**計算機圖學單元介紹**

1. 英文主題：

Chapter 05: Viewing

1. 中文主題：

單元05：視角變化

1. 組別：

第04組

1. 組員：

B0729061\_余承叡；B0729062\_李羽喬；B0729063\_黃紹禎；B0729065\_歐妍君；

B0744149\_白儀婕；B0829044\_葉丙俊；B0829051\_吳沛聲

1. 作業分工：

(詳見作業報告)

1. 功能簡述：

本單元並非去轉換幾何元件，而是切換觀察的視角

1. 主要程式碼：

相關檔案：Ch\_05\_tm4\_src1.cpp

|  |
| --- |
| /\* Rotating cube with viewer movement from Chapter 5 \*/  /\* Cube definition and display similar to rotating--cube program \*/  /\* We use the Lookat function in the display callback to point  the viewer, whose position can be altered by the x,X,y,Y,z, and Z keys.  The perspective view is set in the reshape callback \*/  #ifdef \_\_APPLE\_\_  #include <GLUT/glut.h>  #else  #include <GL/glut.h>  #endif  #include <iostream>  #include <cstdlib>  #include <cmath>  #include <ctime>  using namespace std;  GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},  {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},  {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};  GLfloat normals[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},  {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},  {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};  GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},  {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},  {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};  void polygon(int a, int b, int c , int d)  {  glBegin(GL\_POLYGON);  glColor3fv(colors[a]);  glNormal3fv(normals[a]);  glVertex3fv(vertices[a]);  glColor3fv(colors[b]);  glNormal3fv(normals[b]);  glVertex3fv(vertices[b]);  glColor3fv(colors[c]);  glNormal3fv(normals[c]);  glVertex3fv(vertices[c]);  glColor3fv(colors[d]);  glNormal3fv(normals[d]);  glVertex3fv(vertices[d]);  glEnd();  }  void colorcube()  {  polygon(0,3,2,1);  polygon(2,3,7,6);  polygon(0,4,7,3);  polygon(1,2,6,5);  polygon(4,5,6,7);  polygon(0,1,5,4);  }  static GLfloat theta[] = {0.0,0.0,0.0};  static GLint axis = 2;  static GLdouble viewer[]= {0.0, 0.0, 5.0}; // initial viewer location  void display(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  // Update viewer position in modelview matrix  glLoadIdentity();  gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  // rotate cube  glRotatef(theta[0], 1.0, 0.0, 0.0);  glRotatef(theta[1], 0.0, 1.0, 0.0);  glRotatef(theta[2], 0.0, 0.0, 1.0);  colorcube();  glFlush();  glutSwapBuffers();  }  void mouse(int btn, int state, int x, int y)  {  if(btn==GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) axis = 0;  if(btn==GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN) axis = 1;  if(btn==GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) axis = 2;  theta[axis] += 2.0;  if( theta[axis] > 360.0 ) theta[axis] -= 360.0;  display();  }  void keys(unsigned char key, int x, int y)  {  // Use x, X, y, Y, z, and Z keys to move viewer  if(key == 'x') viewer[0]-= 1.0;  if(key == 'X') viewer[0]+= 1.0;  if(key == 'y') viewer[1]-= 1.0;  if(key == 'Y') viewer[1]+= 1.0;  if(key == 'z') viewer[2]-= 1.0;  if(key == 'Z') viewer[2]+= 1.0;  display();  }  void myReshape(int w, int h)  {  glViewport(0, 0, w, h);  // Use a perspective view  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  if(w<=h) glFrustum(-2.0, 2.0, -2.0 \* (GLfloat) h/ (GLfloat) w,  2.0\* (GLfloat) h / (GLfloat) w, 2.0, 20.0);  else glFrustum(-2.0, 2.0, -2.0 \* (GLfloat) w/ (GLfloat) h,  2.0\* (GLfloat) w / (GLfloat) h, 2.0, 20.0);  // Or we can use gluPerspective  // gluPerspective(45.0, w/h, -10.0, 10.0);  glMatrixMode(GL\_MODELVIEW);  }  int  main(int argc, char \*\*argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  glutInitWindowSize(500, 500);  glutCreateWindow("colorcube");  glutReshapeFunc(myReshape);  glutDisplayFunc(display);  glutMouseFunc(mouse);  glutKeyboardFunc(keys);  glEnable(GL\_DEPTH\_TEST);  glutMainLoop();  } |

