**計算機圖學單元介紹**

1. 英文主題：

Chapter 2: Graphics Programming

1. 中文主題：

單元二

1. 組別：

第2組

1. 組員：

B0729007\_官慶恩；B0729014\_黃建銘；B0729015\_楊金榮；B07290\_楊佶儫；

B0729017\_劉威廷；B0729024\_謝瑞筑；B0829012\_鄭家竣

1. 作業分工：

(詳見作業報告)

1. 功能簡述：

本單元為介紹計算機圖學中，OpenGL的介紹以及範例實作的操演，當中包含了計算機圖學繪圖中可能遭遇到的困難及解決方法。

1. 主要程式碼：

相關檔案：Ch\_00\_tm0\_src1.cpp

|  |
| --- |
| (以1x1表格填寫，文字為 “Segoe UI” 11點字，固定行高12點，內容可變更文字顏色)  #include<stdlib.h>  #include<stdio.h>  #include<time.h>  #include<GL/glut.h>  #include<cmath>  #define pi acos(-1)    void init (void)  {  glClearColor (1.0, 1.0, 1.0, 0.0);  glMatrixMode (GL\_PROJECTION);  gluOrtho2D (0.0, 300.0, 0.0, 300.0);  }  void show (void)  {  glClear (GL\_COLOR\_BUFFER\_BIT);  glLineWidth(1);  glBegin (GL\_LINE\_LOOP);  glColor3f(0.0, 0.0, 1.0);  glVertex2i (25, 205);  glVertex2i (25, 225);  glVertex2i (55, 225);  glVertex2i (55, 205);  glEnd ( );    glBegin (GL\_POLYGON);  glColor3f(0.0, 1.0, 0.0);  glVertex2i (25, 190);  glVertex2i (55, 190);  glVertex2i (55, 170);  glVertex2i (25, 170);  glEnd ( );    glBegin (GL\_POLYGON);  glColor3f(0.0, 0.0, 1.0);  glVertex2i (25, 155);  glVertex2i (55, 155);  glVertex2i (55, 135);  glVertex2i (25, 135);  glEnd ( );    glBegin (GL\_LINE\_LOOP);  glColor3f(1.0, 0.0, 0.0);  glVertex2i (25, 155);  glVertex2i (55, 155);  glVertex2i (55, 135);  glVertex2i (25, 135);  glEnd ( );    glEnable (GL\_LINE\_SMOOTH);  glHint (GL\_LINE\_SMOOTH, GL\_NICEST);  glLineWidth(10);    glBegin (GL\_LINE\_LOOP);  glColor3f(0.0, 0.0, 1.0);  for(float i=0; i<10000; i++)  {  glVertex2f(110+25\*cos(2\*i\*pi/10000), 200+25\*sin(2\*i\*pi/10000));  }  glEnd ( );    glBegin (GL\_LINE\_LOOP);  glColor3f(0.0, 0.0, 0.0);  for(float i=0; i<10000; i++)  {  glVertex2f(175+25\*cos(2\*i\*pi/10000), 200+25\*sin(2\*i\*pi/10000));  }  glEnd ( );    glBegin (GL\_LINE\_LOOP);  glColor3f(1.0, 0.0, 0.0);  for(float i=0; i<10000; i++)  {  glVertex2f(240+25\*cos(2\*i\*pi/10000), 200+25\*sin(2\*i\*pi/10000));  }  glEnd ( );    glBegin (GL\_LINE\_LOOP);  glColor3f(1.0, 1.0, 0.0);  for(float i=0; i<10000; i++)  {  glVertex2f(142.5+25\*cos(2\*i\*pi/10000), 175+25\*sin(2\*i\*pi/10000));  }  glEnd ( );    glBegin (GL\_LINE\_LOOP);  glColor3f(0.0, 1.0, 0.0);  for(float i=0; i<10000; i++)  {  glVertex2f(207.5+25\*cos(2\*i\*pi/10000), 175+25\*sin(2\*i\*pi/10000));  }  glEnd ( );    glFlush ( );  }  int main (int argc, char\*\* argv)  {  glutInit (&argc, argv);  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition (50, 100);  glutInitWindowSize (600, 600);  glutCreateWindow ("B0729056\_Training-1");    init ( );  glutDisplayFunc (show);  glutMainLoop ( );  } |

