Capitolul 2

Grafică rastru 2D

În acest capitol vom prezenta diverşi algoritmi de afişare a unor obiecte matematice simple (segmente de dreaptă, cercuri, elipse, poligoane) pe ecrane rastru (algoritmi de conversie prin scanare$_{\text{scan converting}}$ a primitivelor geometrice în pixeli). Ulterior vom prezenta algoritmi care realizează o colorare uniformă a pixelilor interiori unei primitive geometrice (dreptunghi, poligon oarecare, cerc, elipsă).

Ecranele rastru vor fi modelate prin spaţiul discret $\mathbb{Z} \times \mathbb{Z}$ şi funcţia $value : \mathbb{Z} \times \mathbb{Z} \to V$, unde, în cazul ecranelor rastru monocrome (cu 1 bit / pixel) $V = \mathbb{B}$ ($= \{\mathbf{true}, \mathbf{false}\}$ sau $\{0, 1\}$) iar în cazul ecranelor multicrome (cu $n \geq 2$ biţi / pixel) alegem $V \subseteq \mathbb{N}$ astfel încât $\mid V \mid \geq$ numărul nivelurilor de intensitate (sau numărul de culori).

Din punct de vedere grafic vom reprezenta ecranele rastru printr-o mulţime de drepte paralele (orizontale şi verticale - linii şi coloane) egal distanţate (sub forma unei grile) iar pixelii (aprinşi cu diverse intensităţi sau stinşi) vor fi discuri centrate în punctele de intersecţie ale grilei (denumite vârfurile grilei). În figura 2.1, pag. 21 am reprezentat un ecran rastru.

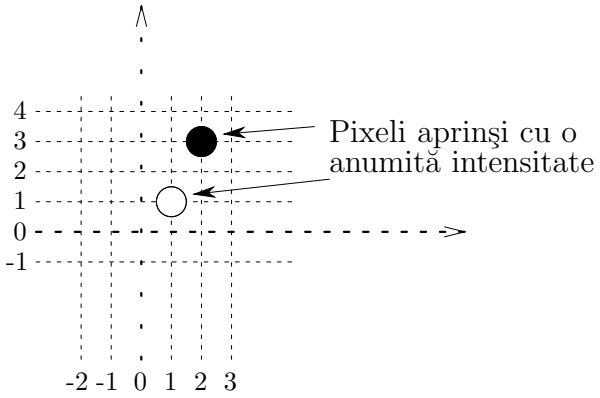

Figura 2.1: Reprezentarea unui ecran rastru

2.1 Afişarea segmentelor de dreaptă

Vom afişa segmente de dreaptă pe ecrane rastru monocrome. Segmentele de dreaptă vor fi aproximate prin mulţimi de pixeli aprinşi. Avem problema următoare :

Problema 2.1

Fie $[P_0P_n]$ un segment de dreaptă, unde $P_0(x_0, y_0)$ şi $P_n(x_n, y_n)$. Mai presupunem că $(x_0, y_0) \in \mathbb{Z} \times \mathbb{Z}$ şi $(x_n, y_n) \in \mathbb{Z} \times \mathbb{Z}$. Să se determine secvenţa de pixeli $M = (V_0, V_1, \ldots, V_n)$ având proprietăţile următoare :

1. $(\forall 0 \leq i \leq n)(V_i(x_i, y_i) \wedge (x_i, y_i) \in \mathbb{Z} \times \mathbb{Z})$ i.e., pixelul V_i este situat la intersecţia dintre coloana $x = x_i$ şi linia $y = y_i$, $\forall 0 \leq i \leq n$.

2. Fie m panta dreptei P_0P_n (ecuaţia dreptei P_0P_n este $y = m \cdot x + k$), unde m şi k se obţin din ecuaţia dreptei care trece prin punctele P_0 şi P_n : $\frac{y-y_0}{y_n-y_0} = \frac{x-x_0}{x_n-x_0}$). Dacă $\mid m \mid \leq 1$ atunci $(\forall 0 \leq i < n)(x_{i+1} = x_i + 1)$. În mod informal, această condiţie semnifică faptul că pentru segmente de dreaptă având panta $\mid m \mid \leq 1$ trebuie să existe un unic pixel aprins în fiecare coloană intersectată de segmentul de dreaptă.

3. Dacă $\mid m \mid > 1$ atunci $(\forall 0 \leq i < n)(y_{i+1} = y_i + 1)$. În mod informal, această condiţie semnifică faptul că pentru segmente de dreaptă având panta $\mid m \mid > 1$ trebuie să existe un unic pixel aprins în fiecare linie intersectată de segmentul de dreaptă.

4. Distanţa dintre pixelul $V_i(x_i, y_i)$ şi punctul de intersecţie dintre $[P_0P_n]$ şi coloana $x = x_i$ (dacă $\mid m \mid \leq 1$) sau dintre $V_i(x_i, y_i)$ şi punctul de intersecţie dintre $[P_0P_n]$ şi linia $y = y_i$ (dacă $\mid m \mid > 1$) trebuie să fie minimă, i.e., pixelul V_i este cel mai apropiat de punct dintre toţi pixelii de pe coloana $x = x_i$ sau, similar, de pe linia $y = y_i$.

▪

Observaţia 2.1

Cerinţele 2 şi 3 ale problemei 2.1, pag. 21 au rolul de a asigura o anumită "continuitate" în reprezentarea unui segment de dreaptă. Încălcarea acestor cerinţe poate conduce la afişări incorecte, de exemplu reprezentarea unui segment de dreaptă continuu printr-un segment de dreaptă discontinuu. ▪

În cele ce urmează vom prezenta o serie de algoritmi pentru rezolvarea problemei de mai sus. Vom pleca de la un algoritm iniţial din care vom obţine algoritmi superiori din punctul de vedere al complexităţii timp. Rafinarea algoritmului iniţial constă în eliminarea operaţiilor de înmulţire şi împărţire şi înlocuirea acestora cu operaţii de adunare şi scădere. Vom încerca să utilizăm variabile de tip **int** în loc de variabile de tip **float** sau **double**.

Algoritmul 1 AfişareSegmentDreaptă1

`Date de intrare :` $(x_0, y_0), (x_n, y_n) \in \mathbb{Z} \times \mathbb{Z}$

`Date de ieşire :` $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, secvenţa de pixeli cerută de problema 2.1, pag. 21. Presupunem că ne aflăm în cazul $\mid m \mid \leq 1$.

`Metodă :`

```
procedure AfisareSegmentDreapta1 (x0,y0,xn,yn : Z,
                                  var M : list of (Z x Z))
{
  double m = (yn - y0) / (xn - x0);
  double k = y0 - m * x0;
  // am calculat dreapta y = m * x + k care trece prin
  // punctele P0 si Pn
  M = \epsilon; // M este initial lista vida
```

```
  for (int x = x0; x <= xn; x++)
  {
    double y;
    y = m * x + k;
    // secventei M i se adauga, prin concatenare, un
    // nou pixel V(x, floor(y + 0.5))
    M = M . (x, floor(y + 0.5));
  }
}
```

Mai sus notaţia `\epsilon` semnifică ϵ (notaţia pentru lista vidă) şi `M = M . (x, floor(y + 0.5));` semnifică concatenarea listei M cu un element : $M = M \cdot (x, floor(y + 0.5))$.