1. 程式說明：

程式目的：創造出立方體，觀察轉軸轉換的變化

程式說明：

1. 執行OpenGL，在標頭檔部份引用GL/glut.h, iostream, cstdlib, cmath與ctime。
2. 定義Cube每面的位置及顏色。
3. Void polygon(**int** a, **int** b, **int** c , **int** d)：定義多邊形。
4. void colorcube()：利用polygon(a, b, c, d);畫出立方體。
5. static GLint axis = 2;：定義初始旋轉軸。
6. static GLfloat theta[]：定義x, z軸初始夾角。
7. static GLdouble viewer[]：定義初始視角的位置。
8. void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);

該函數定義一個視圖矩陣，並與當前矩陣相乘。

第一組eyex, eyey, eyez：Camera在世界坐標的位置

第二組centerx,centery,centerz Camera鏡頭對準的物體在世界坐標的位置

第三組upx,upy,upz：Camera向上的方向在世界坐標中的方向

你把相機想像成為你自己的腦袋：

第一組數據就是腦袋的位置

第二組數據就是眼睛看的物體的位置

第三組就是頭頂朝向的方向（因為你可以歪著頭看同一個物體）。

1. glRotatef 函式：GlRotatef 函式會將目前的矩陣乘以旋轉矩陣。

參數：

角度：旋轉的角度（以度為單位）。

x：向量的 x 座標。

Y：向量的 y 座標。

Z：向量的 y 座標。

1. void keys(unsigned char key, int x, int y)：定義鍵盤事件，在按下x, y, z時會改變Viewers。
2. void mouse(int btn, int state, int x, int y)：定義滑鼠事件，在點下左、右和中間的滑鼠鍵會切換到對應旋轉軸。
3. glEnable(GL\_DEPTH\_TEST)：用來開啟更新深度緩衝區的功能，也就是，如果通過比較後深度值發生變化了，會進行更新深度緩衝區的操作。啟動它，OpenGL就可以跟踪再Z軸上的像素，這樣，它只會再那個像素前方沒有東西時，才會繪畫這個像素。在做繪畫3D時，這個功能最好啟動，視覺效果比較真實。
4. 延伸應用程式碼：

相關檔案：Ch\_05\_tm4\_src2.cpp

|  |
| --- |
| /\*  \* shadow.c  \* This program draws a polygon and its shadow.  \*/      #include<stdlib.h>  #include<stdio.h>  #include<time.h>  #include <GL/glut.h>    GLfloat theta=0.0;    void display(void)    {  /\* set clear color to white and clear window \*/    GLfloat light[3]={0.0, 10.0, 0.0};  GLfloat m[16];    double sin(), cos();    int i;  for(i=0;i<16;i++) m[i]=0.0;    light[0]=10.0\*sin((6.28/180.0)\*theta);  light[2]=10.0\*cos((6.28/180.0)\*theta);  m[0]=m[5]=m[10]=1.0;  m[7]=-1.0/light[1];  glClearColor (1.0, 1.0, 1.0, 0.0);  glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);    /\* set drawing/fill color to red \*/    glColor3f(1.0, 0.0, 0.0);    /\* set up standard orthogonal view with clipping \*/  /\* box as cube of side 2 centered at origin \*/    glMatrixMode (GL\_PROJECTION);  glLoadIdentity ();  glOrtho(-2.0, 2.0, -2.0, 2.0, -5.0, 5.0);    glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0);    /\* define unit square polygon \*/    glBegin(GL\_POLYGON);  glVertex3f(-0.5, 0.5, -0.5);  glVertex3f(-0.5, 0.5, 0.5);  glVertex3f(0.5, 0.5, 0.5);  glVertex3f(0.5, 0.5, -0.5);  glEnd();    glPushMatrix();  glTranslatef(light[0], light[1],light[2]);  glMultMatrixf(m);  glTranslatef(-light[0], -light[1],-light[2]);  glColor3f(0.0,0.0,0.0);  glBegin(GL\_POLYGON);  glVertex3f(-0.5, 0.5, -0.5);  glVertex3f(-0.5, 0.5, 0.5);  glVertex3f(0.5, 0.5, 0.5);  glVertex3f(0.5, 0.5, -0.5);  glEnd();  glPopMatrix();    /\* Swap buffers \*/    glutSwapBuffers();    }    void myidle()  {  theta+=2.0;  if(theta>360.0) theta-=360;  glutPostRedisplay();  }    int main(int argc, char\*\* argv)  {    /\* Initialize mode and open a window in upper left corner of screen \*/  /\* Window title is name of program (arg[0]) \*/    glutInit(&argc,argv);  glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGB |GLUT\_DEPTH);  glutInitWindowSize(500,500);  glutInitWindowPosition(0,0);  glutCreateWindow("shadow");  glutDisplayFunc(display);  glEnable(GL\_DEPTH\_TEST);  glutIdleFunc(myidle);  glutMainLoop();    } |

