**計算機圖學單元介紹**

1. 英文主題：

Chapter 03: Input and Interaction

1. 中文主題：

單元03：輸入與互動

1. 組別：

第02組

1. 組員：

B0729007 官慶恩 B0729014 黃建銘 B0729015 楊金榮

B0729016 楊佶儫 B0729017 劉威廷 B0729024 謝瑞筑 B0829012 鄭家竣

1. 作業分工：

Input device + Building interactive models B0729007 官慶恩

Display list B0729024 謝瑞筑

Programming event + Interaction B0829012 鄭家竣

Picking B0729017 劉威廷

Simple CAD B0729015 楊金榮

PPT + word +Logic + Menu B0729016 楊佶儫

Animating + Design B0729014 黃建銘

1. 功能簡述：

We introduce the variety of devices available for interaction，like Physical device, Logical device.

We then consider client-server network and client-server graphics. Even driven input for the graphics program. We develop a polygon modeling program that demonstrates many of these important features of interactive graphics programming.

1. 程式碼：

延伸檔案1：Ch\_03\_tm2\_src1.cpp

|  |
| --- |
| #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();  }  /\* processHits prints out the contents of the  \* selection array  \*/  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++)  { /\* for each hit \*/  names = \*ptr;  ptr+=3;  for (j = 0; j < names; j++)  { /\* for each name \*/  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;  } |

Picking程式說明：

1. Picking是一種邏輯輸入操作，允許用戶識別顯示器上的對象，這裡使用兩個不同顏色的矩形示範

2. glRenderMode() 提供三種模式:

GL\_RENDER: 會進行柵格化，並產生會寫入畫面格緩衝區中的圖元片段。是一般模式，也 是預設模式。

GL\_SELECT:  不會產生任何圖元片段，也不會對畫面格緩衝區內容進行任何變更。 相反 地，當轉譯模式為此時，所要繪製的基本名稱記錄，會在選取的緩衝區中傳回。

GL\_FEEDBACK: 不會產生任何圖元片段，也不會對畫面格緩衝區內容進行任何變更。 相反 地，已繪製之頂點的座標和屬性會在意見反應緩衝區中傳回，該緩衝區必須建立。

3.柵格化是將向量圖形格式表示的圖像轉換成點陣圖以用於顯示器或者印表機輸出的過程

4. void glSelectBuffer(GLsizei n, GLunint \*buff):設定拾取緩衝區

5. void glInitNames():初始化stack的名字

6. void glPushName(Gluint name): 將名稱輸入剛被初始化的stack

7. void glPopName():顯示當前stack名稱

8. void glLoadName(Gluint name):替換stack頂部的值

9. gluPickMatrix (x, y, w, h, \*vp):將選取的物件轉換為一個projection matrix(w\*h)中心點為(x,y)

主要檔案： Ch\_03\_tm2\_src2.cpp

|  |
| --- |
| /\* 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()  {  /\* display all active polygons \*/  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();  } |

Polygon程式說明:

1. CAD是指運用電腦軟體製作並類比實物設計，展現新開發商品的外型、結構、色彩、質感等 特色的過程。

而OpenGL 的主要關注點，在於如何用CAD渲染出我們想要的幾何體。

2.需要建立一個data struct來容納polygon-model

**#define MAX\_POLYGONS 10 //polygon的最大數量**

**#define MAX\_VERTICES 10 //polygon的最大頂點數量**

**typedef struct polygon**

**3.**宣告polygon的空間，建立完成後，用glClear()顯示出前N個polygon

**polygon polygons[MAX\_POLYGONS]; //宣告MAX\_POLYGONS個polygon**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**4.** 添加新的polygon並刪除現有的polygon

