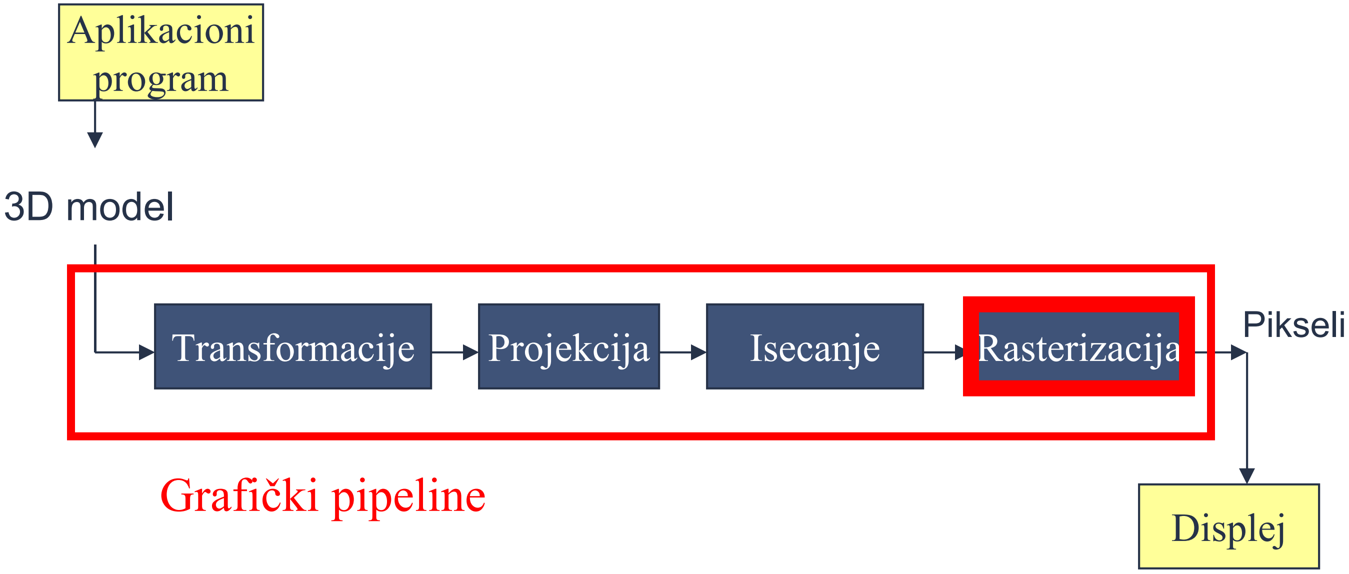
Računarska grafika
(2OER7O02)

Osnovni rasterski grafički algoritmi za crtanje 2D primitiva

Predavanja

Grafička protočna obrada (pipeline)



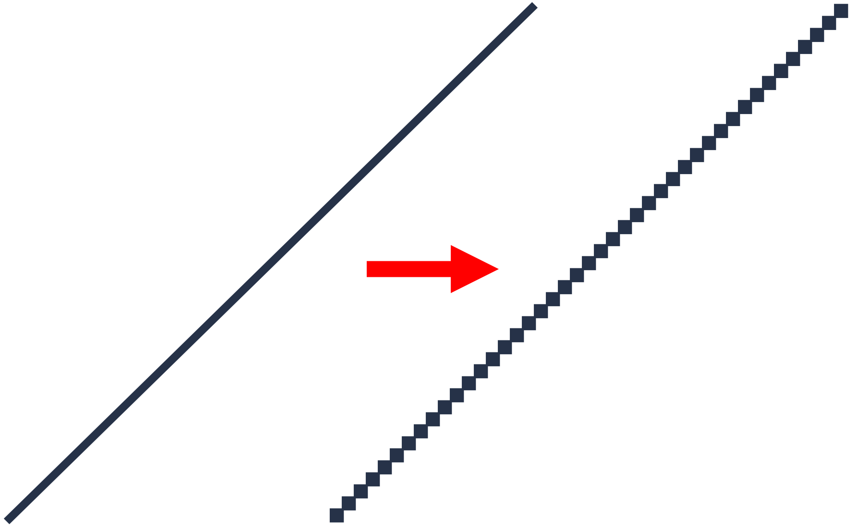
Rasterizacija (skaniranje – *scan conversion*)

- Skaniranje je proces prevođenja kontinualnih opisa grafičkih primitiva u diskretne opise (preko piksela).
- Koordinate piksela moraju biti celi brojevi.
- GAPI imaju ugrađene funkcije za skaniranje.

Skaniranje grafičkih primitiva

- Linije
- Kružnice
- Elipse
- Lukovi

Skaniranje linije



Skaniranje linije - zahtevi

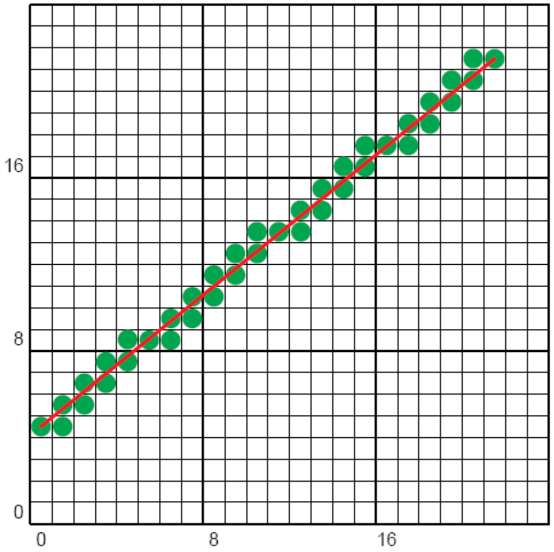
- Niz piksela treba da bude što bliže idealnoj liniji
- *Z*a linije koeficijenta između -1 i 1, u svakoj koloni treba da bude označen tačno jedan piksel. Za linije van tog opsega, u svakoj vrsti treba da bude označen bar jedan piksel.
- *S*ve linije treba da budu iste osvetljenosti i sve tačke svake linije treba da budu iste osvetljenosti (ili da tako izgledaju) bez obzira na njihov nagib i dužinu.
- Crtanje treba da bude što je moguće brže.

Skaniranje linije - algoritmi

- “Navni” algoritam
- Nagibni algoritam
- Inkrementalni algoritam
- Bresenham-ov algoritam

“Naivni” algoritam

- Koje piksele izabrati?



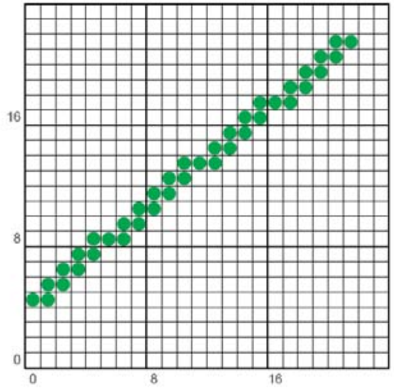
$(21, 21)$

$(0, 4)$

Sve koje matematička linija seče

Naivni algoritam - nedostaci

- Rezultat nije najbolji.
- Potrebno je dosta vremena za izračunavanje preseka piksela i matematičke linije.



