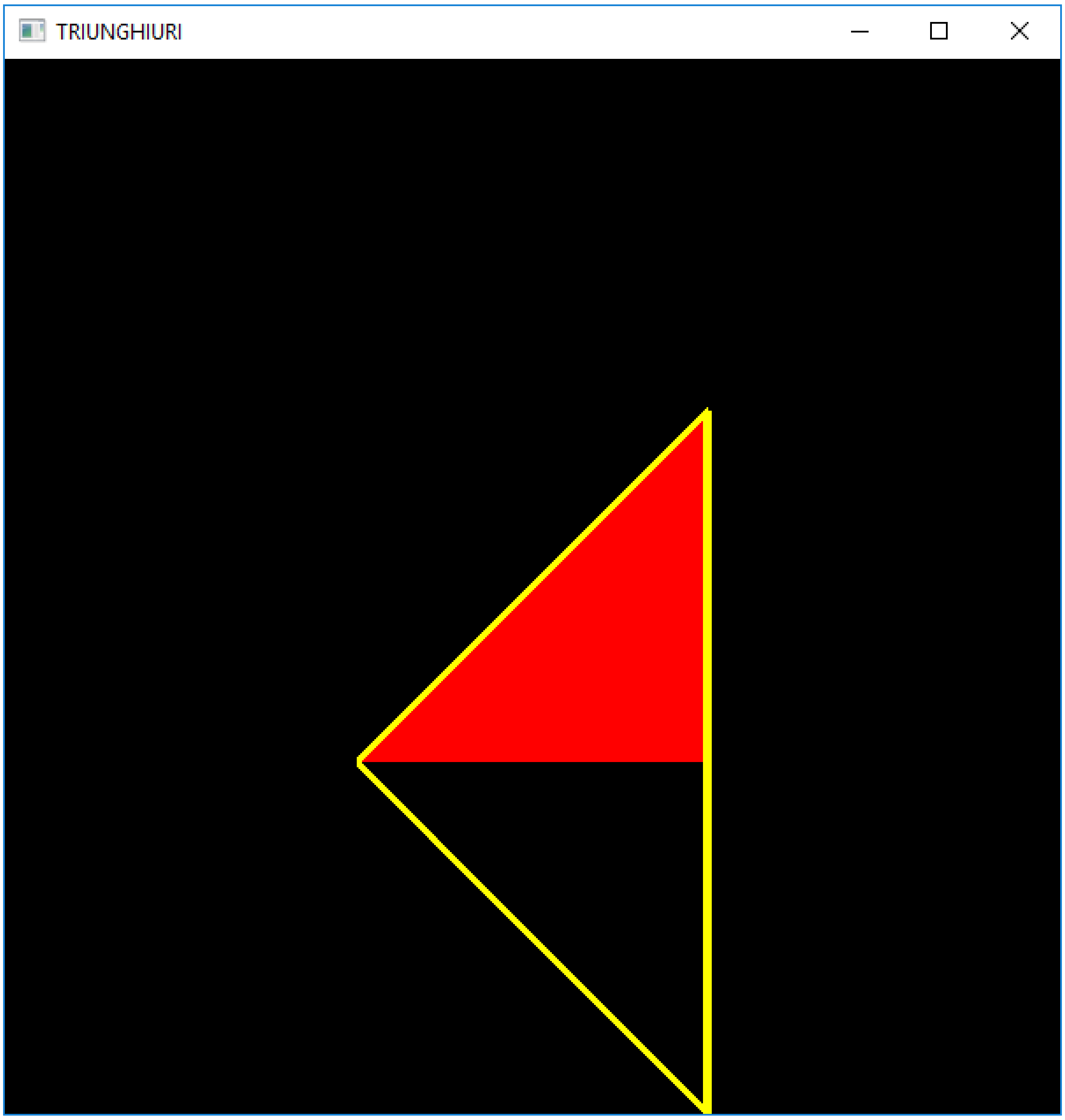
Aplicatii 23.10.2023

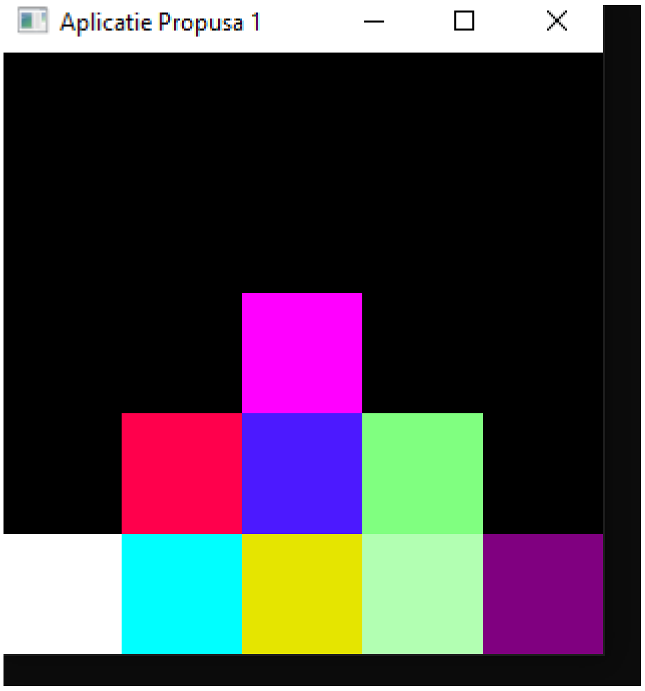
```
#include <iostream>
#include <gl/freeglut.h>

void init()
{
	glClearColor(0.0, 0.0, 0.0, 0.0);
	glPointSize(40.0);
	glShadeModel(GL_FLAT);
}
void desen()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.0, 1.0, 0.0);
	glBegin(GL_POLYGON);//initializare desen poligon
	glVertex2f(0.0, 0.0); //stabilire coordonate triunghi
	glVertex2f(200., 0.);
	glVertex2f(200.0, 200.0);//stabilire coordonate triunghi
	glVertex2f(00.0, 200.0);//stabilire coordonate triunghi
	glEnd();//sfisit desenare
	              //executare functie
	glFlush();


	glColor3f(1.0, 1.0, 0.0);
	glBegin(GL_POINTS);
	glVertex2i(100, 300);
	glVertex2i(200, 300);
	glVertex2i(200, 400);
	glColor3f(1.0, 0.0, 1.0);
	glVertex2i(20, 20);
	glEnd();
	glFlush();



}
void reshape(int w, int h)//functia redesenare
{
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);//stabilirea viewportului la 
dimensiunea ferestrei
	glMatrixMode(GL_PROJECTION);//specificare matrice modificabila la valoare 
argumentului de modificare
	glLoadIdentity();//initializarea sistemului de coordonate
	gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);//stabileste volumul de vedere 
folosind o proiectie ortografica
}
void main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(400, 500);
	glutInitWindowPosition(150, 150);
	glutCreateWindow("Aplicatia 1");
	init();	glutDisplayFunc(desen);	glutReshapeFunc(reshape);
	glutMainLoop();
	//return 0;
}
```

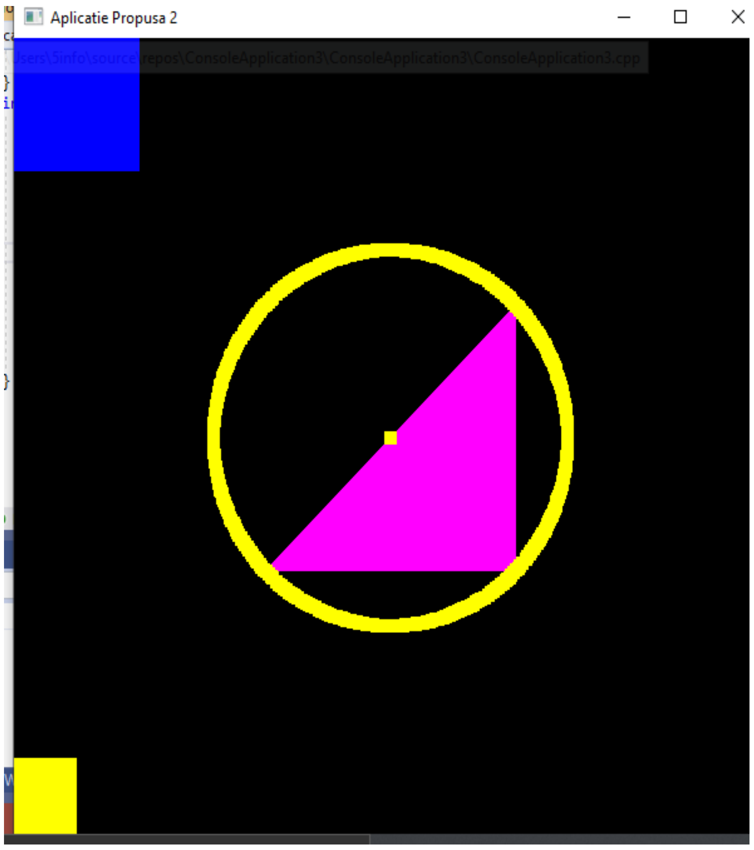


```
#include <gl/freeglut.h>
void init()//functia initiere
{
	// glClearColor (0.0, 0.0, 0.0, 0.0);//stabileste culoarea de sters
	// glShadeModel (GL_FLAT);
}
void display()//functia de desenare si afisare
{
	glClear(GL_COLOR_BUFFER_BIT);//sterge urmele de desene din fereastra curenta
