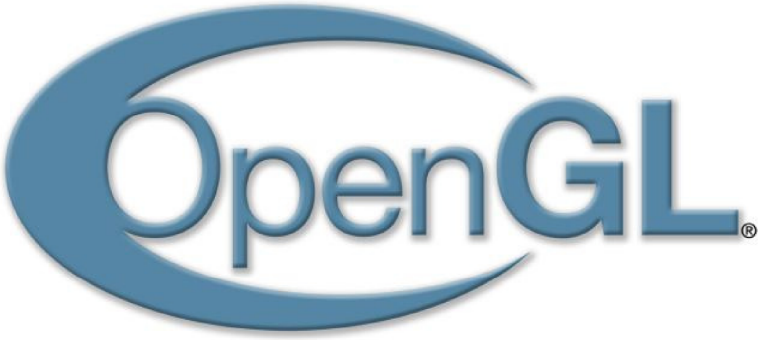
ĐỒ HỌA MÁY TÍNH

Bài thực hành 2

OPENGL

Version 1.0



Lê Viết Tuấn

TP. HCM, 2018

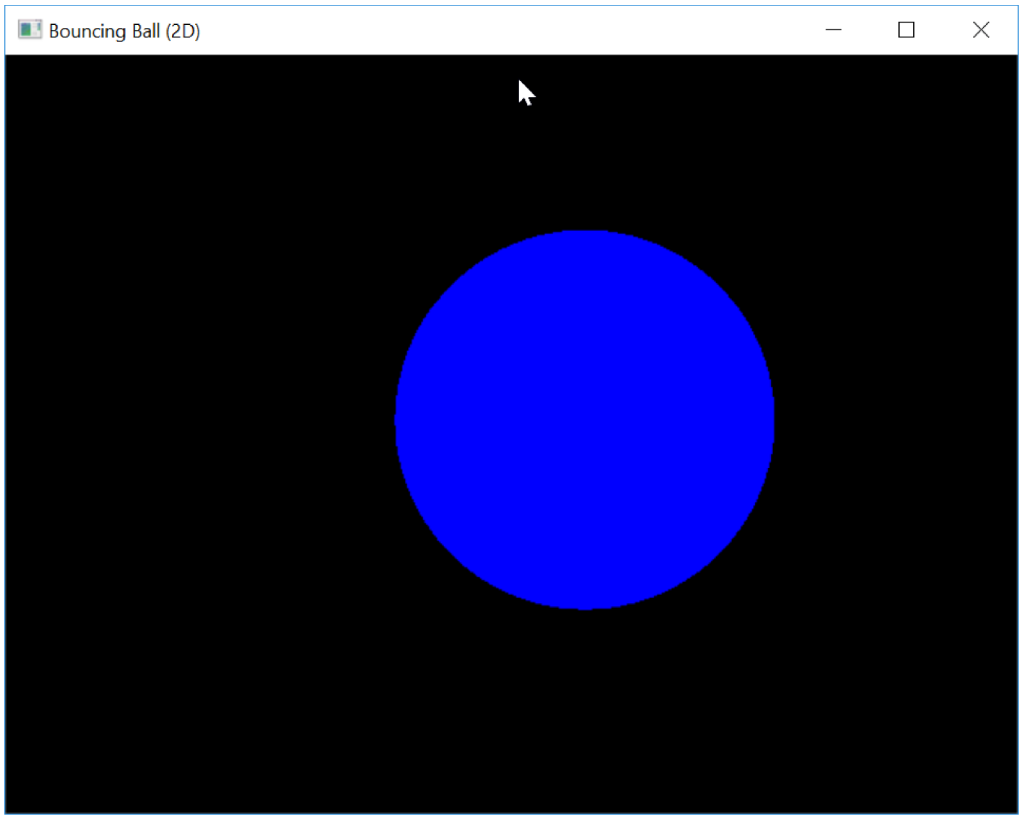
MỤC LỤC

1. Mục tiêu

- Sử dụng OpengL

2. Bài tập hướng dẫn

Bài 1. Vẽ quả bóng nảy ngược khi chạm vào các cạnh của màn hình (Bouncing Ball)



