**CÁC PHÉP BIẾN ĐỔI ĐỒ HỌA 3D**

**(VIEW TRANSFORMATION)**

**Họ và tên Sinh viên:**Phan Trần Nhật Hạ  

**Mã Sinh viên:**102210159     **Nhóm:** 21Nh12

[**1.**     **Chương trình *lab02viewmatrix.cpp xem modelview matrix***](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215930)

[**2.**     **Chương trình *lab02transform.cpp*biểu diễn các đối tượng từ model spaces sang world space**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215931)

[**3.**     **Chương trình *lab02rotation01.cpp* xoay hình vuông một góc 20o. Tâm xoay là gốc tọa độ**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215932)

[**4.**     **Chương trình *lab02rotation02.cpp* xoay hình vuông một góc 20o. Tâm xoay là điểm (x, y)**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215933)

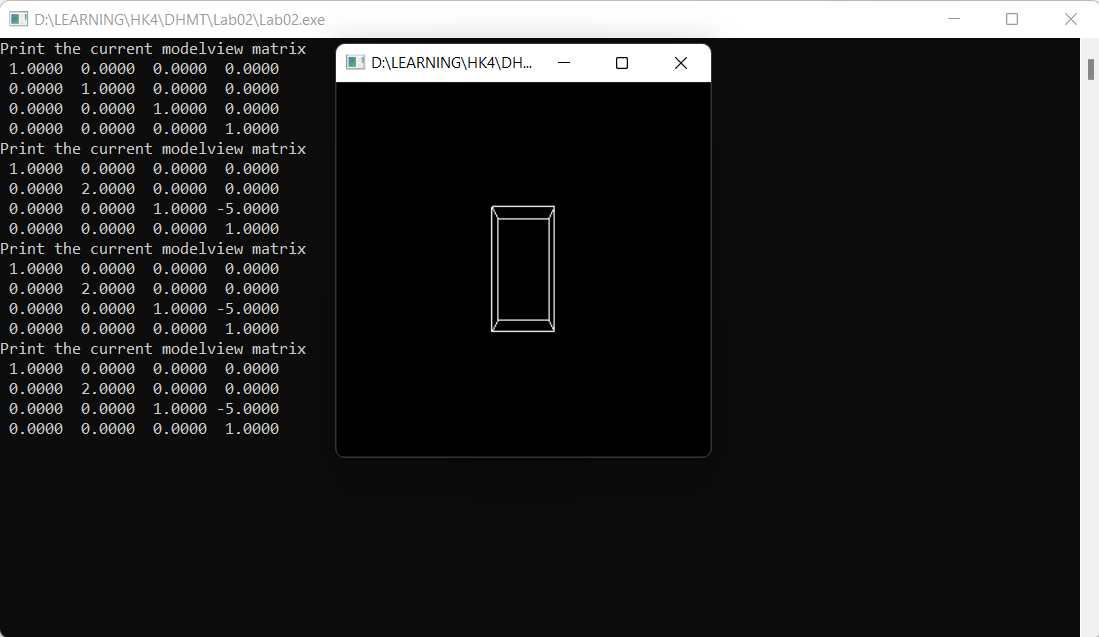
[**5.**     **Chương trình *lab02rotation03.cpp* vẽ hình chữ nhật quay quanh tâm**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215934)

[**6.**     **Chương trình *lab02affine.cpp* minh họa các phép biến đổi affine**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215935)

[**7.**     **Chương trình *lab02earth.cpp* vẽ hình trái đất quay xung quanh mặt trời**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215936)

[**8.**     **Bài tập**](http://itfdut.ddns.net/mod/resource/tknetlab/LabDHMT-CacPhepBienDoiAffine3D.htm#_Toc98215937)

**1.    Chương trình *lab02viewmatrix.cpp xem modelview matrix***

****

**2.    Chương trình *lab02transform.cpp*biểu diễn các đối tượng từ model spaces sang world space**

#include <windows.h>  // for MS Windows

#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/\* Initialize OpenGL Graphics \*/