▪

Exerciţiul 2.1

Modificaţi algoritmul 1, pag. 22 astfel încât să funcţioneze corect şi în cazul în care $\mid m \mid > 1$. În cazul în care avem $\mid m \mid > 1$ procedura *AfisareSegmentDreapta1* poate calcula o secvenţă de pixeli $M = ((x_0, y_0), \ldots, (x, y), (x + 1, y'), \ldots, (x_n, y_n))$, cu $y' > y + 1$ şi $y' \in \mathbb{Z}$ (pentru a nu complica expunerea) ceea ce contravine cerinţei 3 din problema 2.1, pag. 21. ▪

Algoritmul următor provine din algoritmul 1, pag. 22. Este un algoritm *incremental* (aceasta însemnând că valorile (x', y') ale unui pixel sunt obţinute din valorile (x, y) ale pixelului imediat precedent printr-o operaţie de adunare).

Algoritmul 2 AfişareSegmentDreaptă2

`Date de intrare` : $(x_0, y_0), (x_n, y_n) \in \mathbb{Z} \times \mathbb{Z}$

`Date de ieşire` : $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, secvenţa de pixeli cerută de problema 2.1, pag. 21. Presupunem că ne aflăm în cazul $\mid m \mid \leq 1$.

`Metodă` : În algoritmul 1, pag. 22 înlocuim instrucţiunea `y = m * x + k;` : să presupunem că $M = (x_0, y_0) \cdot \ldots \cdot (x_i, y_i)$. La pasul următor lui M i se adaugă pixelul (x_{i+1}, y_{i+1}) : $x_{i+1} = x_i + 1$ şi $y_{i+1} = m \cdot x_{i+1} + k$ deci $y_{i+1} = m \cdot (x_i + 1) + k = (m \cdot x_i + k) + m$. Dar $y_i = m \cdot x_i + k$ şi deci $y_{i+1} = y_i + m$.

```
procedure AfisareSegmentDreapta2 (x0,y0,xn,yn : Z,
                                  var M : list of (Z x Z))
{
  double m = (yn - y0) / (xn - x0);
  double y = y0;
  M = \epsilon; // M este initial lista vida
  for (int x = x0; x <= xn; x++)
  {
    // secventei M i se adauga, prin concatenare, un
    // nou pixel V(x, floor(y + 0.5))
    M = M . (x, floor(y + 0.5));
    y += m;
  }
}
```

▪

Observaţia 2.2

Şi în cazul algoritmului 2, pag. 23 cazul $| m | > 1$ este tratat incorect. Modificaţi algoritmul 2, pag. 23 astfel încât cazul $| m | > 1$ să fie tratat corect. ▪

Algoritmul următor (propus de Pitteway în 1967 şi de van Aken în 1984 şi variantă a unui algoritm propus de Bresenham în 1965) calculează aceeaşi secvenţă de pixeli ca algoritmii anteriori dar utilizează doar aritmetica întregilor (astfel vom renunţa la utilizarea variabilelor de tip **double** sau la utilizarea funcţiei parte întreagă inferioară *floor*).

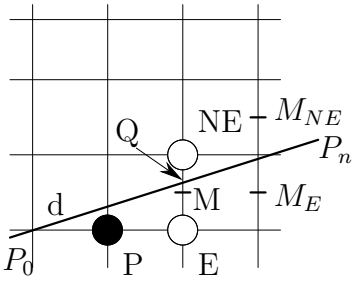

Figura 2.2: Alegerea unuia dintre pixelii E sau NE

Principiul acestui algoritm este următorul : să presupunem că pixelul P este pixelul curent selectat (i.e., ultimul pixel calculat al secvenţei M).

Mai presupunem, fără a restrânge generalitatea, că dreapta d (determinată de segmentul $[P_0P_n]$) are panta $| m | \leq 1$ şi poziţia ca în figura 2.2, pag. 24 (i.e., $x_0 < x_n$, $y_0 < y_n$ şi $\frac{y_n - y_0}{x_n - x_0} < 1$).

În acest caz următorul pixel din secvenţă poate fi fie pixelul E (*est*), fie pixelul NE (*nord-est*). Conform cerinţei 4 din problema 2.1, pag. 21 va trebui ales acel pixel a cărui distanţă până la dreapta d este $\min(\mathit{distanta}(\text{E}, d), \mathit{distanta}(\text{NE}, d))$.

Decizia alegerii unuia dintre cei doi pixeli este luată în funcţie de poziţia punctului M situat la mijlocul distanţei dintre E şi NE. Dacă punctul M este situat *sub* dreapta d (aceasta însemnând că punctul Q, care este intersecţia dreptei d cu segmentul de dreaptă [E NE], este situat între M şi NE) atunci este evident că NE este pixelul cel mai apropiat de d şi va fi selectat. Altfel, dacă M este situat *deasupra* lui d, i.e., Q este situat între M şi E atunci este selectat pixelul E.

Notaţia 2.1

În cele ce urmează folosim următoarele notaţii : $P(x_P, y_P)$, $E(x_P + 1, y_P)$, $NE(x_P + 1, y_P + 1)$, $Q = d \cap [\text{E NE}]$. Ecuaţia dreptei d este $F(x, y) = 0$, unde $F(x, y) = a \cdot x + b \cdot y + c$. Coeficienţii a, b, c se obţin plecând de la ecuaţia dreptei d (care trece prin P_0 şi P_n) :

$$\frac{x - x_0}{x_n - x_0} = \frac{y - y_0}{y_n - y_0} \tag{2.1}$$

Utilizăm notaţiile[1] $d_x \doteqdot x_n - x_0$ şi $d_y \doteqdot y_n - y_0$. Rescriem ecuaţia 2.1 :

$$\frac{x - x_0}{d_x} = \frac{y - y_0}{d_y} \tag{2.2}$$

[1]În cele ce urmează, $x \doteqdot y$ înseamnă x este o notaţie pentru y

şi deci avem $y = \frac{d_y}{d_x} \cdot x + (y_0 - \frac{d_y}{d_x} \cdot x_0)$, şi deci avem

$$d_y \cdot x - d_x \cdot y + (d_x \cdot y_0 - d_y \cdot x_0) = 0 \tag{2.3}$$

În concluzie coeficienţii a, b, c sunt : $a = d_y$, $b = -d_x$, $c = d_x \cdot y_0 - d_y \cdot x_0$. Ecuaţia 2.3 se mai numeşte şi ecuaţia *implicită* a dreptei d. ∎

Proprietatea 2.1 (poziţia punctului M relativ la dreapta d)
Fie punctul $M(x', y')$ şi dreapta d dată prin ecuaţia implicită $F(x, y) = 0$. Atunci :

1. $M \in d \iff F(x', y') = 0$.
2. M este situat sub dreapta d ddacă $F(x', y') > 0$.
3. M este situat deasupra dreptei d ddacă $F(x', y') < 0$.

∎

Demonstraţie

1. Evident.
2. Să presupunem că M este situat sub dreapta d. Fie $V = d \cap \{x \mid x = x'\}$. Deci V are coordonatele $(x', \frac{-a \cdot x' - c}{b})$. Atunci $M(x', y')$ este situat sub d ddacă $y' < \frac{-a \cdot x' - c}{b}$. Dacă $b < 0$ (şi aici suntem în acest caz deoarece $b = -d_x$) atunci avem $M(x', y')$ este situat sub d ddacă $a \cdot x' + b \cdot y' + c > 0 \iff F(x', y') > 0$
3. Se arată în mod similar cu cazul precedent.

□

Am văzut mai sus că alegerea unuia din pixelii E sau NE depinde de poziţia punctului M faţă de dreapta d. Conform proprietăţii 2.1, pag. 25 putem acum să descriem *matematic* acest lucru : punctul M are coordonatele $x_P + 1$ şi $y_P + \frac{1}{2}$ şi notăm $d \doteqdot F(x_P + 1, y_P + \frac{1}{2})$. d este o variabilă de decizie utilizată în algoritm[2]. Dacă $d > 0$ atunci alegem pixelul NE şi dacă $d \leq 0$ alegem pixelul E.