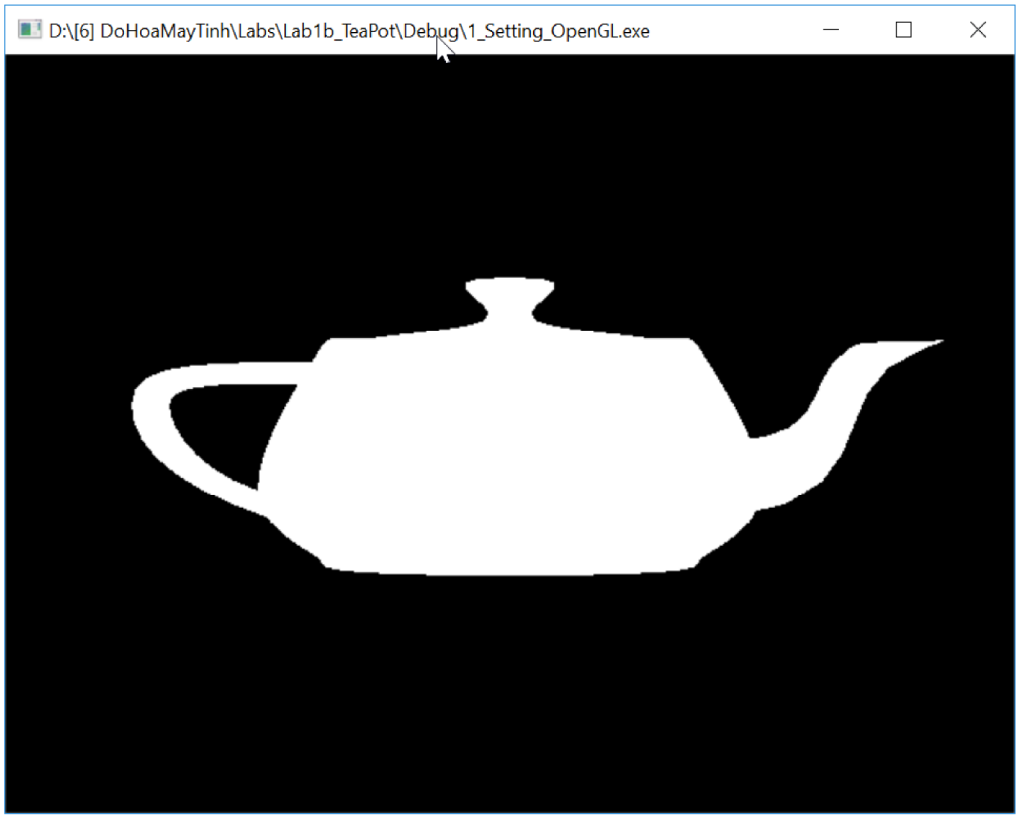
Hướng dẫn:

```
#include "Dependencies\glew\glew.h"
#include "Dependencies\freeglut\freeglut.h"

#include <Math.h>     // Needed for sin, cos
#define PI 3.14159265f

// Global variables
char title[] = "Bouncing Ball (2D)";  // Windowed mode's title
int windowWidth = 640;     // Windowed mode's width
int windowHeight = 480;     // Windowed mode's height
int windowPosX = 50;      // Windowed mode's top-left corner x
int windowPosY = 50;      // Windowed mode's top-left corner y

GLfloat ballRadius = 0.5f;   // Radius of the bouncing ball
GLfloat ballX = 0.0f;         // Ball's center (x, y) position
```

```
GLfloat ballY = 0.0f;
GLfloat ballXMax, ballXMin, ballYMax, ballYMin; // Ball's center (x, y) bounds
GLfloat xSpeed = 0.02f;      // Ball's speed in x and y directions
GLfloat ySpeed = 0.007f;
int refreshMillis = 30;      // Refresh period in milliseconds

                                                // Projection clipping area
GLdouble clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop;

/* Initialize OpenGL Graphics */
void initGL() {
      glClearColor(0.0, 0.0, 0.0, 1.0); // Set background (clear) color to black
}

/* Callback handler for window re-paint event */
void display() {
      glClear(GL_COLOR_BUFFER_BIT);  // Clear the color buffer
      glMatrixMode(GL_MODELVIEW);    // To operate on the model-view matrix
      glLoadIdentity();              // Reset model-view matrix

      glTranslatef(ballX, ballY, 0.0f);  // Translate to (xPos, yPos)
                                                                  // Use triangular
segments to form a circle
      glBegin(GL_TRIANGLE_FAN);
      glColor3f(0.0f, 0.0f, 1.0f);  // Blue
      glVertex2f(0.0f, 0.0f);       // Center of circle
      int numSegments = 100;
      GLfloat angle;
      for (int i = 0; i <= numSegments; i++) { // Last vertex same as first vertex
            angle = i * 2.0f * PI / numSegments;  // 360 deg for all segments
            glVertex2f(cos(angle) * ballRadius, sin(angle) * ballRadius);
      }
      glEnd();

      glutSwapBuffers();  // Swap front and back buffers (of double buffered mode)

                                          //  Animation  Control  -  compute  the
location for the next refresh
      ballX += xSpeed;
      ballY += ySpeed;
      // Check if the ball exceeds the edges
      if (ballX > ballXMax) {