**void initGL() {**

   // Set "clearing" or background color

   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque

}

**void display() {**

   glClear(GL\_COLOR\_BUFFER\_BIT);    // Clear the color buffer

   glMatrixMode(GL\_MODELVIEW);      // To operate on Model-View matrix

   glLoadIdentity();                // Reset the model-view matrix

   glTranslatef(-0.5f, 0.4f, 0.0f); // Translate left and up

   glBegin(GL\_QUADS);               // Each set of 4 vertices form a quad

      glColor3f(1.0f, 0.0f, 0.0f);  // Red

      glVertex2f(-0.3f, -0.3f);     // Define vertices in counter-clockwise (CCW) order

      glVertex2f( 0.3f, -0.3f);     //  so that the normal (front-face) is facing you

      glVertex2f( 0.3f,  0.3f);

      glVertex2f(-0.3f,  0.3f);

   glEnd();

   glTranslatef(0.1f, -0.7f, 0.0f); // Translate right and down

   glBegin(GL\_QUADS);               // Each set of 4 vertices form a quad

      glColor3f(0.0f, 1.0f, 0.0f); // Green

      glVertex2f(-0.3f, -0.3f);

      glVertex2f( 0.3f, -0.3f);

      glVertex2f( 0.3f,  0.3f);

      glVertex2f(-0.3f,  0.3f);

   glEnd();

   glTranslatef(-0.3f, -0.2f, 0.0f); // Translate left and down

   glBegin(GL\_QUADS);                // Each set of 4 vertices form a quad

      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray

      glVertex2f(-0.2f, -0.2f);

      glColor3f(1.0f, 1.0f, 1.0f); // White

      glVertex2f( 0.2f, -0.2f);

      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray

      glVertex2f( 0.2f,  0.2f);

      glColor3f(1.0f, 1.0f, 1.0f); // White

      glVertex2f(-0.2f,  0.2f);

   glEnd();

   glTranslatef(1.1f, 0.2f, 0.0f); // Translate right and up

   glBegin(GL\_TRIANGLES);          // Each set of 3 vertices form a triangle

      glColor3f(0.0f, 0.0f, 1.0f); // Blue

      glVertex2f(-0.3f, -0.2f);

      glVertex2f( 0.3f, -0.2f);

      glVertex2f( 0.0f,  0.3f);

   glEnd();

   glTranslatef(0.2f, -0.3f, 0.0f);     // Translate right and down

   glRotatef(180.0f, 0.0f, 0.0f, 1.0f); // Rotate 180 degree

      glBegin(GL\_TRIANGLES);               // Each set of 3 vertices form a triangle

      glColor3f(1.0f, 0.0f, 0.0f); // Red

      glVertex2f(-0.3f, -0.2f);

      glColor3f(0.0f, 1.0f, 0.0f); // Green

      glVertex2f( 0.3f, -0.2f);

      glColor3f(0.0f, 0.0f, 1.0f); // Blue

      glVertex2f( 0.0f,  0.3f);

   glEnd();

   glRotatef(-180.0f, 0.0f, 0.0f, 1.0f); // Undo previous rotate

   glTranslatef(-0.1f, 1.0f, 0.0f);      // Translate right and down

   glBegin(GL\_POLYGON);                  // The vertices form one closed polygon

      glColor3f(1.0f, 1.0f, 0.0f); // Yellow

      glVertex2f(-0.1f, -0.2f);

      glVertex2f( 0.1f, -0.2f);

      glVertex2f( 0.2f,  0.0f);

      glVertex2f( 0.1f,  0.2f);

      glVertex2f(-0.1f,  0.2f);

      glVertex2f(-0.2f,  0.0f);

   glEnd();

   glFlush();   // Render now

}

/\* Handler for window re-size event. Called back when the window first appears and

   whenever the window is re-sized with its new width and height \*/

**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();

   if (width >= height) {

     // aspect >= 1, set the height from -1 to 1, with larger width

      gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

   } else {

      // aspect < 1, set the width to -1 to 1, with larger height

     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

   }

} //reshape

/\* Main function: GLUT runs as a console application starting at main() \*/

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

**{**

   glutInit(&argc, argv);          // Initialize GLUT

   glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-square

   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner

   glutCreateWindow("Model Transform");  // Create window with the given title

   glutDisplayFunc(display);       // Register callback handler for window re-paint event

