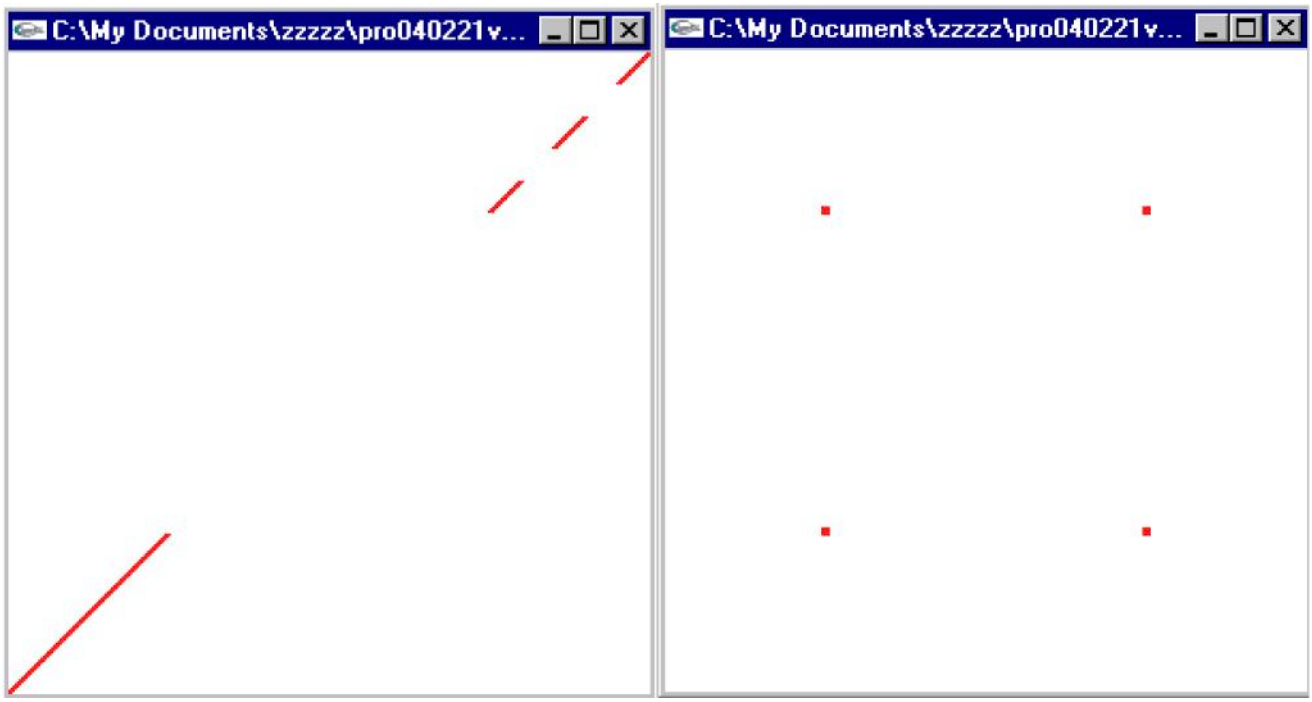
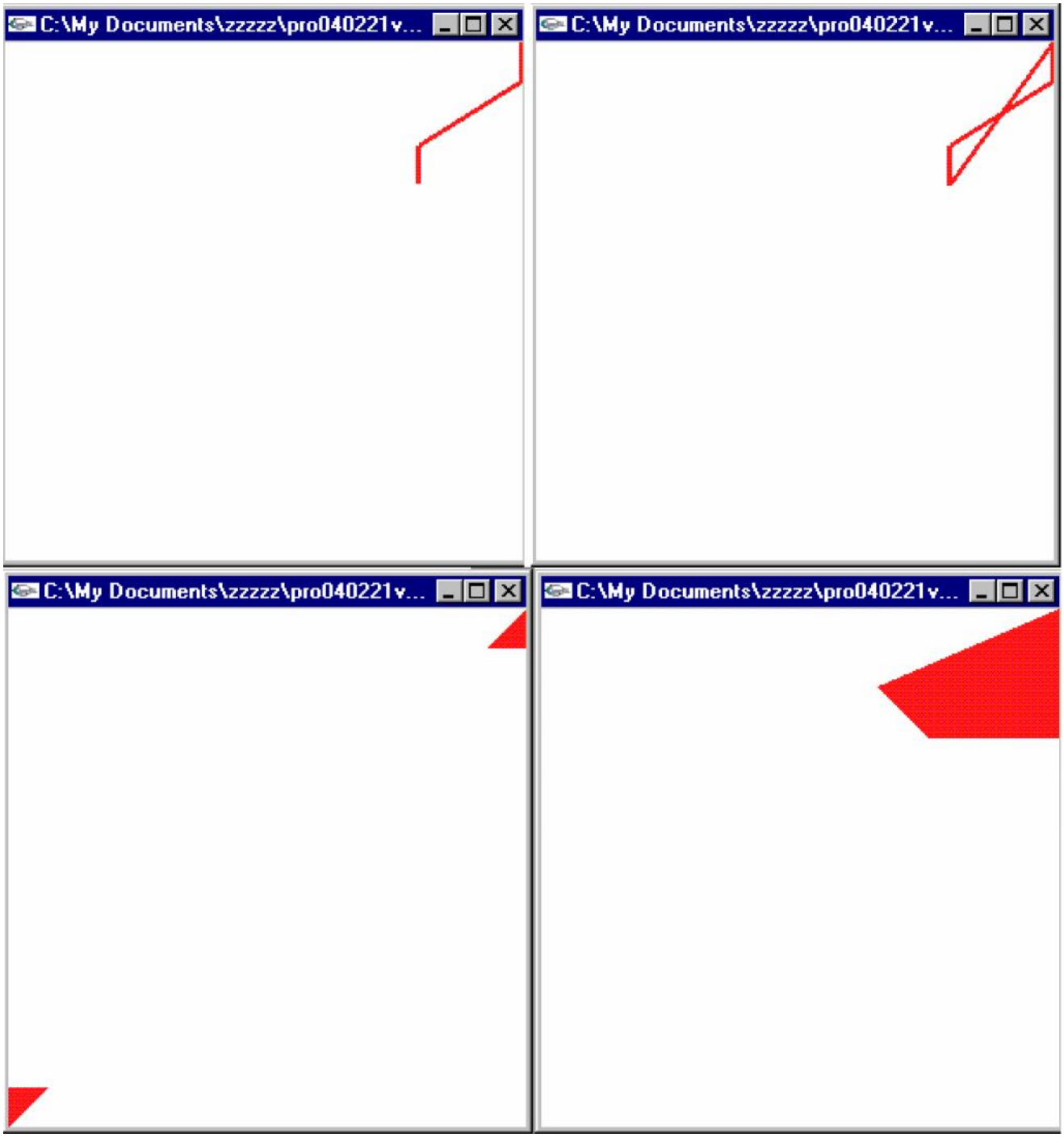
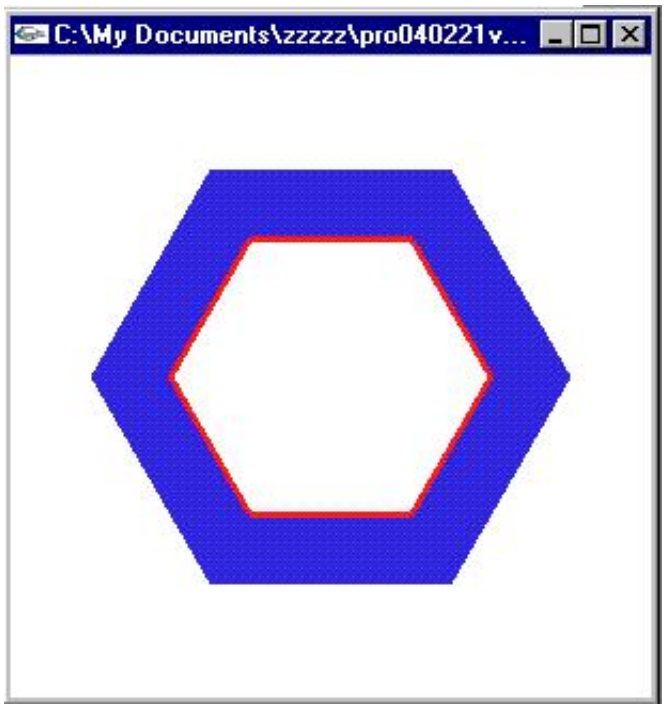
Tema 1.