	glBegin(GL_POLYGON);//initializare desen poligon
	glColor3f(1.0, 0.0, 0.0);//culoarea de desenare
	glVertex2f(200.0, 200.0);//stabilire coordonate triunghi
	glVertex2f(400.0, 200.0);//stabilire coordonate triunghi
	glVertex2f(400.0, 400.0);//stabilire coordonate triunghi
	glEnd();//sfisit desenare
	glFlush();//executare functie
```

```
        glLineWidth(5);
//glPointSize(40.0);


        glBegin(GL_LINE_LOOP);//initializare desen poligon
        glColor3f(1.0, 1.0, 0.0);//culoarea de desenare
        glVertex2f(200.0, 200.0);//stabilire coordonate triunghi
        glVertex2f(400.0, 0.0);//stabilire coordonate triunghi
        glVertex2f(400.0, 400.0);//stabilire coordonate triunghi
        glEnd();//sfarsit desenare
        glFlush();//executare functie

}
void reshape(int w, int h)//functia redesenare
{
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);//stabilirea viewportului la
dimensiunea ferestrei
        glMatrixMode(GL_PROJECTION);//specificare matrice modificabila la valoare
argumentului de modificare
        glLoadIdentity();//initializarea sistemului de coordonate
        gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);//stabileste volumul de vedere
folosind o proiectie ortografica
}
int main(int argc, char** argv) //creare fereastra
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);//se specifica modelul de culoare al
ferestrei: un singur buffer si culoare RGB
        glutInitWindowSize(600, 600);//initiaza dimensiunea ferestrei principale 600x600
pixeli
        glutInitWindowPosition(200, 10);//initiaza in fereastra principala fereastra de
afisare
        glutCreateWindow("TRIUNGHIURI");
        init();
        glutDisplayFunc(display);//se reimprospateaza continutul ferestrei
        glutReshapeFunc(reshape);//functia redesenare
        glutMainLoop();//bucla de procesare a evenimentelor
        return 0;
}
```

