# *Algoritmi de generare a cercurilor si elipselor in spatiul discret*

*Prof. univ. dr. ing. Florica Moldoveanu*

# Aproximarea cercurilor in spatiul discret(1)

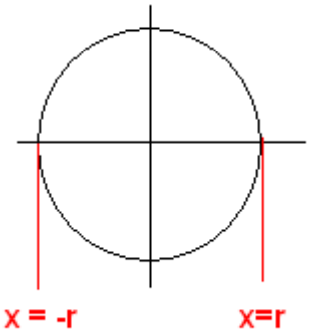
## 1) Folosind ecuatia cercului in coordonate carteziene

$(x - xc)^2 + (y - yc)^2 = r^2$

Generarea cercului cu centrul in (xc,yc) se reduce la generarea cercului centrat in origine, plus o translatie cu (xc, yc):

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

```
for x = -r; x< =r; x++)
{    y = (+/ -) sqrt(r^2 – x^2);
     putpixel (xc + x, yc + y, culoare); }
```




**Dezavantaje:**

- Spatierea inegala a punctelor de pe cerc afisate
- Calcule complexe cu numere reale

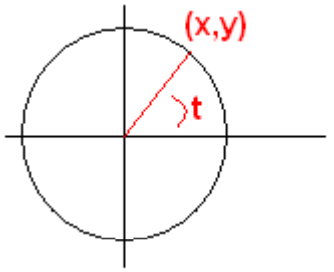
# Aproximarea cercurilor in spatiul discret(2)

## 2) Folosind ecuatia cercului in coordonate polare

$x = xc + r^* \cos(t)$

$y = yc + r^* \sin t(t)$ $\quad 0 <= t <= 2\pi$

Generarea se reduce la generarea cercului centrat in origine, plus o translatie cu (xc, yc):

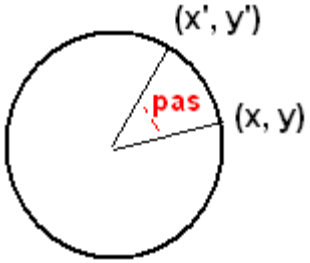



$x = r^* \cos(t)$
$y = r^* \sin t(t)$ $\quad 0 <= t <= 2\pi$

**Cercul poate fi aproximat pintr-o polilinie:**

- daca se calculeaza N puncte de pe cerc, egal distantate, t creste cu pasul $2\pi/N$

**Calculul punctelor succesive de pe cerc, pe baza unei relatii de recurenta:**




- (x, y): punctul curent
- (x', y'): punctul urmator

$x' = r^* \cos(t + pas) = r(\cos(t) * \cos(pas) - \sin(t)* \sin(pas)) = x*\cos(pas) - y*\sin(pas)$

$y' = r^* \sin(t + pas) = r(\cos(t) * \sin(pas) + \sin(t)* \cos(pas)) = x*\sin(pas) + y*\cos(pas)$

# Aproximarea cercurilor in spatiul discret(3)

Avantajul calculului prin recurenta: nu este necesar sa se calculeze sin si cos in fiecare iteratie!

```
// Aproximarea unui cerc printr-o linie poligonala
void Cerc( int xc, int yc, int r, int N)
{ double pas = 2*M_PI /N ; //pasul unghiular
  double u, x, y, x1,y1, s, c;
  c=cos(pas); s=sin(pas); // sin si cos se calculeaza o singura data
  for( int  i = 1, x=r, y=0; i<N; i++)
  { x1 =x*c - y*s; // x'
    y1 =x*s + y*c;  // y'
    line(xc+(int)(x+0.5), yc+(int)(y+0.5), xc+(int)(x1+0.5), yc+(int)(y1+0.5));
    x=x1 ; y= y1 ;
}
line(xc+(int)(x+0.5), yc+(int)(y+0.5),xc+r,yc);
}
```

# Aproximarea cercurilor in spatiul discret(4)

## Simetria punctelor de pe cerc

- **Fiecare punct de pe cerc are 7 puncte simetrice pe cerc:**

  - Este suficient sa se calculeze adresele punctelor dintr-un singur octant al cercului, celelalte puncte de pe cerc find calculate prin simetrie

**// Afiseaza punctele simetrice de pe cerc**

**void punct_simetric(int xc,int yc,int x,int y, int culoare)**

**{** putpixel(xc+x,yc+y,culoare);

putpixel(xc+x,yc-y,culoare);

putpixel(xc-x,yc-y,culoare);

putpixel(xc-x,yc+y,culoare);

