades(tx);}
 60
 61     if ( strncmp("glass",my_obj_name,5)==0 )
 62       { save_nor(pnx,tx); gline(pnx,tx); /*line2(pnx,tx);*/ my_shades(tx);}
 63
 64     if ( strncmp("cig",my_obj_name,3)==0 )
 65       { save_nor(pnx,tx); p_line1(pnx,tx); p_line2(pnx,tx); my_shades(tx);}
 66  }
 67
 68  save_nor(pnx,tx)
 69  double pnx[],tx[];
 70  {
 71  short i;
 72
 73  /*printf("save_nor.................\n");
 74  */
 75          for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
 76             pnx[i]=tx[i];
 77  /*printf("end save_nor.............\n");
 78  */}
 79
 80  reset_nor(pnx,tx)
 81  double pnx[],tx[];
 82  {
 83  short i;
 84
 85  /*printf("reset_nor.................\n");
 86  */
 87          for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
 88             tx[i]=pnx[i];
 89  /*printf("end reset_nor.............\n");
 90  */}
 91
 92  my_shades(tx)
 93  double tx[];
 94  {
 95  short  i;
 96  double sqrt(),pow(),fabs();
 97  double norm_mag,dot_prod,spe_prod,red,grn,blu;
 98  double lx,ly,lz,lit_dst,rflx,rfly,rflz,rfl_dst,eye_mag,atten;
 99  struct { double xn,yn,zn,r,g,b;} hi_normal[MAXLTS];
100
101  /*printf("my_shades.............\n");
102  */
103          norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
104          if (norm_mag<=0.)
105           { printf("less zero\n");
106             norm_mag=1.;  }
107
108          red=grn=blu=0.;
109       for(i=0;i<nlights;i++) {
110          lx=lights[i].x-tx[TX];
111          ly=lights[i].y-tx[TY];
112          lz=lights[i].z-tx[TZ];
```