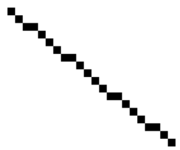
Nagibni algoritam

- Svaka linija je matematički definisana jednačinom prave koja glasi:

$$y = m*x+b$$

- *m* – koeficijent pravca (nagib)

Nagibni algoritam – implementacija



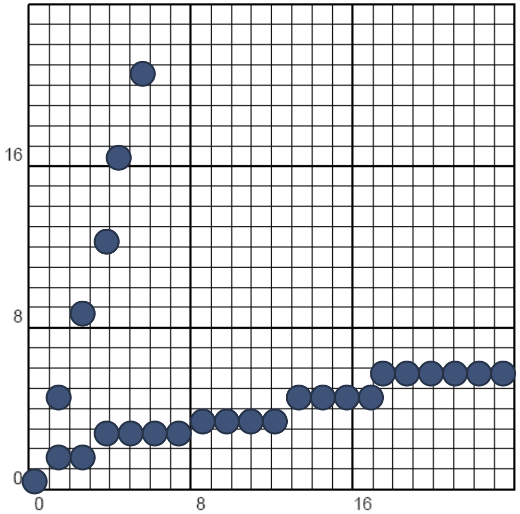
```
void NagibniAlgoritam (CDC* pDC, int x0,int y0,int
x1,int y1,COLORREF value)
{
  float dx, dy, m, y, b;
  int x;
  dx = x1 - x0;
  dy = y1 - y0;
  m  = dy/dx;
  b  = y1 - m*x1;

  for (x = x0; x <= x1; x++)
  {
      y = m*x + b;
      WritePixel(pDC, x, int(y+0.5), value);
  }
}
```

```
void WritePixel(CDC* pDC, int x, int y,
COLORREF value)
{
     pDC->SetPixel(x,y,value);
}
```

$x0 < x1$

Nagibni algoritam - nedostaci

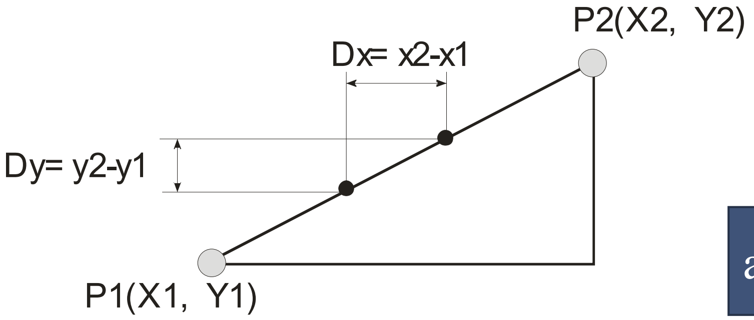
- Koristi se operacija množenja i izračunavanja sa realnim brojevima, što usporava algoritam.
- Različita osvetljenost linije (zavisno od nagiba).



abs(m)<1

Inkrementalni algoritam - DDA

- **DDA** - digitalni diferencijalni analizator



abs(m)<1, x1<x2

Inkrementalni algoritam - DDA

$$m = \frac{dy}{dx} = \frac{y2 - y1}{x2 - x1}$$

$y_i = mx_i + b$

$y_{i+1} = mx_{i+1} + b = m(x_i + dx) + \boxed{b = mx_i + m}dx + b = y_i + mdx$

Ako je $dx = 1$, onda je:

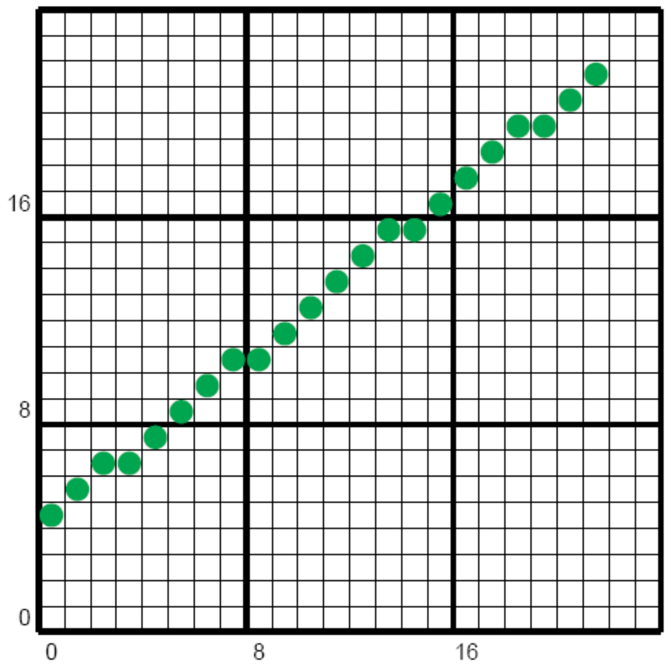
$$\boxed{y_{i+1} = y_i + m}$$

čime je eliminisano množenje.

Inkrementalni algoritam - implementacija

```
void LineDraw(CDC* pDC, int x0, int y0, int x1, int y1,
COLORREF value)
{
	int x;
	double dy = y1 - y0;
	double dx = x1 - x0;
	double m = dy / dx;
	double y = y0;

	for (x = x0; x <= x1; x++)
	{
		WritePixel(pDC, x,int(y+0.5),value);
		y += m;
	}
}
```

Inkrementalni algoritam - rezultat

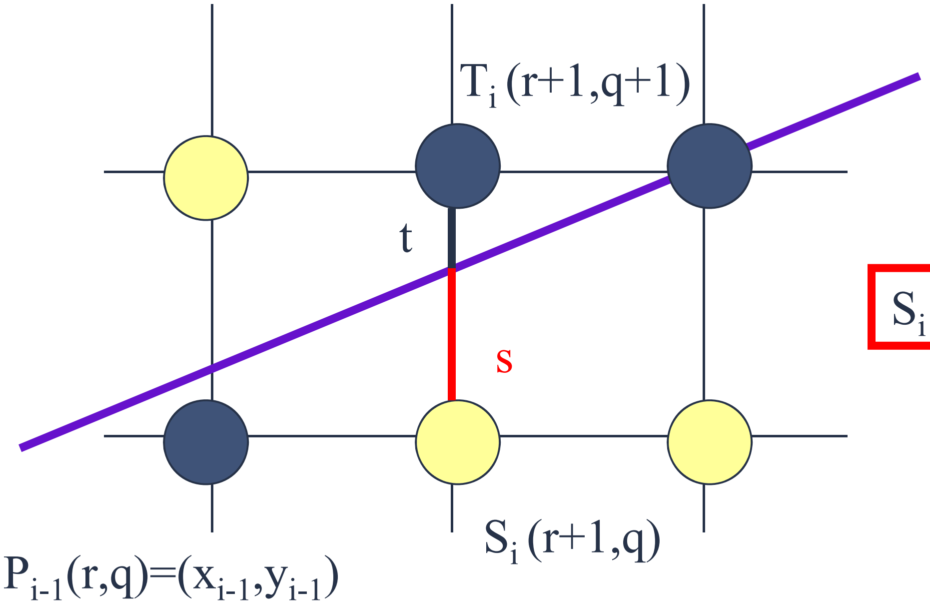


Bresenham-ov algoritam

- 1965. godine J. Bresenham je analizirao problem generisanja prave linije na digitalnom ploteru. Tako je razvio algoritam koji se koristi čak i danas. Njegova tehnika se zasniva na odlučivanju da li, u slučaju kada inkrementiramo vrednost koordinate X, treba inkrementirati vrednost koordinate Y ili ne.