            ballX = ballXMax;
            xSpeed = -xSpeed;
      }
      else if (ballX < ballXMin) {
            ballX = ballXMin;
            xSpeed = -xSpeed;
      }
      if (ballY > ballYMax) {
            ballY = ballYMax;
            ySpeed = -ySpeed;
      }
      else if (ballY < ballYMin) {
            ballY = ballYMin;
            ySpeed = -ySpeed;
      }
}

/* Call back when the windows is re-sized */
void reshape(GLsizei width, GLsizei height) {
      // Compute aspect ratio of the new window
```

```
	if (height == 0) height = 1;                // To prevent divide by 0
	GLfloat aspect = (GLfloat)width / (GLfloat)height;

	// Set the viewport to cover the new window
	glViewport(0, 0, width, height);

	// Set the aspect ratio of the clipping area to match the viewport
	glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
	glLoadIdentity();             // Reset the projection matrix
	if (width >= height) {
		clipAreaXLeft = -1.0 * aspect;
		clipAreaXRight = 1.0 * aspect;
		clipAreaYBottom = -1.0;
		clipAreaYTop = 1.0;
	}
	else {
		clipAreaXLeft = -1.0;
		clipAreaXRight = 1.0;
		clipAreaYBottom = -1.0 / aspect;
		clipAreaYTop = 1.0 / aspect;
	}
	gluOrtho2D(clipAreaXLeft, clipAreaXRight, clipAreaYBottom, clipAreaYTop);
	ballXMin = clipAreaXLeft + ballRadius;
	ballXMax = clipAreaXRight - ballRadius;
	ballYMin = clipAreaYBottom + ballRadius;
	ballYMax = clipAreaYTop - ballRadius;
}

/* Called back when the timer expired */
void Timer(int value) {
	glutPostRedisplay();    // Post a paint request to activate display()
	glutTimerFunc(refreshMillis,  Timer,  0);  //  subsequent  timer  call  at
milliseconds
}

/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
	glutInit(&argc, argv);            // Initialize GLUT
	glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
	glutInitWindowSize(windowWidth, windowHeight);  // Initial window width and
height
	glutInitWindowPosition(windowPosX, windowPosY); // Initial window top-left
corner (x, y)
	glutCreateWindow(title);      // Create window with given title
	glutDisplayFunc(display);      // Register callback handler for window re-
paint
	glutReshapeFunc(reshape);      // Register callback handler for window re-
shape
	glutTimerFunc(0, Timer, 0);   // First timer call immediately
	initGL();                     // Our own OpenGL initialization
	glutMainLoop();               // Enter event-processing loop
	return 0;

}
```