Ştim că primul pixel din secvenţa de pixeli este chiar extremitatea $P_0(x_0, y_0)$ a segmentului $[P_0P_n]$. Valoarea de iniţializare a variabilei de decizie d este atunci $F(x_0 + 1, y_0 + \frac{1}{2}) = a \cdot (x_0 + 1) + b \cdot (y_0 + \frac{1}{2}) + c = a \cdot x_0 + b \cdot y_0 + c + a + \frac{b}{2} = F(x_0, y_0) + a + \frac{b}{2} = a + \frac{b}{2}$ (deoarece $P_0 \in [P_0P_n]$ avem $F(x_0, y_0) = 0$).

Deoarece există posibilitatea ca $\frac{b}{2} \notin \mathbb{Z}$ vom considera $F(x, y) = 2 \cdot (a \cdot x + b \cdot y + c)$ şi se observă că această alegere nu schimbă semnul variabilei de decizie d.

În algoritmul care urmează, după alegerea unuia dintre pixelii E sau NE ca fiind pixel curent, trebuie să modificăm şi punctul de mijloc M (deci şi valoarea variabilei de decizie d). Aşa cum vom vedea această modificare este *incrementală*.

Să presupunem că pixelul E a fost cel selectat. Atunci punctul de mijloc M devine $M_E(x_P + 2, y_P + \frac{1}{2})$. Noua valoare a variabilei de decizie d este $d' = F(M_E) = F(x_P + 2, y_P + \frac{1}{2}) = 2 \cdot (a \cdot (x_P + 2) + b \cdot (y_P + \frac{1}{2}) + c) = 2 \cdot (a \cdot (x_P + 1) + b \cdot (y_P + \frac{1}{2}) + c) + 2 \cdot a = d + 2 \cdot a$.

Să presupunem că pixelul NE a fost cel selectat. Atunci punctul de mijloc M devine $M_{NE}(x_P + 2, y_P + \frac{3}{2})$. Noua valoare a variabilei de decizie d este $d' = F(M_{NE}) = F(x_P + 2, y_P + \frac{3}{2}) = 2 \cdot (a \cdot (x_P + 2) + b \cdot (y_P + \frac{3}{2}) + c) = 2 \cdot (a \cdot (x_P + 1) + b \cdot (y_P + \frac{1}{2}) + c) + 2 \cdot (a + b) = d + 2 \cdot (a + b)$.

[2] a nu se confunda cu dreapta d care trece prin punctele P_0 şi P_n

Pe baza celor de mai sus putem descrie algoritmul de afişare a segmentelor de dreaptă care utilizează tehnica punctului de mijloc.

Algoritmul 3 AfişareSegmentDreaptă3

`Date de intrare :` $(x_0, y_0), (x_n, y_n) \in \mathbb{Z} \times \mathbb{Z}$

`Date de ieşire :` $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, secvenţa de pixeli cerută de problema 2.1, pag. 21. Presupunem că ne aflăm în cazul $\mid m \mid \leq 1$ şi $x_0 < x_n$, $y_0 < y_n$.

`Metodă :`

```
procedure AfisareSegmentDreapta3 (x0,y0,xn,yn : Z,
                                  var M : list of (Z x Z))
{
  // valoarea initiala a variabile de decizie
  // dx, dy sunt constante - a se vedea mai sus
  int d = 2 * dy - dx;
  int dE = 2 * dy;
  int dNE = 2 * (dy - dx);
  int x = x0, y = y0;
  M = (x,y);
  while (x < xn)
  {
    if (d <= 0) { /* alegem E */    d += dE; x++; }
    else        { /* alegem NE */   d += dNE; x++; y++; }
    M = M . (x, y);
  }
}
```

■

Observaţia 2.3

Algoritmii 1, pag. 22, 2, pag. 23 şi 3, pag. 26 de afişare a segmentelor de dreaptă pot fi folosiţi pentru afişarea liniilor poligonale sau a contururilor poligoanelor. ■

2.2 Afişarea cercurilor

Ca şi în cazul afişării segmentelor de dreaptă vom prezenta mai întâi algoritmi relativ simpli de afişare a unui cerc (dar care utilizează operaţii complexe în raport cu operaţia de adunare a întregilor). Ulterior vom prezenta algoritmi mai complicaţi dar având o complexitate timp mai bună (deoarece sunt incrementali şi utilizează doar operaţii de adunare a întregilor).

Vom prezenta algoritmi de afişare a unui cerc având centrul în origine şi rază $R \in \mathbb{Z}$. Ecuaţia unui astfel de cerc este $F(x, y) = 0$, unde $F(x, y) = x^2 + y^2 - R^2$.

Algoritmul 4 AfişareCerc1

`Date de intrare :` $R \in \mathbb{Z}$, raza unui cerc centrat în origine,

`Date de ieşire :` $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, o secvenţă de pixeli care aproximează cadranul I al cercului de rază R centrat în origine (cadran descris matematic astfel : $\{(x, y) \mid (x, y) \in \mathbb{R}^2 \wedge 0 \leq x \leq R \wedge y = \sqrt{R^2 - x^2}\}$).

`Metodă :`

```
procedure AfisareCerc1 (R : Z, var M : list of (Z x Z))
{
  M = \epsilon;
  for (int x = 0; x <= R; x++)
  {
    M = M . (x, floor(sqrt(R*R - x*x) + 0.5));
  }
}
```

■

Observaţia 2.4

Utilizând procedura AfişareCerc1 putem afişa un cerc astfel : dat un pixel $P(x, y) \in M$ putem obţine pixelii simetrici din cadranele 2,3,4 prin simetrie faţă de axa Oy, faţă de origine şi faţă de axa Ox : $P''(-x, y)$, $P'''(-x, -y)$ şi $P^{\dagger}(x, -y)$. ■

Observaţia 2.5

Algoritmul 4, pag. 26 utilizează operaţii relativ costisitoare (*floor*, *sqrt*). În plus prezintă şi dezavantajul de a genera pixeli în mod neuniform : e.g., în cazul unui cerc de rază 17 centrat în origine, primii doi pixeli generaţi sunt $(0, 17)$ şi $(1, 17)$ iar ultimii doi pixeli generaţi sunt $(16, 6)$ şi $(17, 0)$. Acest ultim dezavantaj nu apare în cazul algoritmului următor. ■

Algoritmul 5 AfişareCerc2

`Date de intrare` : $R \in \mathbb{Z}$, raza unui cerc centrat în origine,

`Date de ieşire` : $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, o secvenţă de pixeli care aproximează cadranul I al cercului de rază R centrat în origine.

`Metodă` : Vom folosi coordonatele polare ale pixelilor : $\begin{cases} x &= R \cdot \cos t \\ y &= R \cdot \sin t \end{cases}$, unde $0 \leq t \leq \frac{\pi}{2}$.

```
procedure AfisareCerc2 (R : Z, var M : list of (Z x Z))
{
  M = \epsilon;
  for (double t = 0; t <= PI/2; t+=PI/100)
  {
    M = M . (floor(R*cos(t) + 0.5), floor(R*sin(t) + 0.5));
  }
}
```

■

Şi în cazul cercului putem aplica tehnica punctului de mijloc. Principiul este acelaşi ca în cazul unui segment de dreaptă : plecând de la pixelul curent, următorul pixel din secvenţă este ales dintre doi pixeli, în funcţie de poziţia faţă de cerc a punctului situat la jumătatea distanţei dintre cei doi pixeli. Vom genera prin această metodă doar pixelii din octantul al doilea al cercului de rază R centrat în origine (descris matematic astfel : $\{(x, y) \mid (x, y) \in \mathbb{R}^2 \wedge x^2 + y^2 = R^2 \wedge 0 \leq x \leq \frac{R \cdot \sqrt{2}}{2} \wedge \frac{R \cdot \sqrt{2}}{2} \leq y \leq R\}$).