- Pretpostavke za izvođenje su iste kao kod osnovnog inkrementalnog algoritma:
  - Nagib linije je u granicama $0<m<1$;
  - $x0 < x1$

Bresenham-ov algoritam



$\boxed{S_i \text{ ili } T_i?}$

Bresenham-ov algoritam

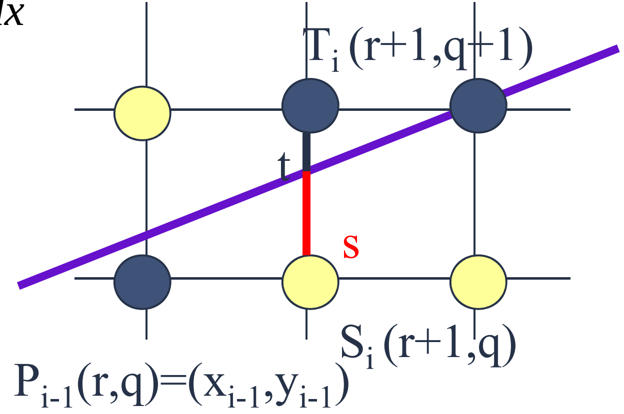
- Ukoliko transliramo liniju u koordinatni početak (za T($-x_0$,$-y_0$)), onda je:

$$dx = x_1 - x_0$$
$$dy = y_1 - y_0$$

$$y = \frac{dy}{dx} x$$

$$\boxed{s = \frac{dy}{dx}(r+1) - q}$$

$$\boxed{t = q + 1 - \frac{dy}{dx}(r+1)}$$



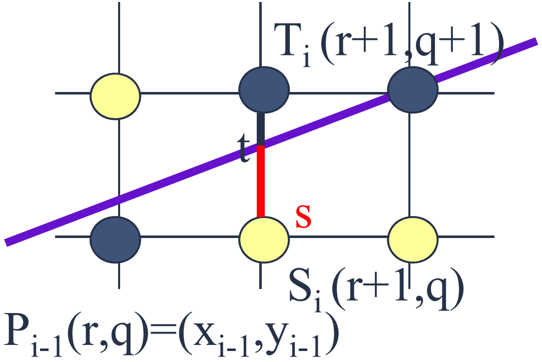
Bresenham-ov algoritam

- Ako napravimo razliku:

$$s - t = 2\frac{dy}{dx}(r+1) - 2q - 1$$

$$\boxed{dx(s-t)} = 2(rdy - qdx) + 2dy - dx$$

d_i



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

$$r = x_{i-1}, q = y_{i-1}$$

$$\boxed{d_i = 2x_{i-1}dy - 2y_{i-1}dx + 2dy - dx}$$

Bresenham-ov algoritam

$$d_{i+1} = 2x_i dy - 2y_i dx + 2dy - dx$$

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

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

Promenljiva odluke

$$d_i = \begin{cases} \geq 0 & y_i = y_{i-1} + 1, \quad d_{i+1} = d_i + 2(dy - dx) \\ < 0 & y_i = y_{i-1}, \quad d_{i+1} = d_i + 2dy \end{cases}$$

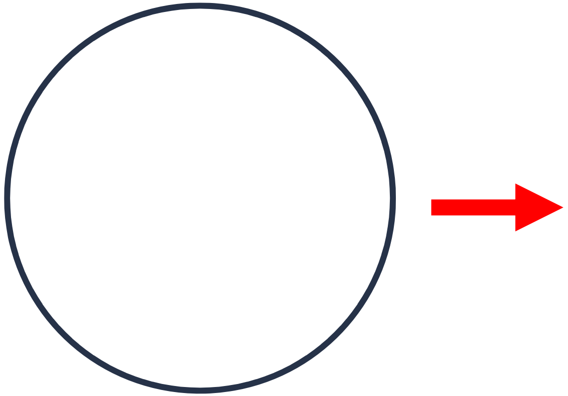
Bresenham-ov algoritam

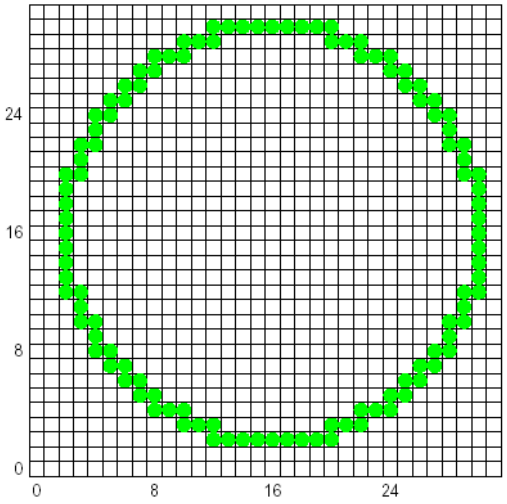
```
void BresenhamLine(int x0,int y0,int x1,int y1,
COLORREF value)
{
	int dx,dy,x,y,incr1,incr2,xend,d;
	dx = abs(x1 – x0);
	dy = abs(y1 – y0);
	d = 2*dy - dx;
	incr1 = 2*dy;
	incr2 = 2*(dy-dx);
	if (x0 > x1){
	x = x1;
	y = y1;
	xend = x0;
	}
	else {
		x = x0;
		y = y0;
		xend = x1;
	}
	WritePixel(pDC, x, y, value);
	while (x < xend) {
		x = x + 1;
		if (d < 0) d += incr1;
		else{
			y = y + 1;
			d += incr2;
		}
		WritePixel(pDC, x, y, value);
	}
}
```

Bresenhamov algoritam - prednosti

- Kompletna računica se svodi na celobrojnu aritmetiku. Koriste se samo sabiranje, oduzimanje i množenje sa 2, što čini ovaj algoritam najbržim do sada.
- Postoji i varijacija ovog algoritma – Midpoint algoritam

Skaniranje kružnice





Skaniranje kruga - zahtevi

- Slično kao za linije: Niz piksela treba da bude što bliže idealnom krugu.

- Nacrtani krug treba da bude ,,neprekidan“.

- Crtanje treba da bude što je moguće brže.

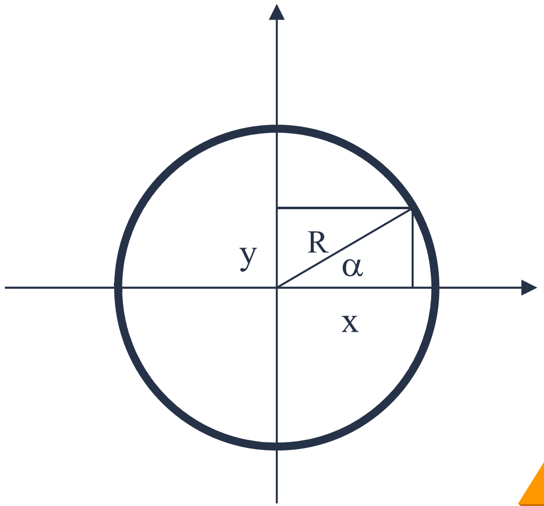
Skaniranje kruga - algoritmi

- Trigonometrijski algoritam
- Polinomni algoritam
- Bresenham-ov algoritam

Trigonometrijski (trivijalan) algoritam

Pretpostavka je da se crta kružnica sa centrom u koordinatnom početku. Trivijalan algoritam radi na osnovu parametarskih jednačina kružnice, odnosno na osnovu trigonometrijskih funkcija.