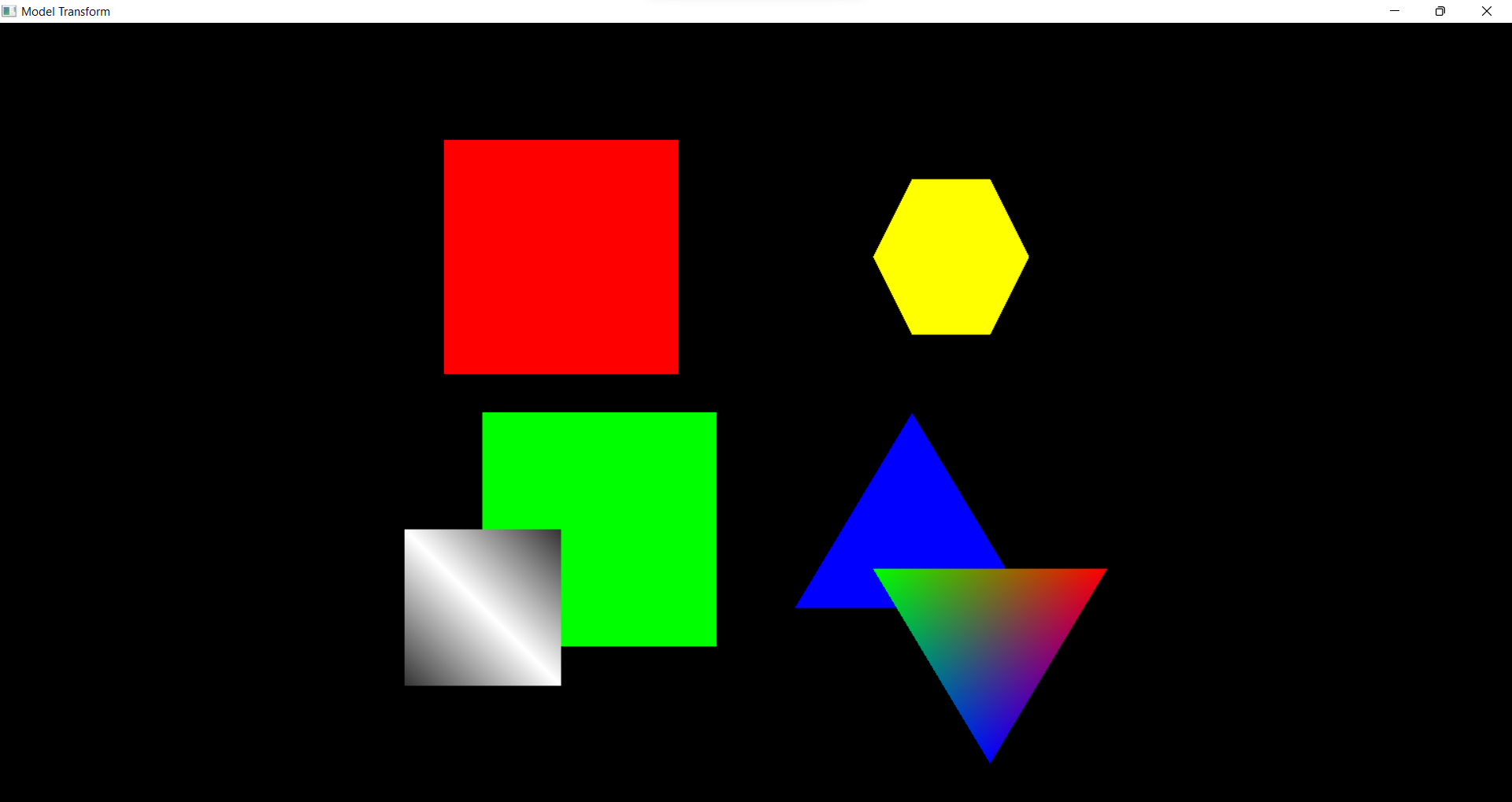
   glutReshapeFunc(reshape);       // Register callback handler for window re-size event

   initGL();                       // Our own OpenGL initialization

   glutMainLoop();                 // Enter the infinite event-processing loop

   return 0;