Presupunând că $P(x_P, y_P)$ este pixelul curent, alegerea se va face între pixelii $E(x_P + 1, y_P)$ (est) şi $SE(x_P + 1, y_P - 1)$ (sud-est), conform figurii 2.3, pag. 28. Punctul de mijloc este $M(x_P + 1, y_P - \frac{1}{2})$. Dacă a fost ales E atunci noul punct de mijloc este $M_E(x_P + 2, y_P - \frac{1}{2})$ iar în cazul alegerii SE noul punct de mijloc este $M_{SE}(x_P + 2, y_P - \frac{3}{2})$.

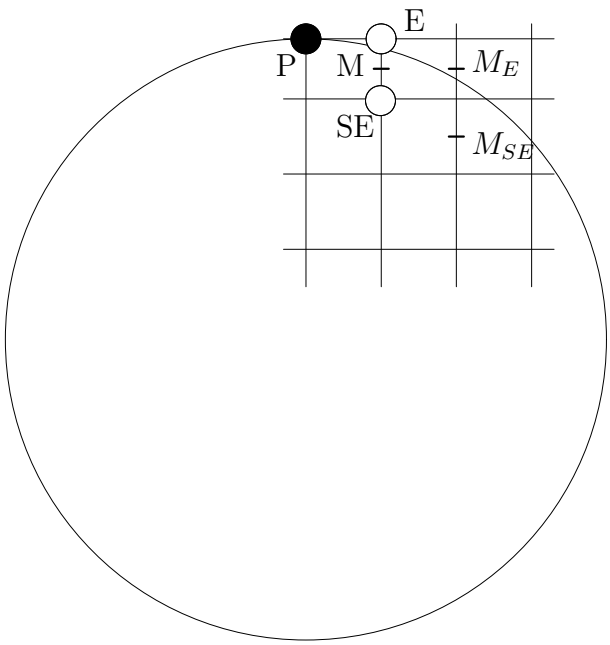

Figura 2.3: Alegerea unuia dintre pixelii E sau SE

Dacă punctul M este situat în interiorul cercului atunci pixelul E este mai apropiat de cerc şi va fi selectat iar dacă M este situat în exteriorul cercului atunci pixelul SE este mai apropiat de cerc şi va fi selectat.

Proprietatea 2.2

Fie $M(x', y')$. Atunci :

1. $M \in$ cercului de rază R centrat în origine ddacă $F(x', y') = 0$,
2. M este situat în interiorul cercului de rază R centrat în origine ddacă $F(x', y') < 0$,
3. M este situat în exteriorul cercului de rază R centrat în origine ddacă $F(x', y') > 0$,

unde $F(x, y) = x^2 + y^2 - R^2$. ∎

Demonstraţie

Ecuaţia cercului de rază R centrat în origine este $F(x, y) = 0$, unde $F(x, y) = x^2 + y^2 - R^2$.

1. Evident.
2. Originea este situată în interiorul cercului şi deci un punct din interiorul cercului $M(x', y')$ are proprietatea că $sgn(F(x', y')) = sgn(F(0, 0)) = sgn(-R^2) = -1$ şi deci $F(x', y') < 0$.
3. Prin excluderea celor două cazuri anterioare.

□

Ca şi în cazul segmentului de dreaptă considerăm o variabilă de decizie $d \doteqdot F(M) = F(x_P + 1, y_P - \frac{1}{2})$. Primul pixel din octantul al doilea care este generat de algoritm are coordonatele $(0, R)$ deci valoarea iniţială a variabilei de decizie este $F(0+1, R-\frac{1}{2}) = F(1, R-\frac{1}{2}) = (1) + (R^2 - R + \frac{1}{4}) - R^2 = \frac{5}{4} - R$.

Dacă $d < 0$ atunci M este în interiorul cercului şi deci alegem pixelul E. În acest caz noua valoare a variabilei de decizie este $d' = F(M_E) = F(x_P+2, y_P-\frac{1}{2}) = (x_P+2)^2+(y_P-\frac{1}{2})^2-R^2 = (x_P+1)^2+(y_P-\frac{1}{2})^2-R^2+2\cdot x_P+3 = d+2\cdot x_P+3$.

Dacă $d > 0$ atunci M este în exteriorul cercului şi deci alegem pixelul SE. În acest caz noua valoare a variabilei de decizie este $d' = F(M_{SE}) = F(x_P+2, y_P-\frac{3}{2}) = (x_P+2)^2+(y_P-\frac{3}{2})^2-R^2 = (x_P+1)^2+(y_P-\frac{1}{2})^2-R^2+2\cdot x_P+3-2\cdot y_P+2 = d+2\cdot x_P-2\cdot y_P+5$.

Pe baza observaţiilor de mai sus putem descrie o versiune preliminară a algoritmului de afişare a cercului care utilizează tehnica punctului de mijloc. Este o versiune care va fi rafinată prin eliminarea valorilor de tip **float** sau **double** (valori posibile pentru variabila de decizie d) precum şi prin eliminarea operaţiilor de înmulţire (e.g., $d' = d+2\cdot x_P+3$ sau $d' = d+2\cdot x_P-2\cdot y_P+5$).

Algoritmul 6 AfişareCerc3

`Date de intrare :` $R \in \mathbb{Z}$, raza unui cerc centrat în origine,

`Date de ieşire :` $M \in$ `list of` $(\mathbb{Z}\times\mathbb{Z})$, o secvenţa de pixeli care aproximează cel mai bine octantul al II-lea al cercului de rază R centrat în origine. Prin simetrie faţă de Oy, O, Ox şi faţă de bisectoarea $y = x$ este afişat cercul în întregime.

`Metodă :`

```
procedure AfisarePuncteCerc3 (x,y : Z, var M : list of (Z x Z))
{
  M = M . (x,y) . (-x,-y) . (-x,y) . (x,-y);
  if (x != y)
    M = M . (y,x) . (-y,-x) . (-y,x) . (y,-x);
}
procedure AfisareCerc3 (R : Z, var M : list of (Z x Z))
{
  int x = 0, y = R;
  double d = 5.0 / 4 - R;
  M = \epsilon;
  AfisarePuncteCerc3(x, y, M);
  while (y > x)
  {
    // conditia y > x caracterizeaza octantul al 2-lea
    if (d < 0)
    {
      // alegem pixelul E
      d += 2 * x + 3;
      x++;
    }
    else
    {
      // alegem pixelul SE
      d += 2 * ( x - y ) + 5;
      x++;
      y--;
    }
    AfisarePuncteCerc3(x, y, M);
```

```
  }
}
```

■

În algoritmul precedent aducem următoarele modificări : mai întâi înlocuim variabila de decizie d cu o nouă variabilă de decizie h cu proprietatea că valorile lui h sunt doar de tip **int** (se observă că valorile variabilei de decizie d sunt de tip **double** : valoarea iniţială $\frac{5}{4} - R$ este de tip **double** iar valorile ulterioare se obţin prin incrementarea cu valori de tip **int**).

Fie $h = d - \frac{1}{4}$. Atunci valoarea iniţială a lui h este $\frac{5}{4} - R - \frac{1}{4} = 1 - R \in \mathbb{Z}$. Observăm că instrucţiunile de incrementare `d += 2 x + 3;` şi `d += 2 ( x - y ) + 5;` nu se modifică (deoarece, de exemplu, putem rescrie instrucţiunea `d += 2 x + 3;` sub forma `d = d + 2 x + 3;` şi efectuând înlocuirea în membrii stâng/drept a variabilei d cu $h + \frac{1}{4}$ vom obţine `h + 1/4 = h + 1/4 + 2 x + 3;` şi deci noua instrucţiune este `h = h + 2 x + 3;` deci `h += 2 x + 3;`).

