RAČUNARSKA GRAFIKA

Oznaka predmeta: RAG
Predavanje broj: 06

Nastavna jedinica: Rasterizacija. Fraktali. Geometrija: odnos tačke.

Nastavne teme:

Algoritam za liniju zasnovan na jednačini prave, DDA algoritam za liniju, Bresenhamov algoritam za liniju, algoritam za krug zasnovan na jednačini kružnice, Bresenhamov algoritam za krug. Fraktali: Van Koch pahuljica.
Geometrija. 2D primitivi. Inverzne transformacije: translacije, rotacije i skaliranja. Odnos tačke i prave. Presek pravolinijskih segmenata. Odnos tačke i poligona: realizacija, problemi, korekcija greške, poboljšanja.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

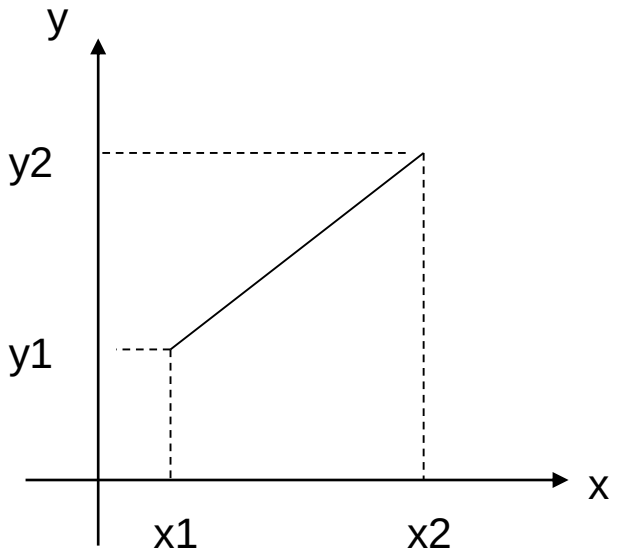
James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: "Computer Graphics: Principles and Practice", 2nd ed. in C, Addison-Wesley, 1996.

Algoritam za crtanje linije baziran na jednačini prave

$$y = mx + b$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

$$b = y_1 - mx_1$$




```
void linijaJP(int x1, int y1, int x2, int y2){
        double m, b;
        int x,y;
        m = (double)(y2-y1)/(double)(x2-x1);
        b = y1-m*x1;
        for(x=x1; x<=x2; x++){
                y=m*x+b;
                set_pixel(x,y);}
        }
}
```

DDA algoritam za liniju

$$y_{i+1} = mx_{i+1} + b$$

$$y_i = mx_i + b$$

$$odavde,$$

$$y_{i+1} = y_i + m$$

$$ili$$

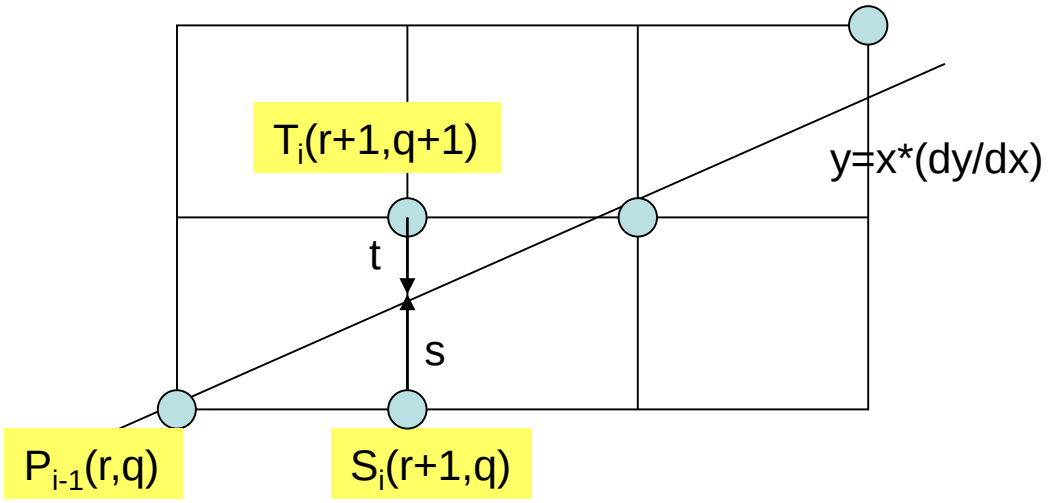
$$x_{i+1} = x_i + 1/m$$

- Kada je nagib
  m<1
  inkrementira se po x osi,
  tada se računa
  $y_{i+1}=y_i+m;$
  odnosno,
  kada je nagib
  m>=1
  inkrementira se po y osi
  tada se računa
  $x_{i+1}=x_i+1/m;$

```
//krajnje tacke A(x1,y1) B(x2,y2)
int dx,dy,steps,k;
double xinc,yinc,x,y;

dx=x2-x1; x=x1;
dy=y2-y1; y=y1;

if(fabs(dx)>fabs(dy)) steps=fabs(dx);
else                  steps=fabs(dy);

xinc=(double)dx/(double)steps;
yinc=(double)dy/(double)steps;

SET_PIXEL((int)x,(int)y);

for( k=1; k<=steps; k++){
  x += xinc;
  y += yinc;
  SET_PIXEL((int)x,(int)y);
}
```

Bresenhamov algoritam za liniju

- Razmatra se sledeći piksel koga bi trebalo upaliti
- Pretpostavke:
  - A(x1,y1) je bliža ishodištu
  - translirana je linija translacijom T(-x1,-y1), sada su:
    A'(0,0)
    B'(dx,dy)



- Sa slike, udaljenosti pixela od prave su:

$s = (dy/dx)(r+1)-q$

$t = q+1 - (dy/dx)(r+1)$

sada je:

$dx(s-t)=2(rdy-qdx)+2dy-dx$

Bresenhamov algoritam za liniju

- Označimo

  $d_i = dx(s_i - t_i)$

- Kako je $dx>0$, ako je $d_i<0$ onda odabiremo tačku S_i

- Sada je: $d_i = 2(rdy-qdx)+2dy-dx$

  odnoso,

  $d_i = 2(x_{i-1}dy-y_{i-1}dx)+2dy-dx$
  $d_{i+1} = 2(x_i dy-y_i dx)+2dy-dx$

  sređivanjem razlike:

  $d_{i+1} - d_i = 2dy(x_i-x_{i-1})-2dx(y_i-y_{i-1})$ uz $x_i-x_{i-1}=1$

  dobija se:

  $d_{i+1} = d_i+2dy - 2dx(y_i-y_{i-1})$

  sada je:

  odabiranjem tačke S_i t.j. za $d_i<0, y_i=y_{i-1}$ sledi $d_{i+1}= d_i+2dy$
  odabiranjem tačke T_i t.j. za $d_i\geq 0, y_i=y_{i-1}+1$ sledi $d_{i+1}= d_i+2(dy - dx)$

- Početni uslovi su: $d=2dy-dx$

Bresenhamov algoritam za liniju

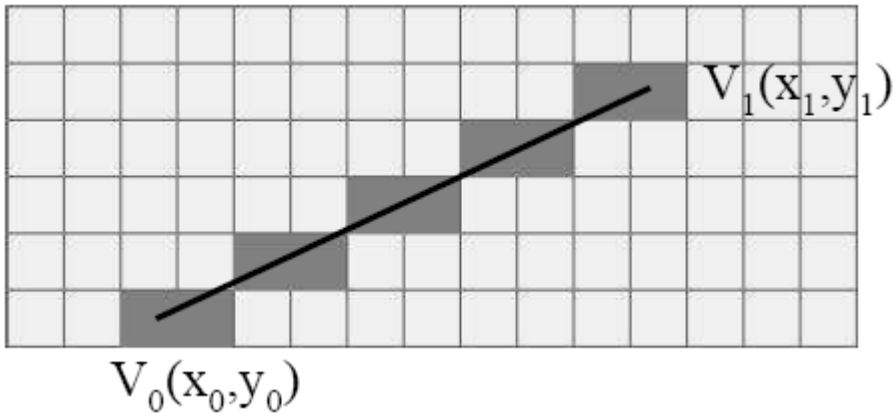
```
void lineBres(int x1, int y1, int x2, int y2)
{
   int dx,dy,incr1,incr2,d,x,y,xend;

   dx = abs(x2-x1);
   dy = abs(y2-y1);
   d = 2*dy-dx;

   incr1 = 2dy;            //za Si, donja tacka
   incr2 = 2*(dy-dx);      //za Ti, gornja tacka

   if(x1>x2) { x=x2; y=y2; xend=x1; }
   else      { x=x1; y=y1; xend=x2; }

   WRITE_PIXEL(x,y);

   while(x<xend)
   {
       x = x+1;
       if(d<0)      {d = d+incr1;              }
       else         {d = d+incr2; y = y+1;     }
       WRITE_PIXEL(x,y);
   }
}
```

Bresenhamov algoritam za liniju

- Prikaz dužine
  - Određuje koje tačke rastera trebaju biti osvetljene kako bi načinili prikaz ravne linije
  - postupak je ostvarljiv upotrebom celobrojnog sabiranja (oduzimanja) i pomaka
- Osnovni algoritam za nagib 0-45°
  - odabiremo celobrojne vrednosti koje odgovaraju središtima piksela $V_0(x_0,y_0)$, $V_1(x_1,y_1)$
  - jednačina pravca kroz V_0, V_1