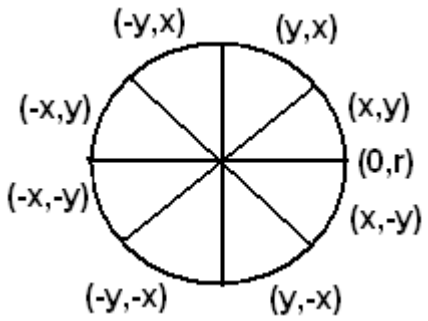
putpixel(xc+y,yc+x,culoare);

putpixel(xc+y,yc-x,culoare);

putpixel(xc-y,yc-x,culoare);

putpixel(xc-y,yc+x,culoare);

}



# Aproximarea cercurilor in spatiul discret(5)

- Pentru afisarea punctelor succesive, pasul unghiular trebuie sa fie aprox. (1/ raza).

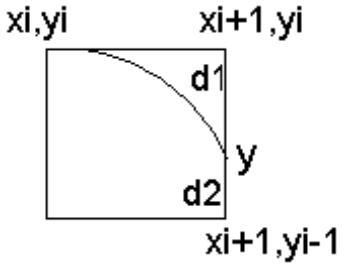
**// Calculeaza numai punctele din primul octant si afiseaza toate punctele simetrice**

```
void Cerc_sim( int xc, int yc, int r, int culoare)
{ double pas, u, x, y, xx, s, c;
  pas=1.0/r; // pas unghiular pentru obtinerea de puncte succesive de pe cerc
  c=cos(pas); s=sin(pas);
  putpixel(xc+r,yc,culoare);
  putpixel(xc-r,yc,culoare);
  putpixel(xc,yc+r,culoare);
  putpixel(xc,yc-r,culoare);
  for(u=pas, x=r, y=0; u<M_PI_4; u+=pas)
  { xx=xx;
    x =xx*c - y*s;
    y =xx*s + y*c;
    punct_simetric(xc, yc,(int)(x+0.5), (int)(y+0.5), culoare);
  }
}
```

# Algoritmul Bresenham pentru rasterizarea cercurilor(1)

- Consideram cercul cu centrul în originea sistemului de coordonate si raza r.
- In cadrul algoritmului **se calculeaza numai punctele din octantul al 2-lea,** celelalte obtinandu-se prin simetrie.
- Se calculeaza punctele de pe cerc incepand cu (x=0,y=r) pana la (x=y)
- Fie $(x_i,y_i)$ ultimul punct al spatiului discret, ales pentru aproximarea cercului.

  Urmatorul punct va fi unul dintre $(x_i+1,y_i)$ si $(x_i+1,y_i-1)$:




Fie y ordonata punctului de pe cercul teoretic, cu abscisa (xi+1).

$$y^2 = r^2-(x_i+1)^2$$

Fie d1 si d2 distantele de la cele 2 puncte ale spatiului discret la punctul de pe cercul teoretic.

Intereseaza semnul diferentei (d1-d2):

- daca $(y_i - y) > (y- (y_i-1))$ atunci

$$(y_i^2 - y^2 ) > (y^2 - (y_i-1)^2 )$$

$$d1' = y_i^2 - y^2 = y_i^2 - r^2 + (x_i+1)^2$$

$$d2' = y^2 - (y_i-1)^2 = r^2 - (x_i+1)^2 - (y_i-1)^2$$

# Algoritmul Bresenham pentru rasterizarea cercurilor(2)

- Notam cu $t_i$ eroarea de aproximare în pasul curent:

$t_i = d1'-d2' = y_i^2 + 2*(x_i+1)^2 + (y_i-1)^2 -2*r^2$

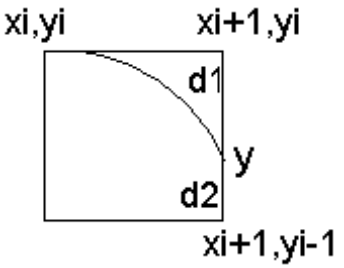
- Se obtine o relatie de recurenta pentru calculul erorii de aproximare in pasul urmator:

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

(1) Daca **$t_i<0$** ($d1' < d2'$) atunci $x_{i+1}=x_i+1$ si $y_{i+1}=y_i$ . Deci,

