Tipurile de primitive geometrice

Proprietate	Primitivă
GL_POINTS	Desenează n puncte
GL_LINES	Desenează segmentele de dreaptă izolate (v_0,v_1), (v_2,v_3),... ş.a.m.d. Dacă n este impar ultimul vârf este ignorat
GL_LINE_STRIP	Desenează linia poligonală formată din segmentele (v_0,v_1), (v_1,v_2), ...(v_{n-2}, v $_{n-1}$)
GL_LINE_LOOP	La fel ca primitiva GL_LINE_STRIP, dar se mai desenează segmentul (v_n, v_0) care închide o buclă.
GL_TRIANGLES	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_3,v_4,v_5), ş.a.m.d. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
GL_TRIANGLE_STRIP	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_2,v_1,v_3),..ş.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeaşi orientare, deci să poată forma o suprafaţă închisă.
GL_TRIANGLE_FAN	Desenează triunghiurile (v_0,v_1,v_2), (v_0,v_2,v_3), ş.a.m.d.
GL_QUADS	Desenează o serie de patrulatere (v_0,v_1,v_2,v_3), (v_4,v_5,v_6,v_7), ş.a.m.d. Dacă n nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
GL_QUADS_STRIP	Desenează o serie de patrulatere (v_0,v_1,v_3,v_2), (v_3,v_2,v_5,v_4), ş.a.m.d Dacă n < 4, nu se desenează nimic. Dacă n este impar, ultimul vârf este ignorat.
GL_POLYGON	Desenează un poligon cu n vârfuri, (v_0,v_1,...v_{n-1}). Dacă poligonul nu este convex, rezultatul este impredictibil.

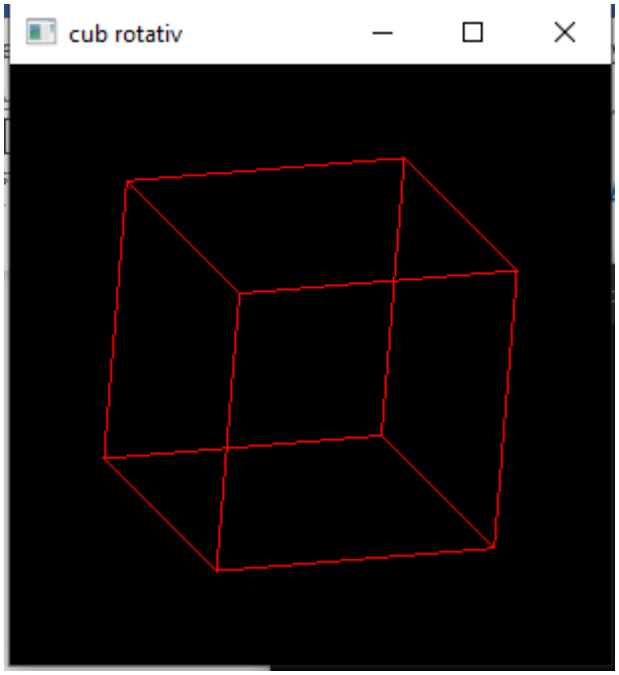
glLoadIdentity ();

Acesta este utilizat împreună cu alte 2 funcții numite glMatrixMode (GL_PROJECTION) și glMatrixMode (GL_MODELVIEW). Aceste funcții sunt utilizate pentru a reseta diverse matrice care se ocupă de volumul de vizionare și orice transformări ale modelelor desenate înainte și după redimensionarea ferestrei dvs.

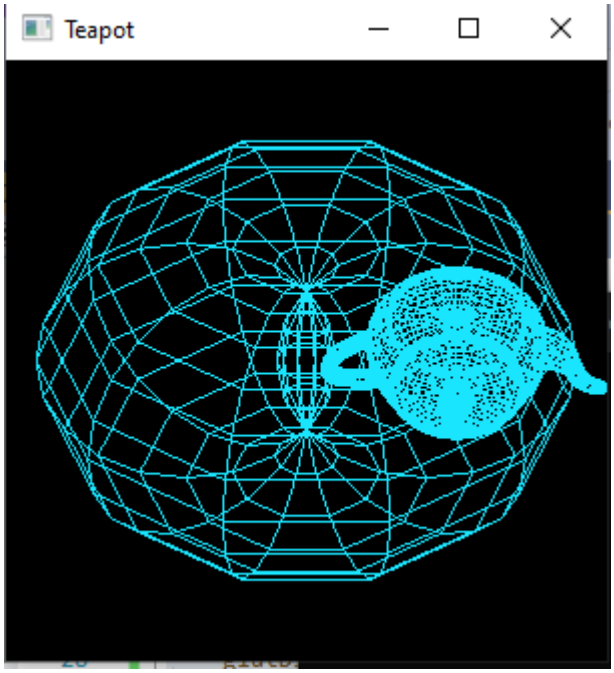
glTranslate și glRotate sunt întotdeauna relative la starea actuală. De exemplu, dacă apelați glTranslate, are efect din „poziția" actuală a matricei, nu din origine. Dar dacă doriți să începeți din nou de la origine, atunci apelați glLoadIdentity() și apoi puteți glTranslate din matricea care este acum situată