Biblioteca OpenGL (si utilitarul GLUT). Notiuni introductive.

1. In exemplul urmator sunt folosite functiile de control ale ferestrei de afisare, functii callback (functiile pentru controlul afisarii, tastaturii, mouse-ului,), primitivele geometrice OpenGL (puncte, linii, poligoane), modelele de culori OpenGL.
2. In exemplul precedent completati codul functiilor Display3, Display4, Display5,
   Display6, Display7, Display8 astfel incat prin apelarea acestor functii sa obtineti
   urmatoarele figuri:




C:\My Documents\zzzzz\pro040221v...
C:\My Documents\zzzzz\pro040221v...
C:\My Documents\zzzzz\pro040221v...
C:\My Documents\zzzzz\pro040221v...



Intrebari, etc. : ghirvu@infoiasi.ro

```
// Daca se doreste utilizarea bibliotecii GLUT trebuie
// inclus fisierul header GL/glut.h (acesta va include
// la GL/gl.h si GL/glu.h, fisierele header pentru
// utilizarea bibliotecii OpenGL). Functiile din biblioteca
// OpenGL sunt prefixate cu gl, cele din GLU cu glu si
// cele din GLUT cu glut.
#include <stdlib.h>
#include <stdio.h>

#include <glut.h>
#include <gl\GL.h>
#include <math.h>

unsigned char prevKey;

void Display1() {
    glColor3f(0.2,0.15,0.88); // albastru
    glBegin(GL_LINES); // trasarea unei linii
    glVertex2i(1,1); // coordonatele unui varf
    glVertex2i(-1,-1);
    glEnd();

    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINES);
    glVertex2i(-1,1);
    glVertex2i(1,-1);
    glEnd();

    glBegin(GL_LINES);
    glVertex2d(-0.5,0);
    glVertex2d(0.5,0);
    glEnd();
}

void Display2() {
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINES);
    glVertex2f(1.0,1.0);
    glVertex2f(0.9,0.9);
    glVertex2f(0.8,0.8);
    glVertex2f(0.7,0.7);
    glVertex2f(0.6,0.6);
    glVertex2f(0.5,0.5);
    glVertex2f(-0.5,-0.5);
    glVertex2f(-1.0,-1.0);
    glEnd();
}

void Display3() {
    // trasare puncte GL_POINTS : deseneaza n puncte
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_POINTS);
    // de completat ...
```

```
    glVertex2f(0.5,0.5);
    glVertex2f(-0.5,-0.5);
    glVertex2f(-0.5,0.5);
    glVertex2f(0.5,-0.5);

    glEnd();
}

void Display4() {
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINES); // trasarea unei linii
    glVertex2f(1,1); // coordonatele unui varf
    glVertex2f(1,0.85);
    glVertex2f(0.5,0.6); // coordonatele unui varf
    glVertex2f(0.5,0.45);
    glEnd();
    // trasare linie poligonala GL_LINE_STRIP : (v0,v1), (v1,v2), (v_{n-2},v_{n-1})
    glBegin(GL_LINE_STRIP);
    // de completat ...
    glVertex2f(1,0.85); // coordonatele unui varf
    glVertex2f(0.5,0.6);
    glEnd();
}

void Display5() {
    glColor3f(1,0.1,0.1); // rosu
    // trasare linie poligonala inchisa GL_LINE_LOOP : (v0,v1), (v1,v2), (v_{n-1},v0)
    glBegin(GL_LINE_LOOP);
    glVertex2f(1,1); // coordonatele unui varf
    glVertex2f(1,0.85);
    glVertex2f(0.5,0.6); // coordonatele unui varf
    glVertex2f(0.5,0.45);
    // de completat ...
    glEnd();
}

void Display6() {
    glColor3f(1,0.1,0.1); // rosu
    // trasare triunghiuri GL_TRIANGLES : (v0,v1,v2), (v3,v4,v5), ...
    glBegin(GL_TRIANGLES);
    glVertex3f(1,1,0);
    glVertex3f(1,0.85,0);
    glVertex3f(0.85,0.85,0);
    glVertex3f(-1,-1,0);
    glVertex3f(-1,-0.85,0);
    glVertex3f(-0.85,-0.85,0);

    // de completat ...
    glEnd();
}

void Display7() {
    // trasare patrulatere GL_QUADS : (v0,v1,v2,v3), (v4,v5,v6,v7), ...
```

```
    glBegin(GL_QUADS);
    // de completat ...
    glColor3f(1,0.1,0.1);//rosu
    glVertex2f(1,1);
    glVertex2f(1,0.6);
    glVertex2f(0.6,0.6);
    glVertex2f(0.4,0.75);
    glEnd();
}

void Display8() {
    // trasare poligon convex GL_QUADS : (v0,v1,v2, ..., v_{n-1})
    glBegin(GL_POLYGON);
    glColor3f(0.2,0.15,0.88);//albastru

    glVertex3f(-0.4,0.6,0);
    glVertex3f(0.4,0.6,0);
    glVertex3f(0.7,0,0);
    glVertex3f(0.4,-0.6,0);
    glVertex3f(-0.4,-0.6,0);
    glVertex3f(-0.7,0,0);
    // de completat ...
    glEnd();

    glBegin(GL_POLYGON);
        glColor3f(1,0.1,0.1);//rosu

        glVertex3f(0.25,0.41,0);
        glVertex3f(0.48,0,0);
        glVertex3f(0.25,-0.41,0);
        glVertex3f(-0.25,-0.41,0);
        glVertex3f(-0.48,0,0);
        glVertex3f(-0.25,0.41,0);
    glEnd();

    //glBegin(GL_POLYGON);
    //  glColor3f(1,1,1);//alb

    //  glVertex3f(0.24,0.40,0);
    //  glVertex3f(0.466,0,0);
    //  glVertex3f(0.24,-0.40,0);
    //  glVertex3f(-0.24,-0.40,0);
    //  glVertex3f(-0.466,0,0);
    //  glVertex3f(-0.24,0.40,0);
    //glEnd();
    glScalef(0.75,0.75,0.75);
    glBegin(GL_POLYGON);
    glColor3f(1,1,1);//albastru

    glVertex3f(-0.4,0.6,0);
    glVertex3f(0.4,0.6,0);
    glVertex3f(0.7,0,0);
    glVertex3f(0.4,-0.6,0);
```

```
    glVertex3f(-0.4,-0.6,0);
    glVertex3f(-0.7,0,0);
    // de completat ...
    glEnd();
}

void Init(void) {
    // specifica culoarea unui buffer dupa ce acesta
    // a fost sters utilizand functia glClear. Ultimul
    // argument reprezinta transparenta (1 - opacitate
    // completa, 0 - transparenta totala)
    glClearColor(1.0,1.0,1.0,1.0);

    // grosimea liniilor
    glLineWidth(3);

    // dimensiunea punctelor
    glPointSize(4);

    // functia void glPolygonMode (GLenum face, GLenum mode)
    // controleaza modul de desenare al unui poligon
    // mode : GL_POINT (numai vf. primitivei) GL_LINE (numai
    //        muchiile) GL_FILL (poligonul plin)
    // face : tipul primitivei geometrice dpdv. al orientarii
    //        GL_FRONT - primitive orientate direct
    //        GL_BACK  - primitive orientate invers
    //        GL_FRONT_AND_BACK  - ambele tipuri
    glPolygonMode(GL_FRONT, GL_LINE);
}

void Display(void) {
    printf("Call Display\n");

    // sterge buffer-ul indicat
    glClear(GL_COLOR_BUFFER_BIT);