Bài 2. Vẽ bình trà đơn giản

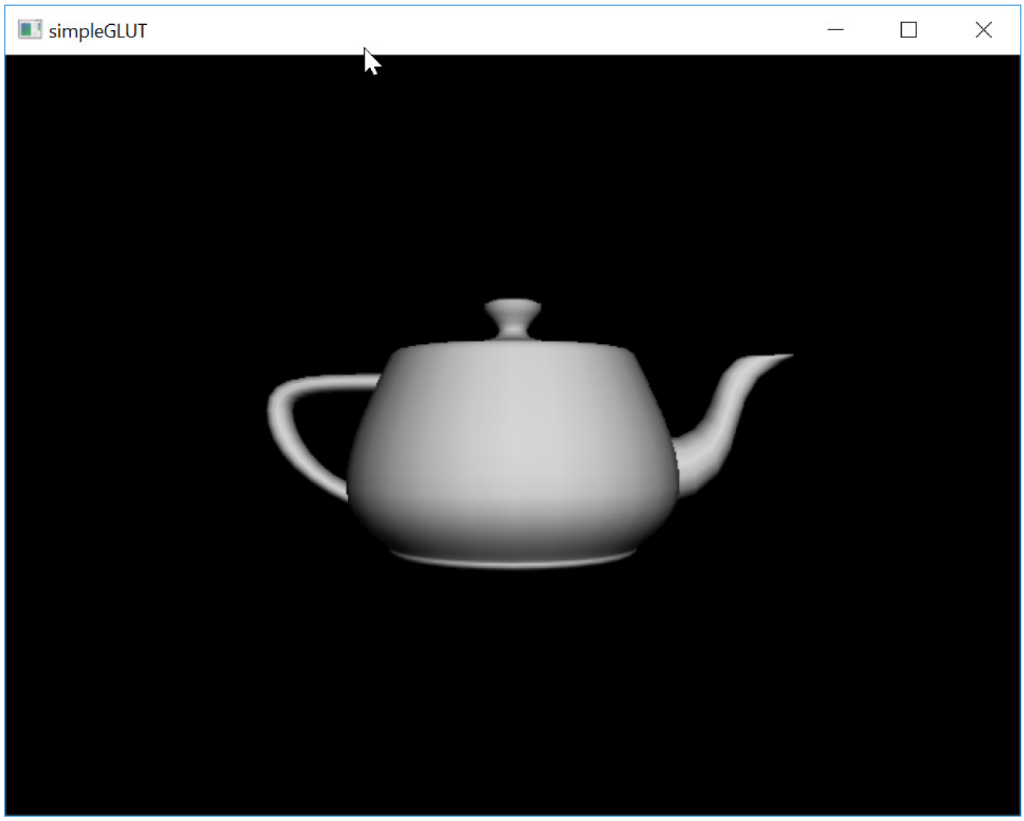


Hướng dẫn:

```
#include "Dependencies\glew\glew.h"
#include "Dependencies\freeglut\freeglut.h"

const int screenWidth = 640;
const int screenHeight = 480;

void init(void)
{
	glClearColor(0.0, 0.0, 0.0, 0.0);//clear black
	glShadeModel(GL_SMOOTH);
}

void display()
{
	glClear(GL_COLOR_BUFFER_BIT); //Clears the color buffer
	glColor3f(1.0, 1.0, 1.0);

	glutSolidTeapot(0.5);

	glFlush();
}

void reshape(int w, int h)
{
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

```
int main(int argc, char **argv)
{
	glutInit(&argc, argv); // Khởi tạo glut
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA); // Khởi tạo chế độ vẽ
single buffer và hệ màu RGB
	glutInitWindowSize(screenWidth, screenHeight); //optional
	glutInitWindowPosition(100, 100);//optional
	glutCreateWindow(argv[0]);
	init();
	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```