1. 程式說明：
   1. 對自定義init函式初始化
   2. glMatrixMode (GL\_PROJECTION):  
       函數會指定哪一個矩陣是目前的矩陣，也就是在要做下一步之前告訴電腦我要對“什麼”進行操作了。

參數有3種模式:：

GL\_PROJECTION 投影

GL\_MODELVIEW 模型視圖

GL\_TEXTURE 纹理

GL\_PROJECTION是投影的意思，目的是對投影相關進行操作，希望把物體投影到一個平面上，如同照相一般，把3維物體投到2維的平面上。這樣，接下來的語句可以是跟透視相關的函數

* 1. gluOrtho2D (0.0, 300.0, 0.0, 300.0);  
     函式會定義 2D正向投射矩陣，參數為(left, right, top, bottom)
  2. 接下來以下列函式進行繪製
  3. glBegin (GL\_LINE\_LOOP);  
      將從 glBegin 和後續 glend函式會分隔基本或類似基本類型群組的頂點。參數GL\_LINE\_LOOP繪製從第一個頂點到最後一個頂點的已連接線段群組，然後回到第一個頂點。 頂點 n 和 n + 1 定義行 n。 但是，最後一行是由頂點 N 和 1 所定義。 會繪製 N 行。
  4. glVertex2i (25, 205);  
      參數(指定頂點的 x 座標, 指定頂點的 y 座標) 皆為integer，x函數命令會在 glBegin / glEnd配對中用來指定點、線條和多邊形頂點。當呼叫 glVertex 時，目前的色彩、標準和材質座標會與頂點相關聯。當僅指定 x 和 y 時， z 的預設值為0.0， w 預設值為1.0。 當指定 x、 y 和 z時， w 預設為1.0。 在 glBegin / glEnd 配對之外叫用 glVertex 會導致未定義的行為。
  5. glEnable (GL\_LINE\_SMOOTH);  
      函數會啟用各種OpenGL功能，GL\_LINE\_SMOOTH若已啟用，請使用正確的篩選繪製行。如果停用，則繪製別名行，可參考glLineWidth()。
  6. glHint (GL\_LINE\_SMOOTH, GL\_NICEST);  
      函式會指定實作為特定的提示，參數為(target,mode)。target指出要控制之行為的符號常數，mode表示所需行為的符號常數，GL\_LINE\_SMOOTH指出反鋸齒線條的取樣品質。 如果套用較大的篩選函式，提示 GL 最好可能會導致在點陣化期間產生更多圖元片段；GL\_NICEST為最正確或最高品質的選項。
  7. glLineWidth(10);  
      GlLineWidth 函式會指定鋸齒和反鋸齒線條的點陣化寬度。使用1.0 以外的線條寬度有不同的效果，這取決於是否已啟用行消除鋸齒功能。藉由呼叫 glEnable 和 glDisable ，並將引數 GL\_LINE\_SMOOTH，以控制線條消除鋸齒。
  8. glVertex2f(110+25\*cos(2\*i\*pi/10000), 200+25\*sin(2\*i\*pi/10000));  
      同glVertex2i，只是參數改為float，在此以迴圈建構出圓形。
  9. glFlush ( );  
      函式會清空所有這些緩衝區，使所有發出的命令都能以實際轉譯引擎接受的速度快速執行。 雖然這項執行可能不會在任何特定時間內完成，但會在有限的時間內完成。
  10. 在main中進行操作
  11. glutDisplayFunc (show);  
       來設置窗口刷新的消息處理函數，其唯一的參數指定了視窗刷新時會調用的函數，即先前定義的show函式。
  12. glutMainLoop ( );