}//main



**3.    Chương trình *lab02rotation01.cpp* xoay hình vuông một góc 20o. Tâm xoay là gốc tọa độ**

#include <gl/glut.h>

#include <gl/gl.h>

void myInit(void)

{

glClearColor(0.7f, 0.7f, 0.7f, 0.0f); //to nen xam

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 640.0, 0.0, 480.0);

glMatrixMode(GL\_MODELVIEW);

}

void myDisplay(void)

{

int x=320, y=240;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0f, 0.0f, 0.0f);

glRectf(x-100, y-100, x+100, y+100);

glColor3f(1.0f, 1.0f, 0.0f);

glPushMatrix(); //save the current matrix

glRotatef(20.0, 0, 0, 1); //Rotate by 20 degrees CCW

glRectf(x-100, y-100, x+100, y+100); //draw the rectangle

glPopMatrix(); //restore the old matrix

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(640, 480);

glutInitWindowPosition(100, 150);

glutCreateWindow("Vi du 4.1");

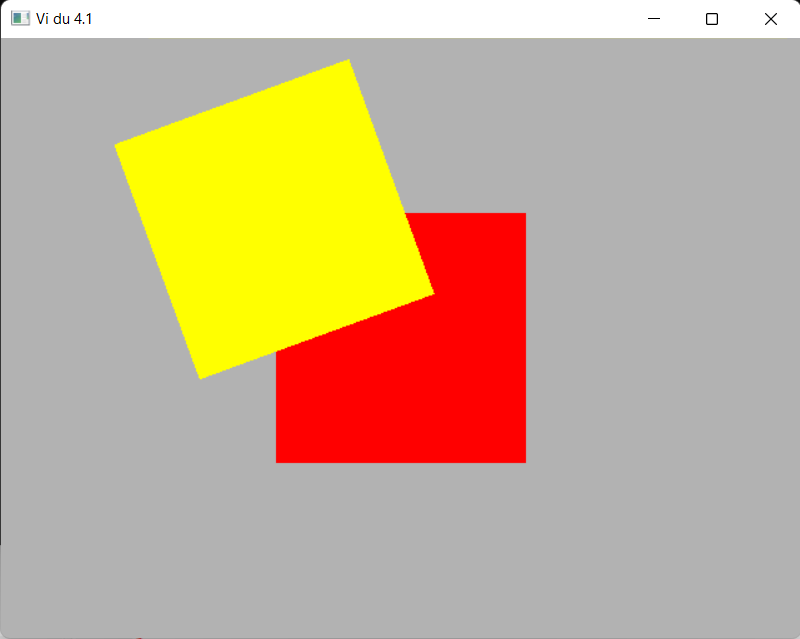
glutDisplayFunc(myDisplay);

myInit();

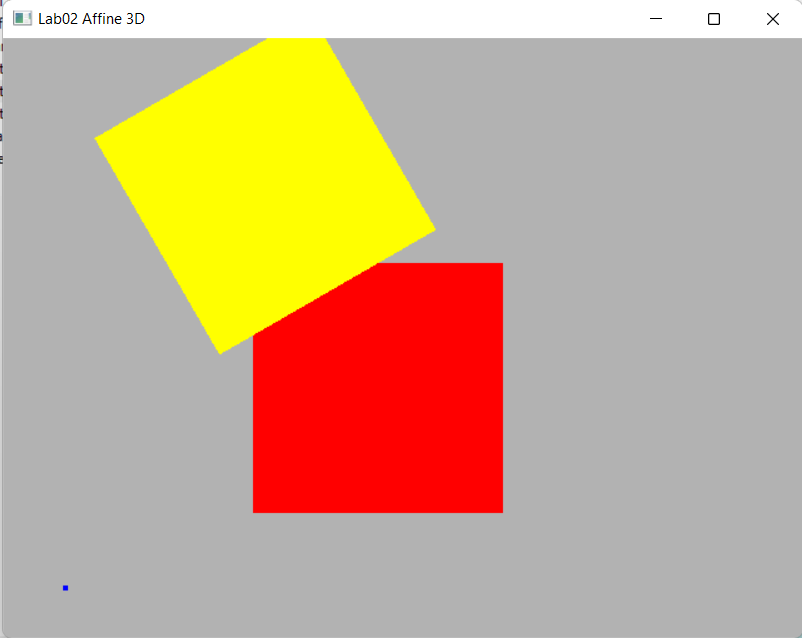
glutMainLoop();

return 0;

