**計算機圖學單元介紹**

一、英文主題：

Chapter 3: Input and Interaction

二、中文主題：

單元03：輸入與互動

三、組別：

第03組

四、組員：

B0729009\_孫維嶺 ; B0729030\_趙威竹 ; B0729036\_林品翰 ; B0729055\_鍾佩宸 ;

B0729058\_林聖凱 ; B0829025\_林宥任

五、作業分工：

(詳見作業報告) …略

六、功能簡述：

將會提到可互動圖形程式(interactive graphics programs)的發展

* 分成3部分來討論

1.介紹各種可互動的裝置，例如:

* + Physical device: 滑鼠、鍵盤...
  + Logical device: 在電腦中應用程式的呈現方式，high-level interface

2.然後是client- Server網路和client- Server圖形。

* + 圖形程式中的事件驅動程式

3.最後介紹一個多邊形建模程式，它演示了可互動式圖形程式的許多重要功能

以此利用滑鼠繪製圖案

七、主要程式碼：

本單元為第一次程式實作，為完整解釋程式流程，故將全部原始碼詳列如下，

相關檔案：Ch\_03\_tm3\_src1.cpp(pick.c)

|  |
| --- |
| #include<stdlib.h>  #include<stdio.h>  #include<time.h>  #include <GL/glut.h>  void init(){  glClearColor (0.0, 0.0, 0.0, 0.0);}  void drawObjects(GLenum mode){  if(mode == GL\_SELECT) glLoadName(1);  glColor3f(1.0, 0.0, 0.0);  glRectf(-0.5, -0.5, 1.0, 1.0);  if(mode == GL\_SELECT) glLoadName(2);  glColor3f(0.0, 0.0, 1.0);  glRectf(-1.0, -1.0, 0.5, 0.5);}  void display(){  glClear(GL\_COLOR\_BUFFER\_BIT);  drawObjects(GL\_RENDER);  glFlush();}  void processHits (GLint hits, GLuint buffer[]){  unsigned int i, j;  GLuint names, \*ptr;  printf ("hits = %d\n", hits);  ptr = (GLuint \*) buffer;  for (i = 0; i < hits; i++)  { names = \*ptr;  ptr+=3;  for (j = 0; j < names; j++)  { if(\*ptr==1) printf ("red rectangle\n");  else printf ("blue rectangle\n");  ptr++;  }  printf ("\n");  }  }  #define SIZE 512  void mouse(int button, int state, int x, int y){  GLuint selectBuf[SIZE];  GLint hits;  GLint viewport[4];  if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {  glGetIntegerv (GL\_VIEWPORT, viewport);  glSelectBuffer (SIZE, selectBuf);  glRenderMode(GL\_SELECT);  glInitNames();  glPushName(0);  glMatrixMode (GL\_PROJECTION);  glPushMatrix ();  glLoadIdentity ();  /\* create 5x5 pixel picking region near cursor location \*/  gluPickMatrix ((GLdouble) x, (GLdouble) (viewport[3] - y),  5.0, 5.0, viewport);  gluOrtho2D (-2.0, 2.0, -2.0, 2.0);  drawObjects(GL\_SELECT);  glMatrixMode (GL\_PROJECTION);  glPopMatrix ();  glFlush ();  hits = glRenderMode (GL\_RENDER);  processHits (hits, selectBuf);  glutPostRedisplay();  }  }  void reshape(int w, int h)  {  glViewport(0, 0, w, h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D (-2.0, 2.0, -2.0, 2.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  void keyboard(unsigned char key,  int x, int y)  {  switch (key)  {  case 27:  exit(0);  break;  }  }  /\* main loop \*/ int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize (500, 500);  glutInitWindowPosition (100, 100);  glutCreateWindow (argv[0]);  init ();  glutReshapeFunc (reshape);  glutDisplayFunc(display);  glutMouseFunc (mouse);  glutKeyboardFunc (keyboard);  glutMainLoop();  return 0;  }  執行結果: |