$t_{i+1}= y_i^2 + 2*((x_i+1)+1)^2 + (y_i-1)^2 -2*r^2$

sau **$t_{i+1} = t_i + 4*x_i + 6$**




(2) Daca **$t_i >=0$** atunci $x_{i+1} = x_i + 1$ si $y_{i+1} = y_i - 1$. Deci,

$t_{i+1} = (y_i-1)^2 + 2*((x_i+1)+1)^2 + ((y_i-1)-1)^2 -2*r^2$

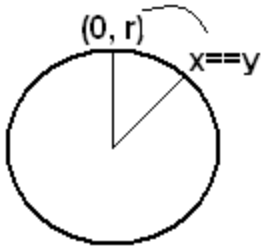
sau **$t_{i+1} = t_i + 4*(x_i - y_i) + 10$**

# Algoritmul Bresenham pentru rasterizarea cercurilor(3)

- Valoarea erorii de aproximare în primul pas:

$t_i = y_i^2 + 2*(x_i+1)^2 + (y_i-1)^2 - 2*r^2$ : $x_1=0$, $y_1 = r$. Rezultă, **t1 = 3-2*r**

```
void Bres_cerc(int xc, int yc, int r, int culoare)
{ int x,y,t;
  t=3-(r<<1);
  putpixel(xc+r,yc,culoare); putpixel(xc-r,yc,culoare);
  putpixel(xc,yc+r,culoare); putpixel(xc,yc-r,culoare);
  for(x=1,y=r; x<y; x++)
  {
    if(t<0)
      t+=6+(x<<2);
    else
     { t+=10+((x-y)<<2); y--;}
    punct_simetric(xc,yc,x,y,culoare);
  }
 putpixel(xc+x,yc+y,culoare); putpixel(xc+x,yc-y,culoare);
 putpixel(xc-x,yc+y,culoare); putpixel(xc-x,yc-y,culoare);
}
```

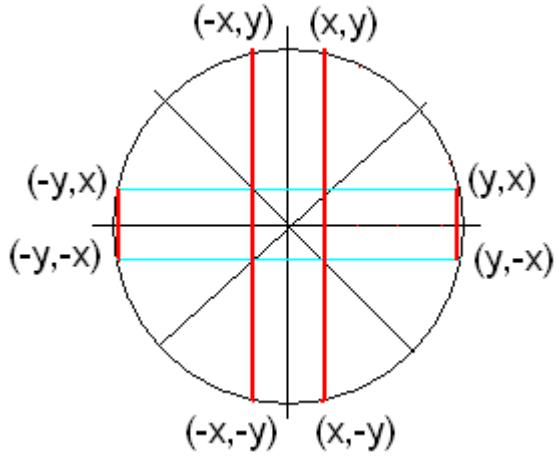


# Generarea unei suprafete circulare

EXERCITIU:

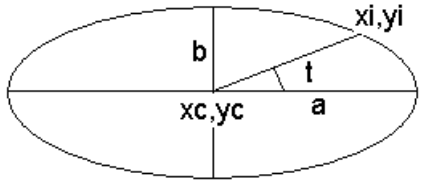
- Modificati algoritmul Bresenham pentru generarea cercurilor astfel incat sa se obtina o suprafata circulara.

```
void Bres_cerc(int xc, int yc, int r, int culoare)
{ int x,y,t;
   t=3-(r<<1);
   line(xc,yc+r,xc, yc-r, culoare); putpixel(xc-r,yc,culoare); putpixel(xc+r,yc,culoare);
   for(x=1,y=r; x<y; x++)
   {
     if(t<0)
       t+=6+(x<<2);
     else
      { t+=10+((x-y)<<2); y--;}
    line_simetric(xc,yc,x,y,culoare);
   }
  putpixel(xc+x,yc+y,culoare); putpixel(xc+x,yc-y,culoare);
  putpixel(xc-x,yc+y,culoare); putpixel(xc-x,yc-y,culoare);
}
```



# Aproximarea elipselor in spatiul discret(1)

- Ecuațiile parametrice ale elipsei sunt:




$x = xc + a*\cos(t)$

$y = yc + b*\sin(t)$ $\quad 0 <= t <= 2*\pi$

unde (xc,yc) este centrul elipsei, iar a si b sunt marimile semiaxelor elipsei.

- Se poate obține o secvență de puncte care aproximează circumferința elipsei dându-i lui t valori de la 0 la 2π cu un pas constant.