$x = R \cos\alpha,$
$y = R \sin\alpha,$
za $\alpha \in (0, 2\pi)$

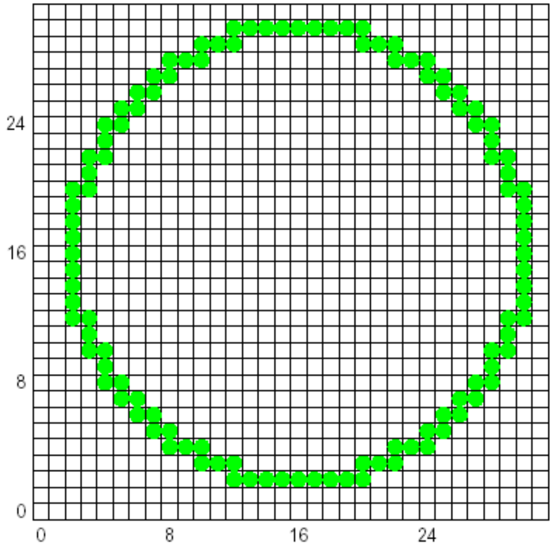


Trigonometrijski algoritam - implementacija

```
void TrigCircle(CDC* pDC, int r, COLORREF value)
{
	float alfa, dvapi = 6.283185;
	float step = dvapi / (7*r);
	int x, y;
	for (alfa = 0; alfa < dvapi; alfa += step)
	{
		x = int(r*cos(alfa) + 0.5);
		y = int(r*sin(alfa) + 0.5);
		WritePixel(pDC, x, y, value);
	}
}
```

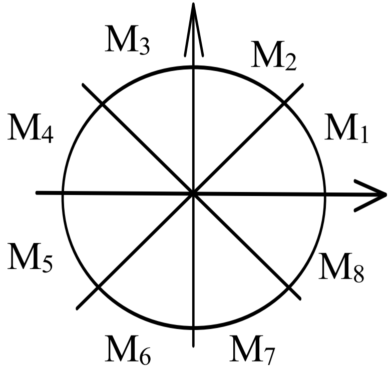
Trigonometrijski algoritam - nedostaci

- Sporost trigonometrijskih funkcija.
- Korak α je direktno proporcionalan poluprečniku kružnice, tako da možemo doći do toga da se veliki krugovi crtaju drastično sporije i lošije od malih.



Efikasniji trigonometrijski algoritam

Treba uočiti da je krug visoko simetrična figura. Delimo ga na lukove koji pripadaju oktantima, pa za sračunatu tačku koja pripada luku u prvom oktantu jednostavno nalazimo simetrične tačke na lukovima u ostalih sedam oktanata.



$M_1 = (x,y)$	$M_5 = (-x,-y)$
$M_2 = (y,x)$	$M_6 = (-y,-x)$
$M_3 = (-y,x)$	$M_7 = (y,-x)$
$M_4 = (-x,y)$	$M_8 = (x,-y)$

Efikasniji trigonometrijski algoritam - implementacija

```
void SimetricCircle(CDC* pDC, int r, COLORREF value)
{
      float alfa, dvapi = 6.283185, pi4 = 0.785398;
      float step = dvapi / (7 * r);
      int x, y;
      for (alfa = 0; alfa < pi4; alfa += step)
      {
            x = int(r*cos(alfa) + 0.5);
            y = int(r*sin(alfa) + 0.5);
            WritePixel8(pDC, x, y, value);
      }
}
```

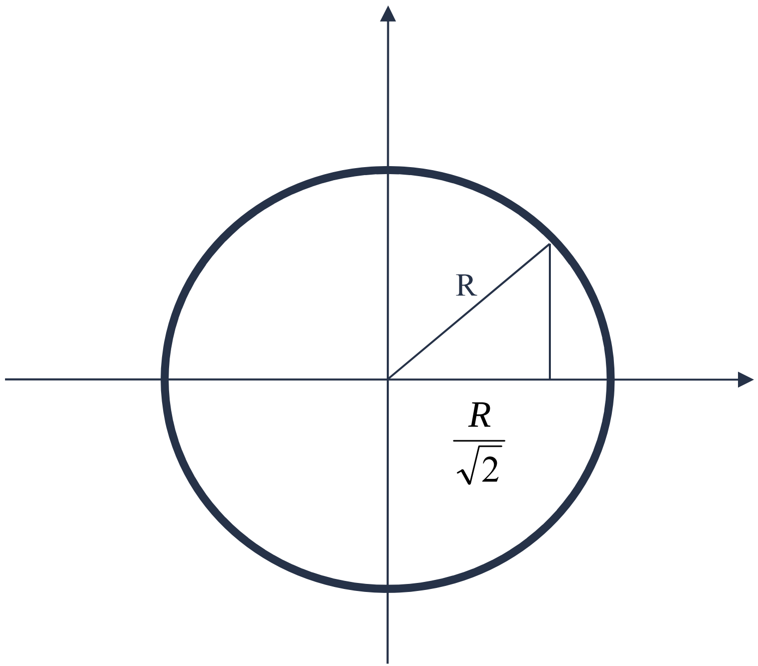
Efikasniji trigonometrijski algoritam - implementacija

```
void WritePixel8(CDC* pDC, int x, int y,
COLORREF value)
{
	WritePixel (pDC, x, y, value);
	WritePixel (pDC, -x, y, value);
	WritePixel (pDC, x,-y, value);
	WritePixel (pDC, -x,-y, value);
	WritePixel (pDC, y, x, value);
	WritePixel (pDC, -y, x, value);
	WritePixel (pDC, y,-x, value);
	WritePixel (pDC, -y,-x, value);
}
```

Polinomni algoritam

$$x^2 + y^2 = R^2$$

$$y = \sqrt{R^2 - x^2}$$



Polinomni algoritam - implementacija

```
void PolinCircle(CDC* pDC, int r,
COLORREF value)
{
	int x, y, xend;
	xend = (int)(r/sqrt(2)+0.5);
	for(x = 0; x <= xend; x++)
	{
		y=(int)(sqrt(r*r-x*x)+ 0.5);
		WritePixel8(pDC, x, y, value);
	}
}
```

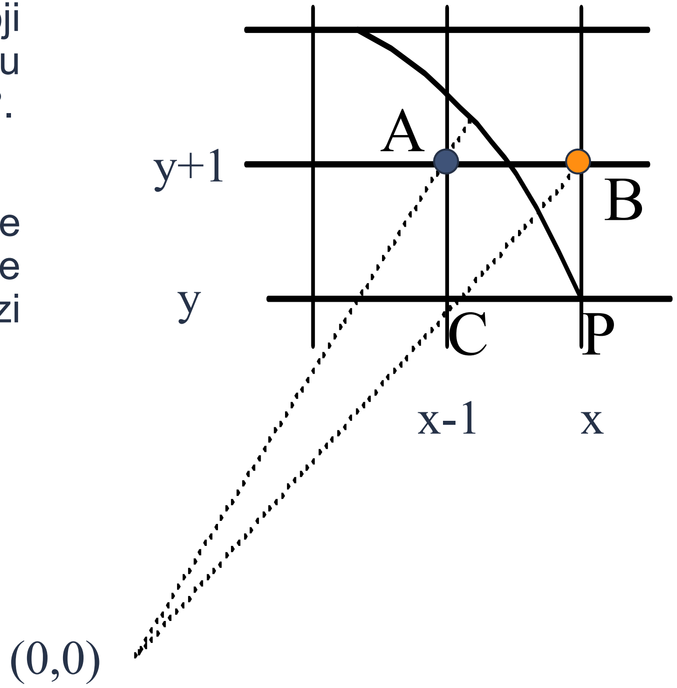
Polinomni algoritam - nedostaci

- Operacije sa realnim brojevima.
- I dalje postoje pozivi transcedentnih funkcija, ali je njihov broj prepolovljen