Atunci când utilizați glTranslate (...) sau glRotate (...) afectați matricea modelview. Aceasta înseamnă că atunci când aplicați mai multe transformări (traduceri și rotații), această matrice se schimbă și ea.

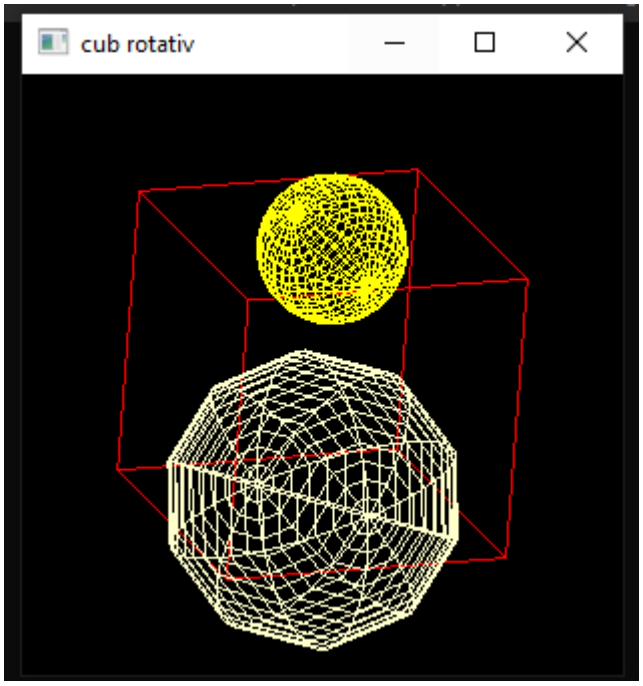
glPushMatrix(); // Setați matricea curentă pe stivă/salvează ecranul curent în stivă
glPopMatrix(); // încarca datele din stivă




```
#include <gl/freeglut.h>
int GAngle = 30;
void display(void) {
	glClear(GL_COLOR_BUFFER_BIT);
	glLoadIdentity();
	glRotated(GAngle, 1, 1, 0);//face o rotatie a unghiului in jurul vectorilor
x,y,z
	glColor3f(1, 0, 0);
	glutWireCube(1);
	//glutWireTeapot(0.3);
	GAngle = GAngle + 1;
	glFlush();
}
void Timer(int extra) {
	glutPostRedisplay();//fereastra curenta este reafisata
	glutTimerFunc(30, Timer, 0);//seteaza timer-ul pentru fereastra curenta
}
int main(int argc, char** argv)
{

	glutInit(&argc, argv);
	glutCreateWindow("cub rotativ");
	glutDisplayFunc(display);
	glutTimerFunc(0, Timer, 0);//????
	glutMainLoop();
	return 0;
}
```