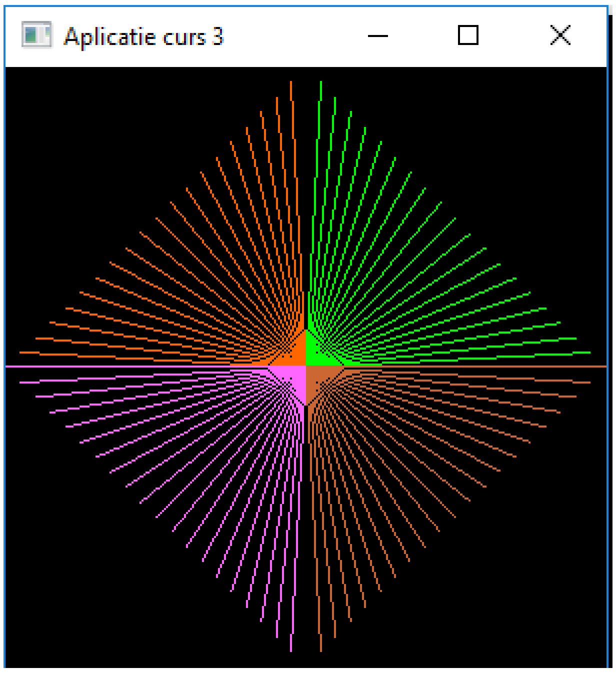
Aplicatie Propusa



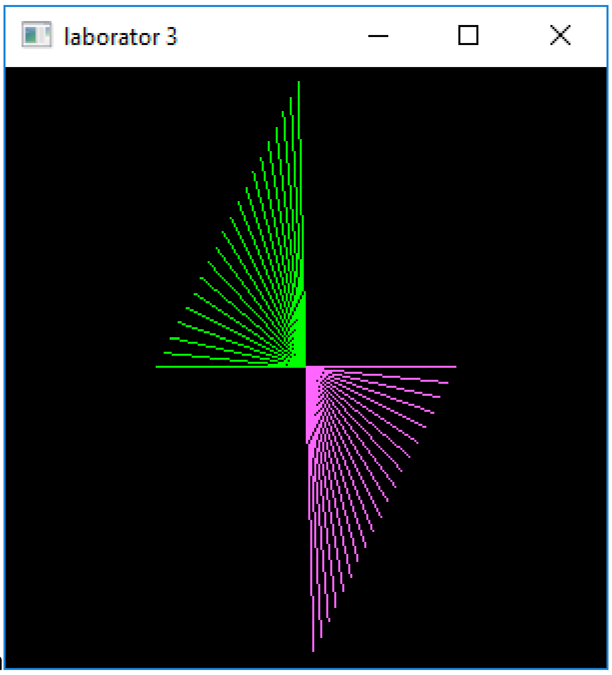

Aplicatie Propusa 2
sers\5info\source\repos\ConsoleApplication3\ConsoleApplication3\ConsoleApplication3.cpp

