Rasterizarea vectorilor

Prof. univ. dr. ing. Florica Moldoveanu

Rasterizarea

- Primitivele geometrice (linii, cercuri, poligoane) sunt definite intr-un plan analogic, XOY, prin coordonate.

-Suprafata de afisare este un spatiu discret, alcatuit din celule de afisare (pixeli) adresate prin coordonate intregi, (0,0) <= (x,y) <= (xmax,ymax)

➢Pentru afisare, este necesara descompunerea primitivelor in fragmente care se afiseaza in celulelele suprafetei de afisare.

➢ **Rasterizarea =** operatia prin care o primitiva grafica este descompusa in fragmente

= aproximarea in spatiul discret a unei primitive definite analitic

Un vector este definit analitic prin coordonatele extremitatilor sale : (x1,y1) – (x2,y2)

➢ **Rasterizarea unui vector =** determinarea pixelilor spatiului discret care sunt cei mai apropiati de vectorul analitic.

Rasterizarea vectorilor

Ecuatia unui vector: y = m*x + b

m = (y2-y1)/(x2-x1) şi b=y1 - m*x1

Fie:

culoare : culoarea in care se afiseaza fragmentele care alcatuiesc vectorul

putpixel(x,y, culoare);// scrie culoarea in celula din memoria imagine(frame buffer), corespunzatoare pixelului de adresa (x,y)

Afisarea vectorului:

```
putpixel(x1,y1, culoare);
for(x=x1+1; x<x2; x++)
   { y=m*x+b ;
     putpixel(x, (int)(y+0.5),culoare);
   }
putpixel(x2,y2,culoare);
```

Dezavantaje algoritm :

- Nu se tine cont de panta dreptei: vectorii cu panta mare sunt aproximati prin cativa pixeli!
- Calculul fiecarei adrese de pixel de pe traseul vectorului contine operatii cu numere reale

Algoritmul Digital Differential Analyser (DDA)

- Tine cont de panta vectorului
- Coordonatele pixelilor de pe traseul vectorului se obtin printr-un calcul incremental (eficient)

Fie (x', y') si (x'',y'') - 2 puncte succesive de pe vector

- (y'' – y') / (x'' – x') = (y2 – y1) / (x2 – x1) = m

|m <1| : incrementare x : x'' = x' +1 → y'' = y' + m

| m >1| : incrementare y: y'' = y' + 1 → x'' = x' +1/m

```
void DDA(int x1, int y1, int x2, int y2, int culoare)
{double m,r; int x, y;
 if(x1==x2) //vector vertical
   {if(y1>y2)
      {y=y1; y1=y2; y2=y;}
    for(y=y1; y<=y2;y++)
      putpixel(x1,y,culoare);
    return;
 }
```

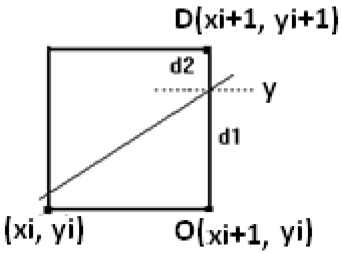
Algoritmul DDA(2)

```
if(y1==y2) //vector orizontal
  {if(x1>x2)
    {x=x1; x1=x2; x2=x;}
   for(x=x1; x<= x2; x++)  putpixel(x,y1,culoare);
   return;
  }
 m=(double)(y2-y1)/(x2-x1); r=abs(m);
// se genereaza vectorul de la (x1,y1) la (x2,y2)
 if(r<=1 && x1>x2  ||  r>1 &&  y1>y2)
      {x=x1; x1=x2; x2=x; y=y1; y1=y2; y2=y;}
 putpixel(x1, y1, culoare);
 if( r<=1)
      for(x=x1+1, r=y1; x<x2; x++)
        {r+=m;  putpixel(x, (int)(r+0.5), culoare); } // y = y + m
else
      {m=1/m;
       for(y=y1+1, r=x1; y<y2; y++)
        {r+=m;  putpixel((int)(r+0.5), y, culoare); } // x = x +1/m
      }
 putpixel(x2, y2, culoare);
}
```