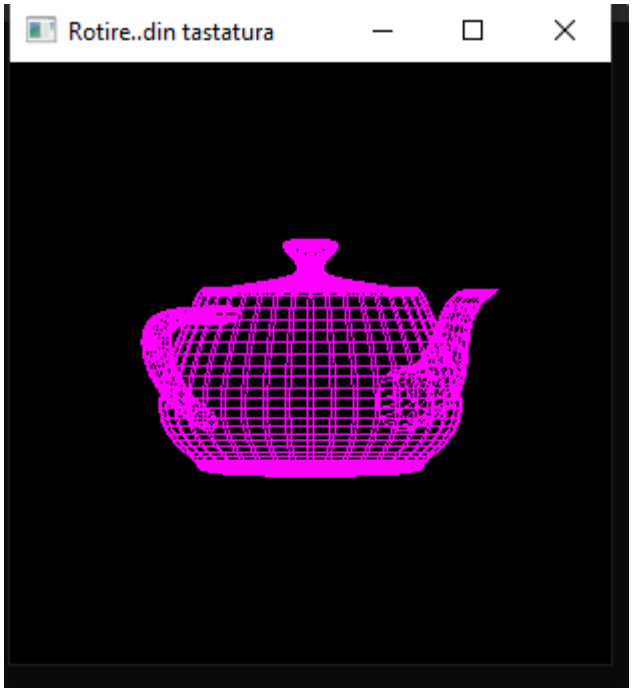



```c
#include <gl/freeglut.h>
void Display()
{
	static float grade = -10;
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.1, 0.9, 1.0);
	glLoadIdentity();
	glPushMatrix();
	glRotatef(grade, 1, 0, 0);
	glutWireTorus(0.5, 0.4, 20, 10);
	glTranslated(0.5, 0, 0);
	glutWireTeapot(0.3);
	glPopMatrix();
	glFlush();
	grade = grade + 0.1;
	glutPostRedisplay();
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	//glutInitWindowSize(640, 480);
	glutInitWindowPosition(10, 10);
	glutCreateWindow("Teapot");
	glutDisplayFunc(Display);
	glutMainLoop();
	return 0;
}
```




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

int GAngle = 30;

void display(void) {
	glClear(GL_COLOR_BUFFER_BIT);

	glLoadIdentity();
	glRotated(GAngle, 1, 1, 0);//face o rotatie a unghiului in jurul vectorilor
x,y,z
	glColor3f(1, 0, 0);
	glutWireCube(1);//cub schelet

	GAngle = GAngle + 1;
	glFlush();
	glPushMatrix();
	glTranslated(-0.45, 0.0, 0.0);
	glRotated(15, 0.0, 1.0, 0.0);
	glColor3d(1.0, 1., 0.8);
	//glScaled(x, y, z);
	glutWireSphere(0.5, 10, 25);
	glPopMatrix();  // Pop matricea veche fără transformări.
	glPushMatrix();  // Setați matricea curentă pe stivă
 // glPushMatrix () este salvează ecranul curent pentru o stiva și glPopMatrix () este
de a încărca datele din stivă
	glTranslated(0.45, 0.0, 0.0);
	glRotated(75, 1.0, 1, 0.0);
	glColor3d(1.0, 1.0, 0.0);
	glScaled(0.5, 0.5, 0.5);
	glutWireSphere(0.5, 25, 25);
	glPopMatrix();
	glFlush();

}
void Timer(int extra) {
	glutPostRedisplay();//fereastra curenta este reafisata
	glutTimerFunc(10, Timer, 0);//seteaza timer-ul pentru fereastra curenta
}
```

```
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	//glutInitWindowSize(640, 480);
	glutCreateWindow("cub rotativ");
	glutDisplayFunc(display);
	glutTimerFunc(0, Timer, 0);
	glutMainLoop();
	return 0;
}
```



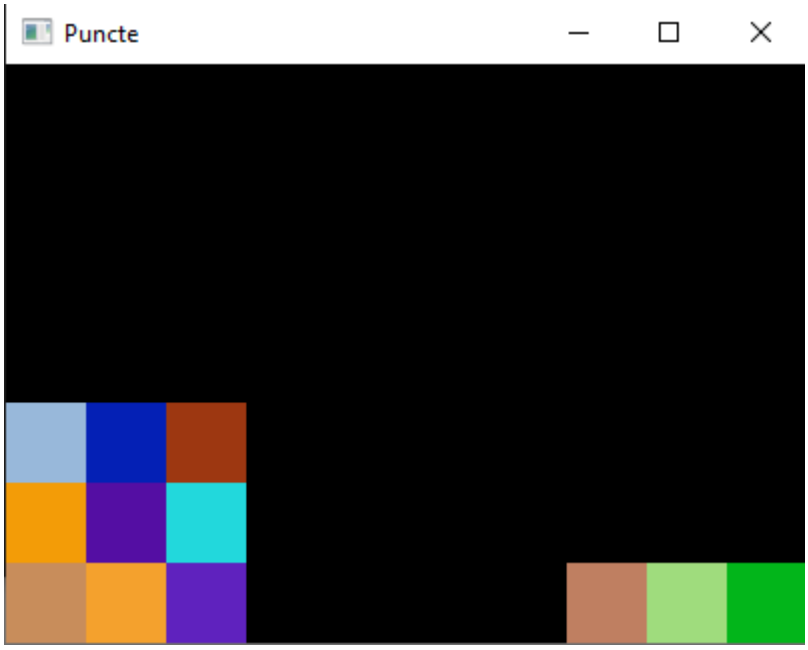