1. 應用說明：

程式目的：說明來自單個多邊形的投影陰影

主要變動或靈活運用之處：

a. set up standard orthogonal view with clipping / box as cube of side 2 centered at origin

(1) glMatrixMode (GL\_PROJECTION)：宣告接下來要進行投影

(2) glLoadIdentity ()：恢復初始座標（將先前有設定的角度與距離都清空）

(3) glOrtho(-2.0, 2.0, -2.0, 2.0, -5.0, 5.0)：以高寬4大小擷取影像

(4) glMatrixMode(GL\_MODELVIEW)：對模型視景矩陣操作，再畫其他物體

(5) gluLookAt(1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0)：建立衍生自眼睛點的視圖矩陣、指出場景中心的參考點（以不同視角去看這個物體）

b. define unit square polygon

(1) glBegin(GL\_POLYGON)、 glEnd()：畫多邊形

(2) glVertex3f(-0.5, 0.5, -0.5)、glVertex3f(-0.5, 0.5, 0.5)、glVertex3f(0.5, 0.5, 0.5)、glVertex3f(0.5, 0.5, -0.5)：它的點

(3) glPushMatrix()：記住自己現在的位置

(4) glMultMatrixf(m)：將矩陣全部乘一個數

(5) glPopMatrix()：回到先前記住的位置

c. void myidle()

(1) glutPostRedisplay()：標記當前窗口需要重新繪製，通過glutMainLoop下一次循環時，窗口顯示將被調回以重新顯示窗口的正常面板

十一、其他範例程式：

1. Ch\_05\_tm4\_src3.cpp

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <ctime>

using namespace std;

#define PI 3.14159265f

//色彩全局常量

GLfloat WHITE[] = { 1, 1, 1 }; //白色

GLfloat RED[] = { 1, 0, 0 }; //紅色

GLfloat GREEN[] = { 0, 1, 0 }; //綠色

GLfloat MAGENTA[] = { 1, 0, 1 }; //洋紅

//攝像機類：水平移動半徑為10，按上下鍵則垂直移動

class Camera {

public:

double theta; //確定x和z的位置

double y; //y位置

double dTheta; //角度增量

double dy; //上下y增量

public:

//類構造函數—默認初始化用法

Camera() : theta(0), y(3), dTheta(0.04), dy(0.2) {}

//類方法

double getX() { return 10 \* cos(theta); }

double getY() { return y; }

double getZ() { return 10 \* sin(theta); }

void moveRight() { theta += dTheta; }

void moveLeft() { theta -= dTheta; }

void moveUp() { y += dy; }

void moveDown() { if (y > dy) y -= dy; }

};

//球類定義

//半徑、顏色、最大高度

//x和z固定

//用lame bouncing algorithm

//每幀上下移動0.05單位

class Ball {

//類的屬性

double radius;

GLfloat\* color;

double maximumHeight;

double x;

double y;

double z;

int direction; //方向

public:

//構造函數

Ball(double r, GLfloat\* c, double h, double x, double z) :

radius(r), color(c), maximumHeight(h), direction(-1),

y(h), x(x), z(z) {

}

//更新和繪制方法

void update() {

//正反運動

y += direction \* 0.05;

if (y > maximumHeight) {

y = maximumHeight;

direction = -1;

}

else if (y < radius) {

y = radius;

direction = 1;

}

glPushMatrix();

//單獨設置每個球的材質參數

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, color);

glTranslated(x, y, z);

//創建球

glutSolidSphere(radius, 30, 30);

glPopMatrix();//

}

};