Dezavantaj: calcule cu numere reale

Algoritmul Bresenham -pentru vectori din primul octant- (1)

- Contine numai operatii cu numere intregi
- Calcul incremental al coordonatelor pixelilor de pe traseul vectorului
- Pentru fiecare valoare a lui x se alege acel punct al spațiului discret care este mai apropiat de punctul de pe vectorul teoretic

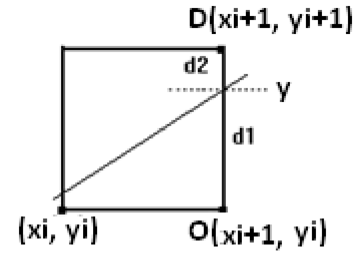


- Fie   m=(y2-y1)/(x2-x1) panta vectorului,
- (x_i, y_i) ultimul punct al spațiului discret ales în procesul de generare a vectorului.
- d1 distanța de la punctul de pe vectorul teoretic, (x_i+1, y), la punctul $O(x_i+1,y_i)$
- d2 distanța de la punctul de pe vectorul teoretic la punctul $D(x_i+1,y_i+1)$.
- O si D sunt adrese de pixeli (puncte ale spatiului discret)

➢ Următorul punct al spatiului discret ales pentru aproximarea vectorului va fi:

- O dacă d1<d2, D în caz contrar.
- Dacă d1=d2 se poate alege oricare dintre cele două puncte.

Algoritmul Bresenham(2)



- Exprimăm difereța d1-d2:

  $y=m*(x_i+1)+b$ este ordonata punctului de pe vectorul teoretic

  $d1=y-y_i=m*(x_i+1)+b-y_i$

  $d2=y_i+1-y=y_i+1-m*(x_i+1)-b$

  $d1-d2=2*m*(x_i+1)-2*y_i+2*b -1$

- Se înlocuieşte m cu dy/dx apoi se înmulțeşte în ambele părți cu dx. Rezultă:

  $t_i=(d1-d2)*dx=2*dy*(x_i+1)-2*dx*y_i+2*b*dx-dx =$

  $2*dy*x_i - 2*dx*y_i + 2*b*dx-dx + 2*dy$

- t_i reprezintă eroarea de aproximare în pasul i: pe baza sa se alege urmatorul punct al spatiului discret

  Notăm cu (x_{i+1},y_{i+1}) punctul care se va alege în pasul curent.

- Expresia erorii de aproximare pentru pasul următor este:

  $t_{i+1} = 2*dy*x_{i+1} - 2*dx*y_{i+1} + 2*b*dx - dx +2*dy$

Algoritmul Bresenham(3)

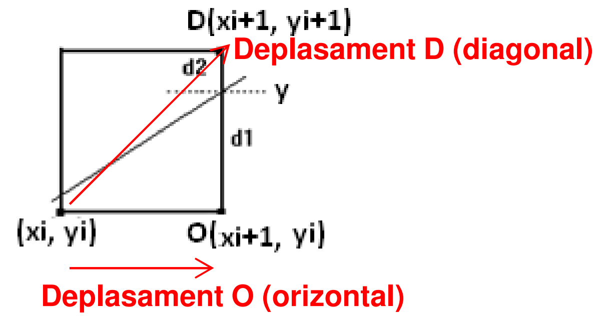
t_i=(d1-d2)*dx   (dx>0)

(1) Dacă $t_i <= 0$, se alege punctul O, deci $x_{i+1} = x_i+1$ şi $y_{i+1} = y_i$

Rezultă:

$$t_{i+1} = 2*dy*(x_i+1) - 2*dx*y_i + 2*b*dx - dx +2*dy$$

sau $\mathbf{t_{i+1} = t_i +2*dy}$



(2) Dacă $t_i > 0$, se alege punctul D, deci $x_{i+1} = x_i + 1$ şi $y_{i+1} = y_i +1$