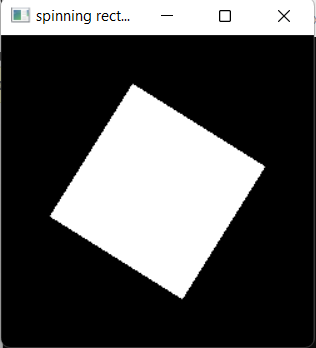
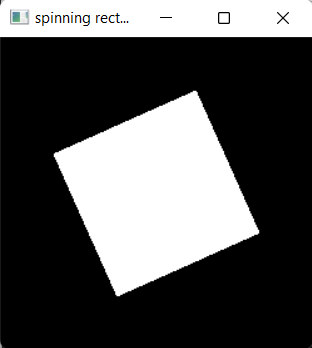
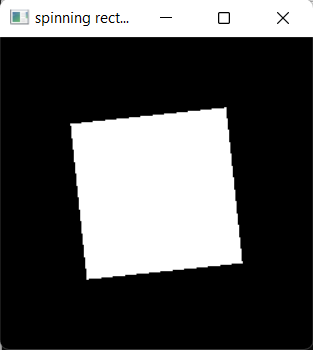
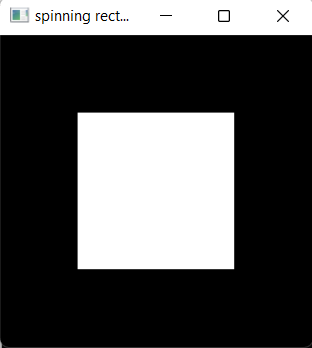
}



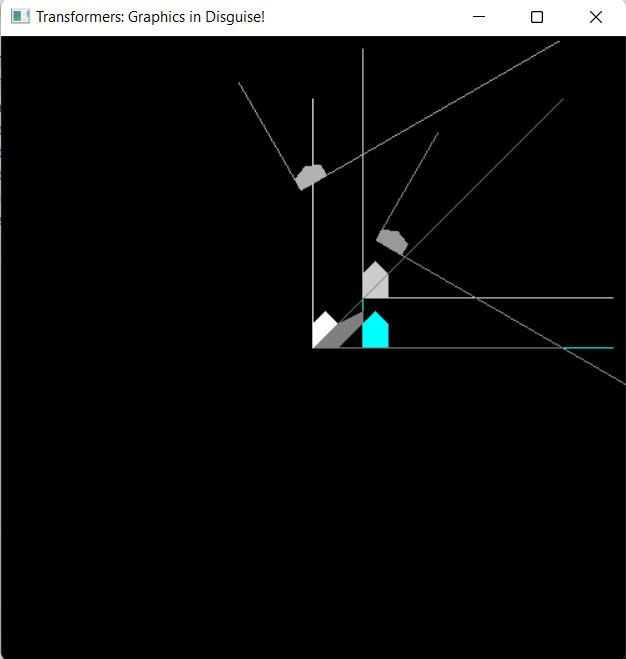
**4.    Chương trình *lab02rotation02.cpp* xoay hình vuông một góc 20o. Tâm xoay là điểm (x, y)**