    switch(prevKey) {
    case '1':
        Display1();
        break;
    case '2':
        Display2();
        break;
    case '3':
        Display3();
        break;
    case '4':
        Display4();
        break;
    case '5':
        Display5();
        break;
    case '6':
```

```
        Display6();
        break;
    case '7':
        Display7();
        break;
    case '8':
        Display8();
        break;
    default:
        break;
    }

    // forteaza redesenarea imaginii
    glFlush();
}

/*
Parametrii w(latime) si h(inaltime) reprezinta noile
dimensiuni ale ferestrei
*/
void Reshape(int w, int h) {
    printf("Call Reshape : latime = %d, inaltime = %d\n", w, h);

    // functia void glViewport (GLint x, GLint y,
    //                          GLsizei width, GLsizei height)
    // defineste poarta de afisare : acea suprafata dreptunghiulara
    // din fereastra de afisare folosita pentru vizualizare.
    // x, y sunt coordonatele pct. din stg. jos iar
    // width si height sunt latimea si inaltimea in pixeli.
    // In cazul de mai jos poarta de afisare si fereastra coincid
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
}

/*
Parametrul key indica codul tastei iar x, y pozitia
cursorului de mouse
*/
void KeyboardFunc(unsigned char key, int x, int y) {
    printf("Ati tastat <%c>. Mouse-ul este in pozitia %d, %d.\n",
        key, x, y);
    // tasta apasata va fi utilizata in Display ptr.
    // afisarea unor imagini
    prevKey = key;
    if (key == 27) // escape
        exit(0);
    glutPostRedisplay();
}

/*
Codul butonului poate fi :
GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
Parametrul state indica starea: "apasat" GLUT_DOWN sau
"eliberat" GLUT_UP
```

```
Parametrii x, y : coordonatele cursorului de mouse
*/
void MouseFunc(int button, int state, int x, int y) {
    printf("Call MouseFunc : ati %s butonul %s in pozitia %d %d\n",
        (state == GLUT_DOWN) ? "apasat" : "eliberat",
        (button == GLUT_LEFT_BUTTON) ?
        "stang" :
    ((button == GLUT_RIGHT_BUTTON) ? "drept": "mijlociu"),
        x, y);
}

int main(int argc, char** argv) {
    // Initializarea bibliotecii GLUT. Argumentele argc
    // si argv sunt argumentele din linia de comanda si nu
    // trebuie modificate inainte de apelul functiei
    // void glutInit(int *argcp, char **argv)
    // Se recomanda ca apelul oricarei functii din biblioteca
    // GLUT sa se faca dupa apelul acestei functii.
    glutInit(&argc, argv);

    // Argumentele functiei
    // void glutInitWindowSize (int latime, int latime)
    // reprezinta latimea, respectiv inaltimea ferestrei
    // exprimate in pixeli. Valorile predefinite sunt 300, 300.
    glutInitWindowSize(300, 300);

    // Argumentele functiei
    // void glutInitWindowPosition (int x, int y)
    // reprezinta coordonatele varfului din stanga sus
    // al ferestrei, exprimate in pixeli.
    // Valorile predefinite sunt -1, -1.
    glutInitWindowPosition(100, 100);

    // Functia void glutInitDisplayMode (unsigned int mode)
    // seteaza modul initial de afisare. Acesta se obtine
    // printr-un SAU pe biti intre diverse masti de display
    // (constante ale bibliotecii GLUT) :
    // 1. GLUT_SINGLE : un singur buffer de imagine. Reprezinta
    //    optiunea implicita ptr. nr. de buffere de
    //    de imagine.
    // 2. GLUT_DOUBLE : 2 buffere de imagine.
    // 3. GLUT_RGB sau GLUT_RGBA : culorile vor fi afisate in
    //    modul RGB.
    // 4. GLUT_INDEX : modul indexat de selectare al culorii.
    // etc. (vezi specificatia bibliotecii GLUT)
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

    // Functia int glutCreateWindow (char *name)
    // creeaza o fereastra cu denumirea data de argumentul
    // name si intoarce un identificator de fereastra.
    glutCreateWindow (argv[0]);

    Init();
```

```
    // Functii callback : functii definite in program si
    // inregistrate in sistem prin intermediul unor functii
    // GLUT. Ele sunt apelate de catre sistemul de operare
    // in functie de evenimentul aparut

    // Functia
    // void glutReshapeFunc (void (*Reshape)(int width, int height))
    // inregistreaza functia callback Reshape care este apelata
    // oridecate ori fereastra de afisare isi modifica forma.
    glutReshapeFunc(Reshape);

    // Functia
    // void glutKeyboardFunc (void (*KeyboardFunc)(unsigned char,int,int))
    // inregistreaza functia callback KeyboardFunc care este apelata
    // la actionarea unei taste.
    glutKeyboardFunc(KeyboardFunc);

    // Functia
    // void glutMouseFunc (void (*MouseFunc)(int,int,int,int))
    // inregistreaza functia callback MouseFunc care este apelata
    // la apasarea sau la eliberarea unui buton al mouse-ului.
    glutMouseFunc(MouseFunc);

    // Functia
    // void glutDisplayFunc (void (*Display)(void))
    // inregistreaza functia callback Display care este apelata
    // oridecate ori este necesara desenarea ferestrei: la
    // initializare, la modificarea dimensiunilor ferestrei
    // sau la apelul functiei
    // void glutPostRedisplay (void).
    glutDisplayFunc(Display);

    // Functia void glutMainLoop() lanseaza bucla de procesare
    // a evenimentelor GLUT. Din bucla se poate iesi doar prin
    // inchiderea ferestrei aplicatiei. Aceasta functie trebuie
    // apelata cel mult o singura data in program. Functiile
    // callback trebuie inregistrate inainte de apelul acestei
    // functii.
    // Cand coada de evenimente este vida atunci este executata
    // functia callback IdleFunc inregistrata prin apelul functiei
    // void glutIdleFunc (void (*IdleFunc) (void))
    glutMainLoop();

    return 0;
}
```

Tema 2.

Utilizarea bibliotecii OpenGL pentru trasarea curbelor plane.