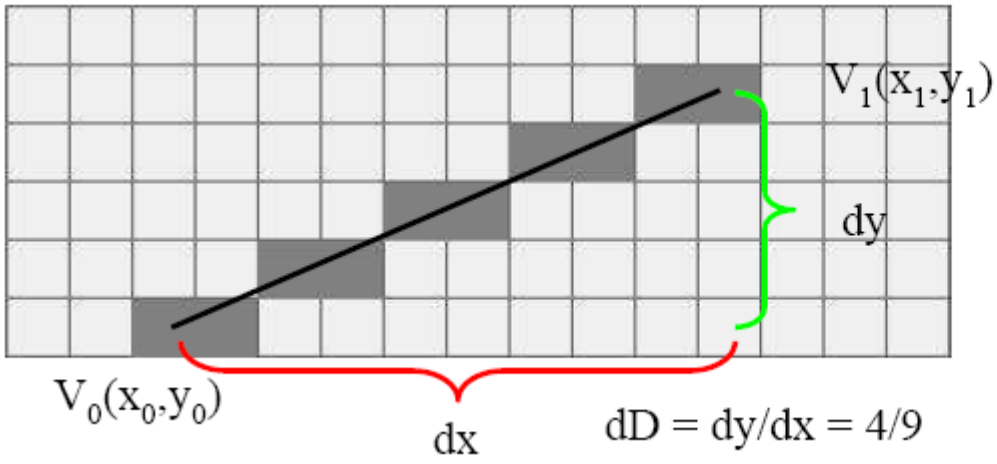
$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0), \qquad y = \frac{\Delta y}{\Delta x}(x - x_0) + y_0, \qquad \Delta y = y_1 - y_0, \quad \Delta x = x_1 - x_0$$

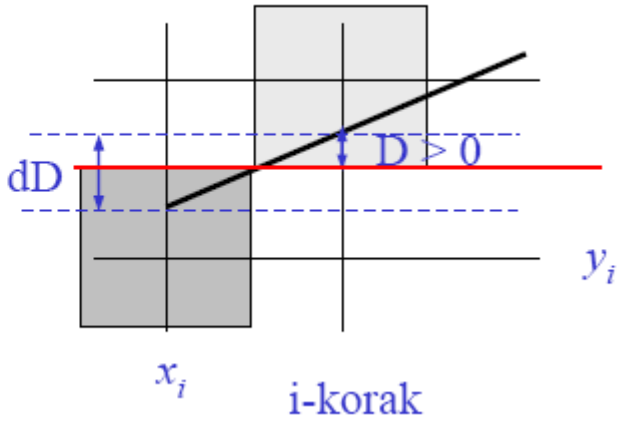
  - x određuje kolonu, a y red piksela koga treba osvetliti
  - potrebno je zaokruživanje (celobrojne koordinate)



Bresenhamov algoritam za liniju

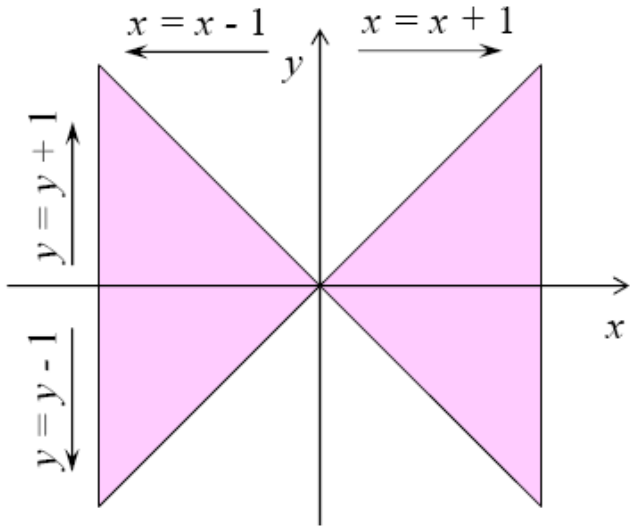
```
//osnovni algoritam Bresenhama za celobrojne koordinate x0,y0,x1,y1
void line(int x0, int y0, int x1, int y1) {
   int dx= x1-x0, dy = y1-y0;  //uzmimo da linija nije vertikalna
   float dD = fabs((float)dy / (float) dx); // tj. dx != 0
   float D = dD - 0.5;
   int y = y0;
   for (int x= x0; x<=x1; x++) {
        crtaj(x,y);
        if ( D ≥ 0 ) {
                y++;
                D--;
        }
        D = D + dD;
   }
}
```





Bresenhamov algoritam za liniju

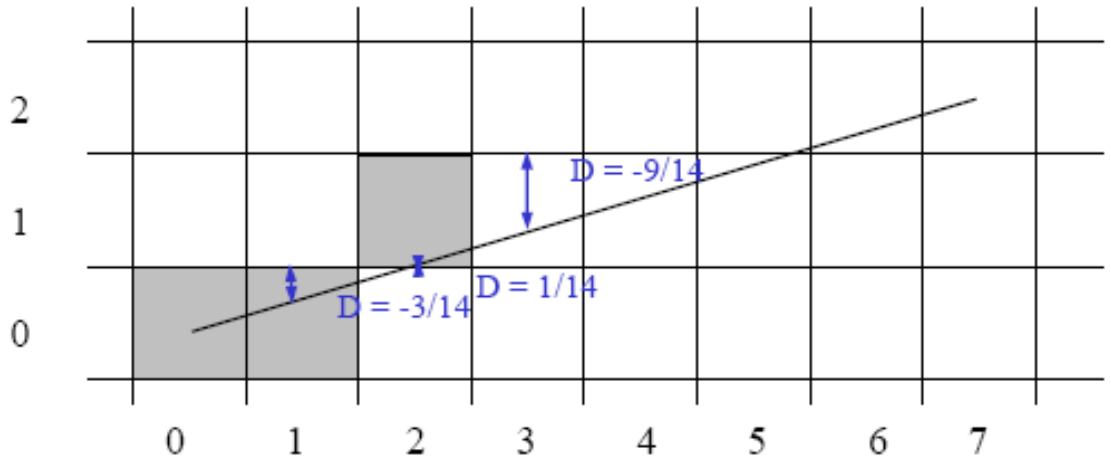
- Proširenje postupka 0-45 na sve nagibe
  - razlikuju se sledeća područja
    - x se povećava/smanjuje za 1
    - y se povećava/smanjuje za 1



- Primer:

  dx = 7, dy = 2
  dD = 2/7
  D = dD - 0.5 = 2/7- 1/2 = -3/14
  y = 0

```
for (x = 0 to 7)
{
   crtaj(x,y);
   if (D ≥ 0)
   {
       y++; D--;
   }
   D = D + dD;
}
```



Algoritam za kruznicu baziran na jednačini kružnice

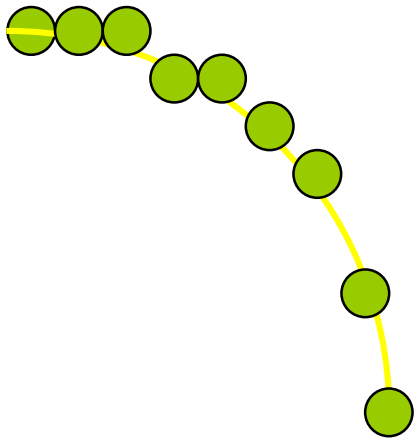
Jednačina kruga radijusa r sa centrom u (0,0) glasi:

$$x^2 + y^2 = r^2,$$

očigledno treba nacrtati:

$$y = \pm sqrt(r^2 - x^2)$$

za -r <= x <= r.



To funkcioniše, ali je neefikasno zbog množenja i kvadratnog korena. Takođe kreira velike greške u krugu za vrednosti x koje su blizu R.

Bolji pristup, koji je još uvek neefikasan, ali izbegava greške je crtanje:

$$x = r \cos ø$$

$$y = r \sin ø$$

tako da ø uzima vrednosti od 0 do 360 stepeni.

Bresenhamov algoritam za kružnicu

- Ideja je da se razmatra osmina kružnice, a onda da se ostale tačke zbog simetričnosti lako odrede.

Plot(x,y);

Plot(y,x);

Plot(y,-x);

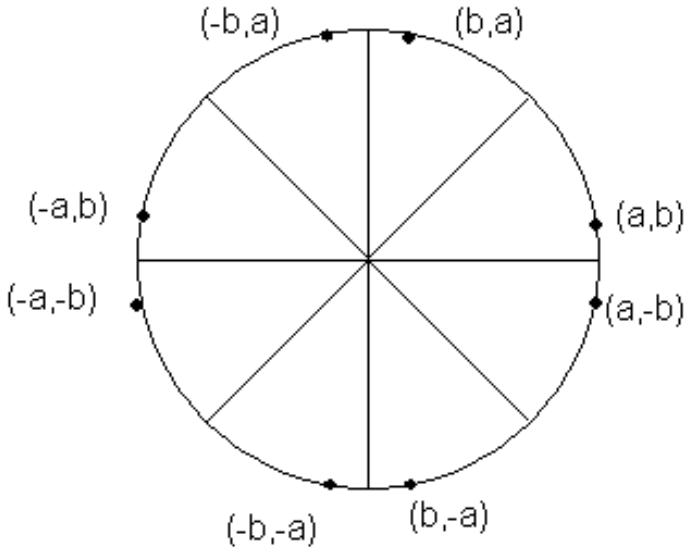
Plot(x,-y);

Plot(-x,-y);

Plot(-y,-x);

Plot(-y,x);

Plot(-x,y)

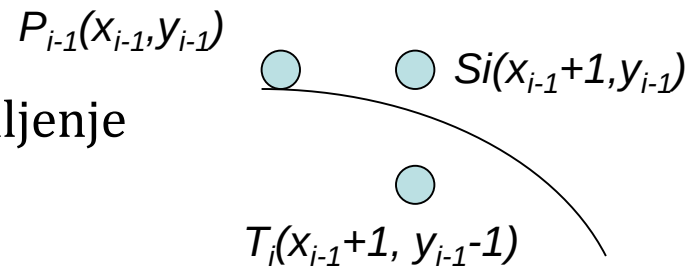


- Kao i kod crtanja linije razmatra se koji će se sledeći piksel upaliti.

Bresenhamov algoritam za kružnicu

- Tačka na kružnici zadovoljava:

$y_i^2 = r^2 - x_i^2$



gledano prema potencijalnim tačkama za paljenje

$S_i(x_{i-1}+1,y_{i-1}$) ili $T_i(x_{i-1}+1, y_{i-1}-1)$

traži se koja je od ovih tačaka bliža odgovarajućoj tački $P(x_i,y_i)$ na kružnici.