Avem de asemenea şi faptul $d < 0 \iff h < 0$ ($d < 0 \iff h < -\frac{1}{4}$ dar valoarea iniţială a lui h este din $\mathbb{Z}$ şi de asemenea şi valorile ulterioare şi deci rezultă că $h \in \mathbb{Z}$. Avem deci $h \in \mathbb{Z} \wedge h < -\frac{1}{4}$ şi deci $h < 0$).

În concluzie, în algoritmul 6, pag. 29 este suficient să modificăm instrucţiunea de iniţializare a variabilei de decizie d din `double d = 5.0 / 4 - R;` în `int d = 1 - R;` . Mai rămân operaţiile de înmulţire. Acestea sunt eliminate în cele ce urmează. Utilizăm următoarele notaţii : $d_E \doteqdot 2 \cdot x_P + 3$, şi $d_{SE} \doteqdot 2 \cdot (x_P - y_P) + 5$. Observăm că expresiile d_E şi d_{SE} depind de pixelul curent $P(x_P, y_P)$. Valorile iniţiale[3] ale acestor expresii sunt 3 şi $-2 \cdot R + 5$. În cazul alegerii pixelului E pixelul curent va avea coordonatele $(x_P + 1, y_P)$. Valorile expresiilor d_E şi d_{SE} se modifică astfel : $d'_E = 2 \cdot (x_P + 1) + 3 = 2 \cdot x_P + 3 + 2 = d_E + 2$, $d'_{SE} = 2 \cdot (x_P + 1 - y_P) + 5 = 2 \cdot (x_P - y_P) + 5 + 2 = d_{SE} + 2$. În cazul alegerii pixelului SE pixelul curent va avea coordonatele $(x_P + 1, y_P - 1)$. Valorile expresiilor d_E şi d_{SE} se modifică astfel : $d'_E = 2 \cdot (x_P + 1) + 3 = 2 \cdot x_P + 3 + 2 = d_E + 2$, $d'_{SE} = 2 \cdot (x_P + 1 - y_P + 1) + 5 = 2 \cdot (x_P - y_P) + 5 + 4 = d_{SE} + 4$.

În concluzie, algoritmul de afişare al cercurilor care utilizează tehnica punctului de mijloc (şi care utilizează doar operaţii de adunare şi valori de tip **int**) este :

Algoritmul 7 AfişareCerc4

`Date de intrare :` $R \in \mathbb{Z}$, raza unui cerc centrat în origine,

`Date de ieşire :` $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, o secvenţă de pixeli care aproximează cercul.

`Metodă :`

```
procedure AfisareCerc4 (R : Z, var M : list of (Z x Z))
{
  int x = 0, y = R;
  int d = 1 - R;
  int dE = 3, dSE = -2 * R + 5;
  M = \epsilon;
  AfisarePuncteCerc3(x, y, M);
  while (y > x)
  {
    if (d < 0)
    {
```

[3]pixelul curent în acest caz este pixelul iniţial $P(0, R)$

```
      d += dE;
      dE += 2;
      dSE += 2;
    }
    else
    {
      d += dSE;
      dE += 2;
      dSE += 4;
      y--;
    }
    x++;
    AfisarePuncteCerc3(x, y, M);
  }
}
```

2.3 Afişarea elipselor

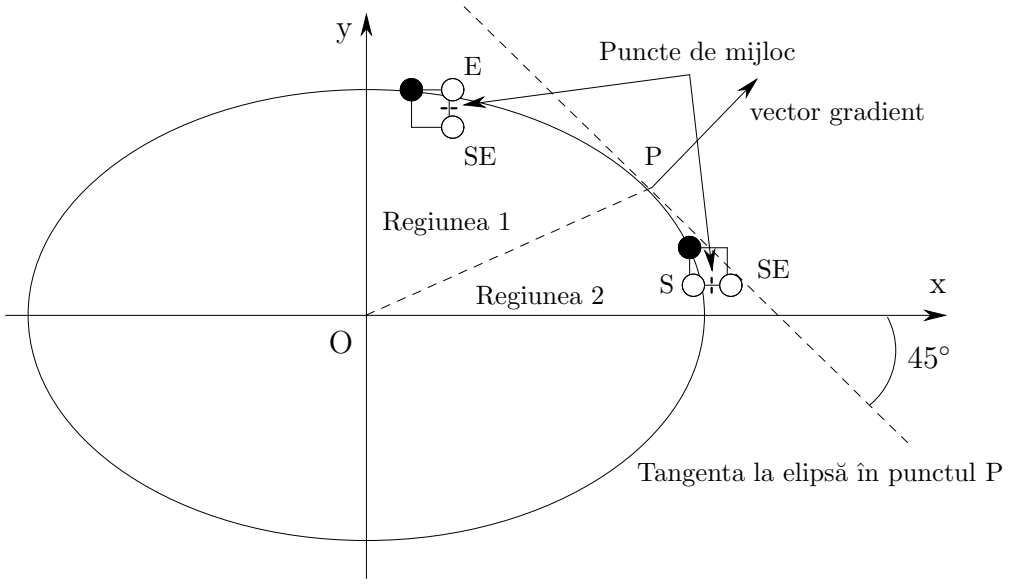

Figura 2.4: Alegerea unuia dintre pixelii E/SE sau S/SE

Vom prezenta algoritmi de afişare a unor elipse având centrul în origine şi semiaxe $a, b \in \mathbb{Z}$. Ecuaţia unei astfel de elipse este $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ şi o rescriem sub forma $F(x, y) = 0$, unde $F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2$.

Şi in cazul elipselor putem aplica algoritmi de genul algoritmului 5, pag. 27 pentru cercuri, în acest caz utilizând coordonatele polare : $\begin{cases} x &= a \cdot \cos t \\ y &= b \cdot \sin t \end{cases}$, unde $0 \leq t < 2\pi$.

Spre deosebire de algoritmii de afişare ai cercurilor, în cazul elipsei nu există simetria faţă de bisectoarea $y = x$. Deci vom efectua afişarea pixelilor în cadranul I şi, prin simetrie faţă de axele Ox, Oy, respectiv origine, afişarea pixelilor din celelalte cadrane.

Afişarea elipsei în cadranul I (vezi figura 2.4, pag. 31) se face în funcţie de una din cele două regiuni. În regiunea 1 trebuie să alegem, la un moment dat, între pixelii E şi SE iar în regiunea 2 între pixelii S şi SE. Alegerea se face în funcţie de poziţia punctului M situat la mijlocul distanţei dintre pixelii între care se face alegerea. Delimitarea celor două regiuni este realizată de către punctul P cu proprietatea că vectorul gradient al elipsei în P (care este perpendicular pe tangenta la elipsă în P) formează un unghi de 45° cu axa Ox (sau o formulare echivalentă : punctul P are proprietatea că tangenta la elipsă în P formează cu axa Ox un unghi de 45°).

Definiţia 2.1

Fie curba dată prin ecuaţia implicită $F(x, y) = 0$. *Vectorul gradient* al curbei este : $gradF(x, y) = \frac{\partial F}{\partial x} \cdot \vec{\imath} + \frac{\partial F}{\partial y} \cdot \vec{\jmath}$. ∎

În cazul elipsei vectorul gradient este : $gradF(x, y) = 2 \cdot b^2 \cdot x \cdot \vec{\imath} + 2 \cdot a^2 \cdot y \cdot \vec{\jmath}$. În regiunea 1 panta vectorului gradient este > 1 iar în regiunea 2 este < 1 sau, echivalent, în regiunea 1 componenta corespunzătoare versorului $\vec{\jmath}$ este mai mare decât componenta corespunzătoare versorului $\vec{\imath}$ şi în regiunea 2 viceversa. Deci un punct P care aparţine elipsei se află în regiunea 1 ddacă $a^2 \cdot y > b^2 \cdot x$ şi se află în regiunea 2 ddacă $a^2 \cdot y < b^2 \cdot x$.