```
113
114         lit_dst=sqrt(sqr(lx)+sqr(ly)+sqr(lz));
115
116         atten=1.0-lit_dst/lights[i].range;
117         atten=(atten>0.)? sqr(atten): 0.;
118
119         if (norm_mag*lit_dst == 0.) dot_prod=0.0;
120         else
121            dot_prod=(tx[XN]*lx+tx[YN]*ly+tx[ZN]*lz)/(norm_mag*lit_dst);
122         if (dot_prod<0.) dot_prod=0.0;
123
124         eye_mag=sqrt(sqr(tx[TX])+sqr(tx[TY])+sqr(tx[TZ]));
125         rflx=(lx/lit_dst-tx[TX]/eye_mag)/2.;
126         rfly=(ly/lit_dst-tx[TY]/eye_mag)/2.;
127         rflz=(lz/lit_dst-tx[TZ]/eye_mag)/2.;
128         rfl_dst=sqrt(sqr(rflx)+sqr(rfly)+sqr(rflz));
129  /*
130         rflx=rflx/rfl_dst;
131         rfly=rfly/rfl_dst;
132         rflz=rflz/rfl_dst;
133  */
134  /*my_hilit=150;*/
135
136         if (norm_mag*rfl_dst == 0.) spe_prod=0.;
137         else
138            spe_prod=(tx[XN]*rflx+tx[YN]*rfly+tx[ZN]*rflz)/(norm_mag*rfl_dst)
139         if (spe_prod<0.) spe_prod=0.;
140         else spe_prod=pow(spe_prod,(double)my_hilit);
141
142  /*       spe_prod=atten*(1-ambnt)*spe_prod*(1-ambntcomp);*/
143         spe_prod=spe_prod;
144         dot_prod=dot_prod*ambntcomp+ambnt;
145
146         red=red+tx[TR]*dot_prod*lights[i].r*atten
147               +spe_prod*(lights[i].r-tx[TR]*255.);
148         grn=grn+tx[TG]*dot_prod*lights[i].g*atten
149               +spe_prod*(lights[i].g-tx[TG]*255.);
150         blu=blu+tx[TB]*dot_prod*lights[i].b*atten
151               +spe_prod*(lights[i].b-tx[TB]*255.);
152
153      }/*for nlights*/
154
155   /*      red=red*tx[TR];
156         grn=grn*tx[TG];
157         blu=blu*tx[TB];
158  */
159      { double fac;
160          fac=255.;
161          if (red > fac) fac=red;
162          if (grn > fac) fac=grn;
163          if (blu > fac) fac=blu;
164          fac = 255./fac;
165          tx[R]=red; tx[G]=grn; tx[B]=blu;
166      }
167  /*printf("end my_shades.....................\n");
168  */}
```

169

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)    ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
extern short    pert_flag;


corn1(pnx,tx)
double pnx[],tx[];
{
int i;
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double all_fuv,fu;
double cx,cy,cz;
double cru,crw;
double xx,yy,zz;
double snx1,sny1,snz1,sx1,sy1,sz1;
double w,u;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/
```

```
/*        for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
           pnx[i]=tx[i];
*/
      all_fuv=0.;
      cru=(int)tx[TX_X]+.5; crw=(int)tx[TX_Y]+.5;
      all_fuv=sqrt(sqr(tx[TX_X]-cru)+sqr(tx[TX_Y]-crw));
/*printf("tx=%f,ty=%f\n",tx[TX_X],tx[TX_Y]);
printf("cru=%f,crw=%f,all_fuv=%f\n",cru,crw,all_fuv);
*/
   if (all_fuv<0.47) {

       snx1=sny1=snz1=0.;

/*      tx[TR]=0.; tx[TG]=0.; tx[TB]=1.;
*/
          cx=cy=cz=0.;
          my_cal_xyz(cru,crw,&xx,&yy,&zz,tx);
          cx=xx-tx[TX]; cy=yy-tx[TY]; cz=zz-tx[TZ];/**/
       /**/
       /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
          cx=tx[TX]-xx; cy=tx[TY]-yy; cz=tx[TZ]-zz;/**/
       /**/

          xx=cx;
          yy=cy;
          zz=cz;

/*        fuv=.25/fuv; fuv=1./exp(fuv*1.5);
       fuv=fuv*fuv*fuv*fuv;
*/     all_fuv=all_fuv/0.47;
       all_fuv=2.*all_fuv*all_fuv*all_fuv-3.*all_fuv*all_fuv+1.;
       norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
/*        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));*/

       fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
       if (fxyz_mag==0.) fxyz_mag=1.;

       snx1=all_fuv*(tx[XN]/norm_mag)+xx/fxyz_mag;/*per NOR*/
       sny1=all_fuv*(tx[YN]/norm_mag)+yy/fxyz_mag;
       snz1=all_fuv*(tx[ZN]/norm_mag)+zz/fxyz_mag;
/*
       snx1=all_fuv*(pnx[XN]/norm_mag)+xx/fxyz_mag;