```
#include <gl/freeglut.h>

#include <math.h>

void init()
{
	glClearColor(0.0, 0.0, 0.0, 0.0); //seteaza culoarea de stergere
	glShadeModel(GL_FLAT); // seteaza tipul umbrei
}

void Display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1, 0, 1);
	glutWireTeapot(0.5);  //deseneaza ceainic

	glFlush();//executare functie
}

void rotate(int x_grade, int y_grade, int z_grade)
{
	if (x_grade) glRotatef(x_grade, 1, 0, 0); //rotire pe axa x
	if (y_grade) glRotatef(y_grade, 0, 1, 0); //rotire pe axa y
	if (z_grade) glRotatef(z_grade, 0, 0, 1); //rotire pe axa z
	if (x_grade || y_grade || z_grade) glutPostRedisplay(); // redesenare
}

void OnKey(unsigned char key, int x, int y)
{
	if (key == 27)
		exit(0);

	int x_grade = 0, y_grade = 0, z_grade = 0;
	switch (key)
	{
	case 'q':
	case 'Q':
```

```
                z_grade++;
                rotate(x_grade, y_grade, z_grade);
                break;
        case 'e':
        case 'E':
                z_grade--;
                rotate(x_grade, y_grade, z_grade);
                break;
        case 'w':
        case 'W':
                x_grade++;
                rotate(x_grade, y_grade, z_grade);
                break;
        case 's':
        case 'S':
                x_grade--;
                rotate(x_grade, y_grade, z_grade);
                break;
        case 'a':
        case 'A':
                y_grade++;
                rotate(x_grade, y_grade, z_grade);
                break;
        case 'd':
        case 'D':
                y_grade--;
                rotate(x_grade, y_grade, z_grade);
                break;
        }
}

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) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        //glutInitWindowSize(600, 600);
        glutInitWindowPosition(150, 150);
        glutCreateWindow("Rotire..din tastatura");
        init();
        glutDisplayFunc(Display);
        glutReshapeFunc(reshape);

        glutKeyboardFunc(OnKey);

        glutMainLoop();
        return 0;
}
```

```
#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();

}
```


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

int sectiuni_i = 10;
GLfloat Pi = 2 * 3.14;
GLfloat R = 0.9;
void init()
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glPointSize(50.0);
	glShadeModel(GL_FLAT);
}


void display()
{
	//float rad3=0.8;
	glClear(GL_COLOR_BUFFER_BIT);

	glBegin(GL_TRIANGLE_FAN);
	glColor3f(1, 0, 0);
	R = 0.7;
	glVertex2f(0.0, 0.0);
	for (int i = 0; i <= sectiuni_i; ++i)
	{
		glVertex2f(R * cos(i * Pi / sectiuni_i), R * sin(i * Pi / sectiuni_i));
	}
	glEnd();
	R = 0.4;

	glBegin(GL_TRIANGLE_FAN);
	glColor3f(1, 1, 0);
	glVertex2f(0.0, 0.0);
```