1. In exemplul urmator am utilizat primitiva grafica OpenGL de trasare a liniilor pentru a trasa
   1. graficul functiei : $|\sin x| \cdot e^{-\sin x}, x \in [0,8\pi]$ si
   2. graficul concoidei lui Nicomede (concoida dreptei) :
      $x = a \pm b \cdot \cos t, \quad y = a \cdot \operatorname{tg} t \pm b \cdot \sin t, \quad t \in (-\pi/2, \pi/2)$.
2. Integrati in exemplul precedent functii C care realizeaza :
   1. afisarea functiei :

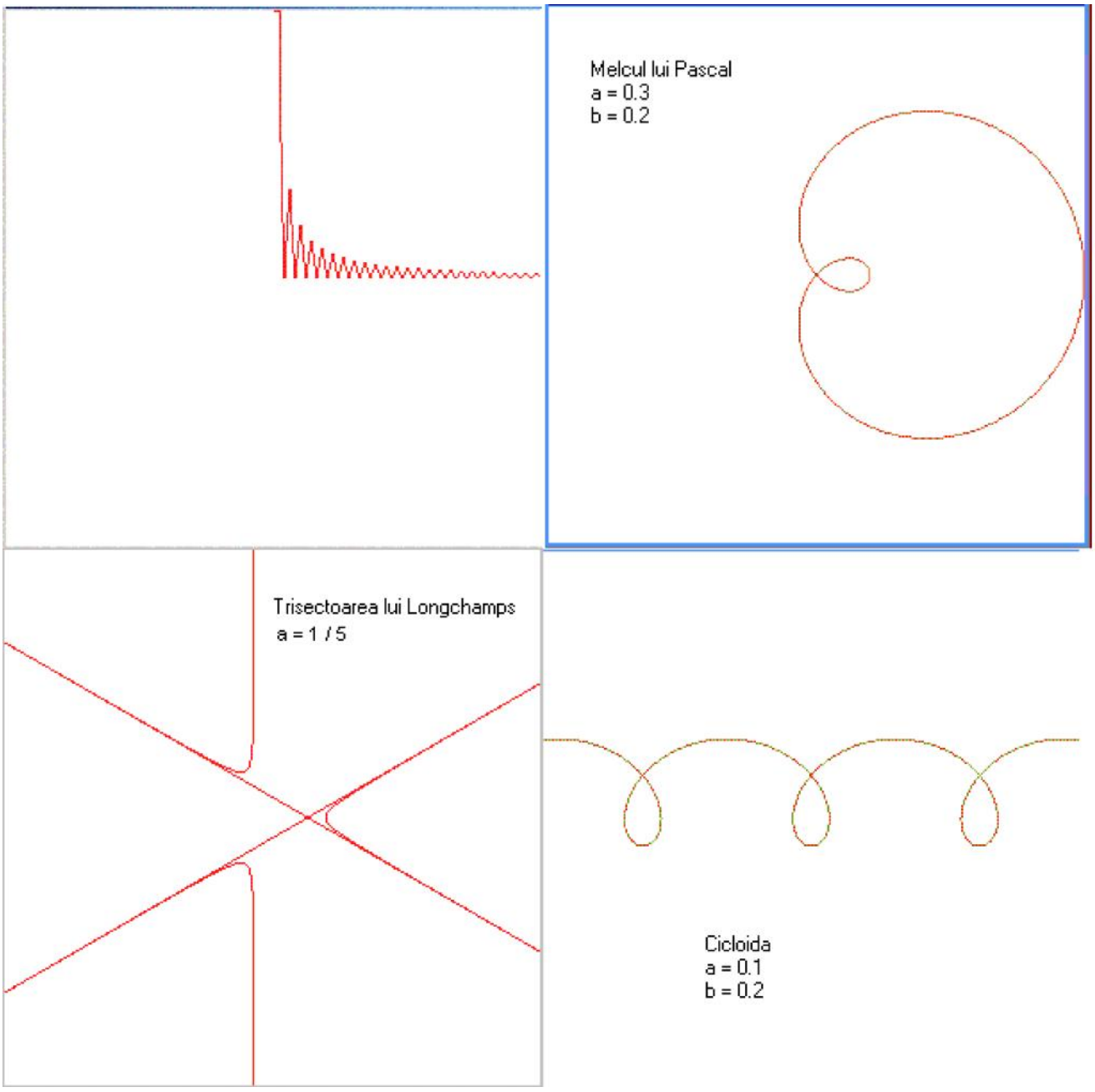
$$f(x) = \begin{cases} 1, & \text{pentru } x = 0 \\ \dfrac{d(x)}{x}, & \text{pentru } x > 0 \end{cases},$$

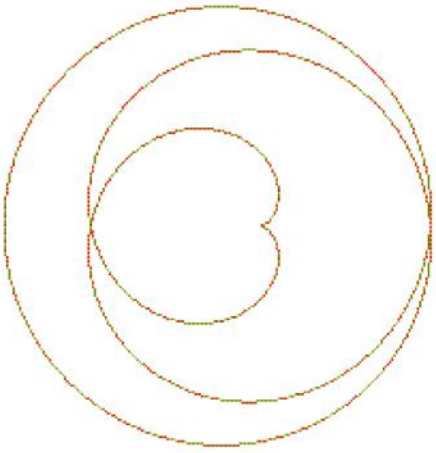
      unde *d(x)* este distanta de la x la cel mai apropiat intreg, pe intervalul [0,100].
   2. afisarea urmatoarelor curbe date prin ecuatii parametrice :
      1. melcul lui Pascal (concoida cercului) :
         $x = 2 \cdot (a \cdot \cos t + b) \cdot \cos t, \quad y = 2 \cdot (a \cdot \cos t + b) \cdot \sin t, \quad t \in (-\pi, \pi)$
      2. trisectoarea lui Longchamps :
         $$x = \frac{a}{4 \cdot \cos^2 t - 3}, \quad y = \frac{a \cdot \operatorname{tg} t}{4 \cdot \cos^2 t - 3}, \quad t \in (-\pi/2, \pi/2) \setminus \{\pm \pi/6\}$$
      3. cicloida :
         $x = a \cdot t - b \cdot \sin t, \quad y = a - b \cdot \cos t, \quad t \in \Re$
      4. epicicloida :
         $$x = (R + r) \cdot \cos\left(\frac{r}{R} \cdot t\right) - r \cdot \cos\left(t + \frac{r}{R} \cdot t\right),$$
         $$y = (R + r) \cdot \sin\left(\frac{r}{R} \cdot t\right) - r \cdot \sin\left(t + \frac{r}{R} \cdot t\right), \quad t \in [0, 2\pi]$$
      5. hipocicloida :
         $$x = (R - r) \cdot \cos\left(\frac{r}{R} \cdot t\right) - r \cdot \cos\left(t - \frac{r}{R} \cdot t\right),$$
         $$y = (R - r) \cdot \sin\left(\frac{r}{R} \cdot t\right) - r \cdot \sin\left(t - \frac{r}{R} \cdot t\right), \quad t \in [0, 2\pi]$$
3. Curbe date de ecuatii polare : coordonatele polare sunt *(r,t)*, unde $t \in [a,b]$ iar *r=f(t)*.
   Transformarea in coordonate carteziene a coordonatelor polare *(r,t)* este
   $x = r \cdot \cos t$
   $y = r \cdot \sin t$
   Sa se reprezinte urmatoarele curbe date prin ecuatii polare:
   1. lemniscata lui Bernoulli :

$$r = \pm a \cdot \sqrt{2 \cdot \cos(2 \cdot t)}, \quad t \in (-\pi/4, \pi/4)$$

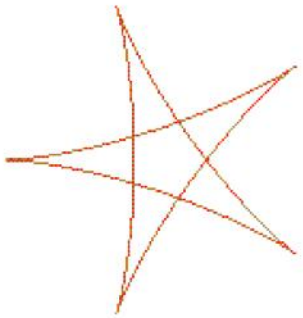
2. spirala logaritmica :

$$r = a \cdot e^{1+t}, \quad t \in (0, \infty)$$

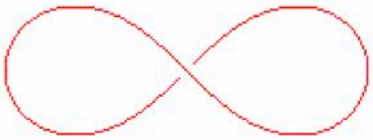




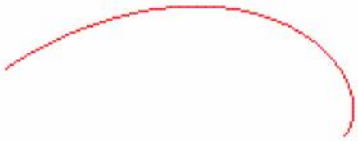
Epicicloida
R = 0.1 r = 0.3



Hipocicloida
R = 0.1
r = 0.3



Lemniscata lui Bernoulli
a = 0.4



Spirala logaritmica
a = 0.02

rebari, etc. : ghirvu@infoiasi.ro

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <limits>

#include <glut.h>

// dimensiunea ferestrei in pixeli
#define dim 300

unsigned char prevKey;