Bài 3. Vẽ bình trà cho phép quay, phóng to thu nhỏ, …



Hướng dẫn:

```
#include <math.h>
#include <iostream>
#include "Dependencies\glew\glew.h"

#include "Dependencies\freeglut\freeglut.h"
```

```
void initGlut(int argc, char **argv);
void displayFunc(void);
void idleFunc(void);
void reshapeFunc(int width, int height);
void mouseFunc(int button, int state, int x, int y);
void mouseMotionFunc(int x, int y);
void keyboardFunc(unsigned char key, int x, int y);
void specialFunc(int key, int x, int y);
//------------------------------------------------------------------------
----

// other [OpenGL] functions
void countFrames(void);
void renderBitmapString(float x, float y, float z, void *font, char *string);


//------------------------------------------------------------------------
----

bool bUsePredefinedCamera = true;

bool bFullsreen = false;
int nWindowID;

// camera attributes
float viewerPosition[3] = { 0.0, 0.0, -50.0 };
float viewerDirection[3] = { 0.0, 0.0, 0.0 };
float viewerUp[3] = { 0.0, 1.0, 0.0 };

// rotation values for the navigation
float navigationRotation[3] = { 0.0, 0.0, 0.0 };

//------------------------------------------------------------------------
----

// parameters for the framecounter
char pixelstring[30];
int cframe = 0;
int time = 0;
int timebase = 0;

//------------------------------------------------------------------------
----

// parameters for the navigation

// position of the mouse when pressed
int mousePressedX = 0, mousePressedY = 0;
float lastXOffset = 0.0, lastYOffset = 0.0, lastZOffset = 0.0;
// mouse button states
int    leftMouseButtonActive    =    0,    middleMouseButtonActive    =    0,
rightMouseButtonActive = 0;
// modifier state
int shiftActive = 0, altActive = 0, ctrlActive = 0;



//------------------------------------------------------------------------
----
bool init = false;
//------------------------------------------------------------------------
----
```

```
//------------------------------------------------------------------------
----

void displayFunc(void) {


	// clear the buffers
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glEnable(GL_DEPTH_TEST);
	glDepthFunc(GL_LESS);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(50.0, 1.33, 1.0, 100000.0);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	//	gluLookAt(	viewerPosition[0],		viewerPosition[1],
viewerPosition[2],
	//				viewerDirection[0],	viewerDirection[1],
viewerDirection[2],
	//				viewerUp[0], viewerUp[1], viewerUp[2]);


	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);

	GLfloat	lightpos[4] = { 5.0, 15.0, 10.0, 1.0 };
	glLightfv(GL_LIGHT0, GL_POSITION, lightpos);

	glTranslatef(viewerPosition[0],		viewerPosition[1],
viewerPosition[2]);
	// add navigation rotation

	glRotatef(navigationRotation[0], 1.0f, 0.0f, 0.0f);
	glRotatef(navigationRotation[1], 0.0f, 1.0f, 0.0f);

	glutSolidTeapot(10.0);

	countFrames();

	glutSwapBuffers();
}

//------------------------------------------------------------------------
----

void initGlut(int argc, char **argv) {

	// GLUT Window Initialization:
	glutInit(&argc, argv);
	glutInitWindowSize(640, 480);
	glutInitWindowPosition(100, 100);
	glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
	nWindowID = glutCreateWindow("simpleGLUT");

	// Register callbacks:
	glutDisplayFunc(displayFunc);
```