       sny1=all_fuv*(pnx[YN]/norm_mag)+yy/fxyz_mag;
       snz1=all_fuv*(pnx[ZN]/norm_mag)+zz/fxyz_mag;
*/        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<0.2*/

/*   my_shades(tx);
printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)    ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
extern short    pert_flag;


golf1(pnx,tx)
double pnx[],tx[];
{
int i;
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double all_fuv,fu;
double cx,cy,cz;
double cru,crw;
double xx,yy,zz;
double snx1,sny1,snz1,sx1,sy1,sz1;
double w,u;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/
```

```
/*       for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          pnx[i]=tx[i];
*/
      all_fuv=0.;
      cru=(int)tx[TX_X]+.5; crw=(int)tx[TX_Y]+.5;
      all_fuv=sqrt(sqr(tx[TX_X]-cru)+sqr(tx[TX_Y]-crw));
/*printf("tx=%f,ty=%f\n",tx[TX_X],tx[TX_Y]);
printf("cru=%f,crw=%f,all_fuv=%f\n",cru,crw,all_fuv);
*/
   if (all_fuv<0.45)
/*   if (tx[TX_X]>3. && tx[TX_X]<13.)*/ {

        snx1=sny1=snz1=0.;

/*       tx[TR]=0.; tx[TG]=0.; tx[TB]=1.;
*/
           cx=cy=cz=0.;
           my_cal_xyz(cru,crw,&xx,&yy,&zz,tx);
           cx=tx[TX]-xx; cy=tx[TY]-yy; cz=tx[TZ]-zz;/**/
        /**/
        /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
           cx[i]=xx-tx[TX]; cy[i]=yy-tx[TY]; cz[i]=zz-tx[TZ];/**/
        /**/

           xx=cx;
           yy=cy;
           zz=cz;

/*       fuv=.25/fuv; fuv=1./exp(fuv*1.5);
        fuv=fuv*fuv*fuv*fuv;
*/      all_fuv=all_fuv/0.47;
        all_fuv=2.*all_fuv*all_fuv*all_fuv-3.*all_fuv*all_fuv+1.;
        norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
/*        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));*/

        fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
        if (fxyz_mag==0.) fxyz_mag=1.;

        snx1=all_fuv*(tx[XN]/norm_mag)+.08*(xx/fxyz_mag);/*per NOR*/
        sny1=all_fuv*(tx[YN]/norm_mag)+.08*(yy/fxyz_mag);
        snz1=all_fuv*(tx[ZN]/norm_mag)+.08*(zz/fxyz_mag);
/*
        snx1=all_fuv*(pnx[XN]/norm_mag)+xx/fxyz_mag;
        sny1=all_fuv*(pnx[YN]/norm_mag)+yy/fxyz_mag;
        snz1=all_fuv*(pnx[ZN]/norm_mag)+zz/fxyz_mag;
*/        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<0.2*/

/*   my_shades(tx);
printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS      6
#define MAXLTS         16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
extern short    pert_flag;

int g2_n_cir,g2_sp_read_flag=0;
double g2_cru[300],g2_crw[300],g2_Rr[300];

a_paint(pnx,tx)
double pnx[],tx[];
{
FILE  *ip_f;
char  string[512];
int i;
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double all_fuv,fu;
double fuv[300];
double cx[300],cy[300],cz[300];
double xx,yy,zz;
double snx1,sny1,snz1,sx1,sy1,sz1;
double r1,r2,g1,g2,b1,b2;
```

```
double w,u;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*        for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          pnx[i]=tx[i];
*/
     if (g2_sp_read_flag==0) {
       ip_f=fopen("sp2.cir","r");
       if (ip_f==NULL) printf("error sp\n");
       fgets(string,512,ip_f);
       sscanf(string,"%d",&g2_n_cir);
printf("n_cir=%d\n",g2_n_cir);
     }
       pert_flag=0; all_fuv=0.;

      for (i=0;i<g2_n_cir;i++) {
       if (g2_sp_read_flag==0) {
         fgets(string,512,ip_f);
         sscanf(string,"%F %F %F",&g2_cru[i],&g2_crw[i],&g2_Rr[i]);
       }
       fuv[i]=0.;
       my_field_f(g2_cru[i],g2_crw[i],g2_Rr[i],&fuv[i],tx);
       all_fuv=all_fuv+fuv[i];
      }
      if (g2_sp_read_flag==0) fclose(ip_f);
      g2_sp_read_flag=1;

/*        my_field_f(cru1,crw1,Rr,&fuv1,tx);*/

   if (all_fuv>=0.05) {

       tx[TR]=0.; tx[TG]=1.; tx[TB]=0.;

    }/*if fu<0.2*/