Rezultă:

$$t_{i+1} = 2*dy*(x_i+1) - 2*dx*(y_i+1) + 2*b*dx - dx +2*dy$$

sau $\mathbf{t_{i+1} = t_i + 2*dy - 2*dx}$

- Valoarea variabilei de test se obtine prin calcul incremental: adunarea unei constante intregi

Eroarea de aproximare pentru primul pas: $x_i = x1$ şi $y_i = y1$:

$$t_1 = 2*dy*x1 - 2*dx*y1 + 2*dy – dx + 2*dx(y1 - (dy/dx)*x1)$$

sau $\mathbf{t_1 = 2*dy-dx}$

Algoritmul Bresenham(4)

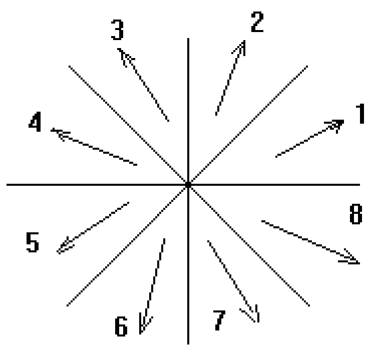
Implementare in C:

```
void Bres_vect(int x1, int y1, int x2, int y2, int culoare)
{ //pentru vectori cu panta cuprinsă între 0 şi 1
  int dx, c1, c2, x, y, t;
  dx=x2-x1;
  c1=(y2-y1)<<1;  // 2*dy
  c2=c1-(dx<<1);  // 2*dy – 2*dx
  t=c1-dx;  // 2*dy - dx
  putpixel(x1, y1, culoare);
  for(x=x1+1, y=y1; x<x2; x++)
   {if(t<0) t+=c1; //deplasament  O
      else  { t+=c2; y++;} // deplasament D
    putpixel(x,y,culoare);
   }
  putpixel(x2,y2,culoare);
}
```

Algoritmul Bresenham- generalizare(1)

Generalizarea algoritmului Bresenham pentru vectori de orice panta

- Vectorii definiți în spațiul 2D pot fi clasificați, pe baza pantei, în opt clase geometrice, numite **“octanți“**
- Un vector care apartine unui octant O are 7 vectori simetrici in ceilalti 7 octanti



dx=x2 – x1 şi dy=y2 – y1

octantul 1: dx>0 şi dy>0 şi dx>=dy;

octantul 2: dx>0 şi dy>0 şi dx<dy;

octantul 3: dx<0 şi dy>0 şi abs(dx)<dy;

octantul 4: dx<0 şi dy>0 şi abs(dx)>=dy;

octantul 5: dx<0 şi dy<0 şi abs(dx)>=abs(dy);

octantul 6: dx<0 şi dy<0 şi abs(dx)<abs(dy);

octantul 7: dx>0 şi dy<0 şi dx<abs(dy);

octantul 8: dx>0 şi dy<0 şi dx>=abs(dy);

Algoritmul Bresenham –generalizare(2)

Notam cu:

+h, deplasamentul orizontal spre dreapta (în sensul crescător al axei x),

-h, deplasamentul orizontal spre stânga (în sensul descrescător al axei x),

+v, deplasamentul vertical în sus (în sensul crescător al axei y),

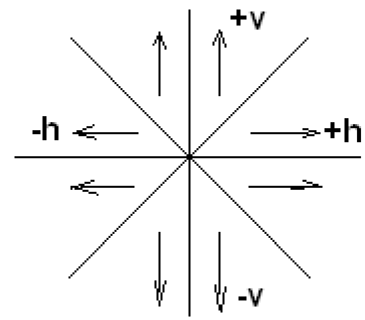
-v, deplasamentul vertical în jos (în sensul descrescător al axei y),

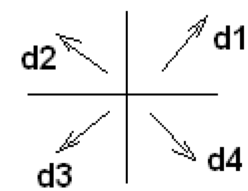
d1, deplasamentul diagonal dreapta-sus,

d2, deplasamentul diagonal stânga-sus,

d3, deplasamentul diagonal stânga-jos,

d4, deplasamentul diagonal dreapta-jos.