// concoida lui Nicomede (concoida dreptei)
// $x = a + b \cdot cos(t), y = a \cdot tg(t) + b \cdot sin(t)$. sau
// $x = a - b \cdot cos(t), y = a \cdot tg(t) - b \cdot sin(t)$. unde
// $t \in (-\pi / 2, \pi / 2)$
void Display1() {
    double xmax, ymax, xmin, ymin;
    double a = 1, b = 2;
    double pi = 4 * atan(1.0);
    double ratia = 0.05;
    double t;

    // calculul valorilor maxime/minime ptr. x si y
    // aceste valori vor fi folosite ulterior la scalare
    xmax = a - b - 1;
    xmin = a + b + 1;
    ymax = ymin = 0;
    for (t = - pi/2 + ratia; t < pi / 2; t += ratia) {
        double x1, y1, x2, y2;
        x1 = a + b * cos(t);
        xmax = (xmax < x1) ? x1 : xmax;
        xmin = (xmin > x1) ? x1 : xmin;

        x2 = a - b * cos(t);
        xmax = (xmax < x2) ? x2 : xmax;
        xmin = (xmin > x2) ? x2 : xmin;

        y1 = a * tan(t) + b * sin(t);
        ymax = (ymax < y1) ? y1 : ymax;
        ymin = (ymin > y1) ? y1 : ymin;

        y2 = a * tan(t) - b * sin(t);
        ymax = (ymax < y2) ? y2 : ymax;
        ymin = (ymin > y2) ? y2 : ymin;
    }

    xmax = (fabs(xmax) > fabs(xmin)) ? fabs(xmax) : fabs(xmin);
    ymax = (fabs(ymax) > fabs(ymin)) ? fabs(ymax) : fabs(ymin);

    // afisarea punctelor propriu-zise precedata de scalare
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
```

```
    for (t = - pi/2 + ratia; t < pi / 2; t += ratia) {
        double x1, y1, x2, y2;
        x1 = (a + b * cos(t)) / xmax;
        x2 = (a - b * cos(t)) / xmax;
        y1 = (a * tan(t) + b * sin(t)) / ymax;
        y2 = (a * tan(t) - b * sin(t)) / ymax;

        glVertex2f(x1,y1);
    }
    glEnd();

    glBegin(GL_LINE_STRIP);
    for (t = - pi/2 + ratia; t < pi / 2; t += ratia) {
        double x1, y1, x2, y2;
        x1 = (a + b * cos(t)) / xmax;
        x2 = (a - b * cos(t)) / xmax;
        y1 = (a * tan(t) + b * sin(t)) / ymax;
        y2 = (a * tan(t) - b * sin(t)) / ymax;

        glVertex2f(x2,y2);
    }
    glEnd();
}

// graficul functiei
// $f(x) = \bar sin(x) \bar \cdot e^{-sin(x)}, x \in \langle 0, 8 \cdot \pi \rangle$,
void Display2() {
    double pi = 4 * atan(1.0);
    double xmax = 8 * pi;
    double ymax = exp(1.1);
    double ratia = 0.05;

    // afisarea punctelor propriu-zise precedata de scalare
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for (double x = 0; x < xmax; x += ratia) {
        double x1, y1;
        x1 = x / xmax;
        y1 = (fabs(sin(x)) * exp(-sin(x))) / ymax;

        glVertex2f(x1,y1);
    }
    glEnd();
}
void Display3() {
    double ratia = 0.05;
    double xmax = 100;
    double ceilValue, floorValue;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for (double x = 0; x <=xmax; x += ratia) {
        double x1=x/100, y1;
        if(x==0)
```

```
            y1 = 1;
        else {
            ceilValue = ceil(x)-x;
            floorValue = x-floor(x);
            if(floorValue<ceilValue){
                y1 = floorValue/x;
            } else {
                y1 = ceilValue/x;
            }
        }

        glVertex2f(x1,y1);
    }
    glEnd();
}
void Display4() {
    double xmax = 100;
    double ratia = 0.05;
    double pi = 4 * atan(1.0);
    double t,x1,y1, a= 0.3, b= 0.2;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
        for(t = -pi+ratia; t < pi; t+=ratia){
            x1 = 2*(a*cos(t)+b)*cos(t);
            y1 = 2*(a*cos(t)+b)*sin(t);
            glVertex2f(x1,y1);
        }

    glEnd();
}
void Display5() {
    double xmax = 100;
    double ratia = 0.05;
    double pi = 4 * atan(1.0);
    double piPe2 = pi/2;
    double piPe6 = pi/6;
    double t,x1,y1, a= 0.2;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for(t = -piPe2+ratia; t < piPe2; t+=ratia){
        if(t!=piPe6 || t!=(-piPe6)){
            x1 = a/(4*pow(cos(t),2)-3);
            y1 = (a*tan(t))/(4*pow(cos(t),2)-3);
        }

        glVertex2f(x1,y1);
    }

    glEnd();
}
void Display6() {
    double xmax = 100;
    double ratia = 0.05;
```

```
    double pi = 4 * atan(1.0);
    double t,x1,y1, a=0.1, b=0.2;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for(t = -(4*pi); t <= (4*pi); t+=ratia){
        x1 = a*t-b*sin(t);
        y1 = a-b*cos(t);
        glVertex2f(x1,y1);
    }

    glEnd();
}
void Display7() {
    double xmax = 100;
    double ratia = 0.05;
    double pi = 4 * atan(1.0);
    double t,x1,y1, R=0.1, r=0.3;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for(t = 0; t <= (2*pi); t+=ratia){
        x1 = (R+r)*cos((r/R)*t)-r*cos(t+(r/R)*t);
        y1 = (R+r)*sin((r/R)*t)-r*sin(t+(r/R)*t);
        glVertex2f(x1,y1);
    }

    glEnd();
}
void Display8() {
    double xmax = 100;
    double ratia = 0.05;
    double pi = 4 * atan(1.0);
    double t,x1,y1, R=0.1, r=0.3;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for(t = 0; t <= (2*pi); t+=ratia){
        x1 = (R-r)*cos((r/R)*t)-r*cos(t-(r/R)*t);
        y1 = (R-r)*sin((r/R)*t)-r*sin(t-(r/R)*t);
        glVertex2f(x1,y1);
    }

    glEnd();
}
void Display9() {
    double xmax = 100;
    double ratia = 0.005;
    double pi = 4 * atan(1.0);
    double piPe4 = pi/4;
    double t,x1,y1, a=0.4,r;
    glColor3f(1,0.1,0.1); // rosu
    glBegin(GL_LINE_STRIP);
    for(t = piPe4-ratia; t > -piPe4; t-=ratia){
        r=a*sqrt(2*cos(2*t));
        x1 =r*cos(t);
```