În fiecare regiune evaluăm $F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2$ în fiecare punct de mijloc $M(x, y)$ şi determinăm poziţia acestuia faţă de elipsă :

- dacă $F(x, y) = 0$ atunci M aparţine elipsei,
- dacă $F(x, y) < 0$ atunci M se găseşte (ca şi originea de altfel : $F(0, 0) = -a^2 \cdot b^2 < 0$) în interiorul elipsei,
- dacă $F(x, y) > 0$ atunci M se găseşte în exteriorul elipsei.

În regiunea 1 dacă punctul de mijloc M se găseşte în interiorul elipsei atunci este selectat pixelul E (deoarece este cel mai apropiat de elipsă) iar dacă M se găseşte în exteriorul elipsei atunci este selectat pixelul SE.

În regiunea 2 dacă punctul de mijloc M se găseşte în interiorul elipsei atunci este selectat pixelul SE altfel este selectat pixelul S.

Trecerea într-o altă regiune (din 1 în 2) se face când punctul de mijloc $M(x, y)$ nu mai aparţine regiunii, deci când $a^2 \cdot y \leq b^2 \cdot x$.

Vom folosi următoarele notaţii : pixelul curent (cel deja selectat) este $P(x_P, y_P)$, punctul de mijloc în regiunea 1 este $M(x_P + 1, y_P - \frac{1}{2})$ iar în regiunea 2 este $M(x_P + \frac{1}{2}, y_P - 1)$.

Variabila de decizie în regiunea 1 este $d_1 = F(x_P + 1, y_P - \frac{1}{2})$ iar în regiunea 2 este $d_2 = F(x_P + \frac{1}{2}, y_P - 1)$.

Valoarea iniţială a variabilei de decizie în regiunea 1 se obţine pentru $P(0, b)$ şi $M(0 + 1, b - \frac{1}{2})$: $d_1^{\text{init}} = F(1, b - \frac{1}{2}) = b^2 - a^2 \cdot b + \frac{a^2}{4}$. În regiunea 2 valoarea iniţială a variabilei de decizie este $d_2^{\text{init}} = F(x_P + \frac{1}{2}, y_P - 1)$ (pentru primul punct de mijloc care aparţine regiunii 2).

Valorile variabilelor de decizie se calculează incremental astfel :

- în regiunea 1 dacă $d_1 < 0$ alegem E, altfel SE. Dacă am ales pixelul E atunci $d_1{}' = F(x_P + 2, y_P - \frac{1}{2}) = d_1 + b^2 \cdot (2 \cdot x_P + 3)$. Dacă am ales pixelul SE atunci $d_1{}' = F(x_P + 2, y_P - \frac{3}{2}) = d_1 + b^2 \cdot (2 \cdot x_P + 3) + a^2 \cdot (-2 \cdot y_P + 2)$.
- în regiunea 2 dacă $d_2 < 0$ alegem SE, altfel S. Dacă am ales pixelul SE atunci $d_2{}' = F(x_P + \frac{3}{2}, y_P - 2) = d_2 + b^2 \cdot (2 \cdot x_P + 2) + a^2 \cdot (-2 \cdot y_P + 3)$. Dacă am ales pixelul S atunci $d_2{}' = F(x_P + \frac{1}{2}, y_P - 2) = d_2 + a^2 \cdot (-2 \cdot y_P + 3)$.

Obţinem astfel algoritmul de afişare utilizând tehnica punctului de mijloc (algoritm propus de Pitteway în 1967, van Aken în 1984 şi Kappel în 1985) :

Algoritmul 8 AfişareElipsă

`Date de intrare` : $a, b \in \mathbb{Z}$, semiaxele unei elipse centrate în origine.

`Date de ieşire` : $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, o secvenţă de pixeli care aproximează cel mai bine elipsa.

`Metodă` :

```
procedure AfisarePuncteElipsa (x,y : Z, var M : list of (Z x Z))
{
  M = M . (x,y) . (-x,-y) . (-x, y) . (x,-y);
}
procedure AfisareElipsa (a,b : Z, var M : list of (Z x Z))
{
  int x = 0, y = b;
  double d1 = b*b - a*a*b + a*a/4.0;
  double d2;
  M = \epsilon;
  AfisarePuncteElipsa(x, y, M);
  while (a*a*(y - 0.5) > b*b*(x+1))
  {
    // atat timp cat punctul de mijloc apartine regiunii 1
    if (d1 < 0)
    {
      // este selectat E
      d1 += b*b*(2*x+3);
      x++;
    }
    else
    {
      // este selectat SE
      d1 += b*b*(2*x+3) + a*a*(-2*y+2);
      x++;
      y--;
    }
    AfisarePuncteElipsa(x, y, M);
  }
  // trecem in regiunea 2 si initializam variabila de decizie
  d2 = b*b*(x+0.5)*(x+0.5) + a*a*(y-1)*(y-1) - a*a*b*b;
  while (y > 0)
  {
    if (d2 < 0)
    {
      // este selectat SE
      d2 += b*b*(2*x+2) + a*a*(-2*y+3);
      x++;
      y--;
    }
```

```
    else
    {
      // este selectat S
      d2 += a*a*(-2*y+3);
      y--;
    }
    AfisarePuncteElipsa(x, y, M);
  }
}
```

■

2.4 Colorarea uniformă a dreptunghiurilor

Problema care se pune este următoarea : să presupunem că pe grila cartezianǎ (a se vedea figura 2.5, pag. 34) avem suprapus un dreptunghi ale cărui vârfuri sunt date prin intersecţia dreptelor $x = x_{\min}$, $x = x_{\max}$, $y = y_{\min}$, $y = y_{\max}$, unde $x_{\min}, x_{\max}, y_{\min}, y_{\max} \in \mathbb{Z}$. Se cere să se determine şi să se coloreze uniform pixelii din interiorul dreptunghiului.