Bresenham-ov algoritam

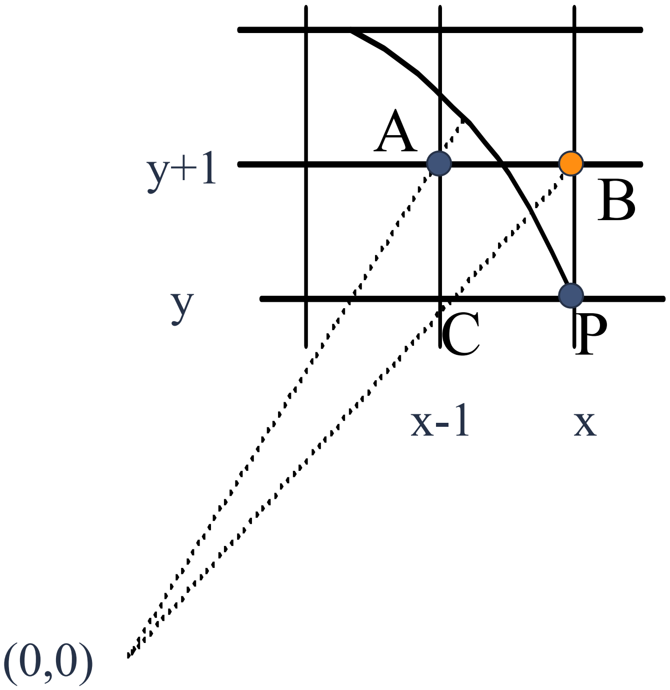
- Posmatra se luk u prvom oktantu, koji počinje tačkom (R, 0) i napreduje u smeru nasuprot kretanja kazaljke sata, do ugla 45°.

- Bresenham-ov algoritam za crtanje kružnice se svodi na odlučivanje o izboru između dve kandidat-tačke (A i B) između kojih prolazi kružna linija.



Bresenham-ov algoritam

- Pretpostavimo da je kružnica prošla kroz tačku P(x,y) i da treba odlučiti da li će “proći” kroz A(x-1, y+1) ili B(x, y+1).

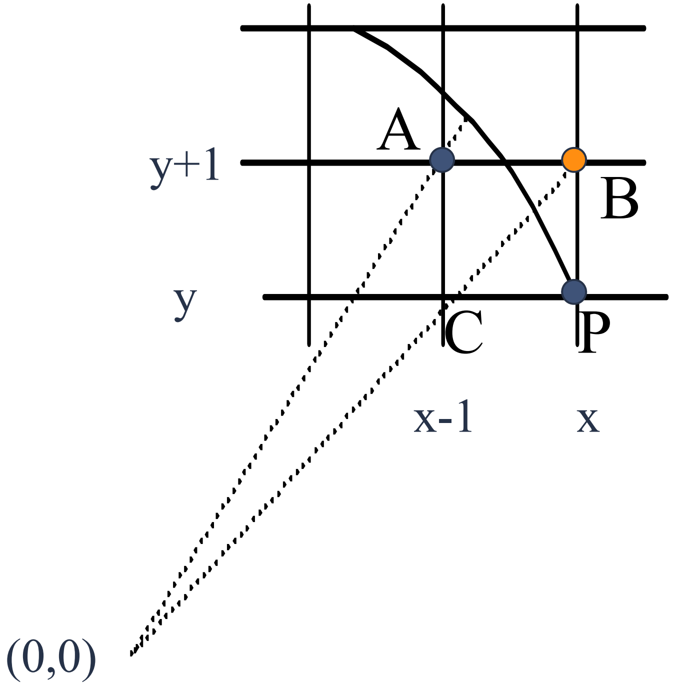


Bresenham-ov algoritam

- Uvodi se pojam kvadratne greške (odstupanja u odnosu na geometrijski tačnu kružnicu) u tački K:

$$\boxed{e(K) = x_k^2 + y_k^2 - R^2}$$

- e(K) > 0, za K izvan kruga,
- e(K) = 0, za K na kružnici,
- e(K) < 0, za K unutar kruga.



Bresenham-ov algoritam

- Algebarski zbir grešaka je:

$$d = e(A) + e(B)$$

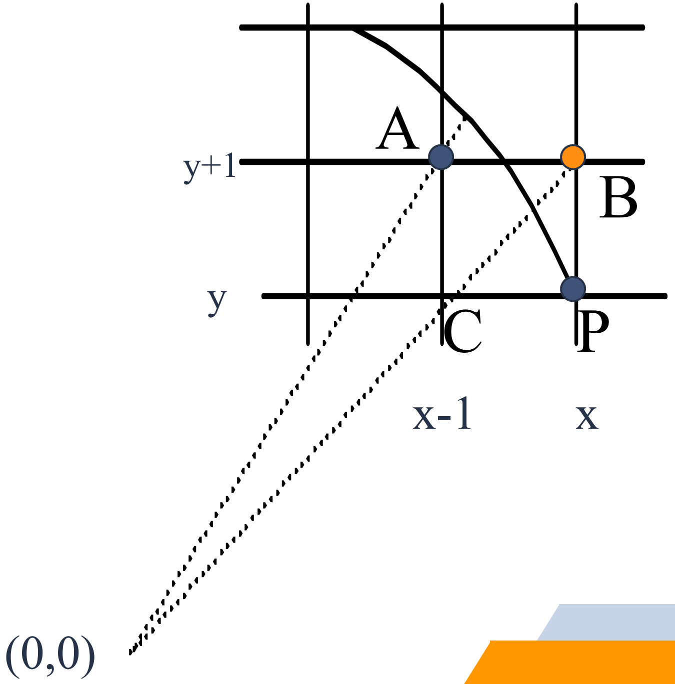
- A, B unutar kruga (teoretski): $e(A)<0$, $e(B)<0$ => **d < 0, bira se B**

- A, B izvan kruga (teoretski): $e(A)>0$, $e(B)>0$ => **d > 0, bira se A**

- A unutar, B izvan kruga: $e(A)<0$, $e(B)>0$

abs(e(A)) > abs(e(B)) => **d < 0, bira se B**

abs(e(A)) < abs(e(B)) => **d > 0, bira se A**

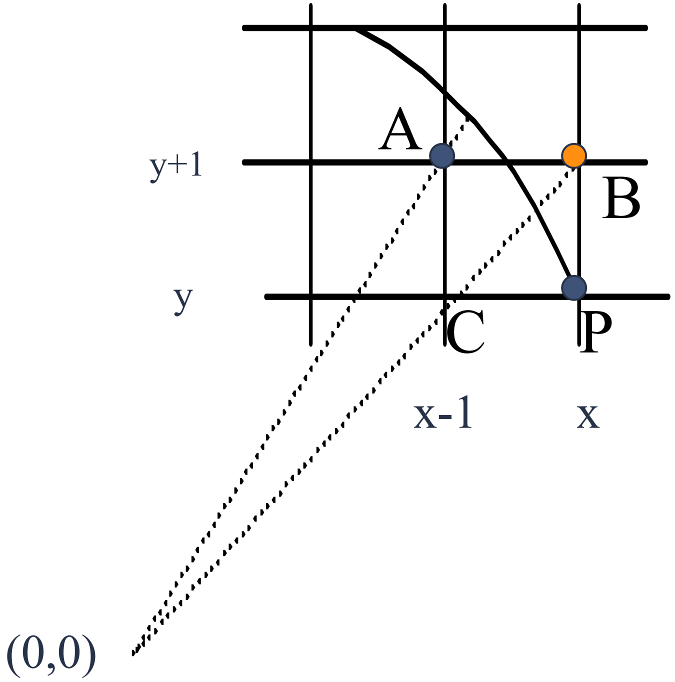


Bresenham-ov algoritam

- Uočava se zakonitost:

d < 0, bira se B

d > 0, bira se A



Bresenham-ov algoritam

- U i-tom koraku:

$$d_i = (x-1)^2 + (y+1)^2 - R^2 + x^2 + (y+1)^2 - R^2 =$$
$$= 2x^2 + 2y^2 - 2R^2 + 3 - 2x + 4y$$