ex 1

```
#include <iostream>
#include <gl/freeglut.h>
void Display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0, 1, 0);
	glBegin(GL_LINES);
	// Cadran 1
	for (int i = 0; i < 20; i++)
	{
		glVertex3f(0, 0, 0);
		glVertex3f(1 - i / 20.0, i / 20.0, 0);
	} // Cadran 2
	glColor3f(1, 0.4, 0);
	for (int i = 0; i < 20; i++)
	{
		glVertex3f(0, 0, 0);
		glVertex3f(-1 + i / 20.0, i / 20.0, 0);
	} // Cadran 3
	glColor3f(1, 0.4, 1);
	for (int i = 0; i < 20; i++)
	{
		glVertex3f(0, 0, 0);
		glVertex3f(-1 + i / 20.0, -i / 20.0, 0);
	}
	// Cadran 4
	glColor3f(0.8, 0.4, 0.2);
	for (int i = 0; i < 20; i++)
	{
		glVertex3f(0, 0, 0);
		glVertex3f(1 - i / 20.0, -i / 20.0, 0);
	}
	glEnd();    glFlush();  glColor3f(0, 0, 1);
}
int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(600, 600);
	//se specifica modelul de culoare al ferestrei: un singur buffer si culoare RGB
	glutCreateWindow("laborator 3");

	glutDisplayFunc(Display);
	glutMainLoop();
	return 0;
}
```



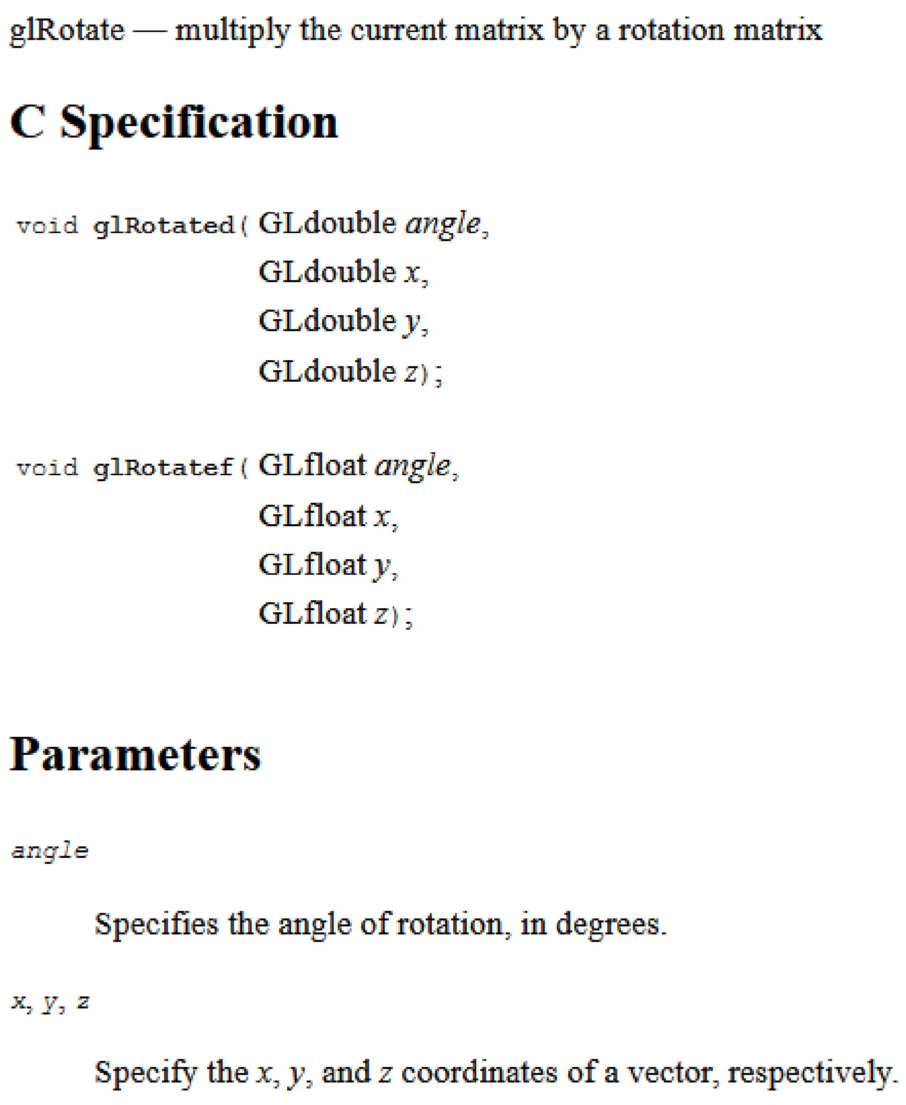
Tema



Se se roteasca imaginea

glutPostRedisplay marchează fereastra curentă ca fiind nevoie să fie reafișată
Usage
void glutPostRedisplay(void);
Description
Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.
Logically, normal plane damage notification for a window is treated as a glutPostRedisplay on the damaged window. Unlike damage reported by the window system, glutPostRedisplay will not set to true the normal plane's damaged status (returned by glutLayerGet(GLUT_NORMAL_DAMAGED

glRotatef



glRotate — multiply the current matrix by a rotation matrix

C Specification

```
void glRotated( GLdouble angle,
                GLdouble x,
                GLdouble y,
                GLdouble z);

void glRotatef( GLfloat angle,
                GLfloat x,
                GLfloat y,
                GLfloat z);
```

Parameters

`angle`

Specifies the angle of rotation, in degrees.