進入GLUT事件處理循環。在一個GLUT程式中，這個例程被調用一次 。一旦被調用，這個程式將永遠不會返回 。它將調用必要的任何已註冊的回調

1. 延伸應用程式碼：

相關檔案：CH2.8 The Gasket Program

|  |
| --- |
| (以1x1表格填寫，文字為 “Segoe UI” 11點字，固定行高12點，內容可變更文字顏色)  #include<stdlib.h>  #include<stdio.h>  #include<time.h>  #include <GL/glut.h>  void myinit()  {  /\* attributes \*/  glClearColor(1.0, 1.0, 1.0, 1.0); /\* white background \*/  glColor3f(1.0, 0.0, 0.0); /\* draw in red \*/  /\* set up viewing \*/  /\* 500 x 500 window with origin lower left \*/  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0.0, 50.0, 0.0, 50.0);  glMatrixMode(GL\_MODELVIEW);  }  void display( void )  {  GLfloat vertices[3][2]={{0.0,0.0},{25.0,50.0},{50.0,0.0}}; /\* A triangle \*/  int j, k;  GLfloat p[2] ={7.5,5.0}; /\* An arbitrary initial point inside traingle \*/  glClear(GL\_COLOR\_BUFFER\_BIT); /\*clear the window \*/  /\* compute and plots 5000 new points \*/  glBegin(GL\_POINTS);  for( k=0; k<5000; k++)  {  j=rand()%3; /\* pick a vertex at random \*/  /\* Compute point halfway between selected vertex and old point \*/  p[0] = (p[0]+vertices[j][0])/2.0;  p[1] = (p[1]+vertices[j][1])/2.0;  /\* plot new point \*/  glVertex2fv(p);  }  glEnd();  glFlush(); /\* clear buffers \*/  }  int main(int argc, char\*\* argv)  {  /\* Standard GLUT initialization \*/  glutInit(&argc,argv);  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB); /\* default, not needed \*/  glutInitWindowSize(500,500); /\* 500 x 500 pixel window \*/  glutInitWindowPosition(0,0); /\* place window top left on display \*/  glutCreateWindow("Sierpinski Gasket"); /\* window title \*/  glutDisplayFunc(display); /\* display callback invoked when window opened \*/  myinit(); /\* set attributes \*/  glutMainLoop(); /\* enter event loop \*/  } |