- U početnom koraku:

$$d_0 = d_i \Big|_{R,0} = 3 - 2R$$

Bresenham-ov algoritam

- U i+1-om koraku:

  Za $d_i > 0$ (A):

$$d_{i+1} = (x-2)^2 + (y+2)^2 - R^2 + (x-1)^2 + (y+2)^2 - R^2 =$$
$$= 2x^2 + 2y^2 - 2R^2 + 3 - 2x + 4y + 10 - 4x + 4y =$$
$$= d_i + 4(y-x) + 10$$

Bresenham-ov algoritam

- U i+1-om koraku:

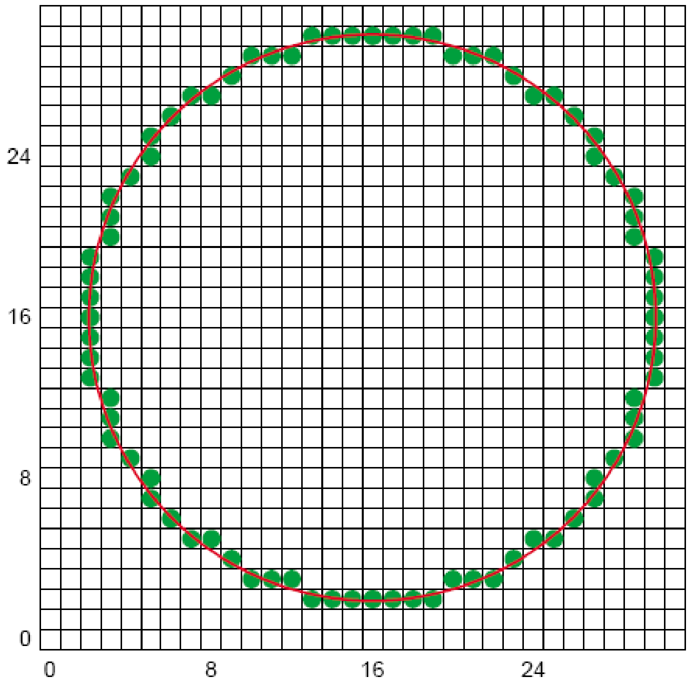
  Za $d_i < 0$ (B):

$$d_{i+1} = (x-1)^2 + (y+2)^2 - R^2 + x^2 + (y+2)^2 - R^2 =$$
$$= 2x^2 + 2y^2 - 2R^2 + 3 - 2x + 4y + 6 + 4y =$$
$$= d_i + 4y + 6$$

Bresenhamov algoritam - implementacija

```
 void BresenhamCircle(CDC* pDC, int r, COLORREF value)
{
	int x, y, d;
	x = r;	y = 0;
	d = 3 - 2*r;
	while(x > y){
		WritePixel8(pDC, x, y, value);
		if(d < 0) d += 4*y + 6;
		else  {
			d += 4*(y-x) + 10;
			x--;
		}
		y++;
	}
	if(x == y) WritePixel8(pDC, x, y, value);
}
```

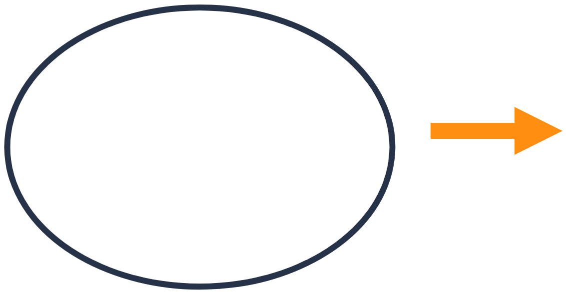
Bresenhamov algoritam - rezultat

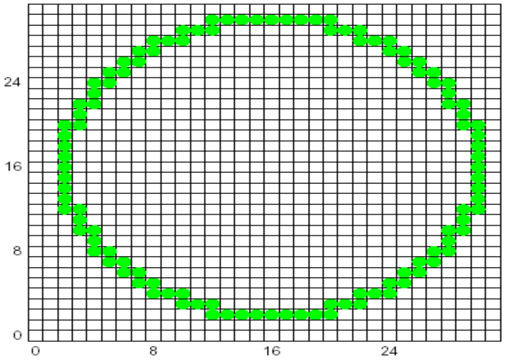


Bresenhamov algoritam - prednosti

- Kompletna računica se svodi na celobrojnu aritmetiku. Koriste se samo sabiranje, oduzimanje i množenje.
- Postoji i varijacija ovog algoritma – **Midpoint** algoritam

Skaniranje elipse





Skaniranje elipse - zahtevi

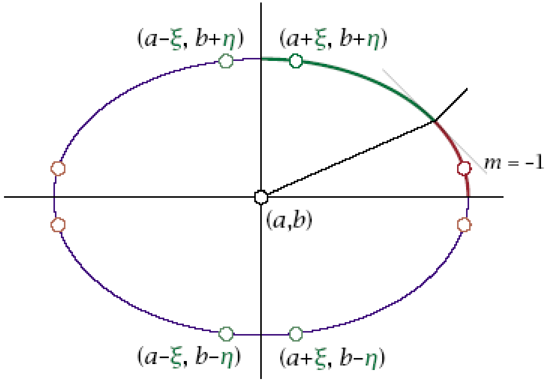
- Slično kao za krug: Niz piksela treba da bude što bliže idealnoj elipsi.
- Nacrtana elipsa treba da bude „neprekidna“.
- Crtanje treba da bude što je moguće brže.

Skaniranje elipse - algoritmi

- Trigonometrijski algoritam
- Polinomni algoritam
- Bresenham-ov algoritam
- Diferencijalni algoritmi
  - I reda
  - II reda

Simetrija kod elipsi

Treba uočiti da je elipsa takođe simetrična figura. Delimo je na lukove koji pripadaju kvadrantima, pa za sračunatu tačku koja pripada luku u prvom kvadrantu jednostavno nalazimo simetrične tačke na lukovima u ostala tri kvadranta

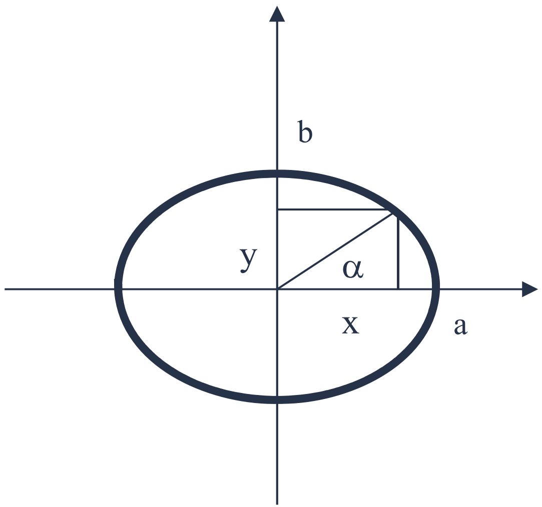


$M_1 = (x,y)$	$M_3 = (-x,-y)$
$M_2 = (-x,y)$	$M_4 = (x,-y)$

Trigonometrijski algoritam

Pretpostavka je da se crta elipsa sa centrom u koordinatnom početku, sa osama koje se poklapaju sa koordinatnim osama. Trigonometrijski algoritam radi na osnovu parametarskih jednačina elipse, odnosno na osnovu trigonometrijskih funkcija.