//棋盤格：沿x和z平面分布

//點光源位置設置為(4, 3, 7).

class Checkerboard {

int displayListId;

int width;

int depth;

public:

//構造函數

Checkerboard(int width, int depth) : width(width), depth(depth) {}

//中心X

double centerx() { return width / 2; }

//中心Y

double centerz() { return depth / 2; }

//創建方法

void create() {

displayListId = glGenLists(1); //每個顯示列表對應1個編號——關聯起來

//新建操作表

glNewList(displayListId, GL\_COMPILE); //把下述命令裝入顯示列表但不顯示

//光源位置參數

GLfloat lightPosition[] = { 4, 3, 7, 1 };

//設置光源位置

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

//開始繪制四邊形

glBegin(GL\_QUADS);

//法向量方向

glNormal3d(0, 1, 0);

for (int x = 0; x < width - 1; x++) {

for (int z = 0; z < depth - 1; z++) {

//設置每個格子的材質屬性

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE,

(x + z) % 2 == 0 ? RED : WHITE);

//四邊形的4個點坐標

glVertex3d(x, 0, z);

glVertex3d(x + 1, 0, z);

glVertex3d(x + 1, 0, z + 1);

glVertex3d(x, 0, z + 1);

}

}

glEnd();

glEndList();

}

//按列表編號繪制棋盤格

void draw() {

glCallList(displayListId);

}

};

//全局變量：棋盤格、相機和3個球的數組

Checkerboard checkerboard(8, 8);

Camera camera;

//創建3個小球的數組

Ball balls[] = {

Ball(1, GREEN, 7, 6, 1),

Ball(1.5, MAGENTA, 6, 3, 4),

Ball(0.4, WHITE, 5, 1, 7)

};

//自定義初始化方法

void init() {

//允許深度測試

glEnable(GL\_DEPTH\_TEST);

//設置散射和鏡像反射為白光

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, WHITE);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, WHITE);

//設置前表面的高光鏡像反射為白光

glMaterialfv(GL\_FRONT, GL\_SPECULAR, WHITE);

//設置前表面散射光反光系數

glMaterialf(GL\_FRONT, GL\_SHININESS, 30);

//允許燈光

glEnable(GL\_LIGHTING);

//打開0#燈

glEnable(GL\_LIGHT0);

//創建棋盤格

checkerboard.create();

}

//自定義繪制函數

//通過類繪制各對象，display函數代碼得以簡化

void display() {

//清除前一幀繪圖結果

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

//裝入單位陣

glLoadIdentity();

//設置視角——攝像機參數

gluLookAt(camera.getX(), camera.getY(), camera.getZ(), //攝像機位置

checkerboard.centerx(), 0.0, checkerboard.centerz(), //焦點坐標

0.0, 1.0, 0.0); //攝像機機頂方向矢量

//繪制棋盤

checkerboard.draw();

//繪制小球

for (int i = 0; i < sizeof balls / sizeof(Ball); i++) {

//更新位置並繪圖

balls[i].update();

}

//glFlush();

glutSwapBuffers();

}

//窗口調整大小時調用的函數

void reshape(GLint w, GLint h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(40.0, GLfloat(w) / GLfloat(h), 1.0, 150.0);

glMatrixMode(GL\_MODELVIEW);

}

//自定義計時器函數

void timer(int v) {

//當計時器喚醒時所調用的函數

glutPostRedisplay();

//設置下一次計時器的參數

glutTimerFunc(1000 / 60, timer, v);//timer:函數名

}

//鍵盤處理函數

void onKey(int key, int, int) {

//按鍵：上下左右

switch (key) {

case GLUT\_KEY\_LEFT: camera.moveLeft(); break;

case GLUT\_KEY\_RIGHT: camera.moveRight(); break;

case GLUT\_KEY\_UP: camera.moveUp(); break;

case GLUT\_KEY\_DOWN: camera.moveDown(); break;

}

glutPostRedisplay();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowPosition(80, 80);

glutInitWindowSize(800, 600);

glutCreateWindow("跳躍的球");

glutDisplayFunc(display); //設置顯示函數

glutReshapeFunc(reshape); //設置窗口調整大小的函數

glutSpecialFunc(onKey); //設置按鍵處理函數

glutTimerFunc(100, timer, 0); //設置計時器函數--每100ms被調用1次

init();//自定義初始化函數

glutMainLoop();//進入opengl主循環

return 0;