相關檔案：CH2.9 POLYGONS and RECURSION

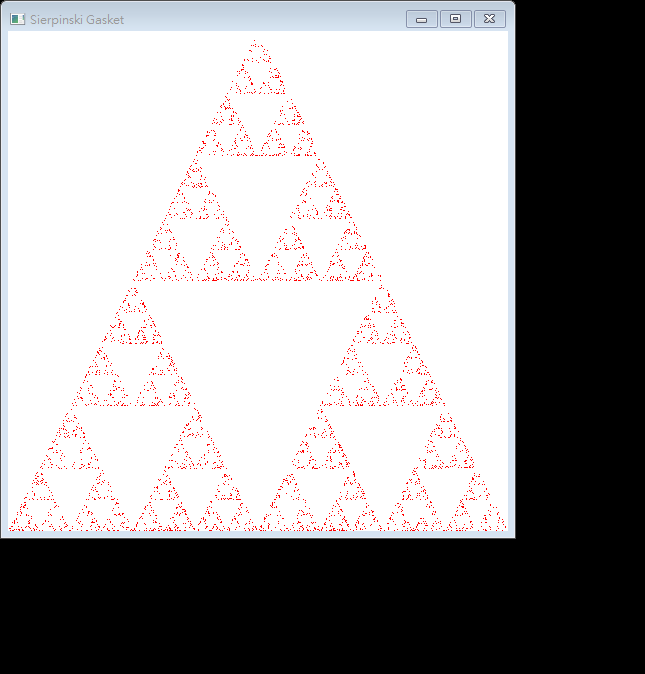
|  |
| --- |
| (以1x1表格填寫，文字為 “Segoe UI” 11點字，固定行高12點，內容可變更文字顏色)  #include<stdlib.h>  #include<stdio.h>  #include<time.h>  #include <GL/glut.h>  /\* initial triangle \*/  GLfloat v[3][2]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};  int n;  void triangle( GLfloat \*a, GLfloat \*b, GLfloat \*c)  /\* specify one triangle \*/  {  glVertex2fv(a);  glVertex2fv(b);  glVertex2fv(c);  }  void divide\_triangle(GLfloat \*a, GLfloat \*b, GLfloat \*c, int m)  {  /\* triangle subdivision using vertex numbers \*/  GLfloat v0[2], v1[2], v2[2];  int j;  if(m>0)  {  for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;  for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;  for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;  divide\_triangle(a, v0, v1, m-1);  divide\_triangle(c, v1, v2, m-1);  divide\_triangle(b, v2, v0, m-1);  }  else triangle(a,b,c); /\* draw triangle at end of recursion \*/  }  void display()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_TRIANGLES);  divide\_triangle(v[0], v[1], v[2], n);  glEnd();  glFlush();  }  void myinit()  {  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(-2.0, 2.0, -2.0, 2.0);  glMatrixMode(GL\_MODELVIEW);  glClearColor (1.0, 1.0, 1.0, 1.0);  glColor3f(0.0,0.0,0.0);  }  int main(int argc, char \*\*argv)  {  n=5; /\* or set number of subdivision steps here \*/  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize(500, 500);  glutCreateWindow("Sierpinski Gasket");  glutDisplayFunc(display);  myinit();  glutMainLoop();  } |

相關檔案：CH2.10 The Three Dimensional Gasket

|  |
| --- |
| (以1x1表格填寫，文字為 “Segoe UI” 11點字，固定行高12點，內容可變更文字顏色)  void divide\_tetra(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, int m){  GLfloat mid[6][3];  int j;  if(m>0)  { /\* compute six midpoints \*/  for(j=0; j<3; j++) mid[0][j]=(a[j]+b[j])/2;  for(j=0; j<3; j++) mid[1][j]=(a[j]+c[j])/2;  for(j=0; j<3; j++) mid[2][j]=(a[j]+d[j])/2;  for(j=0; j<3; j++) mid[3][j]=(b[j]+c[j])/2;  for(j=0; j<3; j++) mid[4][j]=(c[j]+d[j])/2;  for(j=0; j<3; j++) mid[5][j]=(b[j]+d[j])/2;  /\* create 4 tetrahedrons by subdivision \*/  divide\_tetra(a, mid[0], mid[1], mid[2], m-1);  divide\_tetra(mid[0], b, mid[3], mid[5], m-1);  divide\_tetra(mid[1], mid[3], c, mid[4], m-1);  divide\_tetra(mid[2], mid[4], d, mid[5], m-1);  } else(tetra(a,b,c,d)); /\* draw tetrahedron at end of recursion \*/  }  void display()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_TRIANGLES);  //------------------------------------------------  divide\_tetra(v[0], v[1], v[2], v[3], n);  //------------------------------------------------  glEnd();  glFlush();  } |

1. 應用說明：

2.8 The Gasket Program: 生成Sierpinski gasket.

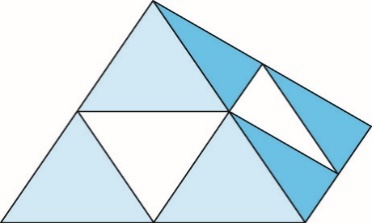


2.9 POLYGONS and RECURSION: 生成 Sierpinski 墊片式的第二種方法

主要變動或靈活運用之處: 該方法使用多邊形而不是點，並且不需要使用隨機數生成器，意味著你可以控制輸出的圖形。

2.10 The Three Dimensional Gasket:

基於two dimensional gasket 再加上一個維度所構成，因此我們需要四個初始的三維頂點來定義四面體的結構。先前tetra() function建立的立體圖形有一些我們不需要的區塊(如下圖)