$$x = a\cos\alpha,$$
$$y = b\sin\alpha,$$
$$\text{za } \alpha \in (0, 2\pi)$$



Trigonometrijski algoritam - implementacija

```
void TrigElipsa(CDC* pDC, int a, int b, COLORREF value)
{
      float t, tend, tstep, dvapi = 6.283185;
      int x, y;
      t = 0.0;
      tend = 1.570796; // pi/2
      tstep = dvapi / (7 * max(a,b));
      while (t <= tend) {
            x = int(a*cos(t) + 0.5);
            y = int(b*sin(t) + 0.5);
            WritePixel4(pDC, x, y, value);
            t += tstep;
      }
}
```

Efikasniji trigonometrijski algoritam - implementacija

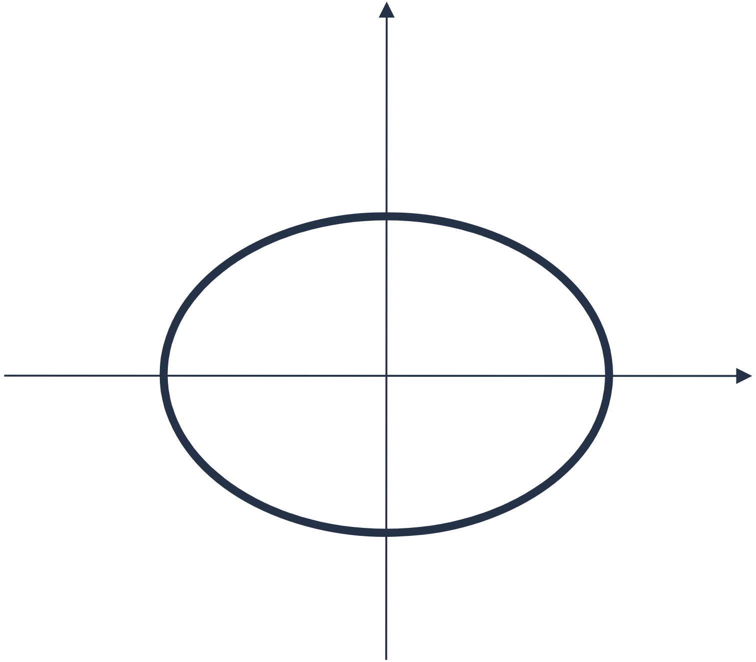
```
void WritePixel4(int x, int y, COLORREF value)
{
     WritePixel (pDC, x, y, value);
     WritePixel (pDC,-x, y, value);
     WritePixel (pDC, x,-y, value);
     WritePixel (pDC,-x,-y, value);
}
```

Polinomni algoritam

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

$$\Downarrow$$

$$y = \sqrt{b^2\left(1 - \frac{x^2}{a^2}\right)}$$

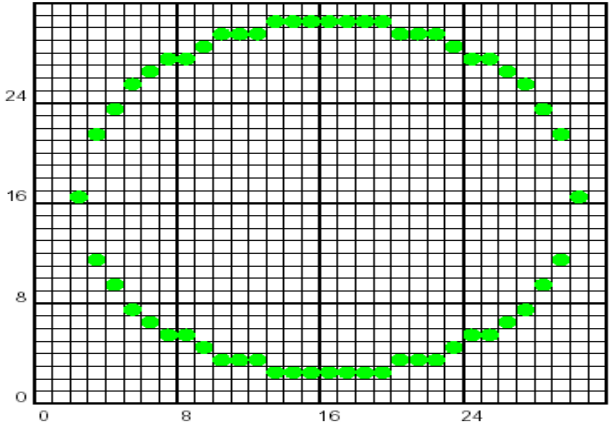


Polinomni algoritam - implementacija

```
void ElipsaPolinom(CDC* pDC, int a, int b,
COLORREF value)
{
	int x;
	float y,a2;
	a2 = a*a;
	for(x = 0; x < a; x++){
		y=(int)(b*sqrt(1-x*x/a2)+0.5);
		WritePixel4(pDC, x, y, value);
		}
}
```

Polinomni algoritam - nedostaci

- Operacije sa realnim brojevima.
- Loš rezultat



Bresenham-ov algoritam - implementacija

```
void BresenhamEllipse(CDC* pDC, int a, int b, COLORREF value)
{
 int x, y, a2 = a*a, b2 = b*b, s, t;
 x = 0; y = b;
 s = a2*(1-2*b) + 2*b2;
 t = b2 - 2*a2*(2*b-1);
 WritePixel4(pDC,x,y,value);
 while (y>0){
    if (s<0){
        s += 2*b2*(2*x+3);
        t += 4*b2*(x+1);
        x++;
       }
      else if (t<0){
          s += 2*b2*(2*x+3) - 4*a2*(y-1);
          t += 4*b2*(x+1) - 2*a2*(2*y-3);
          x++;
          y--;
         }
        else {
          s -= 4*a2*(y-1);
          t -= 2*a2*(2*y-3);
          y--;
         }
    WritePixel4(pDC,x,y,value);
  }
}
```

Diferencijalni algoritam I reda

$$x = a\cos\varphi$$

$$y = b\sin\varphi$$

$$\Downarrow$$

$$x' = -a\sin\varphi = -a\frac{y}{b} = -\frac{a}{b}\cdot y$$

$$y' = b\cos\varphi = \frac{b}{a}\cdot x$$

Diferencijalni algoritam I reda

$$\frac{\Delta x}{\Delta \varphi} = x' = \frac{x_i - x_{i-1}}{\Delta \varphi} = -\frac{a}{b} \cdot y_{i-1}$$

$$\Downarrow$$

$$\boxed{\begin{aligned} x_i &= x_{i-1} - \frac{a}{b} y_{i-1} \cdot \Delta \varphi \\ y_i &= y_{i-1} + \frac{b}{a} x_{i-1} \cdot \Delta \varphi \end{aligned}}$$