`x, y, z`

Specify the *x*, *y*, and *z* coordinates of a vector, respectively.

Description

glRotate produces a rotation of *angle* degrees around the vector xyz. The current matrix (see glMatrixMode) is multiplied by a rotation matrix with the product replacing the current matri
as if glMultMatrix were called with the following matrix as its argument:

$$\begin{array}{cccc} x^2 1-c+c & xy1-c-zs & xz1-c+ys & 0 \\ yx1-c+zs & y^2 1-c+c & yz1-c-xs & 0 \\ xz1-c-ys & yz1-c+xs & z^2 1-c+c & 0 \\ 0 & 0 & 0 & 1 \end{array}$$

Where $c = cosangle$, $s = sinangle$, and $xyz = 1$ (if not, the GL will normalize this vector).

If the matrix mode is either GL_MODELVIEW or GL_PROJECTION, all objects drawn after glRotate is called are rotated. Use glPushMatrix and glPopMatrix to save and restore the unrotated coordinate system.

Notes

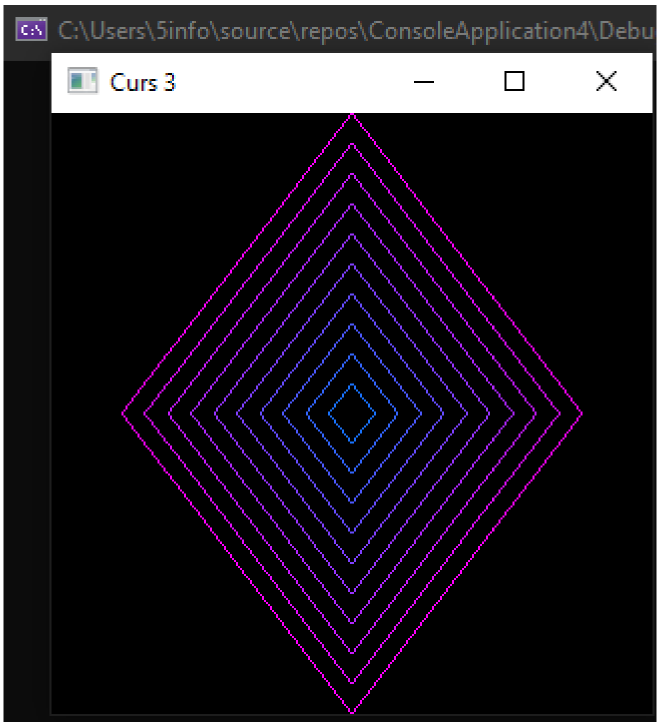
This rotation follows the right-hand rule, so if the vector xyz points toward the user, the rotation will be counterclockwise.

Errors

GL_INVALID_OPERATION is generated if glRotate is executed between the execution of glBegin and the corresponding execution of glEnd.

```
#include <GL/glut.h>

#include <stdlib.h>

#include <iostream>
#include <gl/freeglut.h>
void roteste_Y(int p_grade)
{
	glRotatef(p_grade, 0.0, 1.0, .0);
	glutPostRedisplay();
}
void roteste_X(int p_grade)
{
	glRotatef(p_grade, 0., 1., .0);
	glutPostRedisplay();
}
void OnKeyPress(unsigned char key, int x, int y)
{
	if (key == 27)
		exit(0);
	switch (key)
	{
	case 'q':
	case 'Q':

		roteste_Y(3);
		break;
	case 'w':
	case 'W':
		roteste_Y(-3);
		break;

	case 'a':
	case 'A':

		roteste_X(3);
		break;
	case 's':
	case 'S':
		roteste_X(-3);
		break;
	}
}
void OnMouseClick(int button, int state, int x, int y)
```

```
{
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		roteste_Y(20);
	}
	if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
	{
		roteste_Y(-20);
	}
}

void display(void)
{

	glClear(GL_COLOR_BUFFER_BIT);

	int l = 10;

	for (double i = 0; i <= l; i++) {
		glBegin(GL_LINE_LOOP);
		glColor3f(1 - i / 10, i / 20, 1);
		glVertex3f(1 - i / l, 0, 0);
		glVertex3f(0, 1 - i / l, 0);
		glVertex3f(-(1 - i / l), 0, 0);
		glVertex3f(0, -(1 - i / l), 0);
		glEnd();
	}

	glFlush();

}

int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);//se specifica modelul de culoare al
ferestrei: un singur buffer si culoare RGB
	glutCreateWindow("Curs 07.11.2020");
	glutKeyboardFunc(OnKeyPress);
	glutMouseFunc(OnMouseClick);

	glutDisplayFunc(display);
	glutMainLoop();
	return 0;
}
```