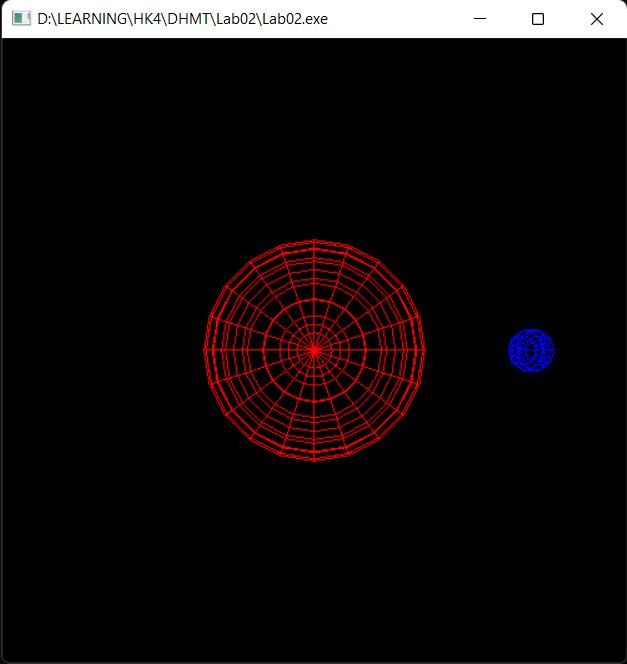
**5.    Chương trình *lab02rotation03.cpp* vẽ hình chữ nhật quay quanh tâm**

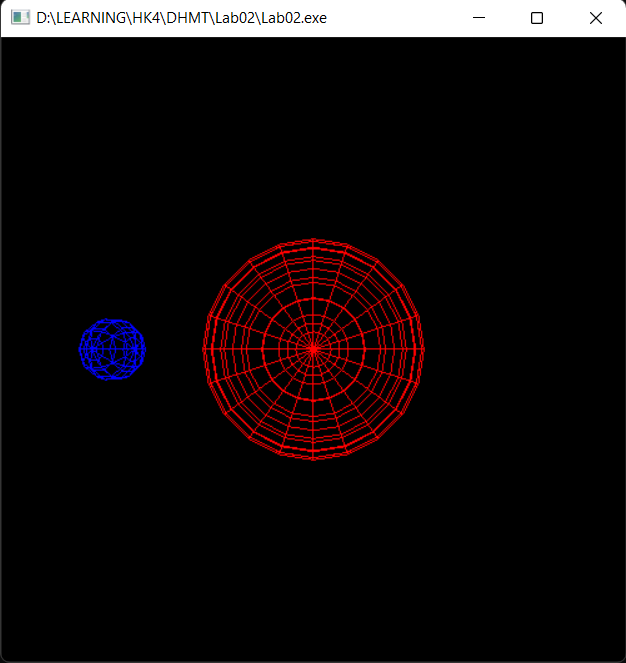
****

**6.****Chương trình *lab02affine.cpp* minh họa các phép biến đổi affine**

****

**7.    Chương trình *lab02earth.cpp* vẽ hình trái đất quay xung quanh mặt trời**





**8.    Bài tập**

Cho đường thẳng PQ có P(1, 2, 0), Q(7, 14). Lập trình biểu diễn các phép biến đổi đường thẳng PQ thông qua điều khiển bàn phím như sau:

1) Nhấn phím t, T: Dịch chuyển PG một đoạn theo khoảng cách (5, 0, 0)

2) Nhấn phím r, R: Quay PG một góc quanh trục Oy một góc 30o

3) Nhấn phím p, P: Lấy đối xứng của PG qua trục Ox

4) Nhấn phím f, F: Lấy đối xứng của PG qua trục MN

5) Nhấn phím h, H: Quay PG quanh trục MN một góc 30o

- Code:

#include <iostream>

#include <stdio.h>

#include <GL\glut.h>

#include <cmath>

const float PI = 3.14159265358979323846;

/\*

x = at + x0

y = bt + y0

z = ct + z0

\*/

class Line

{

public:

GLdouble a, b, c;

GLdouble x0, y0, z0;

Line(GLdouble a, GLdouble b, GLdouble c, GLdouble x0, GLdouble y0, GLdouble z0) {

this->a = a;

this->b = b;

this->c = c;

this->x0 = x0;

this->y0 = y0;

this->z0 = z0;

}

};

class LineSegment

{

public:

GLdouble x1, y1, z1;

GLdouble x2, y2, z2;

GLdouble r, g, b;

LineSegment(GLdouble x1, GLdouble y1, GLdouble z1, GLdouble x2, GLdouble y2, GLdouble z2, GLdouble r=1.0, GLdouble g=1.0, GLdouble b=1.0)