/*   my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];
extern char     my_obj_name[255];
extern short    pert_flag;

int n_cir,cone_read_flag=0;
double cru[20],crw[20],Rr[20];

cup_purt(pnx,tx)
double pnx[],tx[];
{
FILE  *ip_f;
char  string[512];
int i;
double z,xn,yn,zn,tm,det,sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double all_fuv,fu;
double fuv[20];
double cx[20],cy[20],cz[20];
double xx,yy,zz;
double snxl,snyl,snzl,sxl,syl,szl;
double r1,r2,g1,g2,b1,b2;
```

```
double w,u;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*        for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
          pnx[i]=tx[i];
*/
     if (cone_read_flag==0) {
       ip_f=fopen("cup.cir","r");
       if (ip_f==NULL) printf("error==\n");
       fgets(string,512,ip_f);
       sscanf(string,"%d",&n_cir);
printf("n_cir=%d\n",n_cir);
     }
       pert_flag=0; all_fuv=0.;

      for (i=0;i<n_cir;i++) {
       if (cone_read_flag==0) {
         fgets(string,512,ip_f);
         sscanf(string,"%F %F %F",&cru[i],&crw[i],&Rr[i]);
/*printf("cru1=%f\n",cru[i]);*/
       }
       fuv[i]=0.;
       my_field_f(cru[i],crw[i],Rr[i],&fuv[i],tx);
       all_fuv=all_fuv+fuv[i];
      }

      if (cone_read_flag==0) fclose(ip_f);
      cone_read_flag=1;

/*      my_field_f(cru1,crw1,Rr,&fuv1,tx);*/

   if (all_fuv>=0.25) {

       if (all_fuv>1.) all_fuv=1.;

       pert_flag=1;
       snx1=sny1=snz1=0.;

/*       tx[TR]=1.; tx[TG]=0.; tx[TB]=1.;
*/
       tx[TR]=1.; tx[TG]=1.; tx[TB]=1.;

       for (i=0;i<n_cir;i++) {
          cx[i]=cy[i]=cz[i]=0.;
          my_cal_xyz(cru[i],crw[i],&xx,&yy,&zz,tx);
          cx[i]=tx[TX]-xx; cy[i]=tx[TY]-yy; cz[i]=tx[TZ]-zz;
       /**/
       /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
          xx=tx[TX]-xx; yy=tx[TY]-yy; zz=tx[TZ]-zz;*/
       /**/
       }

       xx=yy=zz=0.;
       for (i=0;i<n_cir;i++) {
```

```
            xx=xx+fuv[i]*cx[i];
            yy=yy+fuv[i]*cy[i];
            zz=zz+fuv[i]*cz[i];
          }

/*          fuv=.25/fuv; fuv=1./exp(fuv*1.5);
          fuv=fuv*fuv*fuv*fuv;
*/        all_fuv=(1.-all_fuv)/(1.-0.25);
          all_fuv=2.*all_fuv*all_fuv*all_fuv-3.*all_fuv*all_fuv+1.;

          norm_mag=sqrt(sqr(tx[XN])+sqr(tx[YN])+sqr(tx[ZN]));
/*          norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));*/

          fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
          if (fxyz_mag==0.) fxyz_mag=1.;

          snx1=all_fuv*(tx[XN]/norm_mag)+xx/fxyz_mag;/*per NOR*/
          sny1=all_fuv*(tx[YN]/norm_mag)+yy/fxyz_mag;
          snz1=all_fuv*(tx[ZN]/norm_mag)+zz/fxyz_mag;
/*
          snx1=all_fuv*(pnx[XN]/norm_mag)+xx/fxyz_mag;
          sny1=all_fuv*(pnx[YN]/norm_mag)+yy/fxyz_mag;
          snz1=all_fuv*(pnx[ZN]/norm_mag)+zz/fxyz_mag;
*/          tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

      }/*if fu<0.2*/

/*    my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

pieline1(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx,cy,cz,xx,yy,zz;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru,crw,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
       Rr=.040;
```

```
        fu=fuv=0.;
  for (i=0;i<8;i++) {
     if (i==0){ cru=.98; crw=tx[TX_Y];}
     if (i==1){ cru=.65; crw=tx[TX_Y];}
     if (i==2){ cru=.32; crw=tx[TX_Y];}
     if (i==3){ cru=.02; crw=tx[TX_Y];}
     if (i==4){ cru=tx[TX_X]; crw=.98;}
     if (i==5){ cru=tx[TX_X]; crw=.65;}
     if (i==6){ cru=tx[TX_X]; crw=.32;}
     if (i==7){ cru=tx[TX_X]; crw=.02;}

     fu=sqrt(sqr(tx[TX_X]-cru)+sqr(tx[TX_Y]-crw));

   if (fu<=Rr) {

        snx1=sny1=snz1=0.;
        cx=cy=cz=0.;

/*        tx[TR]=1.; tx[TG]=0.; tx[TB]=0.;*/
/**/

           my_cal_xyz(cru,crw,&xx,&yy,&zz,tx);