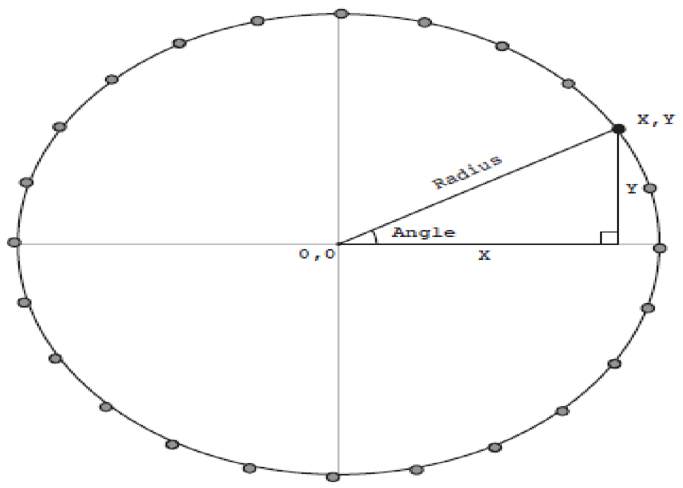


Aplicatie propusa

Drawing Circles with OpenGL

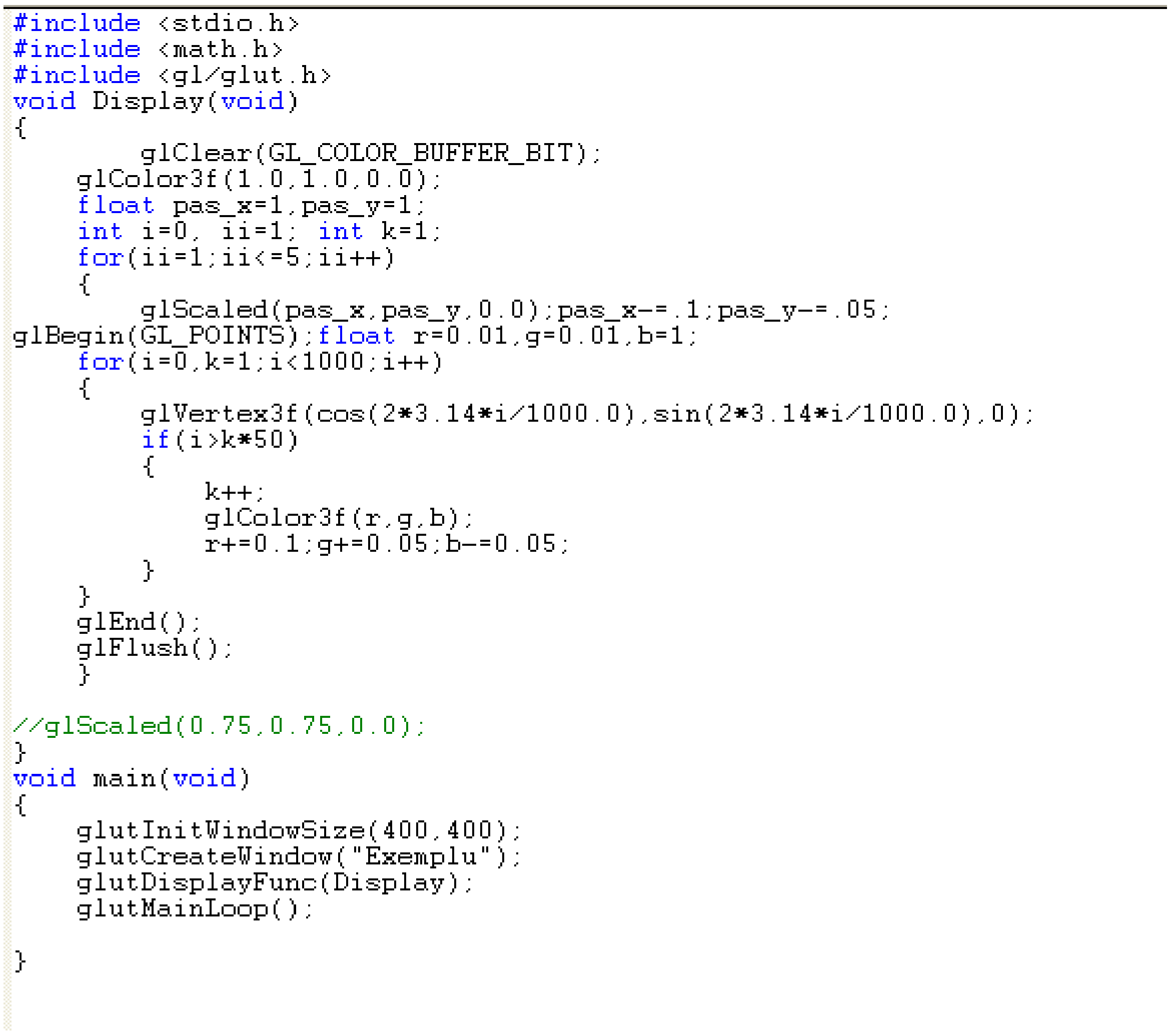
```
void drawCircle( float Radius, int numPoints )

{

glBegin( GL_LINE_STRIP );

for( int i=0; i<numPoints; i++ )

{

float Angle = i * (2.0*PI/numPoints); // use 360 instead of 2.0*PI if

float X = cos( Angle )*Radius; // you use d_cos and d_sin

float Y = sin( Angle )*Radius;

glVertex2f( X, Y );

}

glEnd();

}
```