}

OpenGL 函數:

1) glPushMatrix:記住物件自己現在的位置。

2) glPopMatrix:回到之前記住的位置

3) glMaterialfv:設置物體材質。

4) glTranslated:會將目前的矩陣與轉譯矩陣相乘，產生由 (x、 y、 z) 指定

的平移。

5) glutSolidSphere:創建球體。

6) glGenLists:會產生一組連續的空白顯示清單。

7) GlNewList 和 glEndList 函數會建立或取代顯示清單。

8) glLightfv::設置光源位置。

9) glBegin、glend: GlBegin 和 glend 函式會分隔基本或類似基本類型群組的

頂點。

10) glNormal3d:設定目前的一般向量。 11) glVertex3d:指定頂點。

12) glCallList:會執行顯示清單。

13) glClearColor:設置背景顏色。

14) glClear:清除前一幀繪圖結果

15) glLoadIdentity:會將目前的矩陣取代為識別矩陣。

16) gluLookAt:設置視角，建立衍生自眼睛點的視圖矩陣、指出場景中心的參

考點，以及向上向量。

17) glFlush():會清空所有網路緩衝區和圖形加速器本身，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。

18) glutSwapBuffers:用於實現雙緩衝技術的一個功能是重要的交換功能。

19) glViewport:指定從標準化裝置座標到視窗座標之 x 和 y 的仿射轉換。 讓

(xnd ， ynd ) 為標準化裝置座標。

20) glMatrixMode:設定目前的矩陣模式。

21) glutPostRedisplay:標記當前視窗需要重新繪製。 通過 glutMainLoop 下一

次循環時，視窗顯示將被回調以重新顯示視窗的正常面板。

22) glutInitDisplayMode:在創建窗口的時候，指定其顯示模式的類型。

23) glutInitWindowPosition:确定窗口位置。

24) glutInitWindowSize:确定窗口大小。

25) glutCreateWindow:設定窗口名稱。

26) glutDisplayFunc:設置顯示函數。

27) glutReshapeFunc:設置窗口調整大小的函數。

28) glutKeyboardFunc:設置鍵盤處理函數。

29) glutMouseFunc:設置滑鼠處理函數。

30) glutTimerFunc:設置 Timer 函數。

31) glutMainLoop:這裡讓整個繪圖循環進行。

程式設計邏輯:

用 class 建構三種類別:Ball, Camera, Flat，透過全域變數建立 Ball, Flat, Camera

等物件。其中 Ball 創建球的物件，他的建構函式有球的初始 x, y, z、顏色、半徑、 上下方向和可到達最大高度，其他函式還有 update( )，用來更新現在球的位置， moveX( )用來變動 x 方向速度，Pause( )用來暫停球的移動(將速度設為 0，即暫停)， Start( )用來啟動球的移動，還有 ballState( )用來回傳現在球的狀態(暫停 / 啟動)。 Camera 設定現在的視角，其函式都是根據方向鍵，操控使用者觀看視角。Flat 用來 建立 Ball 彈跳的平面，求將會在此範圍進行移動。在 display 函式會在每次 glutMainLoop 時根據移動的結果重新繪製 Flat 與 Ball，並調整 Camera 視角。 Reshape( )於視窗大小變動時，調整對應視角。onKey( )用來控制上下左右，操作視 角。Keyboard( )設定 ESC 可以結束視窗，空白鍵進行 x 方向移動。Mouse( )控制滑 鼠左右鍵，選擇開始或是暫停動畫。在 init( )設置背景顏色、光線，並且創建平面， 最後在 main 設定視窗大小及位置，並呼叫 display, reshape, onKey, keyboard, mouse 等函式。

十二、注意事項：

在OpenGL中沒有單獨的camera (view) matrix。因此，為了模擬camera或者view的變換，其中的場景(3D物體和光照)必需通過和view相反的方向變換。也就是說，OpenGL總是將camera定義在(0,0,0)點，并且強制在eye space坐標系的-Z軸方向，而且不能變換。

十三、參考資料：

1. <https://documen.site/download/interactive-computer-graphics-chapter-5_pdf>
2. <https://blog.csdn.net/ivan_ljf/article/details/8764737>
3. <https://www.cnblogs.com/arxive/p/7002319.html>