           cx=tx[TX]-xx; cy=tx[TY]-yy; cz=tx[TZ]-zz;/**/
        /**/
        /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
           cx=xx-tx[TX]; cy=yy-tx[TY]; cz=zz-tx[TZ];/**/
        /**/

           xx=cx;
           yy=cy;
           zz=cz;

      fuv=fuv/Rr; fuv=2*fuv*fuv*fuv-3*fuv*fuv+1;

        fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
        if (fxyz_mag==0.) fxyz_mag=1.;
        norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));

        snx1=fuv*(pnx[XN]/norm_mag)+(xx/fxyz_mag);/*per NOR*/
        sny1=fuv*(pnx[YN]/norm_mag)+(yy/fxyz_mag);
        snz1=fuv*(pnx[ZN]/norm_mag)+(zz/fxyz_mag);
        tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<=Rr*/
  } /* for cru crw */
/*my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)    ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

gline2(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx,cy,cz,xx,yy,zz;
double snxl,snyl,snzl;
double w,u,cru,crw,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
       Rr=.27;
       fu=fuv=0.;
```

```
  for (i=0;i<56;i++) {
     if (i==0){ crw=1.; cru=tx[TX_X];}
     if (i==1){ crw=2.; cru=tx[TX_X];}
     if (i==2){ crw=3.; cru=tx[TX_X];}
     if (i==3){ crw=4.; cru=tx[TX_X];}
     if (i==4){ crw=5.; cru=tx[TX_X];}
     if (i==5){ crw=6.; cru=tx[TX_X];}
     if (i==6){ crw=7.; cru=tx[TX_X];}
     if (i==7){ crw=1.5; cru=tx[TX_X];}
     if (i==8){ crw=2.5; cru=tx[TX_X];}
     if (i==9){ crw=3.5; cru=tx[TX_X];}
     if (i==10){ crw=4.5; cru=tx[TX_X];}
     if (i==11){ crw=5.5; cru=tx[TX_X];}
     if (i==12){ crw=6.5; cru=tx[TX_X];}
     if (i==13){ crw=7.5; cru=tx[TX_X];}
     if (i==14){ crw=8.; cru=tx[TX_X];}
     if (i==15){ crw=9.; cru=tx[TX_X];}
     if (i==16){ crw=10.; cru=tx[TX_X];}
     if (i==17){ crw=11.; cru=tx[TX_X];}
     if (i==18){ crw=12.; cru=tx[TX_X];}
     if (i==19){ crw=13.; cru=tx[TX_X];}
     if (i==20){ crw=14.; cru=tx[TX_X];}
     if (i==21){ crw=8.5; cru=tx[TX_X];}
     if (i==22){ crw=9.5; cru=tx[TX_X];}
     if (i==23){ crw=10.5; cru=tx[TX_X];}
     if (i==24){ crw=11.5; cru=tx[TX_X];}
     if (i==25){ crw=12.5; cru=tx[TX_X];}
     if (i==26){ crw=13.5; cru=tx[TX_X];}
     if (i==27){ crw=14.5; cru=tx[TX_X];}
     if (i==28){ crw=15.; cru=tx[TX_X];}
     if (i==29){ crw=16.; cru=tx[TX_X];}
     if (i==30){ crw=17.; cru=tx[TX_X];}
     if (i==31){ crw=18.; cru=tx[TX_X];}
     if (i==32){ crw=19.; cru=tx[TX_X];}
     if (i==33){ crw=20.; cru=tx[TX_X];}
     if (i==34){ crw=21.; cru=tx[TX_X];}
     if (i==35){ crw=15.5; cru=tx[TX_X];}
     if (i==36){ crw=16.5; cru=tx[TX_X];}
     if (i==37){ crw=17.5; cru=tx[TX_X];}
     if (i==38){ crw=18.5; cru=tx[TX_X];}
     if (i==39){ crw=19.5; cru=tx[TX_X];}
     if (i==40){ crw=20.5; cru=tx[TX_X];}
     if (i==41){ crw=21.5; cru=tx[TX_X];}
     if (i==42){ crw=21.; cru=tx[TX_X];}
     if (i==43){ crw=22.; cru=tx[TX_X];}
     if (i==44){ crw=23.; cru=tx[TX_X];}
     if (i==45){ crw=24.; cru=tx[TX_X];}
     if (i==46){ crw=25.; cru=tx[TX_X];}
     if (i==47){ crw=26.; cru=tx[TX_X];}
     if (i==48){ crw=27.; cru=tx[TX_X];}
     if (i==49){ crw=22.5; cru=tx[TX_X];}
     if (i==50){ crw=23.5; cru=tx[TX_X];}
     if (i==51){ crw=24.5; cru=tx[TX_X];}
     if (i==52){ crw=25.5; cru=tx[TX_X];}
     if (i==53){ crw=26.5; cru=tx[TX_X];}
     if (i==54){ crw=27.5; cru=tx[TX_X];}
```

```
     if (i==55){ crw=28.5; cru=tx[TX_X];}

     fu=sqrt(sqr(tx[TX_X]-cru)+sqr(tx[TX_Y]-crw));

   if (fu<=Rr) {

       snx1=sny1=snz1=0.;
       cx=cy=cz=0.;

/*       tx[TR]=0.; tx[TG]=1.; tx[TB]=0.;*/
/**/
          my_cal_xyz(cru,crw,&xx,&yy,&zz,tx);
          cx=tx[TX]-xx; cy=tx[TY]-yy; cz=tx[TZ]-zz;/**/
       /**/
       /*printf("x= %f,y= %f,z= %f\n",xx,yy,zz);