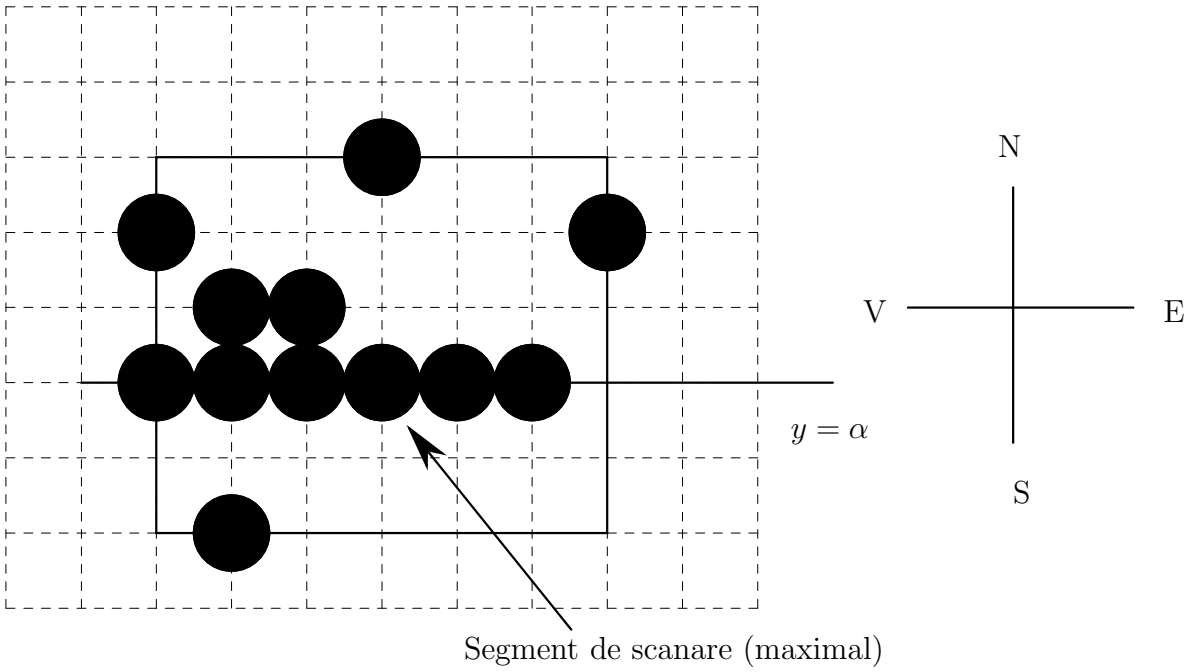

Figura 2.5: Pixelii interiori

Aşa cum se poate observa în figura 2.5, pag. 34 determinarea pixelilor strict interiori este evidentă (sunt acei pixeli $P(x_P, y_P)$ cu proprietatea că $x_{\min} < x_P < x_{\max}$ şi că $y_{\min} < y_P < y_{\max}$). În cazul pixelilor de pe frontiera dreptunghiului (cea de nord, sud, est sau vest) recurgem la următoarea convenţie : pixelii de pe frontiera de nord şi de est *nu* sunt interiori dreptunghiului iar cei de pe frontiera de vest şi de sud sunt[4].

[4]În general, regula este următoarea : un pixel situat pe o muchie nu este pixel interior dacă semiplanul definit de muchia respectivă şi care conţine obiectul se află în stânga sau sub muchie

Această convenţie se impune deoarece colorarea pixelilor trebuie să se realizeze prin apelarea *o singură dată* a primitivei **writePixel(int x, int y, int valoare)** care realizează colorarea unui pixel având coordonatele (x, y). Să presupunem că nu am fi respectat convenţia sus-menţionată. Atunci, în cazul a două dreptunghiuri care partajează o muchie, am fi în situaţia în care procedura **writePixel** ar fi apelată de două ori.

Având în vedere cele expuse mai sus, algoritmul care efectuează colorarea uniformă a pixelilor interiori unui dreptunghi este :

Algoritmul 9 Colorare uniformă dreptunghi

`Date de intrare :` $x_{\min}, x_{\max}, y_{\min}, y_{\max} \in \mathbb{Z}$ şi $val \in \mathbb{Z}$ parametrul care determină culoarea pixelilor.

`Date de ieşire :` Algoritmul coloreaza pixelii interiori dreptunghiului cu culoarea *val*.

`Metodă :`

```
procedure ConvScanDreptunghi (xm,xM,ym,yM,val : Z)
{
  for (int x = xm; x < xM; x++)
    for (int y = ym; y < yM; y++)
      writePixel(x,y,val);
}
```

■

2.5 Colorarea uniformă a poligoanelor

Algoritmul care va fi prezentat poate fi aplicat nu numai poligoanelor convexe ci şi celor concave sau celor care au găuri. Acest algoritm calculează pixelii interiori ai unui poligon utilizând intersecţia unor drepte de scanare cu poligonul (determinând astfel segmente de scanare maximale[5]).

Un exemplu de segment de scanare maximal este prezentat în figura 2.5, pag. 34 : dreapta de scanare $y = \alpha$, $\alpha \in \mathbb{Z}$, intersectează dreptunghiul iar pixelii interiori sunt cei reprezentaţi în figură.

Un alt exemplu este cel din figura 2.6, pag. 36 : în acest caz dreapta de scanare $y = 8$ intersectează poligonul $ABCDEF$ (unde $A(2,3)$, $B(7,1)$, $C(13,5)$, $D(13,11)$, $E(7,7)$ şi $F(2,9)$), mai precis muchiile sale AF, EF, DE, CD, în punctele $a(2,8)$, $b(\frac{9}{2},8)$, $c(\frac{17}{2},8)$ şi $d(13,8)$. Se poate observa că pixelii strict interiori poligonului de pe dreapta de scanare $y = 8$ sunt cei marcaţi în figură, cu excepţia pixelului $(13,8)$ care nu este considerat interior conform convenţiei precizate anterior. Pentru acelaşi poligon $ABCDEF$, pixelii interiori sunt prezentaţi în figura 2.7, pag. 37. Pixelii interiori sunt marcaţi utilizând simbolul ●, prin simbolul ■ am marcat câţiva pixeli care sunt strict exteriori poligonului $ABCDEF$ iar prin simbolul ⭘ am marcat câţiva pixeli care, deşi s-ar părea că sunt interiori poligonului, datorită convenţiei din secţiunea 2.4, pag. 34, nu sunt interiori poligonului $ABCDEF$.

Algoritmul are trei etape :

1. pentru toate dreptele de scanare care intersectează poligonul, se calculează intersecţia acestora cu muchiile poligonului,

[5] un segment de scanare maximal este acel segment de pe o dreaptă de scanare, complet inclus în interiorul obiectului şi care este maximal cu această proprietate, i.e., nu există un alt segment de scanare care să-l includă