- Sada su razlike radijusa do tačaka S_i i T_i i radijusa r kao što sledi:

$s_i = ((x_{i-1}+1)^2+y_{i-1}^2) - (r^2)$

$t_i = (r^2) - ((x_{i-1}+1)^2 + (y_{i-1}-1)^2)$

$d_i = s_i - t_i = 2(x_{i-1}+1)^2+(y_{i-1})^2+(y_{i-1}-1)^2 - 2r^2$

odnosno,

$d_{i+1}=d_i+4x_{i-1}+6$ za $d_i<0$, S_i (tačka $x_{i+1} = x_i+1, y_{i+1} = y_i$)

$d_{i+1}=d_i+4(x_{i-1}-y_{i-1})+10$ za $d_i\geq 0$, T_i (tačka $x_{i+1} = x_i+ 1, y_{i+1} = y_i-1$)

početni uslovi su: x=0, y=r, tada je d=3-2r

Bresenhamov algoritam za kružnicu

```
void krugBresenham(int radius)
   //razmislite o krugu sa centrom u C(x1,y1)
{
   int x, y, d;
   x=0;
   y=radius;
   d=3-2*radius;
   while(x<=y)
   {
        circle_points(x,y); //paljenje 8 simetricnih tacaka
        if(d<0)
        {
               d=d+4*x+6;
        }
        else
        {
               d=d+4*(x-y)+10;
               y=y-1;
        }
        x=x+1;
    }
    circle_points(x,y);
}
```

Euklidska geometrija

- Standardni objekti (napravljeni ljudskom rukom) se mogu predstaviti Euklidskom geometrijom
- Opisani su jednačinama (funkcijama)
- Tako se dobiju glatki, pravilni objekti: lopte, poligoni, …
- Prirodni objekti (oblaci, lišće, stene) se bolje modeliraju korišćenjem fraktalne geometrije
- Objekti se predstavljaju procedurama umesto jednačinama
- Ponavljanjem procedure fraktala dobiju se sve kompleksniji detalji
- Karakteristika fraktala je neograničen proces ponovljenih transformacija invarijantne geometrijske forme.

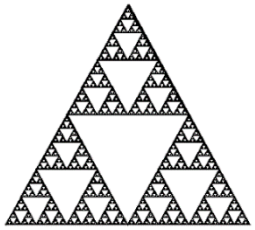
Kako su otkriveni fraktali?

- Zaposlenik IBMa, Benoit Mandelbrot bio je matematičar koji je ispitivao fluktuacije cena pamuka. Bez obzira na način analiziranja, podaci nikad nisu sledili normalnu distribuciju.
- Kad je Mandelbrot dobio sve podatke o cenama od 1900 i analizirao ih pomoću IBM računara, primetio je da brojevi koji su izazivali odstupanja od normalne distribucije dovode do simetrije skaliranja.
- Sekvenca promena je bila nezavisna od skale: krivulje za dnevne i za mesečne promene cena su se savršeno poklapale.

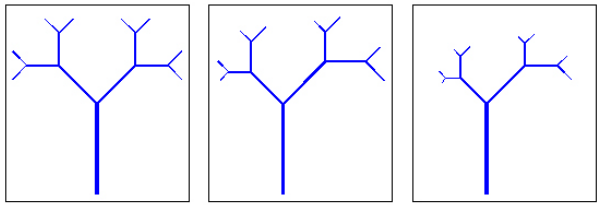
- Karakteristike fraktala:
  - U svakoj tački fraktala ima beskonačno mnogo detalja
  - Postoji sličnost između delova objekata i objekta kao celine
  - Dimenzije fraktala nisu celi brojevi (1D, 2D, 3D)
  - Nemaju određenu veličinu ili skalu

Podela fraktala

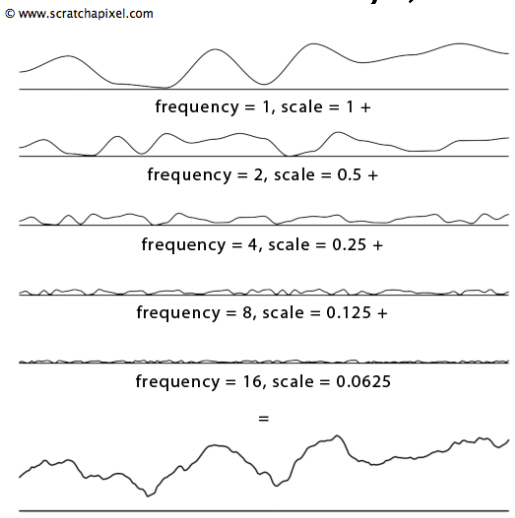
- Samoslični fraktali (*Self-similar*) delovi su umanjene verzije početnog objekta
  - Deterministički "self-similar“
  - Statistički "self-similar"



    - Sadrže određen stepen slučajnosti, vrše se slučajne varijacije na pod-delovima



    - Perlinov šum spada u kvazi-samoslične fraktale.
      - uvećavanjem slike uvek se dobija još veći nivo detalja,ali ipak nijedan deo slike nije sličan (u matematičkom smislu) celini.
      - Prikaz planina



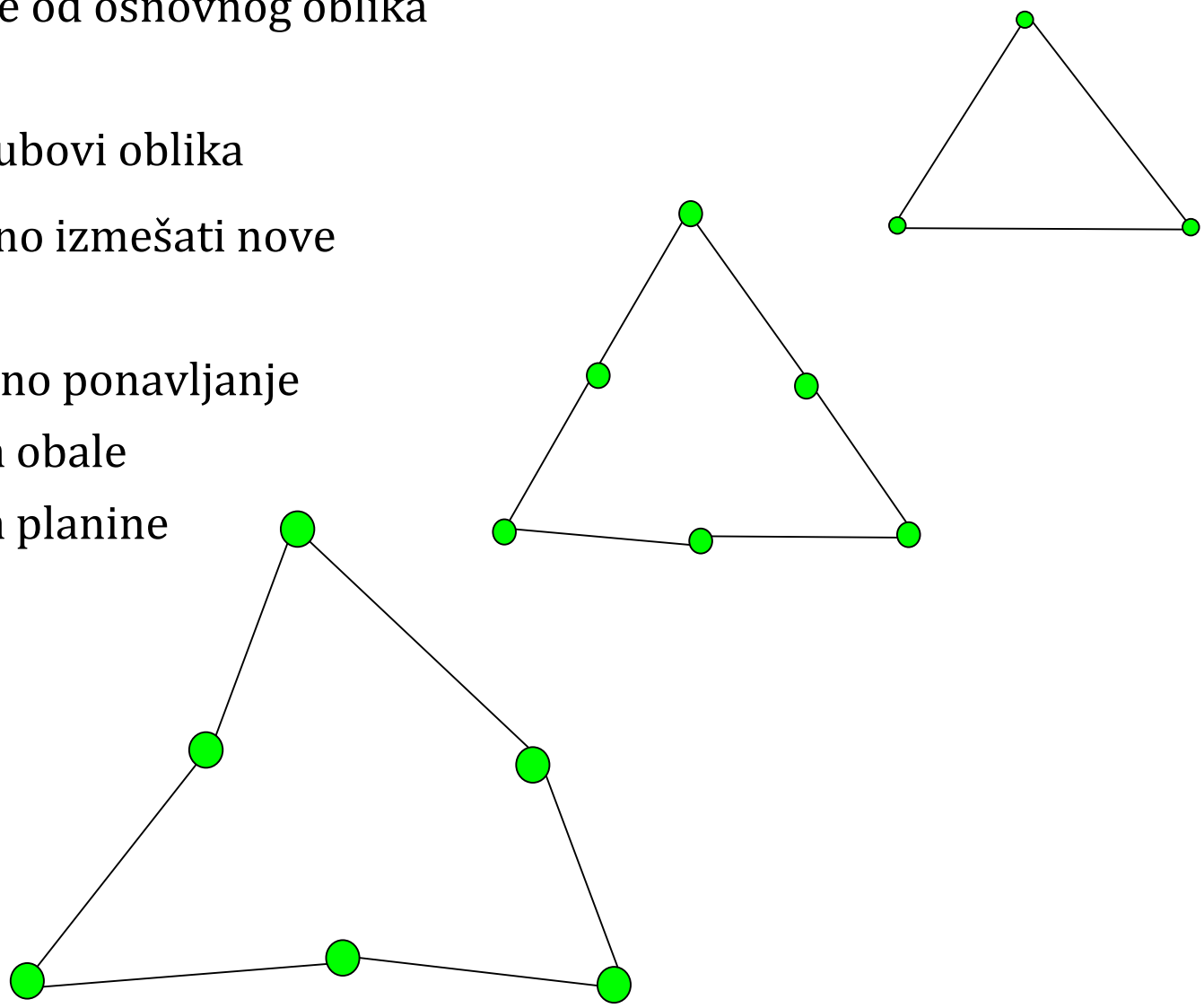
- Afini fraktali (*Self-affine*)
  - Različiti parametri skaliranja u različitim smerovima koordinata

Samoslični fraktali

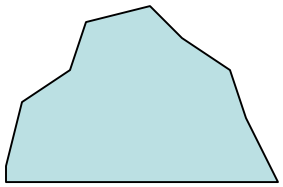
- Delovi su umanjene verzije celog objekta
  - Polazi se od početnog oblika
  - Kreiraju se pod-delovi dupliranjem i skaliranjem početnog oblika
- Za različite delove se mogu koristiti različiti faktori skaliranja
- Primer: von Koch pahuljica
  - Mogu se uvesti i slučajne varijacije
    - Ti fraktali su “statistički samoslični”
  - Koriste se za modeliranje drveća, lišća,...
- **Generisanje fraktala**
- Fraktal se generiše uzastopnim ponavljanjem određene transformacije
- Transformacija se može primeniti na set tačaka, set primitiva (linije, krivulje, boje, itd.), ili na bilo šta drugo
- Teoretski, procedura se primenjuje beskonačno mnogo puta
- Praktično se vrši iteracija konačan broj puta, do određene granice.