          cx=xx-tx[TX]; cy=yy-tx[TY]; cz=zz-tx[TZ];/**/
       /**/

          xx=cx;
          yy=cy;
          zz=cz;

      fuv=fuv/Rr; fuv=2*fuv*fuv*fuv-3*fuv*fuv+1;

       fxyz_mag=sqrt(sqr(xx)+sqr(yy)+sqr(zz));
       if (fxyz_mag==0.) fxyz_mag=1.;
       norm_mag=sqrt(sqr(pnx[XN])+sqr(pnx[YN])+sqr(pnx[ZN]));

       snx1=fuv*(pnx[XN]/norm_mag)+(xx/fxyz_mag);/*per NOR*/
       sny1=fuv*(pnx[YN]/norm_mag)+(yy/fxyz_mag);
       snz1=fuv*(pnx[ZN]/norm_mag)+(zz/fxyz_mag);
       tx[XN]=snx1; tx[YN]=sny1; tx[ZN]=snz1;

    }/*if fu<=Rr*/
  } /* for cru crw */
/*my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define sqr(x)     ((x)*(x))
#define c0         1.
#define fx0        3.1416/6.
#define fy0        3.1416/3.
/*#define fy0        3.1416/7.*/
#define afxy       2.
#define aci        .707
#define t0         0.
#define k          1./2.4
#define nloop      3

double  max,min;
short  xi,yi;


double poorman(x,y,z)
double x,y,z;
{
short i;
double sum1,sum2,ci,fxi,fyi,pxi,pyi,ts;
double cf(),px();

        sum1=0; sum2=0;
/*printf("x= %f,y= %f\n",x,y);
*/
        for (i=0;i<nloop;i++) {
           ci=cf(c0,i,aci);  fxi=cf(fx0,i,afxy);  fyi=cf(fy0,i,afxy);
/*printf("c= %f,x= %f,y= %f\n",ci,fxi,fyi);
*/
           pxi=px(fy0,i,afxy,y,z,fx0);   pyi=px(fx0,i,afxy,x,z,fx0);
/*printf("px= %f,py= %f\n",pxi,pyi);
*/
           ts=fxi*x+pxi; sum1=sum1+ci*sin(ts)+t0;
           ts=fyi*y+pyi; sum2=sum2+ci*sin(ts)+t0;
        }
/*printf("sum1= %f,sum2= %f\n",sum1,sum2);
*/      return(sum1*sum2*k);

}

double cf(s,n,a)
double a,s;
short n;
{
short i;
double r;
/*printf("a= %f,s= %f\n",a,s);
*/
        r=s;
        for (i=0;i<n;i++)  r=r*a;
        return(r);
}
```

```
double px(s,n,a,xy,z,fx)
double s,a,xy,z,fx;
short n;
{
double r,cfxy,ts,cfx;

    cfxy=cf(s,n,a);
    ts=.5*cfxy*xy;
    r=1.5708*sin(ts);
    cfx=cf(fx,n,a);
    ts=cfx*z/2.;
    r=r+3.1416*sin(ts);
    return(r);
}
```

```
       tx[TR]=1.; tx[TG]=1.; tx[TB]=0.;

    }/*if fu<=Rr*/
/*my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

p_line2(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx1,cy1,cz1,xx,yy,zz;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru1,crw1,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
/*       Rr=.0100;
```

```
        fu=fuv=0.;
  for (i=0;i<7;i++) {
     if (i==0){ crul=.7; crwl=tx[TX_Y];}
     if (i==1){ crul=.5; crwl=tx[TX_Y];}
     if (i==2){ crul=.3; crwl=tx[TX_Y];}
     if (i==4){ crul=tx[TX_X]; crwl=.7;}
     if (i==5){ crul=tx[TX_X]; crwl=.5;}
     if (i==6){ crul=tx[TX_X]; crwl=.3;}

     fu=sqrt(sqr(tx[TX_X]-crul)+sqr(tx[TX_Y]-crwl));

   if (fu<=Rr)
*/   if ((tx[TX_Y]<1. && tx[TX_Y]>0.) && (tx[TX_X]<2. && tx[TX_X]>1.95)) {

       tx[TR]=0.1; tx[TG]=0.1; tx[TB]=0.1;/**/

    }/*if fu<=Rr*/
/*  } * for cru crw */
/*my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)    ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

p_line3(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx,cy,cz,xx,yy,zz;
double snxl,snyl,snzl;
double w,u,cru,crw,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
       Rr=.027;
       fu=fuv=0.;
```

```
  for (i=0;i<8;i++) {
     if (i==0){ cru=.15; crw=tx[TX_Y];}
     if (i==1){ cru=.25; crw=tx[TX_Y];}
     if (i==2){ cru=.35; crw=tx[TX_Y];}
     if (i==3){ cru=.45; crw=tx[TX_Y];}
     if (i==4){ cru=.55; crw=tx[TX_Y];}
     if (i==5){ cru=.65; crw=tx[TX_Y];}
     if (i==6){ cru=.75; crw=tx[TX_Y];}
     if (i==7){ cru=.85; crw=tx[TX_Y];}

     fu=sqrt(sqr(tx[TX_X]-cru)+sqr(tx[TX_Y]-crw));

   if (fu<=Rr) {

       tx[TR]=1.; tx[TG]=0.; tx[TB]=0.;

    }/*if fu<=Rr*/
  } /* for cru crw */
/*my_shades(tx);*/
/*printf("end pertur.............\n");
*/}
```