所以我們改用recursive subdivision的方法來取得四個需要的四面體

十一、注意事項：

GlVertex\*():其中\*可解釋為nt或ntv形式的兩個或三個字元，其中n表示維度(2,3或4)；t表示資料型態，如整數(i)、符點數(f)或雙精度數(d)；和v，如果存在，索引變量是通過指向數組的指標而不是通過參數列表指定的。

#define GLfloat float(header file):在OpenGL，我們常使用基本OpenGL型態，例如GLfloat和Glint而不是C型態，例如float和int。(改變型態而不改變現有的應用程式)

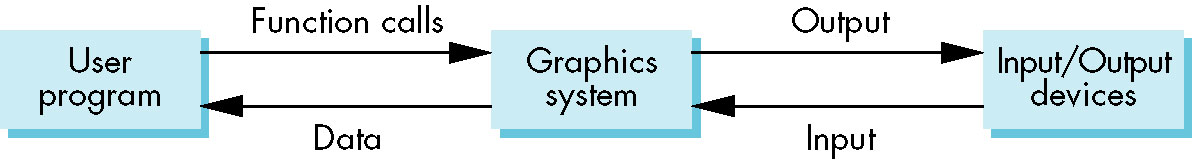
合併上述兩者應用如:

(1)使用者想在二維空間(三維空間)用整數:

glVertex2i(GLint xi,GLint yi)、glVertex3i(GLfloat x, GLfloat y, GLfloat z)

(2)我們使用陣列儲存三維向量資訊:

GLfloat vertex[3],我們可使用glVertex3fv(vertex)

OpenGL結構類似於大多數現代API架構，包括Java3D和DirectX

一個好的API可能包含上百個函數，因此將他們分為七大組:

(1)Primitive functions:定義系統可以顯示的低層物件或atomic entities。(ch1.2)

(2)Attribute functions:允許我們執行從選擇顯示線段的顏色到選擇填充多邊形內部的圖案操作。(ch2)

(3)Viewing functions:允許我們指定各種視圖，儘管API在選擇試圖時提供的靈活性程度不同。(ch5)

(4)Transformation functions:允許進行物件的變換，例如旋轉等。(ch4)

(5)Input functions:對於交互函數API必須提供一組輸入函數，以允許我們處理表徵現代圖形系統的各種輸入形式。(ch3)

(6)Control functions:使我們能夠與窗口系統通性、初始化程式並處理任何錯誤。(ch1)

(7)Query functions:通過一組查詢函數提供特定的操作資訊。

**OpenGL Interface**

大部分的應用將被設計為直接通過三個函式庫存取OpenGL:

(1)函式在主要GL(或OpenGL in windows)函式庫的函式名以字母gl開頭。

(2)OpenGL Utility Library(GLU):這個函式庫僅使用GL函式，但包含用於創建一般物件和簡化Viewing的程式碼(用字母glu)。

(3)OpenGL Utility Toolkit(GLUT):為了與窗口系統交互並從外部設備獲取輸入到我們的程式中，提供了在任何現代窗口系統中都應具備的minimum功能(用字母glut)