Fraktalne planine

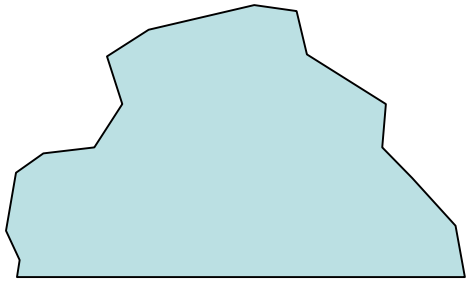
- Počinje se od osnovnog oblika planine
- Dele se rubovi oblika
- Nepravilno izmešati nove vrhove
- Rekurzivno ponavljanje
  - 2D za obale
  - 3D za planine



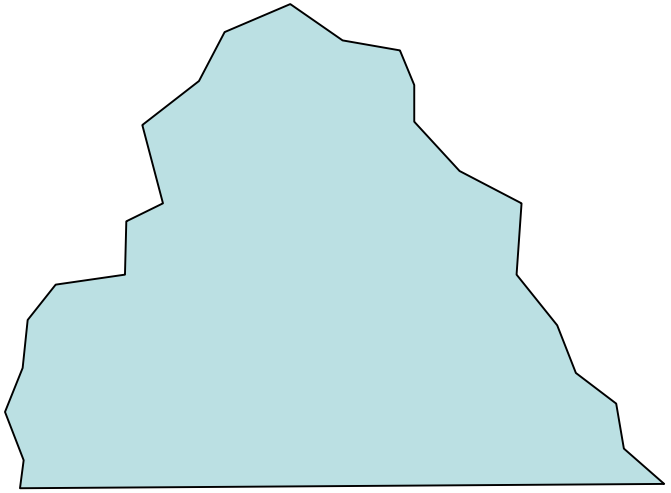
Fraktalne planine



Planina u daljini



Bliži pogled

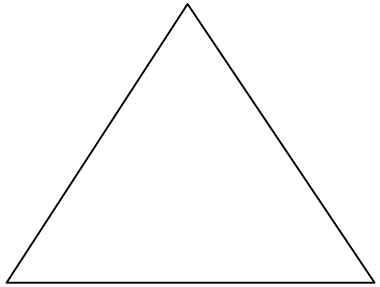


Još bliže

Što se više približimo, vidi se više detalja

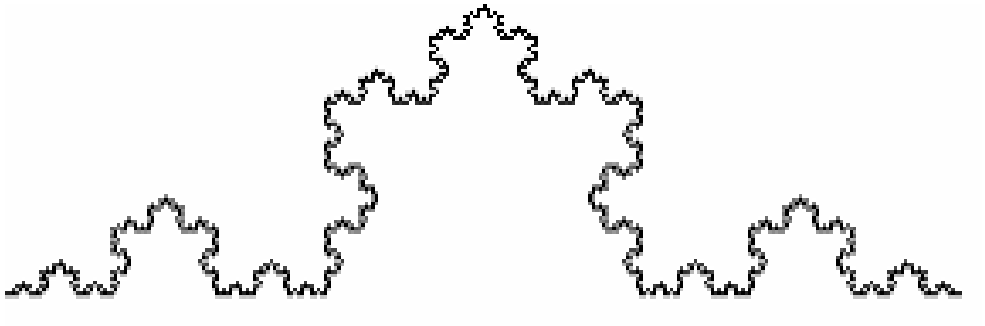
Von Koch pahuljica

- Počne se sa inicijatorom:



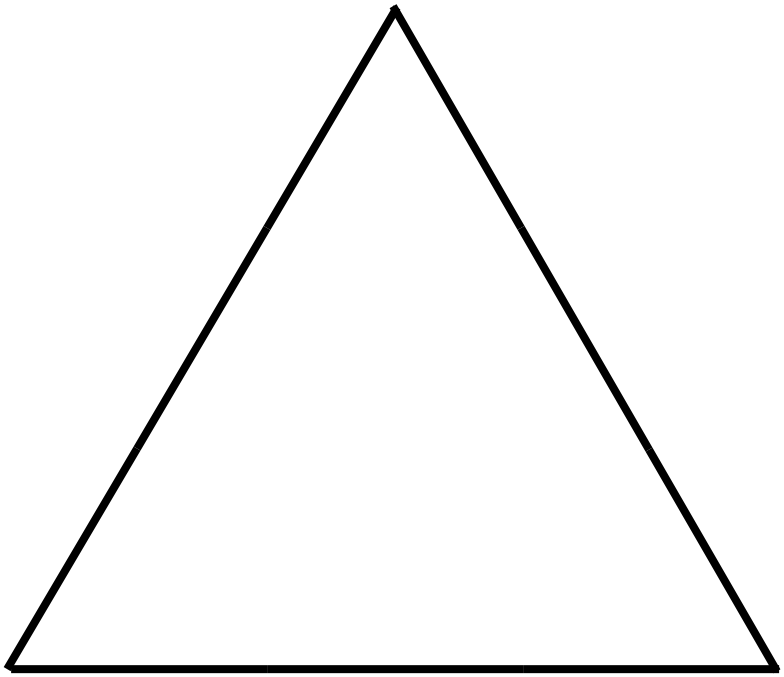
- I generatorom:

- Kod svake iteracije, menja se svaki komad inicijatora generatorom
- Dimenzija Von Koch fraktala: log4/log3 = 1,261859507



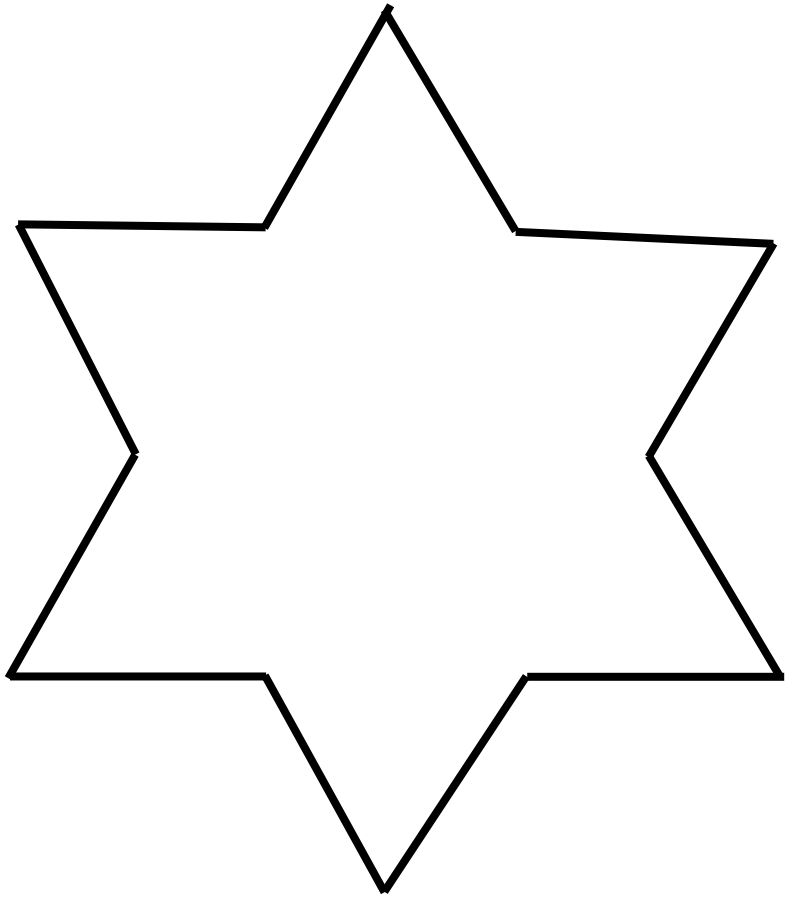
Von Koch pahuljica

- Iteracija 0:



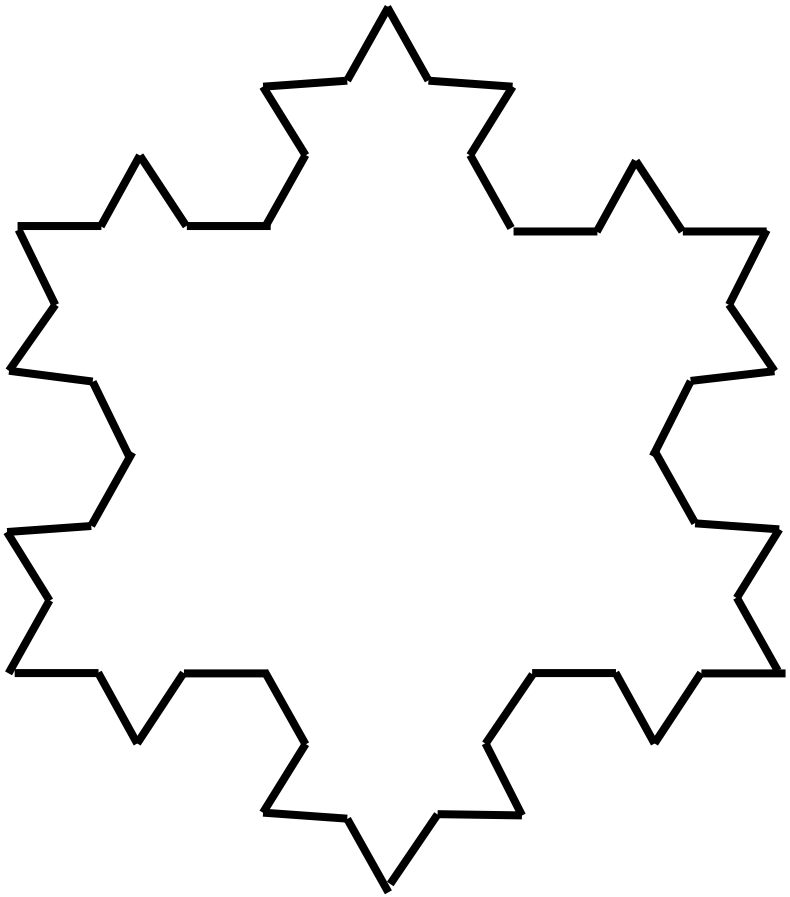
Von Koch pahuljica

- Iteracija 1:



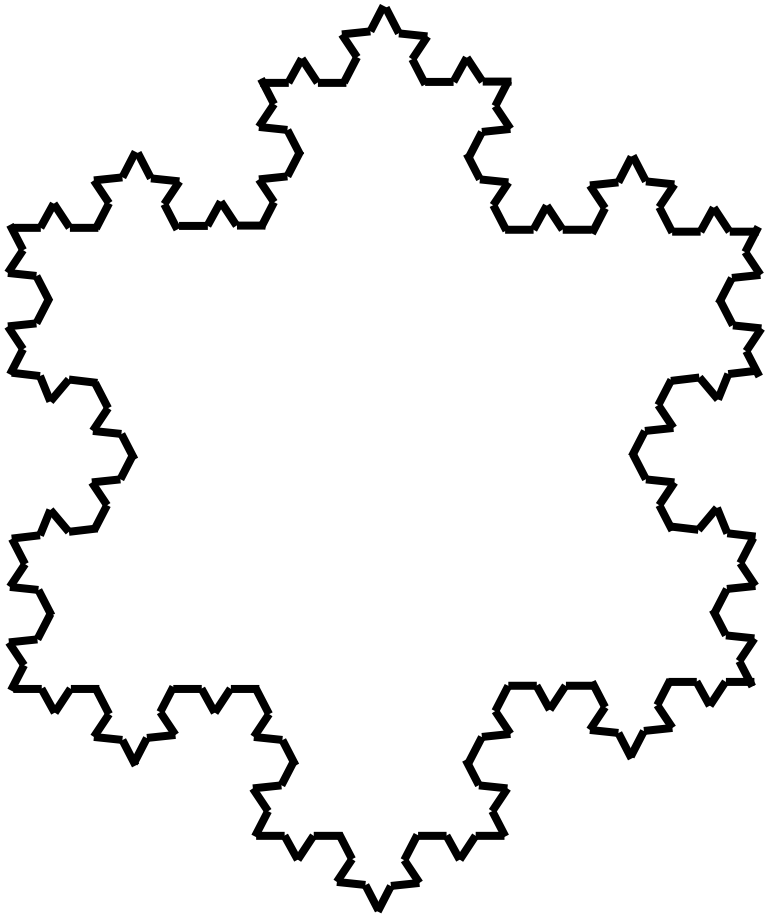
Von Koch pahuljica

- Iteracija 2:



Von Koch pahuljica

- Iteracija 3:



PRIMER 1/4

```
#pragma comment( lib, "opengl32.lib")
#pragma comment( lib, "glu32.lib")
#pragma comment( lib, "glut32.lib")
#include <cmath>
#include <GL/glut.h>
void showLine(int x1, int y1, int x2, int y2){
  glBegin(GL_LINES);
     glVertex2i(x1,y1);     glVertex2i(x2,y2);
  glEnd();
}
void showStar(double ax, double ay, double bx, double by,
    double &cx, double &cy, double &fx, double &fy, double &ex, double &ey)
{
 cx = (ax +0.5*bx)/1.5;  cy = (ay +0.5*by)/1.5;
 ex = (ax +2.0*bx)/3.0;  ey = (ay +2.0*by)/3.0;
 fx = cx+(ex-cx)*0.5-(ey-cy)*0.5*sqrt(double(3));
 fy = cy+(ex-cx)*0.5*sqrt(double(3))+(ey-cy)*0.5;
 glColor3f(1.0, 1.0, 0); showLine((int)ax,(int)ay,(int)bx,(int)by);
 glColor3f(0, 0, 1.0);
 showLine((int)ax,(int)ay,(int)cx,(int)cy);
 showLine((int)cx,(int)cy,(int)fx,(int)fy);
 showLine((int)fx,(int)fy,(int)ex,(int)ey);
 showLine((int)ex,(int)ey,(int)bx,(int)by);
}
```

PRIMER 2/4

```
void VonKochLine(double ax, double ay, double bx, double by, int level)
{
     double cx,cy;
     double fx,fy;
     double ex,ey;

     if(level>0)
     {
      showStar(ax,ay,bx,by,cx,cy,fx,fy,ex,ey);
      VonKochLine(ex,ey,bx,by,level-1);
      VonKochLine(fx,fy,ex,ey,level-1);
      VonKochLine(cx,cy,fx,fy,level-1);
      VonKochLine(ax,ay,cx,cy,level-1);
     }
}

struct point { float x; float y; }; //moglo bi se iskoristiti

void setPixel(int x,int y)
{
     glBegin(GL_POINTS);
        glVertex2i(x,y);
     glEnd();
}
```

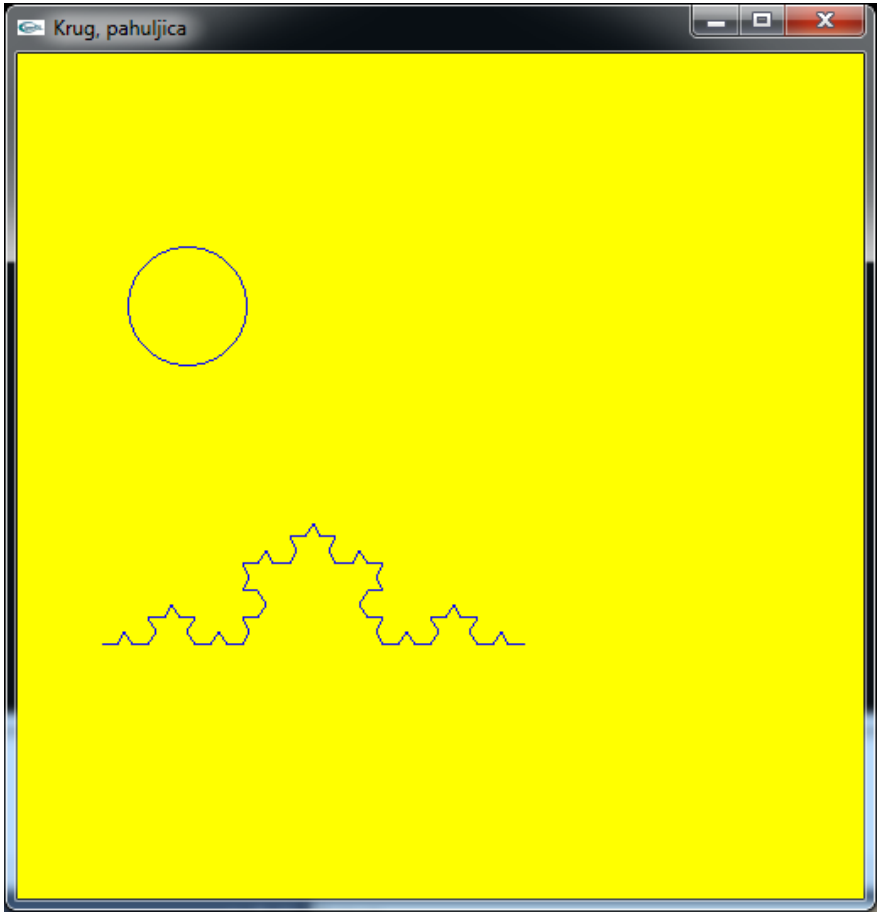
PRIMER 3/4

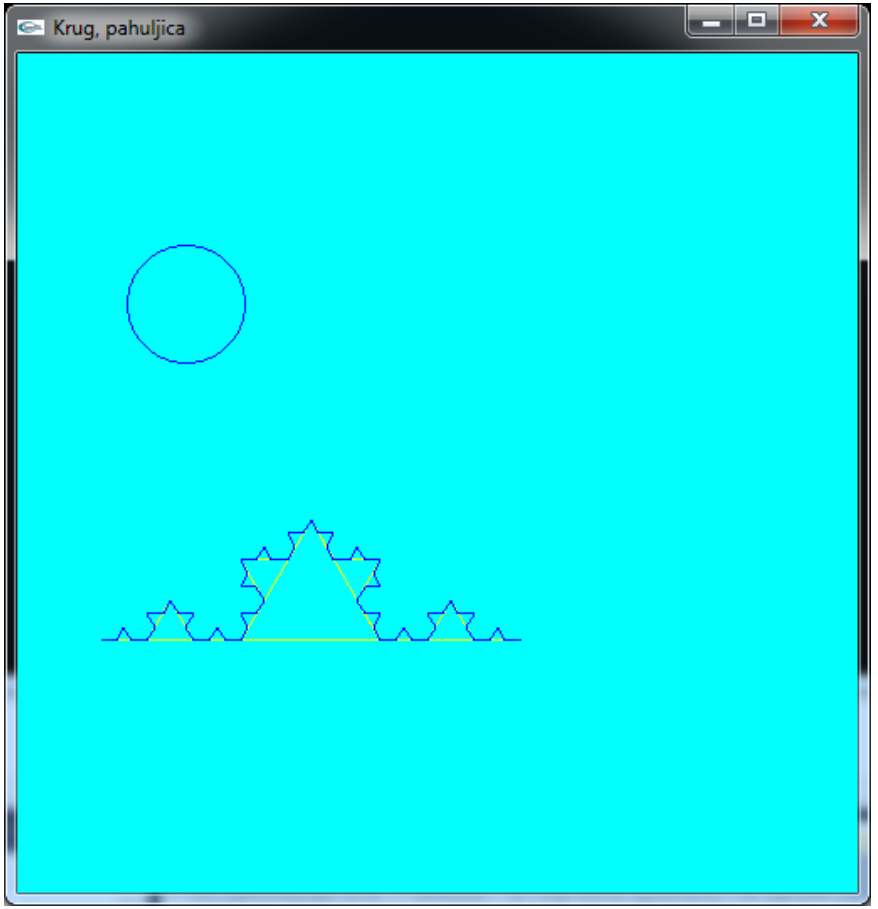
```
void BresenhamKrug(){
  int xCenter=100,yCenter=350,r=35;
  int x=0,y=r;
  int d=3-(2*r);
  glColor3f(0,0,1.);
  while(x<=y)    {
    setPixel(xCenter+x,yCenter+y);setPixel(xCenter+y,yCenter+x);
    setPixel(xCenter-x,yCenter+y);setPixel(xCenter+y,yCenter-x);
    setPixel(xCenter-x,yCenter-y);setPixel(xCenter-y,yCenter-x);
    setPixel(xCenter+x,yCenter-y);setPixel(xCenter-y,yCenter+x);
    if (d<0)  d += (4*x)+6;
    else{     d += (4*(x-y))+10;
              y--;
    }
    x++;
  }
  glFlush();
}
void displayCB(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  BresenhamKrug();  VonKochLine(50.0, 150.0, 300.0, 150.0, 3);
    glFlush();
}
```

PRIMER 4/4

```
void keyCB(unsigned char key, int x, int y){
  if( key == 'q' ) exit(0);
  if( key == 'a' )
  {
      glClearColor(0.0,1.0,1.0,0.0);
      displayCB();
  }
}

int main(int argc, char *argv[]){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowSize(500,500);
  int win = glutCreateWindow("Krug, pahuljica");

  glClearColor(1.0,1.0,0.0,0.0);
  gluOrtho2D(0,500,0,500);

  glutDisplayFunc(displayCB);
  glutKeyboardFunc(keyCB);
  glutMainLoop();

  return 0;
}
```

Izlaz





Homogene koordinate

- Prikaz tačke parom brojeva (x,y) se zamenjuje prikazom sa tri tačke (x',y',h).
- Preslikava se: Wc→Nc[0,1]→Sc

  Nove vrednosti su: x=x'/h; y=y'/h.