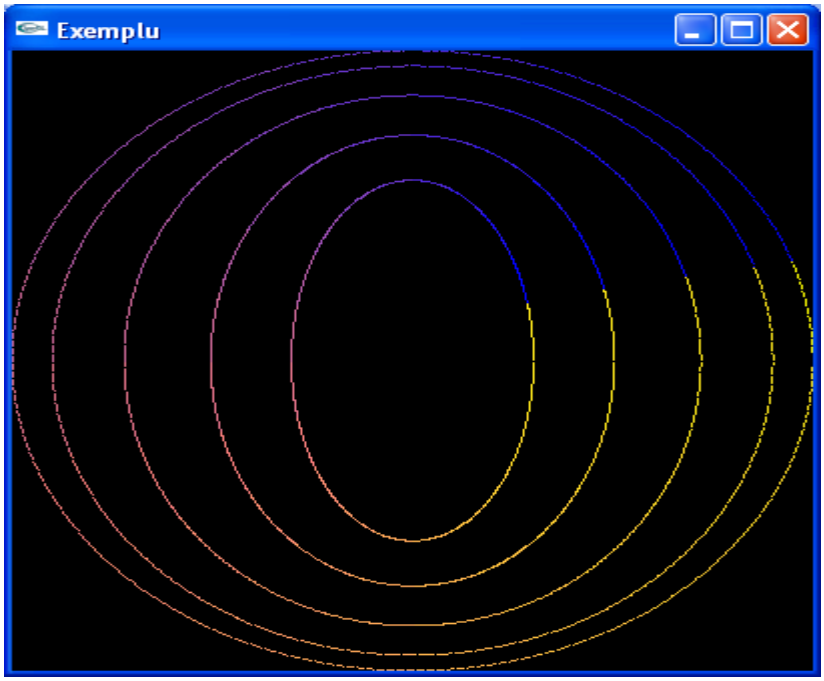
```
	for (int i = 0; i <= sectiuni_i; ++i)
	{
		glVertex2f(R * cos(i * Pi / sectiuni_i), R * sin(i * Pi / sectiuni_i));
	}
	glEnd();


	glBegin(GL_TRIANGLE_FAN);
	glColor3f(1, 0, 1);
	R = 0.1;
	glVertex2f(0.0, 0.0);
	for (int i = 0; i <= sectiuni_i; ++i)
	{
		glVertex2f(R * cos(i * Pi / sectiuni_i), R * sin(i * Pi / sectiuni_i));
	}
	glEnd();

	glFlush();

}

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


	glutInit(&argc, argv);

	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(400, 400);
	glutInitWindowPosition(400, 100);
	glutCreateWindow("GL_TRIANGLE_FAN");
	init();
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```

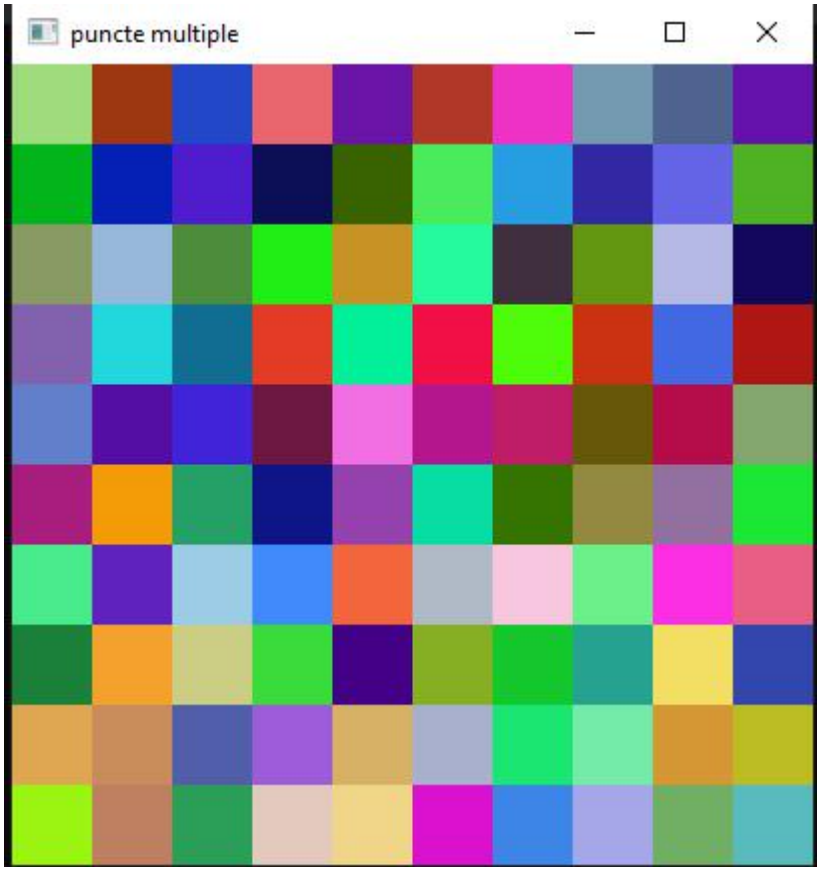



```
#include <gl/freeglut.h>
#include <math.h>

#include<math.h>
int width = 400;
int height = 400;
int psize = 40;
int distx = 0;
int disty = 0;
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glPointSize(psize);

	//varianat 1
	glBegin(GL_POINTS);

	for (GLint i = 0; i < 3; i++)
	{

		double r = ((double)rand() / (RAND_MAX));
		double g = ((double)rand() / (RAND_MAX));
		double b = ((double)rand() / (RAND_MAX));
		glColor3d(r, g, b);
		glVertex2f(400 - (GLfloat)i * 20 - distx, 20);

		distx += 20;
	}
	glEnd();
	glFlush();
	//varianat 2
	glBegin(GL_POINTS);
	disty = 20;
	for (int k = 0; k < 3; k++)
	{
		distx = 20;
	for (int j = 0; j < 3; j++)
	{
		double r = ((double)rand() / (RAND_MAX));
		double g = ((double)rand() / (RAND_MAX));
```