No.

oc Color perturbation

Color perturbation 大致上是參考 gaurand 於 SIGGRAPH 87' 的 paper 加以應用在 polyhedra object 上。可用在自然景觀的模擬，上屆學長林裕隆便是從事這方面的研究。原理亦將此加入 Animator 中：

cppl.c

↓

nscene.c

↓

cloud.c

程式的流程及做法大体与 gaurand 提出的方法相同在此不贅述。

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS           15
#define NORM_PARMS       6
#define MAXLTS          16

#define sqr(x)    ((x)*(x))

extern double poorman();
extern char   my_obj_name[255];

natural_scene(newred,newgrn,newblu,tx,px_trans)
double *newred,*newgrn,*newblu,tx[],*px_trans;
{
    if ( strncmp("cloud1",my_obj_name,6)==0 )
      cloud1(newred,newgrn,newblu,tx,px_trans);
    else if ( strncmp("cloud2",my_obj_name,6)==0 )
      cloud2(newred,newgrn,newblu,tx,px_trans);
    else if ( strncmp("cloud3",my_obj_name,6)==0 )
      cloud2(newred,newgrn,newblu,tx,px_trans);
    else if ( strncmp("cloud4",my_obj_name,6)==0 )
      cloud2(newred,newgrn,newblu,tx,px_trans);
    else if ( strncmp("cloud5",my_obj_name,6)==0 )
      cloud3(newred,newgrn,newblu,tx,px_trans);
    else if ( strncmp("cloud6",my_obj_name,6)==0 )
      cloud3(newred,newgrn,newblu,tx,px_trans);

    if ( strncmp("tree",my_obj_name,4)==0 ) {
      tree(newred,newgrn,newblu,tx,px_trans);
    }

    if ( strncmp("land",my_obj_name,4)==0) {
      land1(newred,newgrn,newblu,tx,px_trans);
    }
}
```