**typedef struct polygon //必須擴增struct polygon**

**{**

**int color; //polygon color**

**bool used; //true : display , false : delete**

**int xmin, xmax, ymin, ymax; //使用邊界框策略來選擇多邊形**

**float xc, yc; // polygon中心位置，協助進行多邊形移動的計算**

**int nvertices;**

**int x[MAX\_VERTICES];**

**int y[MAX\_VERTICES];**

**} polygon;**

**5.** 每個polygon分配一種顏色

**for (i=0; i<MAX\_POLYGONS; i++){ //分配顏色**

**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.** 建立選單，以使用者交互的方式分配顏色、顯示或移動polygon。

**c\_menu=glutCreateMenu(color\_menu);**

**7.** 實現移動、刪除現有多邊形的功能

//添加所需的狀態變數。

**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 \*/**

**8.** **交互的核心是通過 main\_menu\_callback**

**void main\_menu(int index){**

**switch(index)**

**{**

**case(1):**

**{**

**/\* create a new polygon \*/**

**break;**

**{**

**/\* rest of cases\*/**

**}}**

**9.** int pick\_polygon(int x, int y)選擇polygon

用滑鼠點選某個已存在的polygon

我們可以使用簡單的邊界框策略來編寫一個function

該函數返回polygon的標識符

如果鼠標不在任何多邊形的邊界框內，則return -1

10. 在菜單回調中設置狀態標誌後，分別在鼠標和運動回調中處理刪除和移動多邊形

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

**{**

**picking = TRUE;**

**moving = FALSE;**

**break;**

**}**

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

**{**

**moving = TRUE;**

**break;**

**}**

11. 在選單中設置狀態標誌後，分別在鼠標和運動回調中處理刪除和移動多邊形

**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();**

**}**

**}**

12. 檢查狀態，然後移動選定多邊形的中心以跟隨鼠標。

**void myMotion(int x, int y)**

延伸檔案2：Ch\_03\_tm2\_src3.cpp

/\*

\* double.c

\* This program demonstrates double buffering for

\* flicker-free animation. The left and middle mouse

\* buttons start and stop the spinning motion of the square.

\*/

#include<stdlib.h>

#include<stdio.h>

#include<time.h>

#include <GL/glut.h>

#include <math.h>

#define DEGREES\_TO\_RADIANS 3.14159/180.0

static GLfloat spin = 0.0;

GLfloat x, y;

int singleb, doubleb;

void square()

{

glBegin(GL\_QUADS);

glVertex2f(x,y);

glVertex2f(-y,x);

glVertex2f(-x,-y);

glVertex2f(y,-x);

glEnd();

}

void displayd()

{

glClear (GL\_COLOR\_BUFFER\_BIT);

square();

glutSwapBuffers ();

}

void displays()

{

glClear (GL\_COLOR\_BUFFER\_BIT);

square();

glFlush();

}

void spinDisplay (void)

{

spin = spin + 2.0;

if (spin > 360.0) spin = spin - 360.0;

x= 25.0\*cos(DEGREES\_TO\_RADIANS \* spin);

y= 25.0\*sin(DEGREES\_TO\_RADIANS \* spin);

glutSetWindow(singleb);

glutPostRedisplay();

glutSetWindow(doubleb);

glutPostRedisplay();

}

void myinit ()

{

glClearColor (0.0, 0.0, 0.0, 1.0);

glColor3f (1.0, 1.0, 1.0);

glShadeModel (GL\_FLAT);

}

void mouse(int btn, int state, int x, int y)

{

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN) glutIdleFunc(spinDisplay);

if(btn==GLUT\_MIDDLE\_BUTTON && state==GLUT\_DOWN) glutIdleFunc(NULL);

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho (-50.0, 50.0, -50.0\*(GLfloat)h/(GLfloat)w,

50.0\*(GLfloat)h/(GLfloat)w, -1.0, 1.0);

else

glOrtho (-50.0\*(GLfloat)w/(GLfloat)h,

50.0\*(GLfloat)w/(GLfloat)h, -50.0, 50.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity ();

}

/\* Main Loop

\* Open window with initial window size, title bar,

\* RGBA display mode, and handle input events.

\*/

int main(int argc, char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

singleb=glutCreateWindow("single buffered");

myinit ();

glutDisplayFunc(displays);

glutReshapeFunc (myReshape);

glutIdleFunc (spinDisplay);

glutMouseFunc (mouse);

glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowPosition(500,0);

doubleb=glutCreateWindow("double buffered");

myinit ();

glutDisplayFunc(displayd);

glutReshapeFunc (myReshape);

glutIdleFunc (spinDisplay);

glutMouseFunc (mouse);

glutMainLoop();

}

程式說明:

1.將變更的角度回傳到主程式運行的函式

**glutIdleFunc(idle);**

2.變更角度的函式

**void idle()**

3.將滑鼠操作回傳給主程式的函式

**glutMouseFunc(mouse);**

4.滑鼠操作的函式

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

5.設置double buffering的函式 (以往是使用GLUT\_SINGLE)

**glutInitDisplayMode(GLUT\_RGB|GLUT\_DOUBLE)**

6.在呈現回傳的畫面時，第一步先是用glClear清除後緩衝器

**glClear()**

7. **最後一步則是使用此函式切換緩衝器**

**glutSwapBuffers();**

8. 繪製由後緩衝器來進行

**glutDrawBuffer(GL\_BACK)**

9.繪製由前、後緩衝器來進行

**glutDrawBuffer(GL\_FRONT\_AND\_BACK)**

10. Using a Timer

**glutTimerFunc(int display, void (\*timer\_func)(int), int value)**

11. 自行調整計時器頻率

**void myTimer(int v)**

延伸檔案3：Ch\_03\_tm2\_src4.cpp

#include<stdlib.h>

#include<stdio.h>

#include<time.h>

#include <GL/glut.h>

/\* globals \*/

GLsizei wh = 500, ww = 500; /\* initial window size \*/

GLfloat size = 3.0; /\* half side length of square \*/

void drawSquare(int x, int y)