  Za afine koordinate (x,y) → (x',y',h) →(x,y,1)
- Skalarni proizvod homogene koordinate daje istu tačku:
  (2,3,5) = (4,6,10)

  Dokaz je trivijalan.

  `(4,6,10)→(4/10,6/10,10/10)→(0.4,0.6,1)`

  `(2,3, 5)→(2/5 ,3/5 ,5/5 )→(0.4,0.6,1)`
- Bar jedna homogena koordinata mora biti različita od nule; (0,0,0) nije dozvoljena

  Za h=0 dobijaju se tačke "u beskonačnosti"

2D Primitivi

- 2D tačka:

$$V(x,y) \rightarrow X = (x', y', h) \; ili \; X = (x_1, x_2, x_3)$$

$$x = \frac{x_1}{x_3}$$

$$y = \frac{x_2}{x_3}$$

- 2D pravac:
  - implicitni oblik: $a \cdot x + b \cdot y + c = 0$
  - uvođenje homogene koordinate: $a \cdot \frac{x_1}{x_3} + b \cdot \frac{x_2}{x_3} + c = 0$
  - homogena jednačina: $a \cdot x_1 + b \cdot x_2 + c \cdot x_3 = 0$

2D Primitivi

- 2D pravac:
  - matrični zapis:

$$X \cdot G = a \cdot x_1 + b \cdot x_2 + c \cdot x_3 = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0$$

  - vektor normale:

$$n = \begin{bmatrix} a & b \end{bmatrix}$$

  - vektor tangente:

$$t = \begin{bmatrix} b & -a \end{bmatrix}$$

  - dogovor:

$$X \cdot G \begin{cases} > 0, & X \ je "iznad" \ pravca \\ = 0, & X \ je \ na \ pravcu \\ < 0, & X \ je "ispod" \ pravca \end{cases}$$

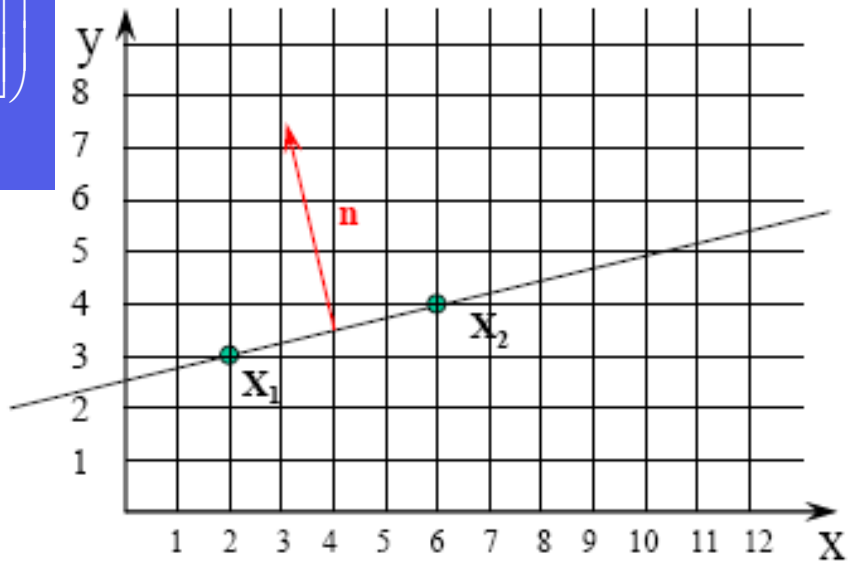
Primer

- X_1=(2 3 1), X_2=(6 4 1) (afine koordinate)

$$G = X_1 \times X_2 = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 2 & 3 & 1 \\ 6 & 4 & 1 \end{vmatrix}$$

$$G = \left(\begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} 3-4 \\ -(2-6) \\ 8-18 \end{bmatrix} \right)^T = \left(\begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 4 \\ -10 \end{bmatrix} \right)^T$$

$$G = -x_1 + 4x_2 - 10x_3 = -x + 4y - 10 = 0$$



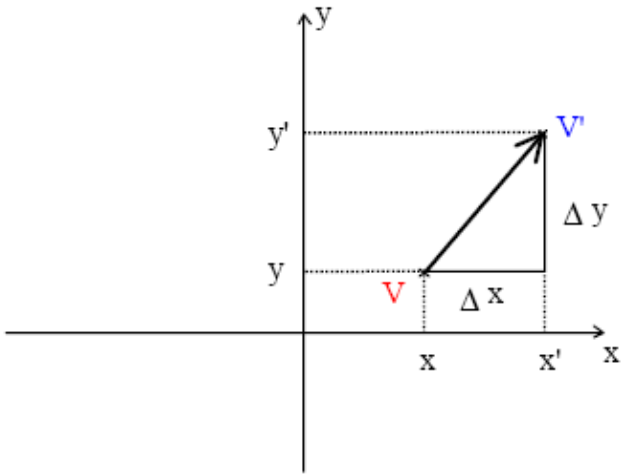
2D translacija i inverzna 2D translacija

$$x' = x + \Delta x$$
$$y' = y + \Delta y$$

$$\begin{bmatrix} x' & y' & h' \end{bmatrix} = \begin{bmatrix} x & y & h \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}$$

$$\mathbf{V}' = \mathbf{V} \cdot \mathbf{T}$$

T- matrica translacija



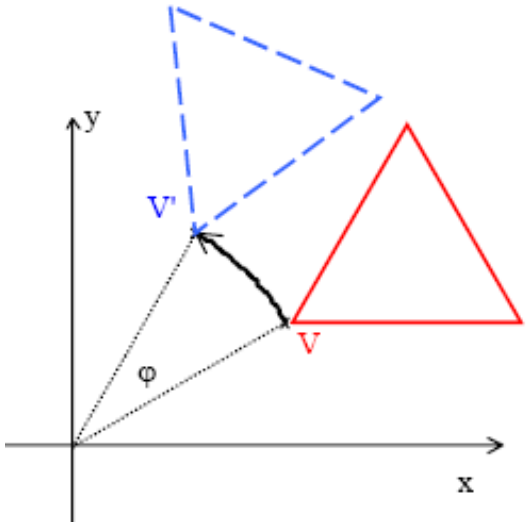
- Inverzna matrica translacije daje pomak u suprotnom smeru

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\Delta x & -\Delta y & 1 \end{bmatrix}$$

2D rotacija i inverzna 2D rotacija

$$x' = x\cos\varphi - y\sin\varphi$$
$$y' = x\sin\varphi + y\cos\varphi$$

$$\begin{bmatrix} x' & y' & h' \end{bmatrix} = \begin{bmatrix} x & y & h \end{bmatrix} \cdot \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{V}' = \mathbf{V} \cdot \mathbf{R}$$



- Inverzna matrica rotacije daje rotaciju u suprotnom smeru

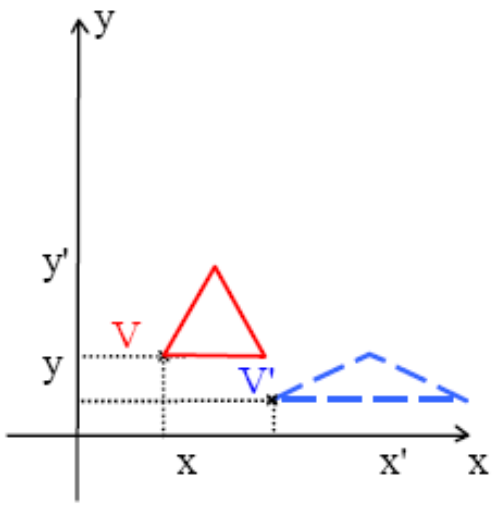
$$\mathbf{R}^{-1} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \cos(-\varphi) & \sin(-\varphi) & 0 \\ -\sin(-\varphi) & \cos(-\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D skaliranje i inverzno 2D skaliranje

$$x' = x\,s_x \qquad y' = y\,s_y$$

$$\begin{bmatrix} x' & y' & h' \end{bmatrix} = \begin{bmatrix} x & y & h \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{V}' = \mathbf{V} \cdot \mathbf{S}$$

$$\begin{bmatrix} x' & y' & h' \end{bmatrix} = \begin{bmatrix} x & y & h \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