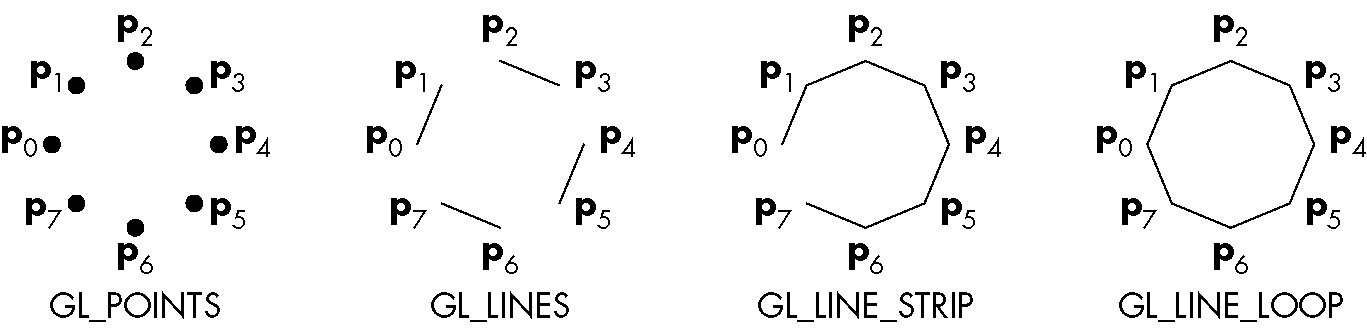
OpenGL提供兩類primitives:

(1)Geometric primitives:特定在問題域中，包括點、線、多邊形、曲線和曲面。這些primitives通過geometric pipeline，存在2或3維空間，可以通過旋轉等操作對其進行操作。

(2)Image or raster primitives:例如像素陣列，缺乏幾何屬性，不能像geometric primitives在空間中進行操作。

我們希望顯示點或線段，在OpenGL中有一些物件，其類型規範包括以下內容。

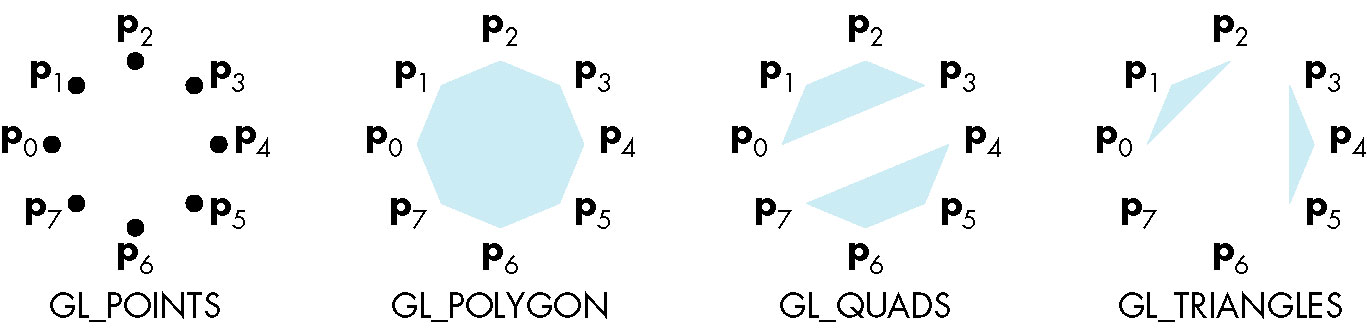
Points(GL\_POINTS)、Line segments(GL\_LINES)、Polylines(GL\_LINE\_STRIP,GL\_LINE\_LOOP)

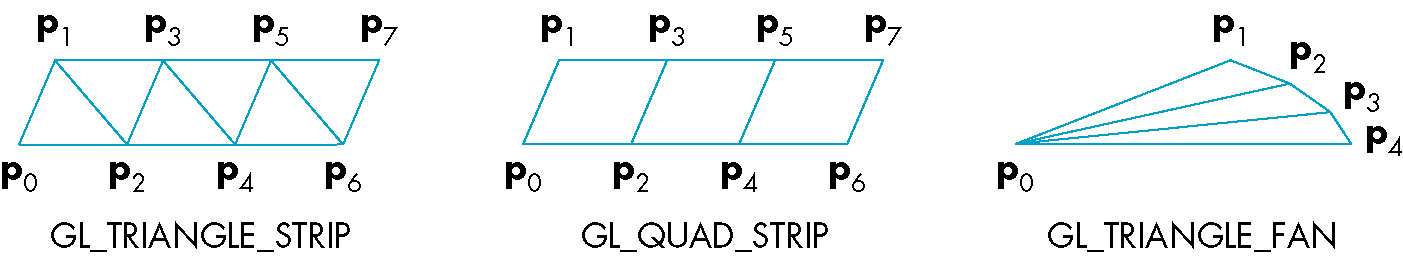


Polygons (GL\_POLYGON):edge與我們使用線循環時的情況相同。我們可以使用函數glPolygonMode告訴renderer只生成edge或只生成頂點的點，而不是填充。