{

y=wh-y;

glColor3ub( (char) rand()%256, (char) rand()%256, (char) rand()%256);

glBegin(GL\_POLYGON);

glVertex2f(x+size, y+size);

glVertex2f(x-size, y+size);

glVertex2f(x-size, y-size);

glVertex2f(x+size, y-size);

glEnd();

glFlush();

}

/\* rehaping routine called whenever window is resized

or moved \*/

void myReshape(GLsizei w, GLsizei h)

{

/\* adjust clipping box \*/

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

/\* adjust viewport and clear \*/

glViewport(0,0,w,h);

glClearColor (0.0, 0.0, 0.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

/\* set global size for use by drawing routine \*/

ww = w;

wh = h;

}

void myinit(void)

{

glViewport(0,0,ww,wh);

/\* Pick 2D clipping window to match size of screen window

This choice avoids having to scale object coordinates

each time window is resized \*/

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0.0, (GLdouble) ww , 0.0, (GLdouble) wh , -1.0, 1.0);

/\* set clear color to black and clear window \*/

glClearColor (0.0, 0.0, 0.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

/\* callback routine for reshape event \*/

glutReshapeFunc(myReshape);

}

void mouse(int btn, int state, int x, int y)

{

if(btn==GLUT\_RIGHT\_BUTTON && state==GLUT\_DOWN) exit(0);

}

/\* display callback required by GLUT 3.0 \*/

void display(void)

{}

int main(int argc, char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutCreateWindow("square");

myinit ();

glutReshapeFunc (myReshape);

glutMouseFunc (mouse);

glutMotionFunc(drawSquare);

glutDisplayFunc(display);

glutMainLoop();

}

程式說明:

1. 該程序說明了使用 glut 庫與窗口系統接口。該程序打開一個窗口，將其清除為黑色，然後每次單擊左鍵時在鼠標位置繪製一個框。當通過將新窗口清除為黑色來移動或調整窗口大小時，右鍵退出程序也會正確反應。

2. 按下滑鼠時，畫出正方形，參數裡面包含滑鼠目前狀態及座標

**glutMouseFunc(mymouse);**

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

3. drawSquare function (top-left and bottom-left problems)

**void drawSquare (int x, int y)**

4.Window Events，大多數的windows 系統都能讓使用者交互的調整視窗的大小，通常就是指使用指向裝置在視窗的邊緣做一個move event

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

**{**

**glMatrixMode(GL\_PROJECTION);//將當前矩陣指定為投影矩陣**

**glLoadIdentity();//重置為單位矩陣**

**gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);//用w、h來指定屏幕區域對應的模型範圍 座標**

**glMatrixMode(GL\_MODELVIEW);//將矩陣切換為模型視圖**

**glLoadIdentity();//重置為單位矩陣**

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

**ww=w;**

**wh=h; //更新圖案新的長與寬值**

**}**

5.負責螢幕顯示畫面上的其他設定

**glutDisplayFunc(myDisplay)**

6. 鍵盤:是當滑鼠在視窗內時，由按下或放開一個鍵盤或滑鼠的按鍵所引發的事件

**glutKeyboardFunc(myKey)** //呼叫鍵盤

**void myKey(unsigned char key, int x, int y)** //按下鍵盤及當前滑鼠的座標

**glutKeyboardFunc**是一個由按下一個鍵所引發的回呼，**glutKeyboardUpFunc**則是放開

**glutGetModifiers**是一個在GLUT函式庫內，能讓使用者使用修飾建(meta keys)來定義其他動作， 例如Control或Alt。

7. 當GLUT斷定視窗需要重新顯示時會引發的回呼，其中一種情況是在視窗剛打開時，另一種則是視窗剛結束一次縮放(resize event)時

**glutDisplayFunc (myDisplay)**

8. 窗口大小改變

**glutReshapeFunc(&reshape)**

9. 有無事件發生

**glutIdelFunc(&IdleFunc)**

10. 在GLUT’s main loop設置一個標示(flag)來紀錄是否需要重新顯示，使用這個函式將能避免掉額外或不必要的螢幕繪製(screen drawing)，GLUT在main loop 的結尾使用這個旗幟來決定顯示函式是否需要被執行。

**glutPostRedisplay();**

1. 應用說明：

**1. Ch\_03\_tm2\_src4.cpp:** draws a small colored square with each mouse click

**2. Ch\_03\_tm2\_src3.cpp:** displays a rotating cube with both single and double

buffering. Also illustrates use of two independent windows in GLUT.

Note: you may want to insert a timer to slow cube display.

**3. Ch\_03\_tm2\_src1.cpp:** Illustrates picking with overlapping red and blue squares

**4. Ch\_03\_tm2\_src2.cpp:** Polygon modeling program

十一、其他範例程式：

（無）

十二、注意事項：

( 無 )

十三、參考資料：

公開資料