```
                double b = ((double)rand() / (RAND_MAX));
                glColor3d(r, g, b);
                glVertex2i(40 * j + distx, 40 * k + disty);

        }
        disty += 0;
}
        glEnd();
        glFlush();
}


void reshape(int w, int h)
{
        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
}//end reshape()

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

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(400, 300);
        glutCreateWindow("Puncte");
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```

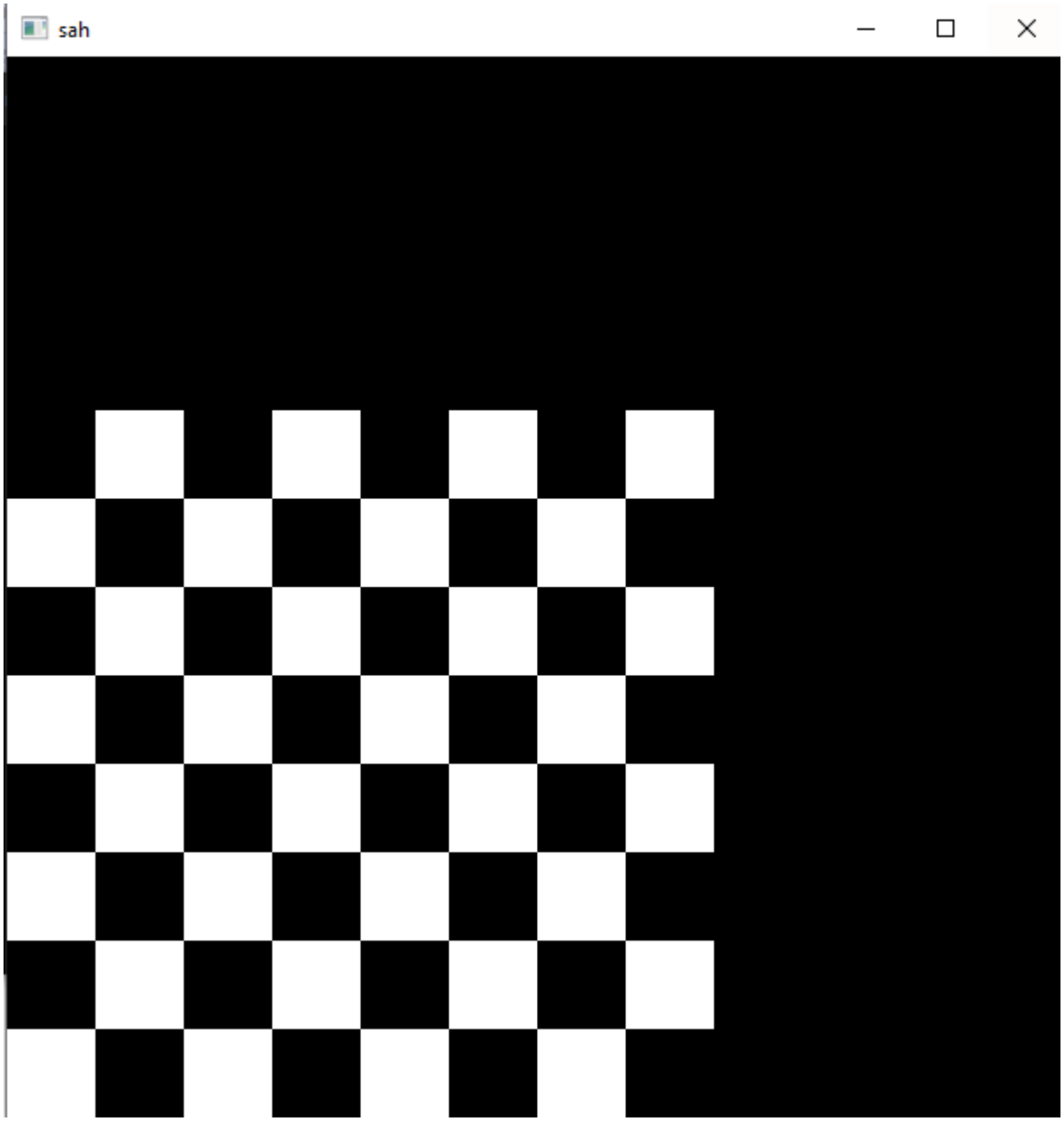
```
#include <stdio.h>
#include <math.h>
#include <gl/glut.h>
void Display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,0.0);
    float pas_x=1,pas_y=1;
    int i=0, ii=1; int k=1;
    for(ii=1;ii<=5;ii++)
    {
        glScaled(pas_x,pas_y,0.0);pas_x-=.1;pas_y-=.05;
glBegin(GL_POINTS);float 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();

}
```