八、程式說明：

(1) 執行OpenGL，在標頭檔部份，只需引用<GL/freeglut.h>即可，往後如需使其他數學與時間函數，可自行引入，但皆與OpenGL繪圖無關。

(2) 我們直接由 main() 主程式開始跑流程；

(3) glutInit(&argc, argv); 用來初始化GLUT，這裡它的參數直接由main()抄過來，main()執行時如果下達 '-sync' 或 '-gldebug' 等參數，就可以直接帶入這裡進行初始化，而達到監視與除錯等目的，這些專門的參數可以自行在網路上查詢。

(4) glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA); 用來設定GLUT核心區域中的兩項狀態參數。GLUT\_DOUBLE和GLUT\_RGBA都是在<GL/freeglut.h>已定義的公用常數，但在執行時才設定給GLUT知道。GLUT\_DOUBLE 是指設置雙緩衝而GLUT\_RGBA 是指色彩模式是RGBA，多了一個透明度(A)，做此設定之後，如果用到色彩，就要用四維描述，比GLUT\_RGB多了一維。

(5) 接下來還有2個狀態參數值要設定，分別是視窗大小(w,h)，以及出現的位置，

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

(6) 以上4個狀態值設完成後，接著執行 glutCreateWindow(argv[0])，來建立一個繪圖視窗，或者說是向系統取得(allocate)一塊記憶體區段(或稱緩衝區、畫布)，並在GLUT核心區域記錄一個index，完成畫布的註記。

(7) glutReshapeFunc (reshape);這程式的目的是將因視窗改變的圖像調整回來，首先先利用glViewport(0, 0, w, h)來設定擷取的影像在視窗的哪一部分顯示，再使用glMatrixMode(GL\_PROJECTION)將影像投影，並透過glLoadIdentity()將其轉為識別矩陣，再利用gluOrtho2D (-2.0, 2.0, -2.0, 2.0)重新擷取圖像，並將其變為新的識別矩陣。

(8) glutDisplayFunc(display);

這個API是把一段我們自已撰寫的程式 'display' 其程式碼的指標值(index)寫到GLUT核心區域的「待執行程序」位置。函數內傳函數指標，是一項高級的程式撰寫技巧，它在其他方面有各種變化與運用方式，而在OpenGL的常用技法中，我們藉此在GLUT的紀錄區域寫上我們所開發的程式碼，以待後續呼叫執行。

(9)glutMouseFunc (mouse);這函式的目的是用來設定滑鼠的回撥函式

(10)glutKeyboardFunc (keyboard);這函式的目的是用來設定鍵盤的回撥函式

(11) glutMainLoop();

glutMainLoop() 是一個GLUT的動作API，它將控制權移送給GLUT，並開始它自己的內部循環。此前，我們曾用glutDisplayFunc(display)在GLUT的核心區域設定一段我們自己撰寫的程式 'display()' 作為它的「待執行程序」，現在GLUT取得控制權之後就會去執行這段程式。

glutMainLoop() 現在做兩件事：1.實際顯示出glutCreateWindow(argv[0])所建立的視窗，2.執行之前指定的自撰程式 'display()'，它的內容如以下(12)所述。

(12) void display(){

glClear(GL\_COLOR\_BUFFER\_BIT);

drawObjects(GL\_RENDER);

glFlush();}

首先，我們會使用glClear(GL\_COLOR\_BUFFER\_BIT)對畫布進行清除，再利用drawObjects(GL\_RENDER)來使用預設的渲染模式來作為我們的繪圖程式，最後利用glFlush()清空緩衝區，將指令送往緩硬件立即執行，但是它是將命令傳送完畢之後立即返回，不會等待指令執行完畢。這些指令會在有限時間內執行完畢。