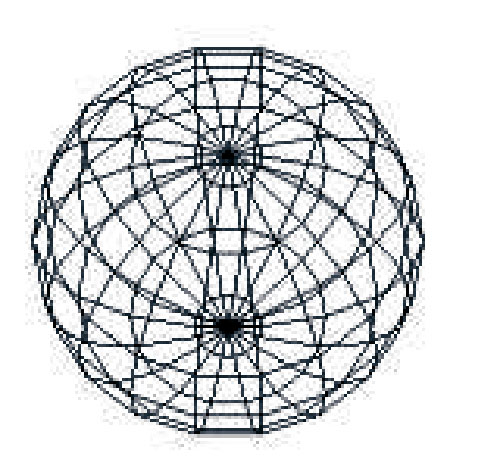
Triangles and Quadrilaterals(GL\_TRIANGLES,GL\_QUADS)

Strips and Fans(GL\_TRIANGLE\_STRIP,GL\_QUAD\_STIP,GL\_TRIANGLE\_FAN)





近似球體扇形和條帶使我們能夠簡單地近似許多曲面。考慮一個單位球體，我們可以用以下三個方程式來描述它。(我們可用四邊形或兩個三角扇型來表示)

x(θ,φ)=sinθcosφ, y(θ,φ)=cosθcosφ, z(θ,φ)=sinφ

Raster text是簡單快捷。自原被定義為位元區塊的位元矩形。每個區塊通過單一區塊中的0和一位元定義每個字元。Raster character可以通過位元位元區塊傳送(bitblt)操作快速放入幀緩衝區，該操作使用單個函式調用移動區塊的位元，你可以通過複製或複製像素來增加raster characters的大小，這一過程會使較大的字源呈現區塊狀外觀。OpenGL沒有text primitives。但是GLUT函式庫提供一些軟體中定義並可移植的預定bitmap和stroke character sets。

GlutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13,c)

GlRasterPos\*去定義位置。

我們用24位元存儲在幀緩衝區中的任何顏色(RGB)，一種自然的技術是使用顏色立方體並將顏色分量指定為0.0和1.0之間的數字，其中1.0表示最大值，例如紅色glColor3f(1.0,0.0,0.0)。

四色(RGBA)系統，第四種顏色(A或alpha)也與RGB值一樣存在幀緩衝區中；它可以用顏色函數的四維版本進行設置。如今啟用混和，那麼OpenGL會將alpha值視為不透明度或透明度值。不透明度值的範圍可以從完全透明(A=0.0)到完全不透明(A=1.0)我們必須在程式中執行的首要任務之一是清除屏幕上的一個區域---一個繪圖窗口---在其中顯示輸出。glClearColor(1.0,1.0,1.0,1.0)(使螢幕上的窗口變為純白色)。

如果我們處於顏色索引模式，則當前顏色由諸如glIndexi(element)之類的函式選擇；從表格中選擇特定顏色。GLUT允許我們通過以下函式位每個窗口設置顏色表中的條目：glutSetColors(int color,GLfloat red,GLfloat blue)，顏色索引模式很重要，因為它需要更少的幀緩衝區內存和更少的其他硬件組件。因此，在大多數情況下，我們將假設我們使用的是RGB顏色。

第一步清除顏色glClearColor(1.0,1.0,1.0,1.0)->設定顏色glColor3f(1.0,0.0,0.0)->設定渲染點大小為2像素寬glPointSize(2.0)