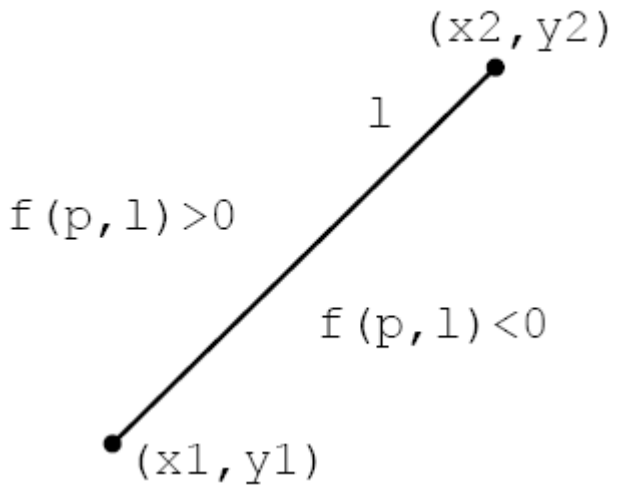


- Inverzna matrica skaliranja odgovara inverznoj transformaciji; povećanje se modifikuje u smanjivanje

$$\mathbf{S}^{-1} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Odnos tačke i prave

- Cilj je utvrditi da li su dve tačke
  - sa iste strane, ili
  - sa suprotnih strana prave
- Polazi se od jednačine prave *l* kroz dve tačke (x1,y1) i (x2,y2):




```
y=[(y2-y1)/(x2-x1)](x-x1) + y1,
```

ili

```
f(p,l)= (x2-x1)(y-y1) - (x-x1)(y2-y1) = 0, za p(x,y)
```

- Ova prava deli ravan na dve poluravni:

u jednoj važi : `f(p,l)=(x2-x1)(y-y1)-(y2-y1)(x-x1)>0`

a u drugoj važi: `f(p,l)=(x2-x1)(y-y1)-(y2-y1)(x-x2)<0`

Izvođenje

- Zamenjujući x i y za tačke p1 i p2 (čiji odnos prema pravoj ispitujemo)

  i množeći odgovarajuće izraze, dobija se izraz:

```
g(p1,p2,l) = f(p1,l)*f(p2,l) =
        ((x2-x1)(p1.y-y1) - (y2-y1)(p1.x-x1))*
        ((x2-x1)(p2.y-y1) - (y2-y1)(p2.x-x1))
```

  koji je:

  > 0, ako su tačke sa iste strane prave linije

  < 0, ako su tačke sa suprotnih strana linije i

  = 0, ako je barem jedna tačka na liniji

- Ako se odnos određuje u ekranskim koordinatama može se javiti problem prekoračenja pri množenju

- Pošto je od interesa samo da li je rezultat funkcije `g(p1,p2,l)` pozitivan, negativan ili jednak nuli uvodi se funkcija koja rezultat funkcije q svodi na vrednosti -1,0,+1:

```
int kaPravoj(int x)
{
        if (x==0) return 0;
        if (x >0) return 1;
        return -1;
}
```

Realizacija sameSide

- Funkcija koja određuje odnos tačaka p1 i p2 prema pravoj liniji *l* određena tačkama a i b:

```
int sameSide(Line l, Point p1, Point p2)//vraca -1,0,+1
{
   int dx,dx1,dx2;
   int dy,dy1,dy2;

   dx = l.b.x - l.a.x; //l.a i l.b su tačke koje odredjuju
   dy = l.b.y - l.a.y; //pravu l, dakle l(a,b)

   dx1 = p1.x - l.a.x;
   dy1 = p1.y – l.a.y;
   dx2 = p2.x - l.a.x;
   dy2 = p2.y - l.a.y;

   return(kaPravoj(dx*dy1 - dy*dx1)*
          kaPravoj(dx*dy2 - dy*dx2)  );
}
```

Presek pravolinijskih segmenata

- Potrebno je utvrditi da li se dva pravolinijska segmenta (linije) *l1* i *l2* seku
- Ovde nije od interesa tačka preseka
- Direktan način (skupo izračunavanje) da se odredi da li se linije seku:
  - naći tačku preseka geometrijskih pravih
  - proveriti da li se tačka nalazi između krajnjih tačaka datih segmenata
- Koristeći sameSide funkciju postojanje preseka se može utvrditi:

```
bool intersect(Line l1, Line l2)
{
  return (sameSide(l1,l2.a,l2.b)<=0)&&
         (sameSide(l2,l1.a,l1.b)<=0));
}
```

- *Napomene*:
  - proglašava se da se dve linije od kojih jedna leži na drugoj seku
  - funkcija proglašava da se dve kolinearne linije seku (greška)

Odnos tačke i poligona

- Glavni cilj je rešiti problem: **da li je tačka unutar ili izvan datog poligona**?
- Ideja rešenja:
  - iz tačke koja se ispituje povuče se linija u bilo kom pravcu do beskonačnosti
  - ako linija seče poligon neparan broj puta - tačka je unutar poligona,
    - inače je izvan
- Algoritam:
  - "beskonačnost" se uzima u pravcu i smeru pozitivne X-ose, pa se ispitna linija (**try**) povuče iz ispitivane tačke (**p**) paralelno sa X-osom
  - u iterativnom postupku, ide se od jedne do druge stranice (**edge**) poligona (***poly***) i ispituje da li se ova seče sa ispitnom linijom
    - ako se seče, inkrementira se broj preseka (**count**)
  - na kraju se odlučuje da je tačka punutar ***poly*** ukoliko je **count** neparan
- Pretpostavka – na kraju niza temena se nalazi ponovljeno prvo teme:

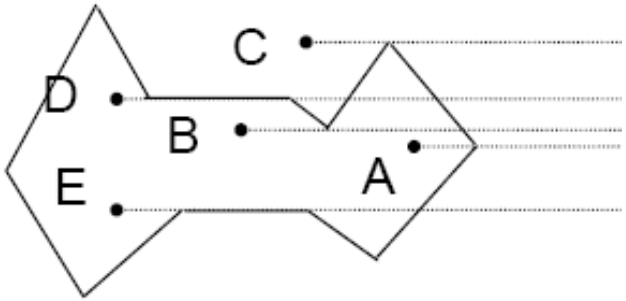
```
Point poly[n];          //vektor n vrhova poligona
//koristiti da n-ti indeks opet bude p[0] t.j. prvo teme
```

Realizacija inside

```
bool inside(Point p, int n, Poly poly){
   int i;
   int count;
   Point infin;
   Line try, edge;

    infin.x = Max_X; //desna granica po x osi
    infin.y = p.y;
    try.a = p;
    try.b = infin;
    count = 0;
    for(i=0; i<n; i++)
    {
       edge.a= poly[i];
       edge.b= poly[(i+1)%n];
       if intersect(try, edge) count = count+1;
    }
    return (count%2!=0)? true : false;
}//umesto int count mozete napisati resenje koristeci bool
 //promenljivu
```

Problemi

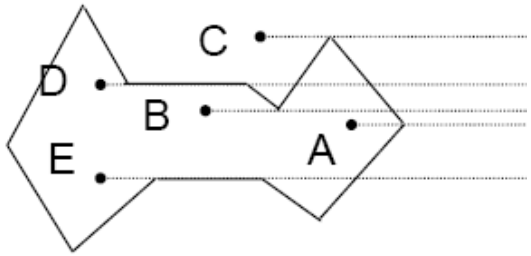
- Navedena funkcija nije korektna za neke slučajeve kada ispitna linija prolazi kroz teme poligona



- Posmatraju se sledeći slučajevi (podatak ukupno se odnosi na broj preseka do `Max_X`):
  - (A) ispitna linija iz unutrašnjosti poligona prolazi kroz teme i nastavlja u spoljašnjosti
    ⇒ count se inkrementira dva puta: **greška** **(biće ukupno 2 inkrementa)**
  - (B) ispitna linija iz unutrašnjosti prolazi kroz teme i ostaje u unutrašnjosti
    ⇒ count se inkrementira dva puta: **nije greška (biće ukupno 3 inkrementa)**
  - (C) ispitna linija iz spoljašnjosti prolazi kroz teme i ostaje u spoljašnjosti
    ⇒ count se inkrementira dva puta: **nije greška (biće ukupno 2 inkrementa)**
  - (D) ispitna linija iz unutrašnjosti polazi kroz dva temena i stranicu koja ih spaja, te nastavlja u spoljašnjosti
    ⇒ count se inkrementira tri puta: **nije greška (biće ukupno 5 inkrementa)**
  - (E) ispitna linija iz unutrašnjosti prolazi kroz dva temena i stranicu koja ih spaja i ostaje u unutrašnjosti
    ⇒ count se inkrementira tri puta: **greška** **(biće ukupno 4 inkrementa)**

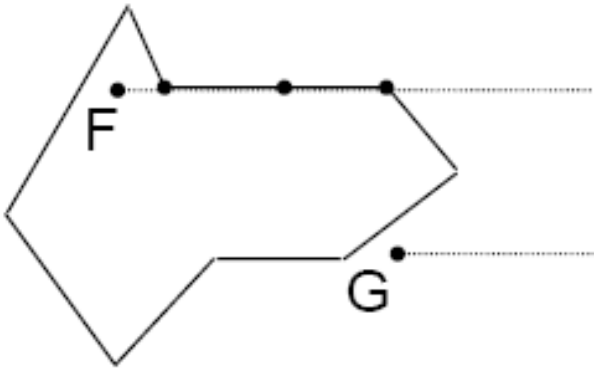
Korekcija greške

- Ideja: korekcija greške u posmatranim slučajevima:
  - za slučajeve A, B i C:
    - posmatrati odnos temena iza i temena ispred kritičnog temena, u odnosu na liniju try:
      - ako su sa iste strane (B i C), sve je u redu
      - ako su susedna temena sa različitih strana (A) treba dekrementirati count



  - slučajeve D i E treba svesti na A i B, respektivno:
    - treba ignorisati horizontalne ivice (ne odbrojavati preseke)
    - treba analizirati odnos temena ispred horizontalne ivice i temena iza te ivice sa linijom try

- Navedena tehnika rešava i probleme:
  - više povezanih horizontalnih ivica (F)
  - lažnih preseka koje vraća *intersect* za kolinearne ivice sa linijom try(G)



Korigovana realizacija (1)

```
bool inside(Point p, int n, Poly py)

{

   int i, count, prev;
   Point infin;
   Line try, edge;

   infin.x= Max_X;
   infin.y= p.y;

   try.a= p;
   try.b= infin;

   count= 0; prev=n-1;

   for(i=0; i<n; i++)

   {

        edge.a= py[i];

        edge.b= py[(i+1)%n];
```