九、延伸應用程式碼： (Ch\_03\_tm3\_src2.cpp)(polygon.c)

|  |
| --- |
| /\* polygon modeler \*/  #define MAX\_POLYGONS 8  #define MAX\_VERTICES 10  typedef int bool;  #define TRUE 1  #define FALSE 0  #include<stdlib.h>  #include<stdio.h>  #include<time.h>  #include <GL/glut.h>  void myMouse(int,int, int, int);  void myMotion(int, int);  void myDisplay();  void myReshape(int, int);  void color\_menu(int);  void main\_menu(int);  int pick\_polygon(int x, int y);  void myinit();  /\* globals \*/  /\* polygon struct \*/  typedef struct polygon  {  int color; /\* color index \*/  bool used; /\* TRUE if polygon exists \*/  int xmin, xmax, ymin, ymax; /\* bounding box \*/  float xc, yc; /\* center of polygon \*/  int nvertices; /\* number of vertices \*/  int x[MAX\_VERTICES]; /\* vertices \*/  int y[MAX\_VERTICES];  } polygon;  /\* flags \*/  bool picking = FALSE; /\* true while picking \*/  bool moving = FALSE; /\* true while moving polygon \*/  int in\_polygon = -1; /\* not in any polygon \*/  int present\_color = 0; /\* default color \*/  GLsizei wh = 500, ww = 500; /\* initial window size \*/  int draw\_mode = 0; /\* drawing mode \*/  GLfloat colors[8][3]={{0.0, 0.0, 0.0}, {1.0, 0.0, 0.0},{0.0, 1.0, 0.0},  {0.0, 0.0, 1.0}, {0.0, 1.0, 1.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 0.0},  {1.0, 1.0, 1.0}};  polygon polygons[MAX\_POLYGONS];  void myReshape(int w, int h)  {  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  glViewport(0,0,w,h);  ww = w;  wh = h;  }  void myinit()  {  int i;  /\* set clear color to grey \*/  glClearColor(0.5, 0.5, 0.5, 1.0);  /\* mark all polygons unused \*/  for(i = 0; i<MAX\_POLYGONS; i++) polygons[i].used = FALSE;  }  void myMouse(int btn, int state, int x, int y)  {  int i,j;  y = wh-y;  if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN &&!picking&&!moving)  /\* adding vertices \*/  {  moving = FALSE;  if(in\_polygon>=0)  {  if(polygons[in\_polygon].nvertices == MAX\_VERTICES)  {  printf("exceeds maximum number vertices\n");  exit(0);  }  i = polygons[in\_polygon].nvertices;  polygons[in\_polygon].x[i] = x;  polygons[in\_polygon].y[i] = y;  polygons[in\_polygon].nvertices++;  }  }  if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN &&picking&&!moving)  {  /\* delete polygon \*/  picking = FALSE;  moving = FALSE;  j = pick\_polygon(x,y);  if(j >= 0)  {  polygons[j].used = FALSE;  in\_polygon = -1;  glutPostRedisplay();  }  }  }  int pick\_polygon(int x, int y)  {  /\* find first polygon in which we are in bounding box \*/  int i;  for(i=0; i<MAX\_POLYGONS; i++)  {  if(polygons[i].used)  if((x>=polygons[i].xmin)&&(x<=polygons[i].xmax)&&(y>=polygons[i].ymin)&&(y<polygons[i].ymax))  {  in\_polygon = i;  moving = TRUE;  return(i);  }  printf("not in a polygon\n");  return(-1);  }  }  void myMotion(int x, int y)  {  /\* find if we are inside a polugon \*/  float dx, dy;  int i,j;  if(moving)  {  y = wh - y;  j = pick\_polygon(x, y);  if(j<0)  {  printf("not in a polygon\n");  return;  }  /\* if inside then move polygon \*/  dx = x - polygons[j].xc;  dy = y - polygons[j].yc;  for(i = 0; i< polygons[j].nvertices; i++)  {  polygons[j].x[i] += dx;  polygons[j].y[i] += dy;  }  /\* update bounding box \*/  polygons[j].xc += dx;  polygons[j].yc += dy;  if(dx>0) polygons[j].xmax += dx;  else polygons[j].xmin += dx;  if(dy>0) polygons[j].ymax += dy;  else polygons[j].ymin += dy;  glutPostRedisplay();  }  }  void color\_menu(int index)  {  present\_color = index;  if(in\_polygon>=0) polygons[in\_polygon].color = index;  }  void main\_menu(int index)  {  int i;  switch(index)  {  case(1): /\* create a new polygon \*/  {  moving = FALSE;  for(i=0; i<MAX\_POLYGONS; i++) if(polygons[i].used == FALSE) break;  if(i == MAX\_POLYGONS)  {  printf("exceeeded maximum number of polygons\n");  exit(0);  }  polygons[i].color = present\_color;  polygons[i].used = TRUE;  polygons[i].nvertices = 0;  in\_polygon = i;  picking = FALSE;  break;  }  case(2): /\* end polygon and find bounding box and center \*/  {  moving = FALSE;  if(in\_polygon>=0)  {  polygons[in\_polygon].xmax = polygons[in\_polygon].xmin = polygons[in\_polygon].x[0];  polygons[in\_polygon].ymax = polygons[in\_polygon].ymin = polygons[in\_polygon].y[0];  polygons[in\_polygon].xc = polygons[in\_polygon].x[0];  polygons[in\_polygon].yc = polygons[in\_polygon].y[0];  for(i=1;i<polygons[in\_polygon].nvertices;i++)  {  if(polygons[in\_polygon].x[i]<polygons[in\_polygon].xmin)  polygons[in\_polygon].xmin = polygons[in\_polygon].x[i];  else if(polygons[in\_polygon].x[i]>polygons[in\_polygon].xmax)  polygons[in\_polygon].xmax = polygons[in\_polygon].x[i];  if(polygons[in\_polygon].y[i]<polygons[in\_polygon].ymin)  polygons[in\_polygon].ymin = polygons[in\_polygon].y[i];  else if(polygons[in\_polygon].y[i]>polygons[in\_polygon].ymax)  polygons[in\_polygon].ymax = polygons[in\_polygon].y[i];  polygons[in\_polygon].xc += polygons[in\_polygon].x[i];  polygons[in\_polygon].yc += polygons[in\_polygon].y[i];  }  polygons[in\_polygon].xc = polygons[in\_polygon].xc/polygons[in\_polygon].nvertices;  polygons[in\_polygon].yc = polygons[in\_polygon].yc/polygons[in\_polygon].nvertices;  }  in\_polygon = -1;  glutPostRedisplay();  break;  }  case(3): /\* set picking mode \*/  {  picking = TRUE;  moving = FALSE;  break;  }  case(4): /\* set moving mode \*/  {  moving = TRUE;  break;  }  case(5): /\* exit \*/  {  exit(0);  break;  }  }  }  void myDisplay(){  int i, j;  glClear(GL\_COLOR\_BUFFER\_BIT);  for(i=0; i<MAX\_POLYGONS; i++){  if(polygons[i].used){  glColor3fv(colors[polygons[i].color]);  glBegin(GL\_POLYGON);  for(j=0; j<polygons[i].nvertices; j++) glVertex2i(polygons[i].x[j], polygons[i].y[j]);  glEnd();  }  }  glFlush();  }  int main(int argc, char\*\* argv){  int c\_menu;  glutInit(&argc,argv);  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize(500, 500);  glutCreateWindow("polygon modeler");  glutDisplayFunc(myDisplay);  myinit ();  c\_menu = glutCreateMenu(color\_menu);  glutAddMenuEntry("Black",0);  glutAddMenuEntry("Red",1);  glutAddMenuEntry("Green",2);  glutAddMenuEntry("Blue",3);  glutAddMenuEntry("Cyan",4);  glutAddMenuEntry("Magenta",5);  glutAddMenuEntry("Yellow",6);  glutAddMenuEntry("White",7);  glutCreateMenu(main\_menu);  glutAddMenuEntry("new polygon", 1);  glutAddMenuEntry("end polygon", 2);  glutAddMenuEntry("delete polygon", 3);  glutAddMenuEntry("move polygon", 4);  glutAddMenuEntry("quit",5);  glutAddSubMenu("Colors", c\_menu);  glutAttachMenu(GLUT\_MIDDLE\_BUTTON);  glutReshapeFunc (myReshape);  glutMouseFunc (myMouse);  glutMotionFunc(myMotion);  glutMainLoop();  }  執行結果:(多邊形繪畫工具) |