```
        y1 =r*sin(t);
        glVertex2f(x1,y1);
    }
    for(t = -piPe4+ratia; t < piPe4; t+=ratia){
        r=-a*sqrt(2*cos(2*t));
        x1 =r*cos(t);
        y1 =r*sin(t);
        glVertex2f(x1,y1);
    }

    glEnd();
}
void Display10() {
    double xmax = 100;
    double ratia = 0.05;
    double pi = 4 * atan(1.0);
    double piPe4 = pi/4;
    double t,x1,y1, a=0.02,r;
    glColor3f(1,0.1,0.1);
    glBegin(GL_LINE_STRIP);
    for(t = 0+ratia; t < (9999*pi); t+=ratia){
        r=a*exp(1+t);
        x1 =r*cos(t);
        y1 =r*sin(t);
        glVertex2f(x1,y1);
    }
    glEnd();
}
void Init(void) {

    glClearColor(1.0,1.0,1.0,1.0);

    glLineWidth(1);

    //   glPointSize(4);

    glPolygonMode(GL_FRONT, GL_LINE);
}

void Display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    switch(prevKey) {
    case '1':
        Display1();
        break;
    case '2':
        Display2();
        break;
    case '3':
        Display3();
        break;
    case '4':
```

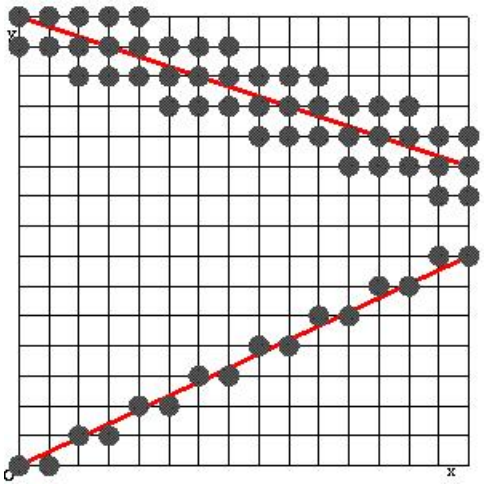
```
        Display4();
        break;
    case '5':
        Display5();
        break;
    case '6':
        Display6();
        break;
    case '7':
        Display7();
        break;
    case '8':
        Display8();
        break;
    case '9':
        Display9();
        break;
    case '0':
        Display10();
        break;
    default:
        break;
    }

    glFlush();
}

void Reshape(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
}

void KeyboardFunc(unsigned char key, int x, int y) {
    prevKey = key;
    if (key == 27) // escape
        exit(0);
    glutPostRedisplay();
}

void MouseFunc(int button, int state, int x, int y) {
}

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

    glutInit(&argc, argv);

    glutInitWindowSize(dim, dim);

    glutInitWindowPosition(100, 100);

    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

    glutCreateWindow (argv[0]);
```

```
    Init();

    glutReshapeFunc(Reshape);

    glutKeyboardFunc(KeyboardFunc);

    glutMouseFunc(MouseFunc);

    glutDisplayFunc(Display);

    glutMainLoop();

    return 0;
}
```

Tema 3.

Desenarea primitivelor grafice 2D pe ecrane rastru.

1. Implementati o clasa GrilaCarteziana prin intermediul careia sa puteti desena o grila carteziana patratica 2D cu urmatoarele caracteristici:
    1. Numarul de linii/coloane sunt parametri ai grilei,
    2. Liniile si coloanele grilei sunt egal spatiate,
    3. In varfurile grilei (intersectiile dintre linii si coloane) sa fie desenati pixeli avand o forma circulara (pot avea si alte forme, patratice de exemplu, dar formele circulare vor primi punctaj maxim),
    4. Pixelii sa fie disjuncti,
    5. Un pixel (i,j) sa fie aprins prin apelul unei metode writePixel avand cel putin 2 argumente de tip intreg: linia i si coloana j.
2. Implementati algoritmul prezentat la curs pentru trasarea unui segment de dreapta ale carui extremitati au coordonate intregi (vezi imaginea). **Vor primi punctaj maxim acele rezolvari care implementeaza algoritmul `AfisareSegmentDreapta3` (modificand-ul corespunzator si explicand aceste modificari).**



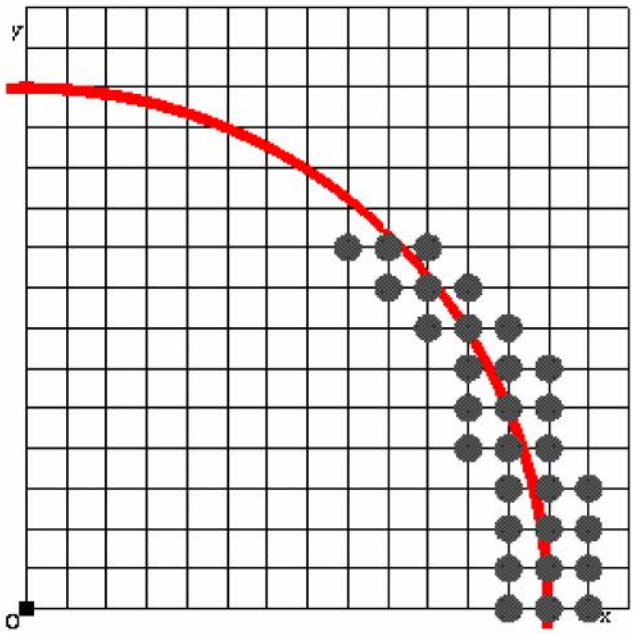
Intrebari, etc. : ghirvu@info.uaic.ro

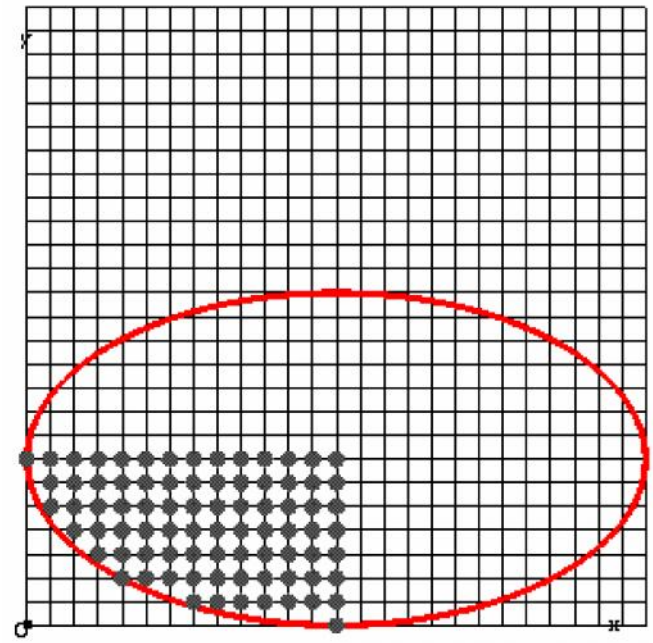
Tema 4

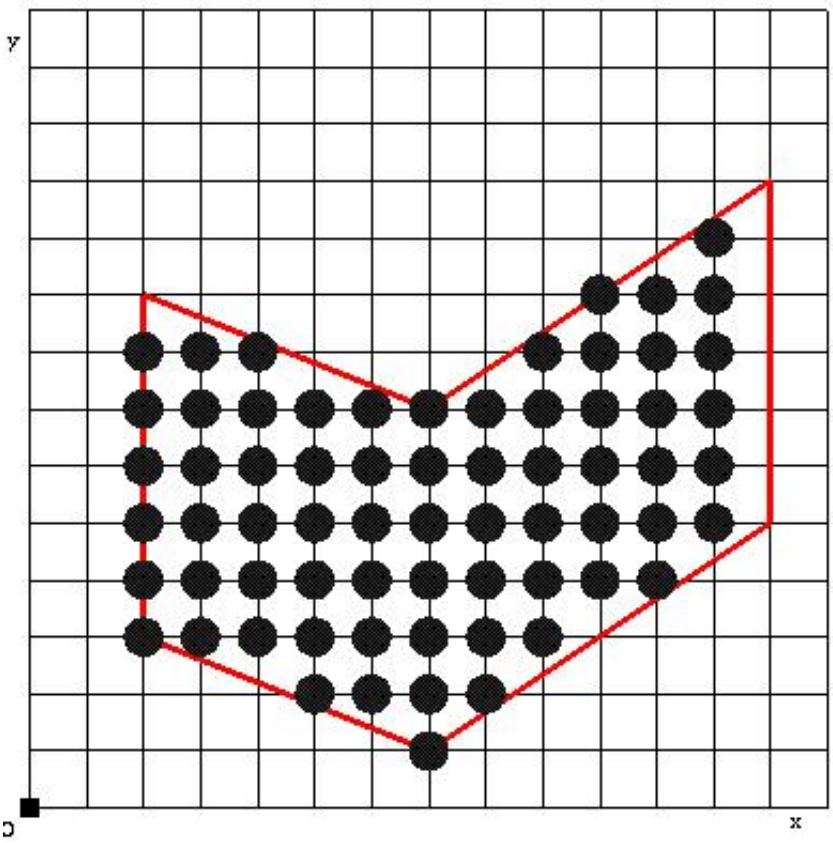
Desenarea primitivelor grafice 2D pe ecrane rastru.

1. La curs a fost prezentat un algoritm pentru trasarea unui cerc cu centrul in origine si de raza din Z. Mai intai se generau pixelii din octantul al 2-lea si ulterior prin simetrie fata de O, Ox, Oy si bisectoare toti pixelii cercului. Modificati algoritmul astfel incat sa fie generati doar pixelii din primul octant si utilizati o tehnica de ingrosare a primitivelor pentru a obtine imaginea.
   Vor primi punctaj maxim acele rezolvari care implementeaza algoritmul `AfisareCerc4` (modificand-ul corespunzator si explicand aceste modificari).
2. La curs a fost prezentat un algoritm pentru colorarea uniforma a unei elipse (avand centrul in origine si semiaxe din Z): se genereaza mai intai pixelii din cadranul 1 si apoi, prin simetrie fata de O, Ox si Oy pixelii din celelalte cadrane. Modificati algoritmul prezentat astfel incat sa fie generati mai intai pixelii din cadranul al 3-lea (vezi imaginea).
   Vor primi punctaj maxim acele rezolvari care modifica algoritmul 11 `UmplereElipsa` dar pastreaza aceleasi principii de obtinere ale extremitatilor segmentelor de scanare maximale.
3. Implementati algoritmul prezentat la curs pentru colorarea pixelilor care sunt interiori unui poligon (vezi imaginea). Varfurile poligonului se vor citi dintr-un fisier. Fisierul va avea urmatorul format: pe prima linie va fi numarul de varfuri si apoi, pe linii consecutive, coordonatele x si y ale varfurilor. Ordinea varfurilor V1, V2, ..., Vn are urmatoarea semnificatie: muchiile poligonului sunt V1V2, V2V3, ..., VnV1.
   De exemplu, pentru poligonul din imagine, fisierul de intrare ar putea fi:

```
6
2 3
7 1
13 5
13 11
7 7
2 9
```


y
O
x


y
O
x



rebari, etc. : ghirvu@info.uaic.ro

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <list>
#include <vector>
#include "glut.h"

using namespace std;

#define dimensiuneFereastra 600
#define NO_LINII_DEFAULT 15
#define NO_COLOANE_DEFAULT 15

unsigned char prevKey;

class Punct{
private:
    int X;
    int Y;
public:
    Punct(int x, int y){
        this->X = x;
        this->Y = y;
    }
    void setX(int x){
        this->X = x;
    }
    void setY(int y){
        this->Y = y;
    }
    int getX(){
        return this->X;
    }
    int getY(){
        return this->Y;
    }
};
list<Punct*> MPixeliDreapta;
list<Punct*> MPixeliCerc;
list<Punct*> MPixeliElipsa;
class Cerc{
private:
    Punct* centru;
    int raza;
public:
    Cerc(Punct* pCentru, int pRaza){
        this->centru = pCentru;
        this->raza = pRaza;
    }
    Punct* getCentru() const { return centru; }
    void setCentru(Punct* val) { centru = val; }
    int getRaza() const { return raza; }
```

```
    void setRaza(int val) { raza = val; }
};
class Elipsa{
private:
    Punct* centru;
    int raza1;
    int raza2;
public:
    Elipsa(Punct* pCentru, int pRaza1, int pRaza2){
        this->centru = pCentru;
        this->raza1 = pRaza1;
        this->raza2 = pRaza2;
    }
    Punct* getCentru() const { return centru; }
    void setCentru(Punct* val) { centru = val; }
    int getRaza1() const { return raza1; }
    void setRaza1(int val) { raza1 = val; }
    int getRaza2() const { return raza2; }
    void setRaza2(int val) { raza2 = val; }
};
class SegmentOrizontal{
public:
    int xMin;
    int xMax;
    int y;
    SegmentOrizontal(int y,int xmin, int xmax){
        this->xMax = xmax;
        this->xMin = xmin;
        this->y = y;
    }
};
list<SegmentOrizontal*> MSegmente;
class GrilaCarteziana{
private:
    int mLinii;
    int mColoane;
    int mDeltaPixeliPerLinie;
    int mDeltaPixeliPerColoana;
protected:
public:
    GrilaCarteziana(){
        this->mLinii = NO_LINII_DEFAULT;
        this->mColoane = NO_COLOANE_DEFAULT;
        this->initializari();
    }
    GrilaCarteziana(int pLinii, int pColoane){
        this->mLinii = pLinii;
        this->mColoane = pColoane;
        this->initializari();
    }
    void initializari(){
        mDeltaPixeliPerLinie = dimensiuneFereastra/(mLinii+1);
        mDeltaPixeliPerColoana = dimensiuneFereastra/(mColoane+1);
```

```
}
void writePixel(int atX, int atY){
    float x,y;
    float PI = 4*atan(1.0);
    float radius = 10;
    float delta_theta = 0.01;
    glColor3f(0,0,0);
    glPolygonMode(GL_FRONT, GL_FILL);
    glBegin( GL_POLYGON );{
        for( float angle = 0; angle < 2*PI; angle += delta_theta )
            glVertex2f( atX+radius*cos(angle),atY+radius*sin(angle));
    }glEnd();
}
void writePixels(list<Punct*> m){
    list<Punct*>::const_iterator iterator;
    for(iterator = m.begin(); iterator!=m.end(); iterator++){
        this->writePixel((*iterator)->getX()*mDeltaPixeliPerLinie, (*iterator)->getY()*
        mDeltaPixeliPerColoana);
    }
}
void writeLinePixels(list<SegmentOrizontal*> segments){

    list<SegmentOrizontal*>::const_iterator iterator;
    for(iterator = segments.begin(); iterator!=segments.end(); iterator++){
        int xMax, xMin, aux;
        if( (*iterator)->xMin < (*iterator)->xMax){
            xMin = (*iterator)->xMin;
            xMax = (*iterator)->xMax;
        } else {
            xMin = (*iterator)->xMax;
            xMax = (*iterator)->xMin;
        }
        for(int j =xMin; j<=xMax; j++){
            this->writePixel(j*mDeltaPixeliPerLinie,(*iterator)->y*mDeltaPixeliPerColoana);
        }
    }
}

void writeRedLine(int fromX, int fromY, int toX, int toY){
    glColor3f(1,0,0);
    glBegin(GL_LINES);{
        glVertex2i(fromX*mDeltaPixeliPerLinie, fromY*mDeltaPixeliPerColoana);
        glVertex2i(toX*mDeltaPixeliPerLinie, toY*mDeltaPixeliPerColoana);
    }glEnd();
}
void writeCircle(Cerc* pCerc){
    float PI = 4*atan(1.0);
    float radius = pCerc->getRaza()*mDeltaPixeliPerColoana;
    float delta_theta = 0.01;
    glColor3f(1,0,0);
    glPolygonMode(GL_FRONT, GL_LINE);
    glBegin( GL_POLYGON );{
```

```
            for( float angle = 0; angle < 2*PI; angle += delta_theta )
                glVertex2f( pCerc->getCentru()->getX()*mDeltaPixeliPerLinie+radius*cos(angle),
                            pCerc->getCentru()->getY()*mDeltaPixeliPerColoana+radius*sin(angle));
        }glEnd();
    }
    void writeElipse(Elipsa* pElipsa){
        float PI = 4*atan(1.0);
        float radius1 = pElipsa->getRaza1()*mDeltaPixeliPerLinie;
        float radius2 = pElipsa->getRaza2()*mDeltaPixeliPerColoana;
        float delta_theta = 0.01;
        glColor3f(1,0,0);
        glPolygonMode(GL_FRONT, GL_LINE);
        glBegin( GL_POLYGON );{
            for( float angle = 0; angle < 2*PI; angle += delta_theta )
                glVertex2f( pElipsa->getCentru()->getX()*mDeltaPixeliPerLinie+radius1*cos(angle),
                            pElipsa->getCentru()->getY()*mDeltaPixeliPerColoana+radius2*sin(