Octant	1	2	3	4	5	6	7	8
Deplas O	+h	+v	+v	-h	-h	-v	-v	+h
Deplas D	d1	d1	d2	d2	d3	d3	d4	d4

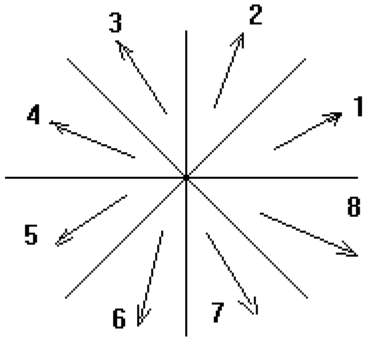
Corespondența între alegerea curentă într-un pas al algoritmului Bresenham (punctul O sau punctul D) şi deplasamentul echivalent în fiecare octant:

Algoritmul Bresenham – generalizare(3)

Algoritmul Bresemham generalizat -implementare in C

```
void Bres_general(int x1, int y1, int x2, int y2, int culoare)
{ int x, y, i, oct, dx, dy, absdx, absdy, c1, c2, t;
  if(x1==x2)
  //vertical
  {if(y1>y2){y=y1; y1=y2; y2=y;}
    for(y=y1; y<=y2;y++)
       putpixel(x1,y,culoare);
    return;
   }
 if(y1==y2)
 //orizontal
   {if(x1>x2) {x=x1; x1=x2; x2=x;}
     for(x=x1; x<= x2; x++)
       putpixel(x,y1,culoare);
    return;
   }
```

Algoritmul Bresenham – generalizare (4)




```
dx=x2-x1; dy=y2-y1;
absdx=abs(dx); absdy=abs(dy);
if(dx>0)//oct=1,2,7,8
   {if(dy>0)// oct=1,2
      if(dx>=dy) oct=1; else oct=2;
    else
      if(dx>=absdy) oct=8; else oct=7;
   }
else//3,4,5,6
   {if(dy>0)// oct=3,4
      if(absdx>=dy) oct=4; else oct=3;
    else
      if(absdx>=absdy) oct=5; else oct=6;
   }
// Numărul de paşi la execuția algoritmului generalizat este maxim(abs(dx), abs(dy))
// Dacă abs(dy) > abs(dx), se inversează rolul variabilelor dx şi dy în calculul constantelor c1 şi c2
if(absdy>absdx) //  adresele de pe traseul vectorului se obtin prin incrementarea lui y
    {x=absdx; absdx=absdy;absdy=x;}
  c1=absdy<<1;  c2=c1-(absdx<<1);
  t=c1-absdx;
```

Algoritmul Bresenham – generalizare (5)

```
putpixel(x1,y1,culoare);
 for(i=1,x=x1,y=y1; i<absdx; i++)
 { if(t<0) // deplasament O
   {t+=c1;
    switch(oct)
    {case 1:  case 8:x++; break; // +h
     case 4:  case 5:x--;break; // -h
     case 2:  case 3:y++;break; //+v
     case 6:  case 7:y--;break; // -v
    }
   }
  else
   {t+=c2 // deplasament D
    switch(oct)
    {case 1: case 2: x++;y++;break; // d1
     case 3: case 4: x--;y++;break; // d2
     case 5: case 6: x--;y--;break; // d3
     case 7: case 8: x++;y--;break; // d4
    }
  }
  putpixel(x,y,culoare);
 } putpixel(x2,y2,culoare);
}
```

Generarea liniilor intrerupte

```
void Bres_gen(int x1, int y1, int x2, int y2, int şablon)
{  // sablon: intreg pe 16 biti
    int val, bit, biţi[16] ;
    ..............................
    for(int i=0,val=1; i<16; i++)
    {    biţi[i] = şablon & val;
         val = val << 1;
    }
    ....................................
   if(biţi[0]]) putpixel(x1,y1,culoare);
   for(i=1,x=x1,y=y1, bit=1; i<absdx; i++)
  { .........................................
     if(biţi[(bit++) % 16]) putpixel(x,y,culoare);
  }
  if(biţi[bit % 16) putpixel(x2,y2,culoare);
}
```