十、應用說明：

(1) 定義兩個全域變數去儲存多邊形的最大數量和每個多邊形的最大頂點數量

#define MAX\_POLYGONS 10

#define MAX\_VERTICES 10

定義一個結構叫做polygon，來存單個多邊形的資料

typedef struct polygon

{

int nvertices;

int x[MAX\_VERTICES];

int y[MAX\_VERTICES];

} polygon;

(2) 配置一個空間給多邊形

We cam allocate space for our polygons with the code

polygon polygons[MAX\_POLYGONS];

如果要指定前N個多邊形，我們顯示他

glClear(GL\_COLOR\_BUFFER\_BIT);

for(i=0;i<N;i++)

{

glBegin(GL\_POLYGON);

for(j=0;j<N;j++)

glVertex2i(polygons[i].x[j], polygons[i].y[j]);

glEnd();

}

(3) 我們想要新增多邊形並移除現有的多邊形，我們希望為每一個多邊形指定顏色。我們還想在顯示螢幕上交互式的移動一個多邊形

typedef struct polygon

{

int color;

bool used;

int xmin, xmax, ymin, ymax;

float xc, yc;

int nvertices;

int x[MAX\_VERTICES];

int y[MAX\_VERTICES];

} polygon;

(4)

* 如果bool used 是 true，我們可以透過將used設定為false來移除多邊形
* 我們還新增了一個中心位置在創建多邊形時做計算和幫助移動多邊形。
* 我們使用bounding-box strategy 去挑多邊形，儘管我們可以使用更多普遍的選擇模式策略。
* bounding-box的資訊包含xmin,xmax,ymin,ymax，這些參數在多邊形創建時設定。

(5) 為了顯示polygons中的所有多邊形，首先我們檢查每個多邊形是否被引用 ，如果是，我們就像之前那樣顯示該多邊形的顏色索引，為其分配顏色

for (i=0; i<MAX\_POLYGONS; i++)

{

if(polygons[i].used)

{

glColor3fv(colors[polygons[i].color]);

glBegin(GL\_POLYGON);

for(j=0;j<polygons[i].nvertices; j++)

glVertex2i(polygon[i].x[j], polygons[i].y[j]);

glEnd();

}}

(6) 我們可以設計允許我們以交互式方式操作的menus

c\_menu=glutCreateMenu(color\_menu);

glutAddMenuEntry("Black",0);

glutAddMenuEntry("Red",1);

glutAddMenuEntry("Green",2);

glutAddMenuEntry("Blue",3);

glutAddMenuEntry("Cyan",4);

glutAddMenuEntry("Magenta",5);

glutAddMenuEntry("Yellow",6);

glutAddMenuEntry("White",7);

(7) 要對存在的多邊形實施移動或刪除功能時，我們必須支持某種選擇動作，我們會添加一些自己的狀態變數。

glutCreateMenu(main\_menu);

glutAddMenuEntry("new polygon", 1);

glutAddMenuEntry("end polygon", 2);

glutAddMenuEntry("delete polygon", 3);

glutAddMenuEntry("move polygon", 4);

glutAddMenuEntry("quit",5);

glutAddSubMenu("Colors", c\_menu);

glutAttachMenu(GLUT\_MIDDLE\_BUTTON);

(8)使用present\_color來當作預設顏色索引。變數pciking和moving分別告訴我們是在選擇多邊形還是移動多邊形

交互的核心是通過 main\_menu\_callback

bool picking = FALSE; /\* true while picking \*/

bool moving = FALSE; /\* true while moving polygon \*/

int in\_polygon = -1; /\* not in any polygon \*/

int present\_color = 0; /\* default color \*/

(9)

void main\_menu(int index)

{

switch(index)

{

case(1):

{

/\* create a new polygon \*/

break;