                            angle));
        }glEnd();
    }
    void draw(){
        glColor3f(0,0,0);
        for(int i = -mLinii-1 ; i<=mLinii; i++){
            glBegin(GL_LINES);{
                glVertex2i(-dimensiuneFereastra*mDeltaPixeliPerLinie,i*mDeltaPixeliPerLinie);
                glVertex2i(dimensiuneFereastra+mDeltaPixeliPerLinie,i*mDeltaPixeliPerLinie);
            }glEnd();
        }
        for(int i = -mLinii-1 ; i<=mLinii; i++){
            glBegin(GL_LINES);{
                glVertex2i(i*mDeltaPixeliPerColoana,-dimensiuneFereastra+mDeltaPixeliPerColoana
                );
                glVertex2i(i*mDeltaPixeliPerColoana, dimensiuneFereastra+mDeltaPixeliPerColoana);
            }glEnd();
        }
    }

};
void AfisarePuncteCerc3(int x, int y){
    MPixeliCerc.push_back(new Punct(x,y));
    MPixeliCerc.push_back(new Punct(x+1,y));
    MPixeliCerc.push_back(new Punct(x-1,y));
    /*
    MPixeliCerc.push_back(new Punct(-x,-y));
    MPixeliCerc.push_back(new Punct(-x,y));
    MPixeliCerc.push_back(new Punct(x,-y));;
    if(x != y){
        MPixeliCerc.push_back(new Punct(y,x));
        MPixeliCerc.push_back(new Punct(-y,-x));
        MPixeliCerc.push_back(new Punct(-y,x));
        MPixeliCerc.push_back(new Punct(y,-x));
    }*/
}
void AfisareCerc4(Cerc* cerc, bool showGrid){
```

```
    GrilaCarteziana* grila = new GrilaCarteziana();
    if(showGrid){
        grila->draw();
    }
    grila->writeCircle(cerc);
    int raza = cerc->getRaza();
    int x = raza, y = 0;
    int d = 1- raza;
    int dN = 3, dNE = -2*raza+5;
    AfisarePuncteCerc3(x+cerc->getCentru()->getX(),y+cerc->getCentru()->getY());
    while(y!=x){
        if(d<0){
            d+=dN;
            dN+=2;
            dNE+=2;
        } else {
            d+=dNE;
            dN+=2;
            dNE += 4;
            x--;
        }
        y++;
        AfisarePuncteCerc3(x+cerc->getCentru()->getX(),y+cerc->getCentru()->getY());
    }
    grila->writePixels(MPixeliCerc);
    //free
    delete(cerc);
    delete(grila);

}
void UmplereElipsa(int x0, int y0, int a, int b){
    int newA = a-1;
    int newB = b-1;
    int xi = 0, x = 0, y = -newB;
    double fxpyp =0.0;
    double deltaV, deltaNV, deltaN;
    GrilaCarteziana* grila = new GrilaCarteziana();
    grila->draw();
    grila->writeElipse(new Elipsa(new Punct(x0, y0), a-1, b-1));
    MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
    while((double)newA*newA*((double)y-0.5)<(double)newB*newB*(x+1)){
        deltaV = (double)newB*newB*(-2*x+1);
        deltaNV = (double)newB*newB*(-2*x+1)+(double)newA*newA*(2*y+1);
        if(fxpyp+deltaV <=0.0){
            fxpyp +=deltaV;
            --x;
            list<SegmentOrizontal*>::const_iterator iterator;
            for(iterator = MSegmente.begin(); iterator!=MSegmente.end(); iterator++){
                if((*iterator)->y == y-y0){
                    (*iterator)->y = y-y0;
                    (*iterator)->xMin = x-x0;
                    (*iterator)->xMax = x0;
                }
```

```
            }
        } else if(fxpyp+deltaNV<=0.0){
            fxpyp += deltaNV;
            --x; ++y;
            MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
        }
    }
    while (y<0){
        deltaNV = (double)newB*newB*(-2*x+1)+(double)newA*newA*(2*y+1);
        deltaN = (double)newA*newA*(2*y+1);
        if(fxpyp+deltaNV<=0){
            fxpyp+=deltaNV;
            --x; ++y;
            MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
        } else {
            fxpyp += deltaN;
            ++y;
        }
        MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
    }
    grila->writeLinePixels(MSegmente);
}
void Init(void) {

   glClearColor(1.0,1.0,1.0,1.0);

   glLineWidth(1);

//   glPointSize(4);

   glPolygonMode(GL_FRONT, GL_LINE);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-dimensiuneFereastra*0.9f, dimensiuneFereastra*0.9f,
        -dimensiuneFereastra*0.9f, dimensiuneFereastra*0.9f);
}
void Display(void) {
   glClear(GL_COLOR_BUFFER_BIT);

   switch(prevKey) {
    case '1':
        AfisareCerc4(new Cerc(new Punct(0,0),10), true);
        break;
    case '2':
        UmplereElipsa(0,0,10,7);
        break;
   default:
      break;
   }

   glFlush();
}

void Reshape(int w, int h) {
```

```
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
}

void KeyboardFunc(unsigned char key, int x, int y) {
   prevKey = key;
   if (key == 27) // escape
      exit(0);
   glutPostRedisplay();
}

void MouseFunc(int button, int state, int x, int y) {
}

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

   glutInit(&argc, argv);

   glutInitWindowSize(dimensiuneFereastra, dimensiuneFereastra);

   glutInitWindowPosition(100, 100);

   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   glutCreateWindow (argv[0]);

   Init();

   glutReshapeFunc(Reshape);

   glutKeyboardFunc(KeyboardFunc);

   glutMouseFunc(MouseFunc);

   glutDisplayFunc(Display);

   glutMainLoop();

   return 0;
}
```