Korigovana realizacija (2)

```
        if ( (p.y!=edge.a.y) || (p.y!=edge.b.y) )
        {
            //try i edge nisu kolinearne
            if ( intersect(try, edge) )
            {
              count= count + 1;
              if (py[i].y == p.y) &&
                 (sameSide(try,py[prev],py[(i+1)%n]) < 0)
              {
                count= count -1; //tačke su na suprotnim stranama
              }
            }
            prev=i;
        }
    }//for
    return (count%2!=0)?true:false;
}
```

Povećanje efikasnosti(1)

- Data implementacija je korektna, ali se može učiniti nešto efikasnijom
- Funkcije *sameSide i intersect* su generalne i ne koriste činjenicu da je *try* horizontalna
- Funkcija *hSameSide* određuje odnos tačaka p1 i p2 prema horizontalnoj liniji čija je krajnja leva tačka p:

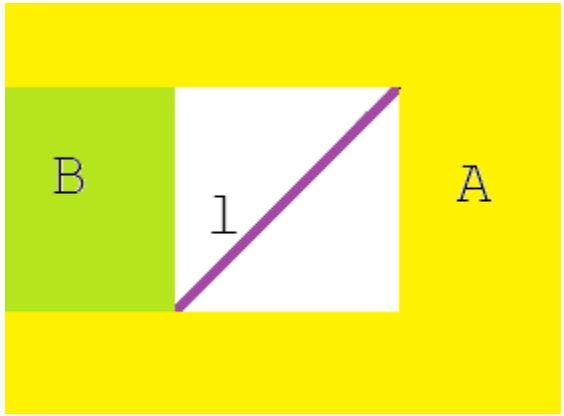
```
int  hSameSide(Point p, Point p1,Point p2)
{
  return kaPravoj((p1.y-p.y)*( p2.y-p.y));
 //dx*dx>0 pa nam ne treba za razmatranje
}
```

- Funkcija *hIntersect* određuje da li se proizvoljna linija *l* seče sa horizontalnom linijom čija je krajnja leva tačka p

Povećanje efikasnosti(2)

- Posmatraju se sledeće oblasti u kojima se može naći tačka p u odnosu na liniju *l* (ljubičasta linija):

```
A: [x>max(l.a.x,l.b.x)] or
   [y>max(l.a.y,l.b.y)] or
   [y<min(l.a.y,l.b.y)]
B: [x<min(l.a.x,l.b.x)] and
   p∉A
```



- Ako je p u oblasti A (žuta oblast)
  - ispitna horizontalna linija sigurno ne seče liniju l
- Ako je p u oblasti B (zelena oblast)
  - ispitna horizontalna linija sigurno seče liniju l
- Ako p nije ni u oblasti A niti u oblasti B (bela oblast)
  - mora se posebno ispitati da li postoji presek

Povećanje efikasnosti(3)

- Presek postoji ako je $p.x <= x_p$
- Iz jednačine prave kroz *l.a* i *l.b*:

  $x_p=(dx/dy)(p.y-l.a.y)+l.a.x$
- Presek postoji ako je:

  $p.x <= (dx/dy)(p.y-l.a.y)+l.a.x$
- Množenjem nejednakosti sa dy:
  - za **dy>0** presek postoji ako je:

    $(p.x-l.a.x)dy - (p.y-l.a.y)dx <= 0 \quad *(-1)$

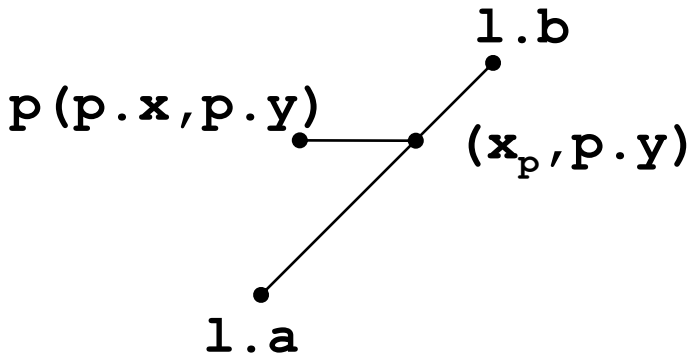
    $(p.y-l.a.y)dx - (p.x-l.a.x)dy >= 0$
  - za **dy<0** presek postoji ako je:

    $(p.x-l.a.x)dy - (p.y-l.a.y)dx >= 0$

    odnosno,

    $-((p.y-l.a.y)dx - (p.x-l.a.x)dy) >= 0$
- Za svako dy presek postoji ako je:

  $[dx(p.y-l.a.y) - dy (p.x-l.a.x)] dy >= 0$



Realizacija funkcije hIntersect

```
bool hIntersect(Point p,Line l)
{
    int dx,dy;
    if ( ( p.x>max(l.a.x,l.b.x)) Or
         ( p.y>max(l.a.y,l.b,y)) Or
         ( p.y<min(l.a.y,l.b,y)) )
    return false;      //oblast A

    if p.x<min(l.a.x,l.b.x) return true;    //oblast B

    //ni u A ni u B oblasti, mora se ispitati
    dx = l.b.x-l.a.x;
    dy = l.b.y-l.a.y;
    return (dx*(p.y-l.a.y) - dy*(p.x-l.a.x)) * dy >=0;
}
```

Zadatak

- Boja pozadine kanvasa je crna
- Dat je poligon određen sa 5 tačaka (šesta je jednaka prvoj)

X	60	90	160	220	170
y	60	200	300	240	100

- Nacrtati beli zatvoren poligon
- Sve tačke u pravougaonom području određenog intervalima:

  $x \in [50,250]$

  $y \in [50,350]$

  - obojiti
    - crvenom bojom ako ona ne pripada datom poligonu
    - obojiti žutom bojom ako se ona nalazi unutar poligona

Primer 1/5

```
#pragma comment( lib, "opengl32.lib")
#pragma comment( lib, "glu32.lib")
#pragma comment( lib, "glut32.lib")
#include <vector>
#include <GL/glut.h>

using namespace std;
const int Max_X = 2000;

class Point{
  public:
    int x;
    int y;
    Point(int x=0, int y=0){ this->x = x;this->y = y; }
};
class Line{
  public:
    Point a;
    Point b;
    Line(Point a=Point(0,0), Point b=Point(0,0)){this->a = a;this->b=b;}
};
```

Primer 2/5

```
class Poly{
    vector<Point> vPoints;
  public:
    int GetSize(){return vPoints.size();}
    void AddPoint(Point p){vPoints.push_back(p);}
    Point GetPoint(int index){return vPoints[index];}
    Point& operator[] (int index){return vPoints[index];}
};
int kaPravoj(int x){  if (x==0) return 0;  if (x >0) return 1;  return -1;}
int hSameSide(Point p, Point p1, Point p2){
  return kaPravoj((p1.y-p.y)*( p2.y-p.y));
}
bool hIntersect(Point p,Line l){
  int dx,dy;
  if ( ( p.x>max(l.a.x,l.b.x)) || ( p.y>max(l.a.y,l.b.y)) ||
       ( p.y<min(l.a.y,l.b.y)) ) return false;//u oblasti A, ne sece
  if (p.x<min(l.a.x,l.b.x)) return true;//nije u A, ali je u oblasti B,sece
  dx = l.b.x-l.a.x; dy = l.b.y-l.a.y; //oblast C, ispitati da li sece
  return ( (dx*(p.y-l.a.y)-(p.x-l.a.x)*dy)*dy >=0 );
}
```

Primer 3/5

```
bool hInside(Point p, Poly py){
  int i, count, prev;
  Point infin;
  Line tryline, edgeline;
  infin.x= Max_X; infin.y= p.y;
  tryline.a= p; tryline.b= infin;
  count= 0; prev=py.GetSize()-1;
  for(i=0; i<py.GetSize(); i++){
    edgeline.a= py[i];
    edgeline.b= py[(i+1)%py.GetSize()];
    if((p.y!=edgeline.a.y) || (p.y!=edgeline.b.y))//try i edge nekolinearne
    {
      if (hIntersect(p, edgeline)){
        count= count + 1;
        if ((py[i].y == p.y) &&
            (hSameSide(p,py[prev],py[(i+1)%py.GetSize()]) < 0)){
          count= count -1;//tacke su na suprotnim stranama
        }
        prev=i;
      }
    }
  }//for
  return (count%2!=0)?true:false;
}
```

Primer 4/5

```
Poly py;
void PopuniPoligon(){
  py.AddPoint(Point( 60, 60)); py.AddPoint(Point( 90,200));
  py.AddPoint(Point(160,300));  py.AddPoint(Point(220,240));
  py.AddPoint(Point(170,100));
}
void displayCB(void)// poziva se kad je potrebno ponovno iscrtavanje
{
  glClear(GL_COLOR_BUFFER_BIT); // obrisi displej
  glColor3f(1.0, 1.0, 1.0);//beli poligon
  glBegin(GL_POLYGON);
    for(int i=0; i<py.GetSize(); i++) glVertex2i(py.GetPoint(i).x,py.GetPoint(i).y);
  glEnd();
  for (int x=50; x<250; x++){
    for (int y=50; y<350; y++){
      if(hInside(Point(x,y),py))  glColor3f(1.0, 1.0, 0.0); //zuta
      else                        glColor3f(1.0, 0.0, 0.0); //crvena
      glBegin(GL_POINTS);glVertex2i(x,y);glEnd();
    }
  }
  glFinish();
}
```

Primer 5/5

```
void keyCB(unsigned char key, int x, int y)//poziva se pritiskom tastera
{
   if( key == 'q' ) exit(0);
}

void main(int argc, char *argv[]){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowSize(500,500);
  int win = glutCreateWindow("Poligon");

  glClearColor(0.5,0.5,0.5,0.0);// boja pozadine
  gluOrtho2D(0,500,0,500);// mapiranje objekta u prozor
  glutDisplayFunc(displayCB);// prikazna funkcija
  glutKeyboardFunc(keyCB);// funkcija tastature

  PopuniPoligon();
  glutMainLoop(); //beskonacna petlja
}
```

Izlaz