**Matrix Modes**

最重要的兩種矩陣：模型觀察矩陣(Model-View Matrix)、投影矩陣(Projection Matrix)

模型觀察矩陣：將模型的座標系對應到空間座標系再對應攝影機所看到的座標系

投影矩陣：將攝影機所看到的座標系對應到螢幕呈現出來的2D座標系

從Model-View Matrix轉換至Projection Matrix

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 50.0, 0.0, 50.0);

glMatrixMode(GL\_MODELVIEW);

50\*50即為Viewing Rectangle的大小

**Control Functions**

Window – 矩形的視窗，是bottom-left左下為(0,0)，x軸往右y軸往上，相關函式有：

glutCreateWindow (char \*title) – 建立一個視窗給定名字為\*title。

glutInitDisplayMode – 指定視窗的顯示模式，主要分為顏色模式跟緩衝區類型。常用的如：GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH | GLUT\_STENCIL

glutInitWindowSize(m,n) – 設定大小為m\*n (pixel為單位)的視窗。

glutInitWindowPosition(x,y) – 設定視窗在螢幕上出現的位置。

Aspect Ratio and Viewports – 有兩個比較重要的投影變換函數， glViewport和glOrtho.

glOrtho只是負責使用什麼樣的視景體來截取圖像，並不負責使用某種規則把圖像呈現在螢幕上，所以可能會變形。

glViewport主要完成這樣的功能。 它負責把視景體截取的圖像按照怎樣的高和寬顯示到螢幕上。

The main, display, and myinit Functions –

glutMainLoop() 則是一個GLUT的動作，它將控制權移送給GLUT，並開始它自己的內部循環，若沒有要執行的會進入等待狀態。

void glutDisplayFunc(void (\*func)(void)) 是display callback的函式將圖形傳送到螢幕視窗。

The function myinit() is used to set the OpenGL state variables dealing with viewing and attributes.

初始化 - glutInit(int \*argc, char \*\*argv)需要在任何GLUT函式之前先設置，用來初始化GLUT和cmd的arguments

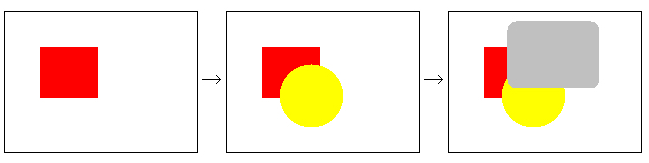
**Hidden-Surface Removal**

在3D計算機圖學上，我們希望可以把隱藏在表面下的線條移除。

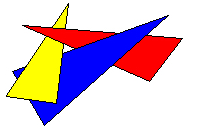
在openGL中我們使用z-buffer的方式去做處理。

z-buffer簡單來說主要目的是用來去除隱藏面( Hiddem surface elimination or Visible surface detemination) 3D繪圖上我們會出現只要出現兩者以上的三角面變就會出現某個物體的三角面被遮住的情形，這在 3D圖學上是種極其常發生的事情，所以在繪製3D場景時，要繪出正確的成果，就一定要處理這個問題。

畫家演算法(Painter’s algo) 透過覆蓋來解決此問題



不過實際上的問題並沒有那麼理想，因3D場景中可能會出現三角面交互重疊的狀況。



此時不管先從哪個物體先話都無法滿足實際狀況。

z-buffer的原理是我們在原本存放繪製結果的frame buffer中再使用額外的空間，來儲存每個pixel與觀察者的距離(也就是 Z值)，繪製場景前 z\_buffer的初始值會被設定成無限遠，然後在繪製三角面時，開始計算 Z值，並和 z\_buffer中存放的 Z值比較。Z值較大則為近，代表者應該被畫上去，並且更新Z值；Z值較小，代表無須被畫上，會由目前frame budder中的pixel遮住，且不必更新Z值。

十三、參考資料：

1. 可參考的公開文件
2. 可參考的公開網址

等等