The Angle variable in this code is in radians, since the built-in cos and sin functions take radians as inputs. To use degrees instead, implement helper functions d_cos and d_sin that convert the angle to radians from degrees, and then return the cosine or sine.

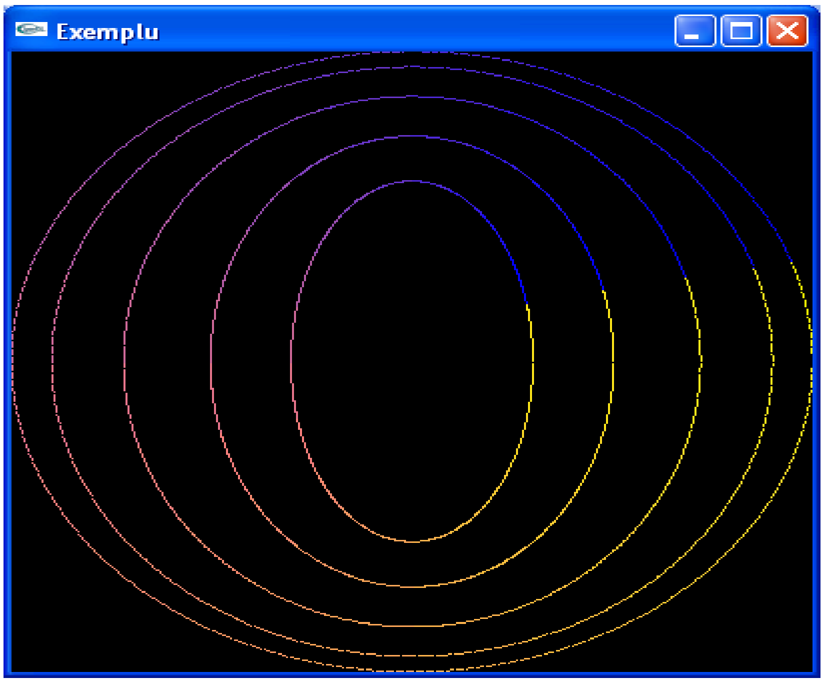


```
X = cos(Angle)*Radius;
Y = sin(Angle)*Radius;

Pythagorean Theorem:
Radius = sqrt(X*X + Y*Y);
```

This circle is shown with 24 points.



```
#include <stdio.h>
#include <math.h>
#include <gl/glut.h>
void Display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,0.0);
    float pas_x=1,pas_y=1;
    int i=0, ii=1; int k=1;
    for(ii=1;ii<=5;ii++)
    {
        glScaled(pas_x,pas_y,0.0);pas_x-=.1;pas_y-=.05;
glBegin(GL_POINTS);float r=0.01,g=0.01,b=1;
    for(i=0,k=1;i<1000;i++)
    {
        glVertex3f(cos(2*3.14*i/1000.0),sin(2*3.14*i/1000.0),0);
        if(i>k*50)
        {
            k++;
            glColor3f(r,g,b);
            r+=0.1;g+=0.05;b-=0.05;
        }
    }
    glEnd();
    glFlush();
    }

//glScaled(0.75,0.75,0.0);
}
void main(void)
{
    glutInitWindowSize(400,400);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();

}
```