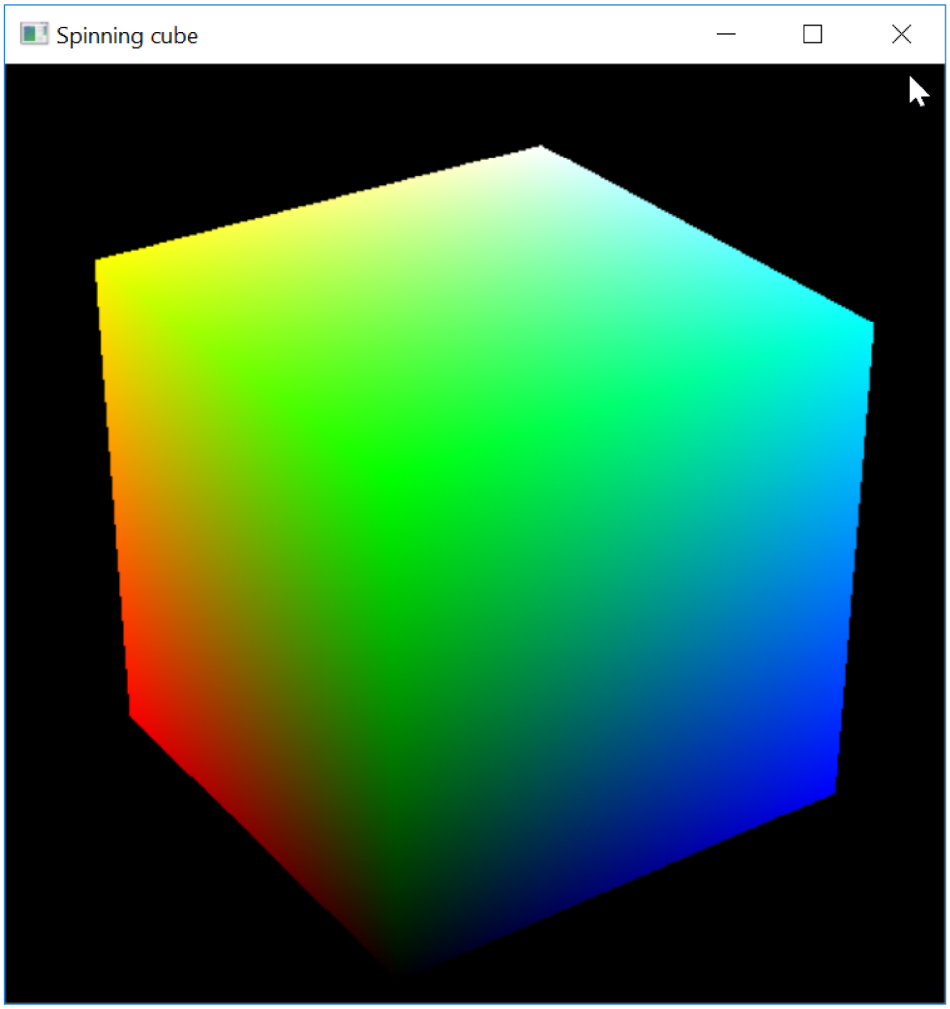
```
	glutReshapeFunc(reshapeFunc);
	glutKeyboardFunc(keyboardFunc);
	glutSpecialFunc(specialFunc);
	glutMouseFunc(mouseFunc);
	glutMotionFunc(mouseMotionFunc);
	glutIdleFunc(idleFunc);
}



//------------------------------------------------------------------------
----

void idleFunc(void) {
	glutPostRedisplay();
}

//------------------------------------------------------------------------
----

void reshapeFunc(int width, int height) {

	glViewport(0, 0, width, height);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(75.0, 1.33, 1.0, 1000.0);

	glMatrixMode(GL_MODELVIEW);

}

//------------------------------------------------------------------------
----


// mouse callback
void mouseFunc(int button, int state, int x, int y) {

	// get the modifiers
	switch (glutGetModifiers()) {

	case GLUT_ACTIVE_SHIFT:
		shiftActive = 1;
		break;
	case GLUT_ACTIVE_ALT:
		altActive = 1;
		break;
	case GLUT_ACTIVE_CTRL:
		ctrlActive = 1;
		break;
	default:
		shiftActive = 0;
		altActive = 0;
		ctrlActive = 0;
		break;
	}

	// get the mouse buttons
	if (button == GLUT_LEFT_BUTTON)
		if (state == GLUT_DOWN) {
			leftMouseButtonActive += 1;
```

```
		}
		else
			leftMouseButtonActive -= 1;
	else if (button == GLUT_MIDDLE_BUTTON)
		if (state == GLUT_DOWN) {
			middleMouseButtonActive += 1;
			lastXOffset = 0.0;
			lastYOffset = 0.0;
		}
		else
			middleMouseButtonActive -= 1;
	else if (button == GLUT_RIGHT_BUTTON)
		if (state == GLUT_DOWN) {
			rightMouseButtonActive += 1;
			lastZOffset = 0.0;
		}
		else
			rightMouseButtonActive -= 1;

	//	if (altActive) {
	mousePressedX = x;
	mousePressedY = y;
	//	}
}

//------------------------------------------------------------------------
----

void mouseMotionFunc(int x, int y) {

	float xOffset = 0.0, yOffset = 0.0, zOffset = 0.0;

	// navigation
	//	if (altActive) {

	// rotatation
	if (leftMouseButtonActive) {

		navigationRotation[0] += ((mousePressedY - y) * 180.0f) /
200.0f;
		navigationRotation[1] += ((mousePressedX - x) * 180.0f) /
200.0f;

		mousePressedY = y;
		mousePressedX = x;

	}
	// panning
	else if (middleMouseButtonActive) {

		xOffset = (mousePressedX + x);
		if (!lastXOffset == 0.0) {
			viewerPosition[0] -= (xOffset - lastXOffset) / 8.0;
			viewerDirection[0] -= (xOffset - lastXOffset) / 8.0;
		}
		lastXOffset = xOffset;

		yOffset = (mousePressedY + y);
		if (!lastYOffset == 0.0) {
			viewerPosition[1] += (yOffset - lastYOffset) / 8.0;
			viewerDirection[1] += (yOffset - lastYOffset) / 8.0;
		}
```

```
                lastYOffset = yOffset;

        }
        // depth movement
        else if (rightMouseButtonActive) {
                zOffset = (mousePressedX + x);
                if (!lastZOffset == 0.0) {