{

/\* rest of cases\*/

}}

新增一個新的多邊形

case(1): /\* create a new polygon \*/

{ moving = FALSE;

for(i=0; i<MAX\_POLYGONS; i++)

if(polygons[i].used == FALSE) break;

if(i == MAX\_POLYGONS)

{ printf("exceeded maximum number of polygons\n"); exit(0); }

polygons[i].color = present\_color;

polygons[i].used = TRUE;

polygons[i].nvertices = 0;

in\_polygon = i;

picking = FALSE;

break;

}

(10) user使用滑鼠左鍵去輸入頂點

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN &&!picking&&!moving)

{

if(in\_polygon>=0)

{

if(polygons[in\_polygon].nvertices == MAX\_VERTICES)

{ printf("exceeds maximum number vertices\n"); exit(0);}

i = polygons[in\_polygon].nvertices;

polygons[in\_polygon].x[i] = x;

polygons[in\_polygon].y[i] = y;

polygons[in\_polygon].nvertices++;

}

}

(12) user返回menu輸入一個多邊形，結束一個多邊依可以計算出他的bounding box和中心

The user returns to the menu to enter a polygon. Ending a polygon allows the program to compute the bounding box for the polygon and its center.

case(2): /\* end polygon and find bounding box and center \*/

{

moving = FALSE;

if(in\_polygon>=0)

{

polygons[in\_polygon].xmax = polygons[in\_polygon].xmin = polygons[in\_polygon].x[0];

polygons[in\_polygon].ymax = polygons[in\_polygon].ymin = polygons[in\_polygon].y[0];

polygons[in\_polygon].xc = polygons[in\_polygon].x[0];

polygons[in\_polygon].yc = polygons[in\_polygon].y[0];

for(i=1;i<polygons[in\_polygon].nvertices;i++)

{

if(polygons[in\_polygon].x[i]<polygons[in\_polygon].xmin)

polygons[in\_polygon].xmin = polygons[in\_polygon].x[i];

else if(polygons[in\_polygon].x[i]>polygons[in\_polygon].xmax)

polygons[in\_polygon].xmax = polygons[in\_polygon].x[i];

if(polygons[in\_polygon].y[i]<polygons[in\_polygon].ymin)

polygons[in\_polygon].ymin = polygons[in\_polygon].y[i];

else if(polygons[in\_polygon].y[i]>polygons[in\_polygon].ymax)

polygons[in\_polygon].ymax = polygons[in\_polygon].y[i];

polygons[in\_polygon].xc += polygons[in\_polygon].x[i];

polygons[in\_polygon].yc += polygons[in\_polygon].y[i];

}

polygons[in\_polygon].xc = polygons[in\_polygon].xc/polygons[in\_polygon].nvertices;

polygons[in\_polygon].yc = polygons[in\_polygon].yc/polygons[in\_polygon].nvertices;

}

in\_polygon = -1;

glutPostRedisplay();

break; }

(13) 我們考慮的其他動作需要看游標在哪個多邊形(如果有的話)。我們可以用一個簡單的bounding-box 測略來寫一個function，回傳多邊形的標示符，如果游標不在任何多邊形的boundary內，回傳-1。

The other actions we consider require picking to find which polygon, if any, the mouse is over. We can use the a simple bounding-box strategy to write a function that returns the identifier of the polygon or –1 if the mouse is not inside the bounding box of any polygon.

int pick\_polygon(int x, int y)

{

int i;

for(i=0; i<MAX\_POLYGONS; i++)

{

if(polygons[i].used)

if((x>=polygons[i].xmin)&&(x<=polygons[i].xmax)&&(y>=polygons[i].ymin)

&&(y<polygons[i].ymax))

{

in\_polygon = i;

moving = TRUE;

return(i);

}

return(-1);

}

}

(14) 在menu回傳中設置狀態的flag後，刪除和移動多邊形分別在滑鼠和運動回傳中處理。

case(3): /\* set picking mode \*/

{

picking = TRUE;

moving = FALSE;

break;

}

case(4): /\* set moving mode \*/

{

moving = TRUE;

break;