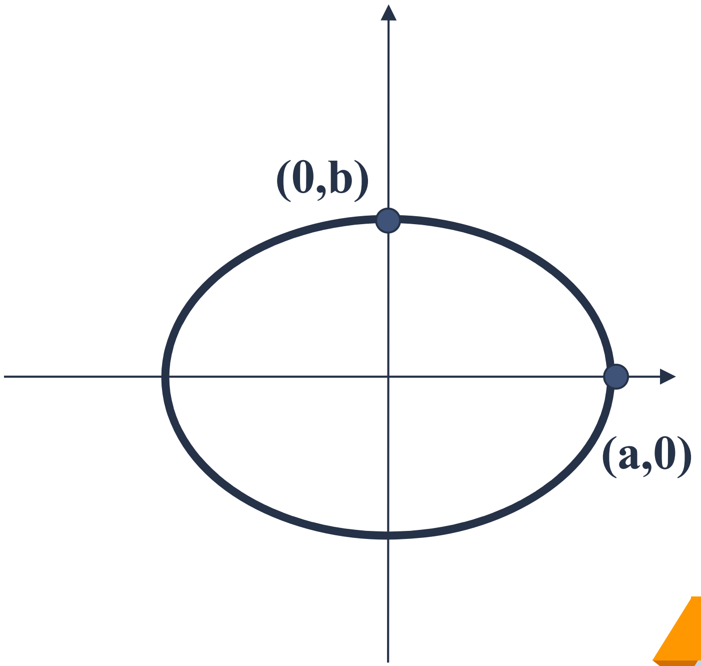
Diferencijalni algoritam I reda-implementacija

```
void DiffEllipse(CDC* pDC, float a, float b, COLORREF value)
{
	float ba,ab,x0,x1,y0,y1,Dphi;
	Dphi = 3.1416/180.0;
	ba = Dphi*b/a; ab = Dphi*a/b;
	x0 = a; y0 = 0;
	for (int j = 0; j < 90; j++){
		x1 = x0 – ab * y0;
		y1 = y0 + ba * x0;
		 BresenhamLine(pDC, x0, y0, x1, y1, value);
		 BresenhamLine(pDC,-x0, y0,-x1, y1, value);
		 BresenhamLine(pDC, x0,-y0, x1,-y1, value);
		 BresenhamLine(pDC,-x0,-y0,-x1,-y1, value);
		x0 = x1;
		y0 = y1;
	}
}
```

Diferencijalni algoritam I reda - nedostaci

- Osnovni nedostatak je taj što se algoritam ne završava tačno u (0,b). To je zbog toga što je diferencijalna razlika procenjivana samo na osnovu dve zadnje tačke x_i i x_{i-1}.

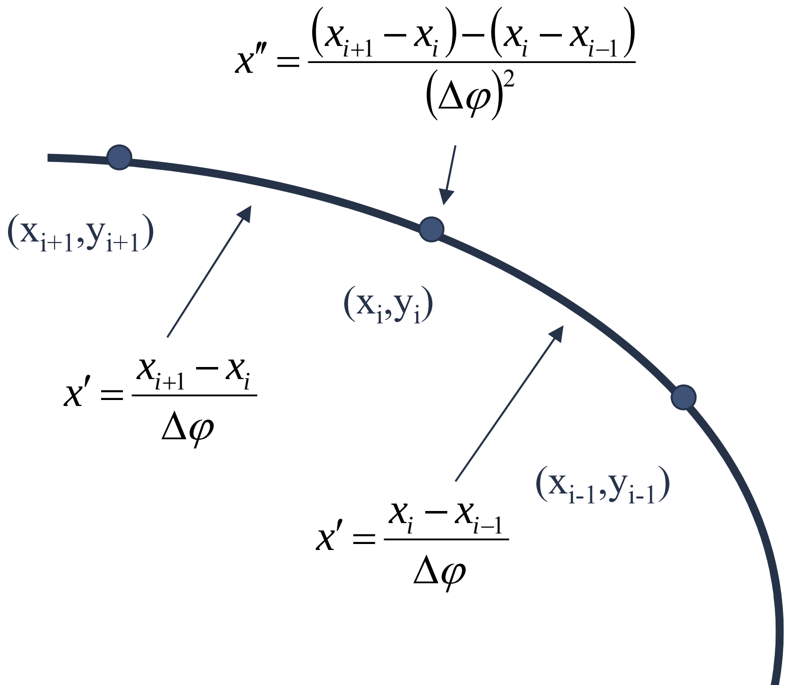
- Poboljšanje ovog algoritma je diferencijalni algoritam II reda.



Diferencijalni algoritam II reda

$$x = a\cos\varphi$$
$$y = b\sin\varphi$$

$$x' = -a\sin\varphi = -a\frac{y}{b} = -\frac{a}{b}\cdot y$$
$$x'' = -a\cos\varphi = -x$$
$$x'' = \frac{x_{i+1} - 2x_i + x_{i-1}}{(\Delta\varphi)^2} = -x_i$$



Diferencijalni algoritam II reda

$$x_{i+1} = \left(2 - (\Delta\varphi)^2\right) \cdot x_i - x_{i-1}$$

$$y_{i+1} = \left(2 - (\Delta\varphi)^2\right) \cdot y_i - y_{i-1}$$

Diferencijalni algoritam II reda- implementacija

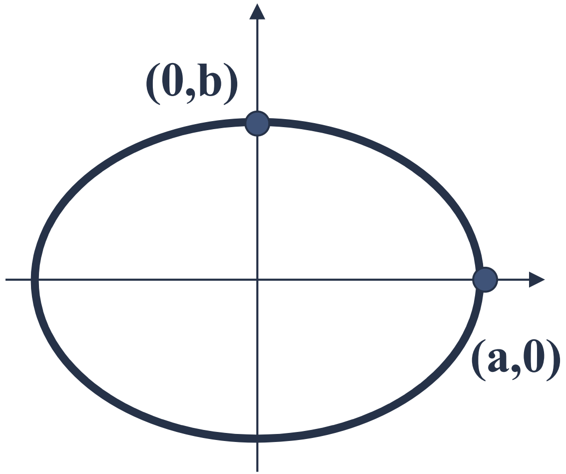
```
void DiffEllipse2(CDC* pDC, float a, float b, COLORREF value)
{
   float x0,x1,y0,y1,x2,y2,k;
   k=3.1416/180.0;
   x0=a; y0=0;
   x1=a*cos(k);
   y1=b*sin(k);
   k=2-k*k;
   for (int j=0; j<90;j++){
       BresenhamLine(pDC, x0, y0, x1, y1, value);
       BresenhamLine(pDC,-x0, y0,-x1, y1, value);
       BresenhamLine(pDC, x0,-y0, x1,-y1, value);
       BresenhamLine(pDC,-x0,-y0,-x1,-y1, value);
       x2=k*x1-x0;
       y2=k*y1-y0;
       x0=x1;
       y0=y1;
       x1=x2;
       y1=y2;
   }
}
```

Diferencijalni algoritam II reda - prednosti

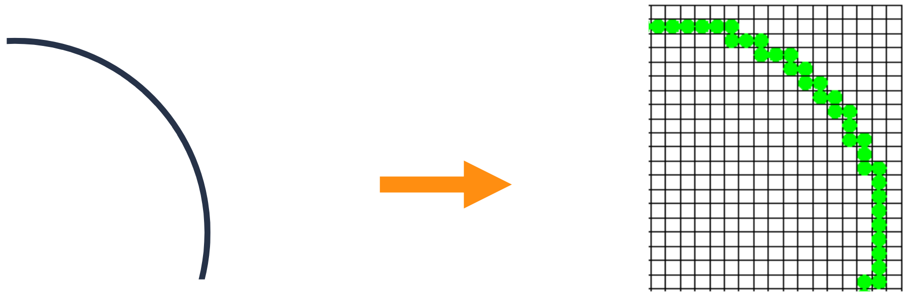
- Iterativna izračunavanja ne zavise od oblika elipse (ne zavise od parametara *a* i *b*)

$$x_{i+1} = \left(2-(\Delta\varphi)^2\right)\cdot x_i - x_{i-1}$$

$$y_{i+1} = \left(2-(\Delta\varphi)^2\right)\cdot y_i - y_{i-1}$$



Skaniranje luka



- Slično kao za krug i elipsu, samo su početni i krajnji ugao ograničeni.

PITANJA