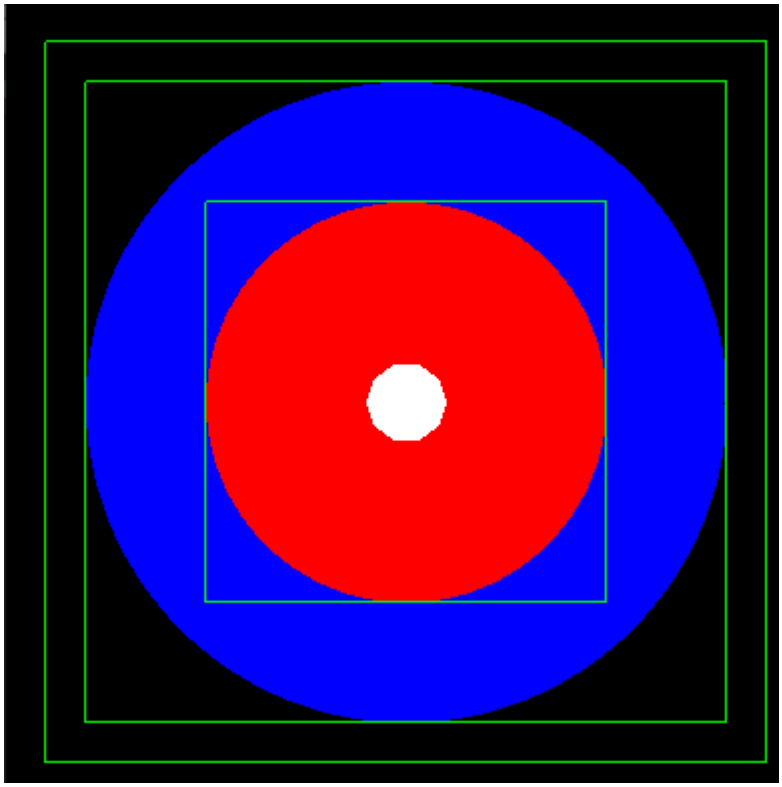

```
 #include <gl/freeglut.h>
#include <math.h>

void init()
{
	//glPointSize(10.0);
}

void Display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1, 1, 1);

	int width = 400;
	int heigth = 400;
	int w = 40, h = 40;
	for (int i = 0; i < width / w; ++i)
	{
		for (int j = 0; j < heigth / h; ++j)
		{
			double red = ((double)rand() / (RAND_MAX));
			double green = ((double)rand() / (RAND_MAX));
			double blue = ((double)rand() / (RAND_MAX));

			//schimba culoarea
			glColor3f(red, green, blue);

			float x1 = i * w;
			float y1 = j * h;
			float x2 = (i + 1) * w;
			float y2 = (j + 1) * h;
```