{

this->x1 = x1;

this->y1 = y1;

this->z1 = z1;

this->x2 = x2;

this->y2 = y2;

this->z2 = z2;

this->r = r;

this->g = g;

this->b = b;

}

void draw()

{

glColor3f(this->r, this->g, this->b);

glVertex3f(this->x1, this->y1, this->z1);

glVertex3f(this->x2, this->y2, this->z2);

}

void translation(GLdouble step\_x, GLdouble step\_y, GLdouble step\_z)

{

glTranslatef(step\_x, step\_y, step\_z);

}

void rotation(GLdouble angle, Line line)

{

glTranslatef(line.x0, line.y0, line.z0);

glRotatef(angle, line.a, line.b, line.c);

glTranslatef(-line.x0, -line.y0, -line.z0);

}

void reflection(Line line)

{

this->rotation(180, line);

}

};

LineSegment PQ(1.0, 2.0, 0.0, 7.0, 14.0, 0.0, 0.0, 0.8, 0.8);

Line Ox(1.0, 0.0, 0.0, 1.0, 0.0, 0.0);

Line Oy(0.0, 1.0, 0.0, 0.0, 1.0, 0.0);

Line MN(5.0, 40.0, 10.0, 5.0, -40.0, 0.0);

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glBegin(GL\_LINES);

PQ.draw();

glEnd();

glPopMatrix();

glFlush();

}

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

std::cout << "Key: " << key << std::endl;

switch (key) {

case 116: // t

case 84: { // T

PQ.translation(5.0, 0.0, 0.0);

glutPostRedisplay();

break;

}

case 114: // r

case 82: { // R

PQ.rotation(30.0, Oy);

glutPostRedisplay();

break;

}

case 112: // p

case 80: { // P

PQ.reflection(Ox);

glutPostRedisplay();

break;

}

case 102: // f

case 70: { // F

PQ.reflection(MN);

glutPostRedisplay();

break;

}

case 104: // h

case 72: { // H

PQ.rotation(30.0, MN);

glutPostRedisplay();

break;

}

}

}

void setupWindow(const char\* title, int width, int height, int pos\_x, int pos\_y) {

glutInitWindowSize(width, height);

glutInitWindowPosition(pos\_x, pos\_y);

glutCreateWindow(title);

}

void init() {

glClearColor(0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-40.0, 40.0, -40.0, 40.0, -40.0, 40.0);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

setupWindow("Lab 02", 640, 480, 100, 100);

init();

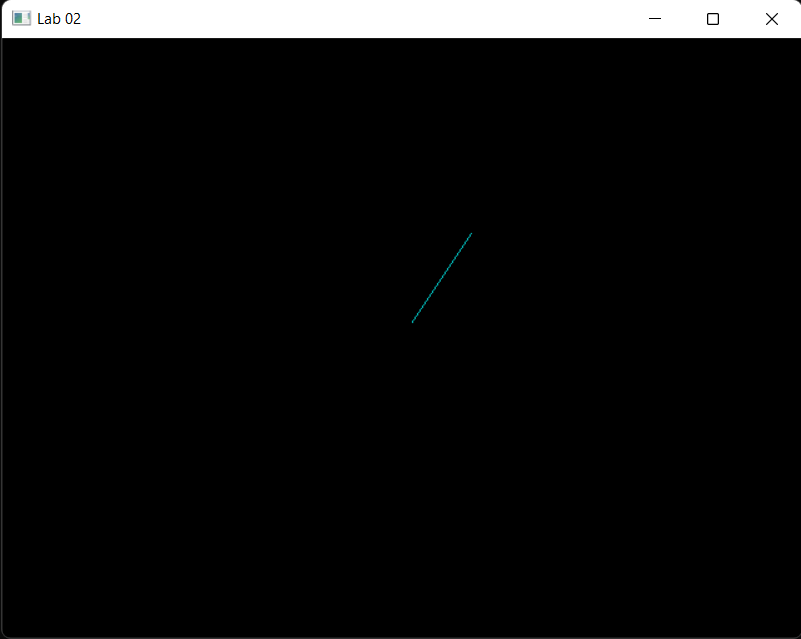
glutDisplayFunc(display);