}

(15) 刪除多邊形的游標回傳code，包含挑選操作

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN &&picking&&!moving)

{

picking = FALSE;

moving = FALSE;

j = pick\_polygon(x,y);

if(j >= 0)

{

polygons[j].used = FALSE;

in\_polygon = -1;

glutPostRedisplay();

}

}

(16) motion回傳確認狀態，然後移動被選擇的多邊形的中心以跟隨游標。

The motion callback checks the state and then moves the center of the selected polygon to follow the mouse.

void myMotion(int x, int y)

{

float dx, dy;

int i,j;

if(moving)

{

y = wh - y;

j = pick\_polygon(x, y);

if(j<0) return;

dx = x - polygons[j].xc;

dy = y - polygons[j].yc;

for(i = 0; i< polygons[j].nvertices; i++)

{

polygons[j].x[i] += dx;

polygons[j].y[i] += dy;

}

polygons[j].xc += dx;

polygons[j].yc += dy;

if(dx>0) polygons[j].xmax += dx;

else polygons[j].xmin += dx;

if(dy>0) polygons[j].ymax += dy;

else polygons[j].ymin += dy;

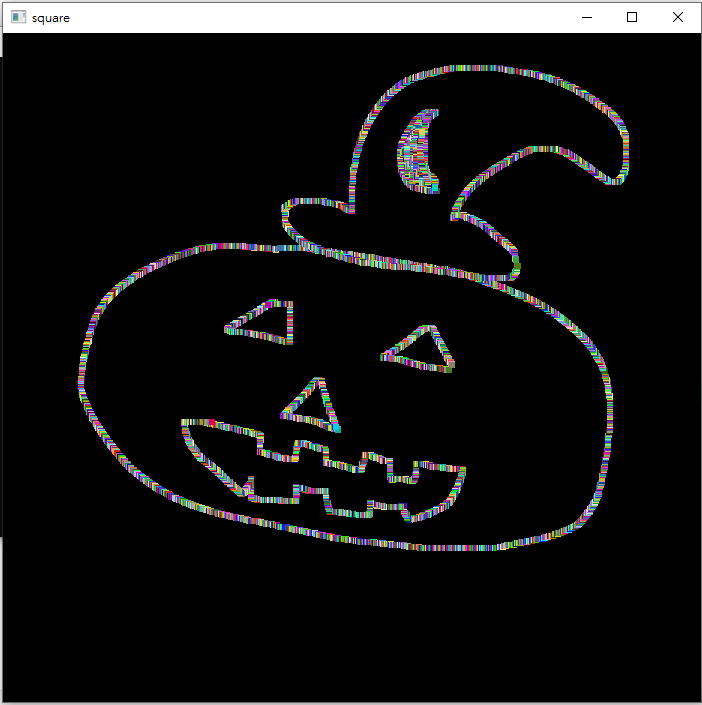
glutPostRedisplay();

}

}

十一、其他範例程式：

Ch\_03\_tm3\_src3.cpp(square.c): 以四邊形繪製出萬聖節南瓜，結果如下



十二、注意事項：

**使用定點裝置**

我們使用最常見的定點裝置-滑鼠來進行移動與點擊的操作，並用此函式

**void mymouse(int button, int state, int x, int y)**

來獲得滑鼠點擊的按鈕，是按下還是釋放與當前的位置。

**Window Event**

當改變視窗大小時須使用reshape功能來同時調整圖形的大小

**Void myReshape(GLsizei w, GLsizei h)**

**{**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**glViewport(0, 0, w, h);**

**ww=w;**

**wh=h;**

**}**

**Keyboard Event**

除了使用滑鼠上的按鍵外，也可以利用鍵盤的按鍵。

**glutKeyboardFunc(myKey);**

**void myKey(unsigned char key, int x, int y)**

**Window Management**

GLUT可以同時開啟多個視窗，利用給予不同的ID來管理。

**id=glutCreateWindow(“second window”);**

**glutSetWindow(id);**

十三、參考資料：

https://cs.appstate.edu/~rt/cs4465/notes/chap3.pdf