- Fie $(x_i,y_i)$ ultimul punct calculat de pe circumferința elipsei:

$x_i = xc + a*\cos(t) = xc + a*dx$ $\quad dx=\cos(t)$

$y_i = yc + b*\sin(t) = yc + b*dy$ $\quad dy = \sin(t)$

- Următorul punct, $(x_{i+1},y_{i+1})$, este distanțat de cel curent cu pasul unghiular:

$x_{i+1} = xc + a*\cos(t+pas) = xc + a*\ dx'$

$y_{i+1} = yc + b*\sin(t+pas) = yc + b*dy'$

$dx'=\cos(t+pas)=\cos(t)*\cos(pas) - \sin(t)*\sin(pas) = dx*c - dy*s$

$dy'=\sin(t+pas)=\cos(t)*\sin(pas) + \sin(t)*\cos(pas) = dx*s + dy*c$

unde c=cos(pas), s=sin(pas) sunt constante

# Aproximarea elipselor in spatiul discret(2)

Rezultă:

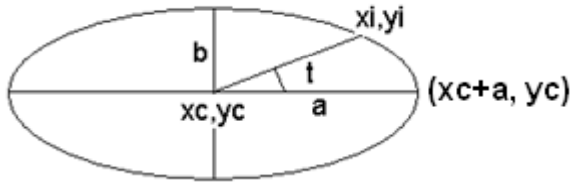
$x_{i+1} = xc+a*dx' = xc + a*(dx*c - dy*s)$

$y_{i+1} = yc+b*dy' = yc + b*(dx*s + dy*c)$

**// Generarea unei elipse cu axele paralele cu axele sistemului de coordonate carteziene 2D**

**void Elipsa(int xc, int yc, int a, int b, int N)**

```
{ double c,s,t,dx,dy,u, pas = 2*M_PI/N; // N: numarul de puncte calculate pe circumferinta elipsei
  int x, y, x1, y1 ;
  c=cos(pas); s=sin(pas);
  x= xc +a; y = yc; // dx = cos(0) = 1; dy = sin(0) = 0
  for(i=1, dx=1, dy=0; i<N; i++)
      {   t=dx;
          dx=dx*c - dy*s; // dx'
          dy=t*s + dy*c; //dy'
          x1 = xc +(int)(a*dx+0.5); y1 = yc+(int)(b*dy+0.5);
          line(x, y, x1, y1);
          x = x1; y = y1;
      } line(x, y, xc+a,yc);
}
```



# Aproximarea elipselor in spatiul discret(3)

**Generarea unei elipse rotite**

- Fie R, o elipsă cu centrul în originea sistemului de coordonate şi semiaxele a, b rotite cu unghiul u.
- Fie E, elipsa cu centrul în origine şi semiaxele a şi b suprapuse peste axele sistemului de coordonate.
- Punctele de pe elipsa R pot fi obținute rotind punctele de pe elipsa E în jurul originii cu unghiul u.
- Dacă centrul elipsei R este în (xc,yc), atunci după rotație se aplică fiecărui punct translația cu (xc,yc).

- Fie: x'= a*dx‘ , y'= b*dy‘ punctul curent de pe elipsa E, unde vectorul [dx',dy'] s-a obținut prin aplicarea formulei de recurență.
- Notăm: dx1=a*dx' , dy1=b*dy‘
- Punctul corespunzător de pe elipsa R este:

  xr = xc + dxr, yr = yc + dyr

  unde (dxr, dyr) se obtin prin rotatia vectorului (dx1,dy1) in jurul originii

  dxr = dx1*cos(u) - dy1*sin(u), dyr = dx1*sin(u) + dy1*cos(u)

# Aproximarea elipselor in spatiul discret(4)

// **Generarea unei elipse cu axele rotite fata de axele sistemului de coordonate carteziene 2D**

```
void Elipsa_rot(int xc, int yc, int a, int b, int N, double u)
{ double c,s,t,dx,dy,dx1,dy1,cu,su, pas = 2*M_PI/N;
  c=cos(pas);s=sin(pas);
  cu=cos(u) ;su=sin(u);
  x= xc+(int)(a*cu+0.5); y= yc+(int)(a*su+0.5); // punctul initial de pe elipsa rotita
  for(i=1, dx=1,dy=0; i< N; i++)
  {  t=dx;
     dx=dx*c-dy*s; //dx'
     dy=t*s+dy*c; // dy'
     dx1=a*dx; dy1=b*dy; // punctul de pe elipsa nerotita cu centrul in origine
     x1  = xc+ (int)(dx1*cu-dy1*su+0.5); y1 = yc + (int)(dx1*su+dy1*cu+0.5);
     line(x, y, x1, y1);
     x = x1; y = y1;
  }
  line(x, y, xc+(int)(a*cu+0.5),yc+(int)(a*su+0.5));
}
```