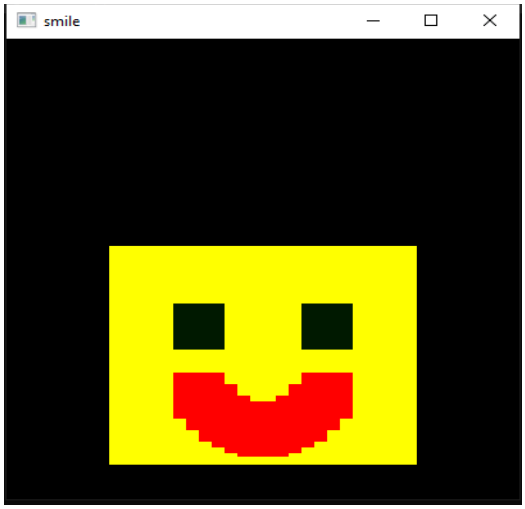
```
                        glBegin(GL_POLYGON);
                        glVertex2f(x1, y1);
                        glVertex2f(x1, y2);
                        glVertex2f(x2, y2);
                        glVertex2f(x2, y1);
                        glEnd();
                }
        }

        glFlush();
}

void reshape(int w, int h)
{
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
}

int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(400, 400);
        glutInitWindowPosition(150, 150);
        glutCreateWindow("puncte multiple");
        init();
        glutDisplayFunc(Display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```




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

#include <math.h>

void init()
{
	//glPointSize(10.0);
}

void Display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1, 1, 1);

	bool negru = true;
	int w = 50, h = 50;

	for (int i = 0; i < 8; ++i)
```

```
	{
		for (int j = 0; j < 8; ++j)
		{
			//schimba alternativ culoarea
			if (negru)
				glColor3f(1, 1, 1);
			else
				glColor3f(0, 0, 0);

			negru = !negru;
			float x1 = i * w;
			float y1 = j * h;
			float x2 = (i + 1) * w;
			float y2 = (j + 1) * h;

			glBegin(GL_POLYGON);
			glVertex2f(x1, y1);
			glVertex2f(x1, y2);
			glVertex2f(x2, y2);
			glVertex2f(x2, y1);
			glEnd();
		}

		negru = !negru; //schimba culoarea la sfarsitul randului
	}

	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
}

int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(600, 600);
	glutInitWindowPosition(150, 150);
	glutCreateWindow("sah");
	glutDisplayFunc(Display);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```





Aplicatie cu meniuri si obiecte 3D

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

static int window;
static int menu_id;
static int submenu_id;
static int value = 0;
void menu(int num) {
	if (num == 0) {
```

```
                glutDestroyWindow(window);
                exit(0);
        }
        else {
                value = num;
        }
        glutPostRedisplay();
}
void createMenu(void) {
        submenu_id = glutCreateMenu(menu);
        glutAddMenuEntry("Sphere", 2);
        glutAddMenuEntry("Torus", 4);
        glutAddMenuEntry("Teapot", 5);
        glutAddMenuEntry("Cube", 6);
        glutAddMenuEntry("Cone", 7);
        glutAddMenuEntry("Tetrahedron", 8);
        menu_id = glutCreateMenu(menu);
        glutAddSubMenu("Draw", submenu_id);
        glutAddMenuEntry("Clear", 1);
        glutAddMenuEntry("Exit", 0);     glutAttachMenu(GLUT_RIGHT_BUTTON);
}
int GAngle = 0;
void display(void) {
        glClear(GL_COLOR_BUFFER_BIT);   if (value == 1) {
                return; //glutPostRedisplay();
        }
        else if (value == 2) {
                static float alpha = 20;
                glPushMatrix();
                glColor3d(1.0, 1.0, 0.0);
                glRotated(alpha, -1.0, 0.0, 0.0);
                glutWireSphere(0.5, 50, 50);
                glPopMatrix();
                alpha = alpha + 0.5;
        }
        else if (value == 4) {
                static float alpha = 20;

                glPushMatrix();
                glColor3d(0.0, 1.0, 1.0);
                glRotatef(alpha, 1.9, 0.6, 0);
                glutSolidTorus(0.3, 0.6, 100, 100);
                glPopMatrix();
                alpha = alpha + 0.5;
        }
        else if (value == 5) {
                static float alpha = 20;
                glPushMatrix();
                glColor3d(0.5, 1.0, 0.5);
                glRotatef(alpha, 1.9, 0.6, 0);
                glutSolidTeapot(0.5);
                glPopMatrix();
                alpha = alpha + 0.5;
        }
        else if (value == 6) {
                glColor3f(0.0, 0.5, 0.0);
                glLoadIdentity();
                glRotated(GAngle, 1, 1, 0);
                glutWireCube(0.5);
                GAngle = GAngle + 1;
        }
        else if (value == 7) {
```

```
                static float alpha = 20;
                glPushMatrix();
                glColor3d(1.0, 1.0, 1.0);
                glRotated(alpha, -1.0, 0.0, 0.0);
                glutWireCone(0.5, 1.0, 50, 50);
                glPopMatrix();
                alpha = alpha + 0.5;
        }
        else if (value == 8) {
                static float alpha = 20;
                glPushMatrix();
                glColor3d(0.0, 1.0, 0.0);
                glRotated(alpha, 0.0, 1.0, 0.0);
                glutWireTetrahedron();
                glPopMatrix();
                alpha = alpha + 0.5;
        }
        glFlush();
}
void Timer(int extra) {
        glutPostRedisplay();
        glutTimerFunc(40, Timer, 0);
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("Aplicatie 05.11.2022");
        createMenu();
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glutDisplayFunc(display);
        glutTimerFunc(0, Timer, 0);  glutMainLoop();

        return 0;
}
```