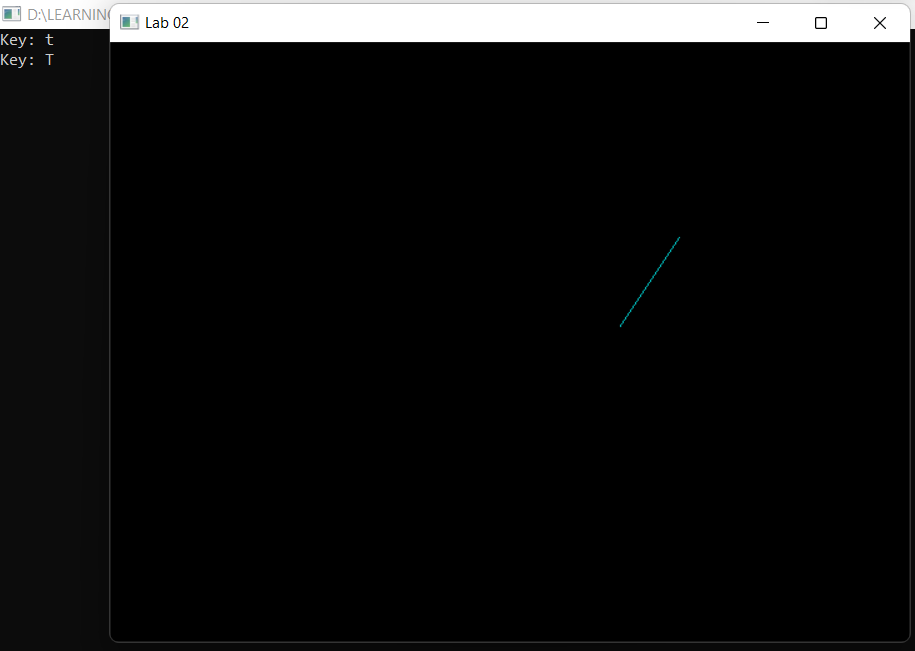
glutKeyboardFunc(keyEvents);

glutMainLoop();

return 0;

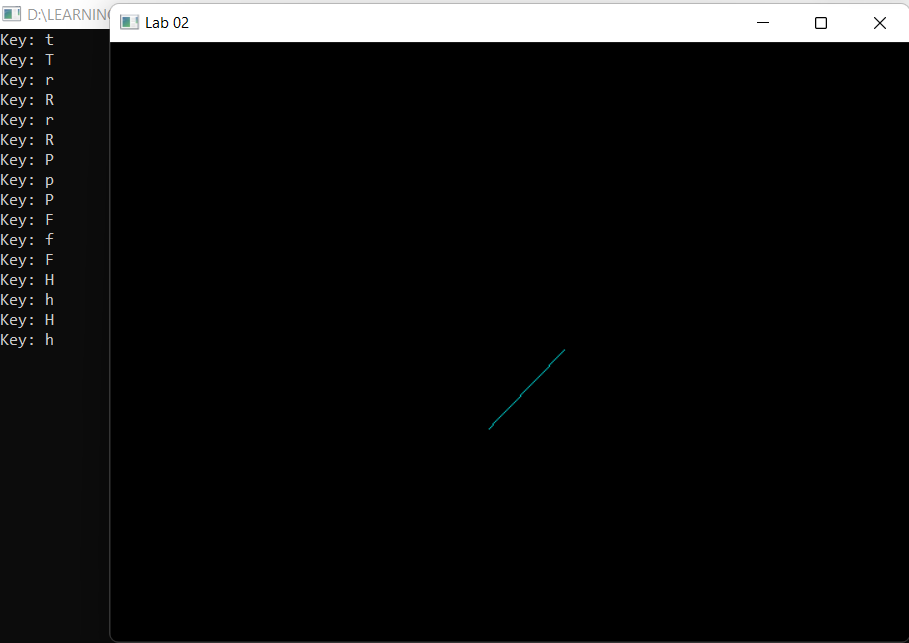
}











------------------------------------------------