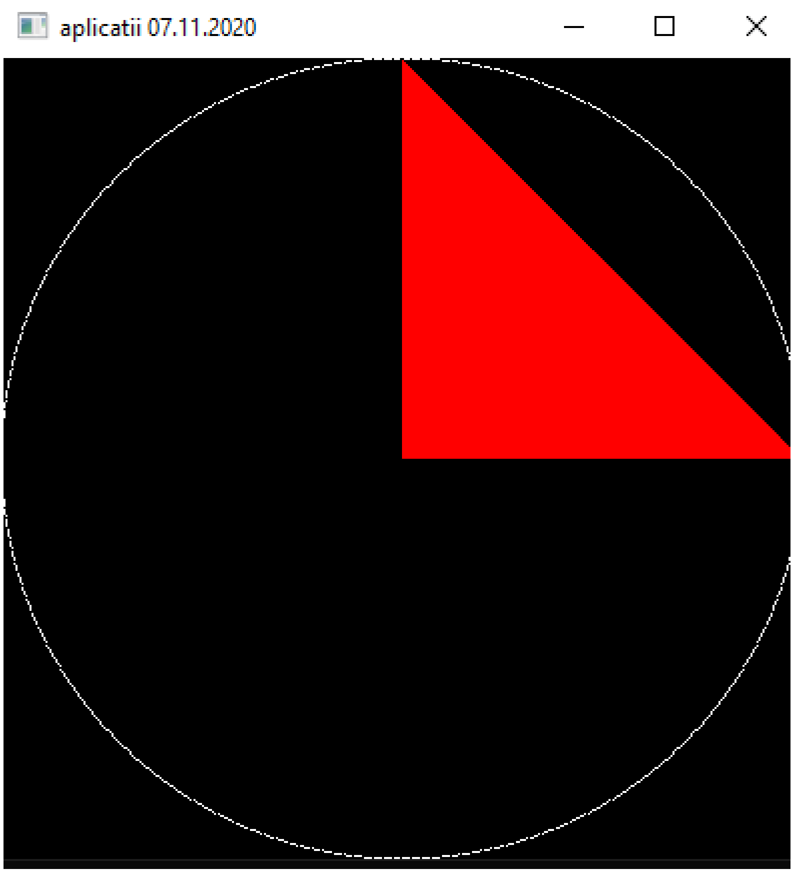


Meniuri

```
#include <iostream>
#include <gl/freeglut.h>
#include<math.h>
#include<stdio.h>
#include <stdlib.h>


void init()
{
	glClearColor(0.0, 0.0, 0.0, 0.0);
	glPointSize(50.0);
	glShadeModel(GL_FLAT);
}
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);

	glBegin(GL_TRIANGLES);
	glColor3f(1.0, 0.0, 0.0);
	glVertex2i(1, 0);
	glVertex2i(0, 0);
	glVertex2i(0, 1);
	glEnd();

	glPointSize(1.0);
	glColor3f(1, 1, 1);
	glBegin(GL_POINTS);
	for (int i = 0; i < 1000; ++i)
	{
		glVertex3f(cos(2 * 3.14159 * i / 1000.0), sin(2 * 3.14159 * i / 1000.0),
0);
	}
	glEnd();

	glFlush();
}
int meniu_1, meniu_2, meniu_3, meniu_main;

void meniu_principal(int key)
{
	if (key == 0)
	{
		exit(0);
	}
}

void callback_1(int key)
{
	switch (key)
	{
	case 0:
		printf("Cerc 1\n");
		break;
	case 1:
		printf("Cerc 2\n");
		break;
	}
}
```

```
void callback_2(int key)
{
	switch (key)
	{
	case 0:
		printf("Ati selectat dreptunghi 1\n");
		break;
	case 1:
		printf("Ati selectat dreptunghi 2\n");
		break;
	}
}

void callback_3(int key)
{
	switch (key)
	{
	case 0:
		printf("Ati selectat triunghi 1\n");
		break;
	case 1:
		printf("Ati selectat triunghi 2\n");
		break;

	}
}
GLint x = 10;
GLint y = 20;
GLint WindowWidth = 400;
GLint WindowHight = 400;

void mouseHandler(int button, int state, int mouse_x, int mouse_y)
{
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		x = mouse_x;
		y = WindowHight - mouse_y;
		glColor3f(1, 0, 0);
		glBegin(GL_POINTS);
		glVertex2i(x, y);
		printf("x=%d , y=%d \n", x, y);
		glEnd();
		glFlush();
		glClear(GL_COLOR_BUFFER_BIT);
	}
}
int main(int argc, char** argv)
{

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(400, 400);
	glutInitWindowPosition(400, 100);
	glutCreateWindow("aplicatii 07.11.2020");
	init();
	glutMouseFunc(mouseHandler);

	glutDisplayFunc(display);
	meniu_1 = glutCreateMenu(callback_1);
	glutAddMenuEntry("cerc1", 0);
	glutAddMenuEntry("cerc2", 1);
```

```
        meniu_2 = glutCreateMenu(callback_2);
        glutAddMenuEntry("dreptunghi1", 0);
        glutAddMenuEntry("dreptunghi2", 1);
        meniu_3 = glutCreateMenu(callback_3);
        glutAddMenuEntry("triunghi1", 0);
        glutAddMenuEntry("triunghi2", 1);

        meniu_main = glutCreateMenu(meniu_principal);
        glutAddSubMenu("cerc", meniu_1);
        glutAddSubMenu("patrat", meniu_2);
        glutAddSubMenu("triunghi", meniu_3);
        glutAddMenuEntry("Exit", 0);
        glutAttachMenu(GLUT_RIGHT_BUTTON);

        glutMainLoop();

        return 0;
}
```


aplicatii 07.11.2020