                        viewerPosition[2] -= (zOffset - lastZOffset) / 5.0;
                        viewerDirection[2] -= (zOffset - lastZOffset) / 5.0;
                }
                lastZOffset = zOffset;
        }
        //      }

}

//------------------------------------------------------------------------
----

void keyboardFunc(unsigned char key, int x, int y) {

        switch (key) {

                // ----------------------------------------

#ifdef WIN32
                // exit on escape
        case '\033':
                exit(0);
                break;
#endif

                // ----------------------------------------

                // switch to fullscreen
        case 'f':

                bFullsreen = !bFullsreen;
                if (bFullsreen)
                        glutFullScreen();
                else {
                        glutSetWindow(nWindowID);
                        glutPositionWindow(100, 100);
                        glutReshapeWindow(640, 480);
                }
                break;

                // ----------------------------------------
        }
}

//------------------------------------------------------------------------
----

void specialFunc(int key, int x, int y) {
        //printf("key pressed: %d\n", key);
}

//------------------------------------------------------------------------
----

void countFrames(void) {
```

```
	time = glutGet(GLUT_ELAPSED_TIME);
	cframe++;
	if (time - timebase > 50) {
		printf(pixelstring, "fps: %4.2f", cframe*1000.0 / (time - 
timebase));
		timebase = time;
		cframe = 0;
		// Draw status text and uni-logo:
	}
	glPushMatrix();
	glLoadIdentity();
	glDisable(GL_LIGHTING);
	glColor4f(1.0, 1.0, 1.0, 1.0);
	glMatrixMode(GL_PROJECTION);
	glPushMatrix();
	glLoadIdentity();
	gluOrtho2D(0, 200, 0, 200);
	glMatrixMode(GL_MODELVIEW);

	// render the string
	renderBitmapString(5, 5, 0.0, GLUT_BITMAP_HELVETICA_10, pixelstring);

	glPopMatrix();
	glMatrixMode(GL_PROJECTION);
	glPopMatrix();
	glMatrixMode(GL_MODELVIEW);
}

//------------------------------------------------------------------------
----

void renderBitmapString(float x, float y, float z, void *font, char *string)
{
	char *c;
	glRasterPos3f(x, y, z);
	for (c = string; *c != '\0'; c++) {
		glutBitmapCharacter(font, *c);
	}
}

//------------------------------------------------------------------------
----


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

	printf("simpleGLUT\n\tGordon Wetzstein [gordon.wetzstein@medien.uni-
weimar.de]\n\n");
	printf("keys:\n\tf\t- toggle fullscreen\n\tesc\t- exit\n\n");
	printf("mouse:\n\tleft button\t- rotation\n\tmiddle button\t-
panning\n\tright button\t- zoom in and out\n");

	initGlut(argc, argv);
	glutMainLoop();


}
```