```
land1(newred,newgrn,newblu,tx,px_trans)
double *newred,*newgrn,*newblu,tx[],*px_trans;
{
double tl_i;

      tl_i =  poorman(tx[TX],tx[TY],tx[TZ]);
      tl_i = tl_i+0.5;
      if (tl_i<0.) tl_i *= 0.-1.;
      if (tl_i>1.) tl_i=1.;
/*printf("It= %f\n",tl_i);
      px_trans = 1.-(tl_i-0.2)/0.5;
      if (px_trans>1.) px_trans=1.;
      if (px_trans<0.) px_trans=0.;
*/
      (*px_trans)=0.;
      (*newred) = 0.8 * (*newred) + 0.5 * tl_i*255.;
      (*newgrn) = 0.8 * (*newgrn) + 0.5 * tl_i*255.;
/*      (*newblu) = 0.8 * (*newblu) + 0.3 * tl_i*255.;
*/}

cloud1(newred,newgrn,newblu,tx,px_trans)
double *newred,*newgrn,*newblu,tx[],*px_trans;
{
double tl_i;

      tl_i =  poorman(tx[TX],tx[TY],tx[TZ]);
      tl_i = tl_i+0.5;
      if (tl_i<0.) tl_i *= 0.-1.;
      if (tl_i>1.) tl_i=1.;
/*printf("It= %f\n",tl_i);
      (*px_trans) = 1.-(tl_i-0.2)/0.5;
*/
      (*px_trans) = 1.-(tl_i-0.4)/1.0;
      if ((*px_trans)>1.) *px_trans=1.;
      if ((*px_trans)<0.) *px_trans=0.;

      (*newred) = 0.8 * (*newred) + 0.3 * tl_i*255.;
      (*newgrn) = 0.8 * (*newgrn) + 0.3 * tl_i*255.;
      (*newblu) = 0.8 * (*newblu) + 0.3 * tl_i*255.;
}


cloud2(newred,newgrn,newblu,tx,px_trans)
double *newred,*newgrn,*newblu,tx[],*px_trans;
{
double tl_i;

      tl_i =  poorman(tx[TX],tx[TY],tx[TZ]);
      tl_i = tl_i+0.5;
      if (tl_i<0.) tl_i *= 0.-1.;
      if (tl_i>1.) tl_i=1.;
/*printf("It= %f\n",tl_i);
      (*px_trans) = 1.-(tl_i-0.2)/0.5;
*/
      (*px_trans) = 1.-(tl_i-0.2)/0.5;
```

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

#define X       0
#define Y       1
#define Z       2

#define R       0
#define G       1
#define B       2
#define T       3
#define XN      4
#define YN      5
#define ZN      6
#define TX_X    7
#define TX_Y    8
#define TX      9
#define TY     10
#define TZ     11
#define TR     12
#define TG     13
#define TB     14

#define NORMS            15
#define NORM_PARMS        6
#define MAXLTS           16

#define sqr(x)   ((x)*(x))

/* global variables for pixel output routine */
extern short    num_lights,hilit_power,txtr,txtr_obj;
extern short    nlights,my_hilit;
extern struct { float x,y,z,r,g,b,range;} lights[MAXLTS];
extern double   ambnt,ambntcomp;                    /* lighting specifications */

extern double   quv[12],tquv[8];

p_line1(pnx,tx)
double pnx[],tx[];
{
short i;
double sqrt(),sin(),cos();
double norm_mag,dist_mag,fxyz_mag;
double fuv,fu;
double cx1,cy1,cz1,xx,yy,zz;
double snx1,sny1,snz1;
double r1,r2,g1,g2,b1,b2;
double w,u,cru1,crw1,Rr;

/*printf(" purter .............\n");*/
/*printf("tx_x= %f,tx_y= %f\n",tx[TX_X],tx[TX_Y]);*/

/*      for (i=0;i<NORMS+nlights*NORM_PARMS;i++)
         pnx[i]=tx[i];
*/
   if ((tx[TX_X]>1. && tx[TX_X]<1.3) && (tx[TX_Y]<1. && tx[TX_Y]>0.)) {
```