Bài 4. Vẽ khối lập phương tự quay



Hướng dẫn:

```
#include "Dependencies\glew\glew.h"
#include "Dependencies\freeglut\freeglut.h"

void          DisplayFunc(void)
{
	static float alpha = 0;

	/* Clear the buffer, clear the matrix */
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();

	/* A step backward, then spin the cube */
	glTranslatef(0, 0, -10);
	glRotatef(30, 1, 0, 0);
	glRotatef(alpha, 0, 1, 0);
```

```
	/* We tell we want to draw quads */
	glBegin(GL_QUADS);

	/* Every four calls to glVertex, a quad is drawn */
	glColor3f(0, 0, 0); glVertex3f(-1, -1, -1);
	glColor3f(0, 0, 1); glVertex3f(-1, -1, 1);
	glColor3f(0, 1, 1); glVertex3f(-1, 1, 1);
	glColor3f(0, 1, 0); glVertex3f(-1, 1, -1);

	glColor3f(1, 0, 0); glVertex3f(1, -1, -1);
	glColor3f(1, 0, 1); glVertex3f(1, -1, 1);
	glColor3f(1, 1, 1); glVertex3f(1, 1, 1);
	glColor3f(1, 1, 0); glVertex3f(1, 1, -1);

	glColor3f(0, 0, 0); glVertex3f(-1, -1, -1);
	glColor3f(0, 0, 1); glVertex3f(-1, -1, 1);
	glColor3f(1, 0, 1); glVertex3f(1, -1, 1);
	glColor3f(1, 0, 0); glVertex3f(1, -1, -1);

	glColor3f(0, 1, 0); glVertex3f(-1, 1, -1);
	glColor3f(0, 1, 1); glVertex3f(-1, 1, 1);
	glColor3f(1, 1, 1); glVertex3f(1, 1, 1);
	glColor3f(1, 1, 0); glVertex3f(1, 1, -1);

	glColor3f(0, 0, 0); glVertex3f(-1, -1, -1);
	glColor3f(0, 1, 0); glVertex3f(-1, 1, -1);
	glColor3f(1, 1, 0); glVertex3f(1, 1, -1);
	glColor3f(1, 0, 0); glVertex3f(1, -1, -1);

	glColor3f(0, 0, 1); glVertex3f(-1, -1, 1);
	glColor3f(0, 1, 1); glVertex3f(-1, 1, 1);
	glColor3f(1, 1, 1); glVertex3f(1, 1, 1);
	glColor3f(1, 0, 1); glVertex3f(1, -1, 1);

	/* No more quads */
	glEnd();

	/* Rotate a bit more */
	alpha = alpha + 0.1;

	/* End */
	glFlush();
	glutSwapBuffers();

	/* Update again and again */
	glutPostRedisplay();
}

/*
** Function called when the window is created or resized
*/
void		ReshapeFunc(int width, int height)
{
	glMatrixMode(GL_PROJECTION);

	glLoadIdentity();
	gluPerspective(20, width / (float)height, 5, 15);
	glViewport(0, 0, width, height);

	glMatrixMode(GL_MODELVIEW);
	glutPostRedisplay();
```

```
}

/*
** Function called when a key is hit
*/
void          KeyboardFunc(unsigned char key, int x, int y)
{
       int foo;

       foo = x + y; /* Has no effect: just to avoid a warning */
       if ('q' == key || 'Q' == key || 27 == key)
              exit(0);
}


int           main(int argc, char **argv)
{
       /* Creation of the window */
       glutInit(&argc, argv);
       glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
       glutInitWindowSize(500, 500);
       glutCreateWindow("Spinning cube");

       /* OpenGL settings */
       glClearColor(0, 0, 0, 0);
       glEnable(GL_DEPTH_TEST);

       /* Declaration of the callbacks */
       glutDisplayFunc(&DisplayFunc);
       glutReshapeFunc(&ReshapeFunc);
       glutKeyboardFunc(&KeyboardFunc);

       /* Loop */
       glutMainLoop();

       /* Never reached */
       return 0;

}
```

3. Tài liệu tham khảo

[1] http://www.glprogramming.com/red/