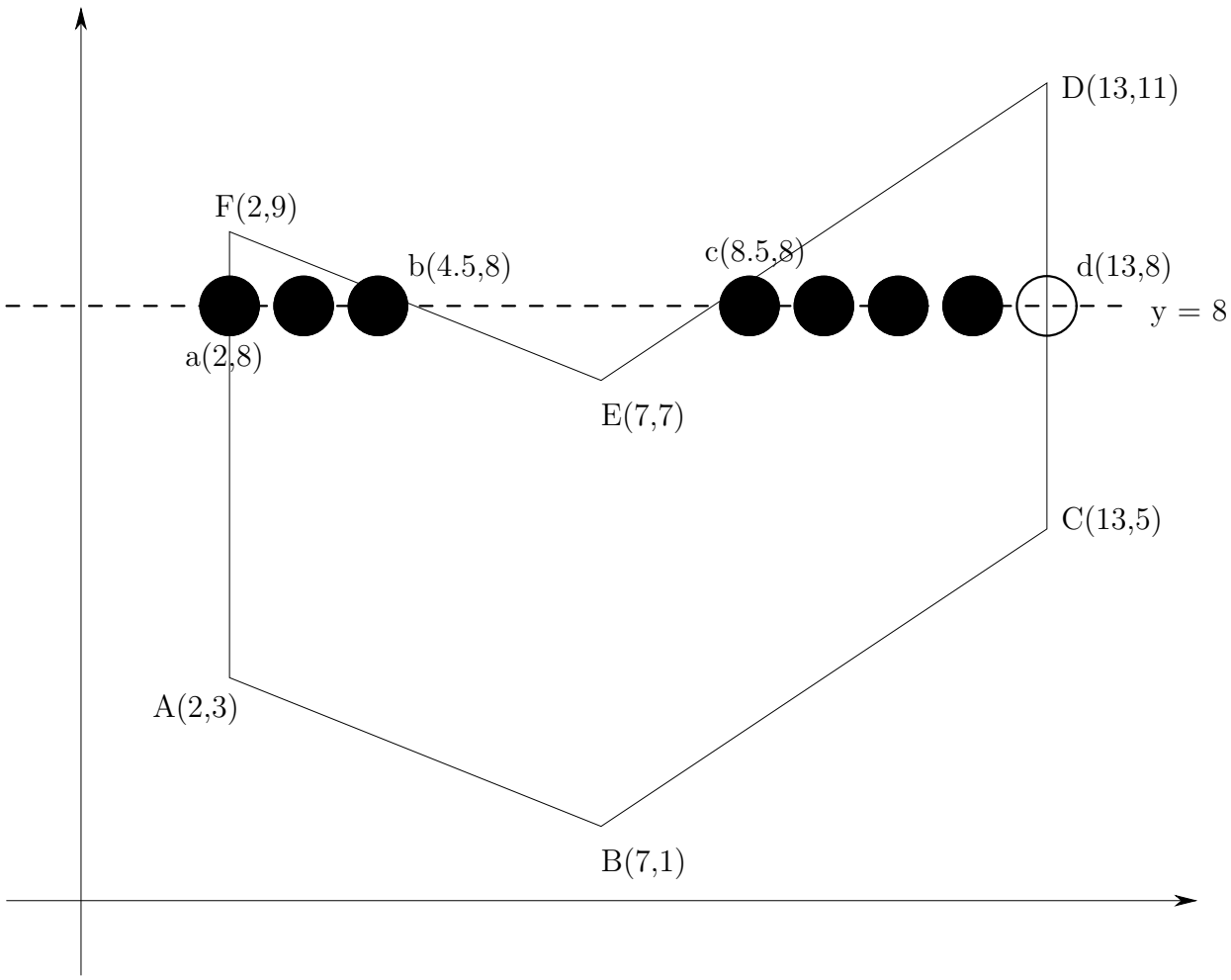

Figura 2.6: Colorarea uniformă a unui poligon. Un segment de scanare maximal.

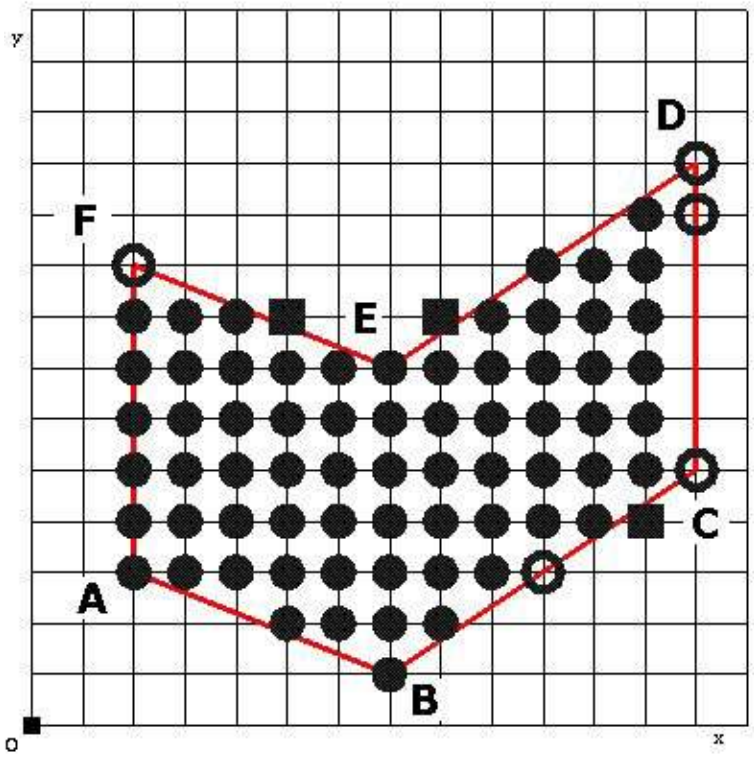

Figura 2.7: Pixeli interiori poligonului $ABCDEF$

2. sortarea punctelor de intersecţie crescător după coordonata x,

3. colorarea tuturor pixelilor dintre două intersecţii succesive, pixeli care sunt strict interiori poligonului. Având la dispoziţie lista punctelor de intersecţie dintre o dreaptă de scanare şi poligon (listă ordonată crescător după coordonata x), pixelii interiori sunt cei dintre punctele de intersecţie 1 şi 2, dintre 3 şi 4, dintre 5 şi 6, etc. Trebuie clarificate următoarele aspecte :

   (a) Cum procedăm în cazul în care coordonata x a unui punct de intersecţie nu are o valoare din $\mathbb{Z}$? În acest caz acest punct are doi pixeli vecini. Pe care îl selectăm ? În acest caz[6] procedăm astfel : în cazul punctelor de intersecţie 1, 3, 5, etc. pixelul interior va avea coordonata $\lceil x \rceil$ iar în cazul punctelor de intersecţie 2, 4, 6, etc. pixelul interior va avea coordonata $\lfloor x \rfloor$. De exemplu, în figura 2.6, pag. 36, cele 2 cazuri se referă la punctele de intersecţie $b(\frac{9}{2}, 8)$ şi $c(\frac{17}{2}, 8)$ (folosind notaţia noastră ele sunt punctele 2 şi 3). Conform regulii de mai sus, pixelul vecin al lui $b(\frac{9}{2}, 8)$ este $(4, 8)$ iar cel vecin al lui $c(\frac{17}{2}, 8)$ este $(9, 8)$ (aşa cum de altfel se poate observa în figura 2.7, pag. 37).

   (b) Cum procedăm cu pixelii care sunt extremităţi ale unui segment de scanare maximal ? Aşa cum am precizat în secţiunea 2.4, pag. 34 pixelul din extremitatea stângă este considerat interior poligonului iar cel din extremitatea dreaptă nu este considerat interior.

   (c) Cum procedăm cu pixelii care sunt în acelaşi timp şi vârfuri ale poligonului ? De ce, de exemplu, în figura 2.7, pag. 37 pixelii A, B şi E sunt interiori poligonului dar

[6]Vom presupune, fără a restrânge generalitatea, că ne aflăm în cadranul 1.

pixelii C, D, F nu ? Vom numi un vârf al unei muchii $y_{\min}$ dacă este vârful care are coordonata y minimă (dintre cele două vârfuri ale muchiei) iar $y_{\max}$ altfel. Regula este : dacă un pixel corespunde unui vârf care este vârf $y_{\min}$ al cel puţin uneia dintre cele două muchii care îl partajează atunci pixelul este considerat interior. În cazul poligonului $ABCDEF$ vârful A este $y_{\min}$ pentru muchia AF şi $y_{\max}$ pentru AB, vârful B este $y_{\min}$ pentru muchiile AB şi BC, vârful C este $y_{\min}$ pentru muchia CD şi $y_{\max}$ pentru muchia BC (dar, aplicând regula 3b, pag. 37 în prealabil, a fost etichetat extern), vârful D este $y_{\max}$ pentru muchiile CD şi DE, vârful E este $y_{\min}$ pentru muchiile DE şi EF, vârful F este $y_{\max}$ pentru muchiile AF şi EF. În concluzie pixelii A, B şi E sunt interiori poligonului iar pixelii C, D, F nu sunt interiori poligonului.

(d) Cum procedăm cu pixelii situaţi pe muchiile orizontale ale poligonului ? De ce, de exemplu, în cazul figurii 2.9, pag. 39 pixelii de pe muchiile AB, respectiv CD sunt interiori iar cei de pe muchia HG nu ? Sau, în cazul figurii 2.8, pag. 39, de ce pixelii de pe muchia BC nu sunt interiori triunghiului ? Vom proceda astfel : vom considera intersecţiile dreptelor de scanare doar cu muchiile care nu sunt orizontale. Vom elimina acele intersecţii cu vârfuri care sunt doar $y_{\max}$. De exemplu, în cazul poligonului $ABCDEFGHI$ (figura 2.9, pag. 39) :

i. dreapta de scanare $y = 10$ intersectează muchiile HI, FG, EF în $H(1, 10)$, $G(8, 10)$, $(10, 10)$. Remarcăm că vârful H este doar vârf de tip $y_{\max}$. Eliminăm acest vârf şi vom obţine doar punctele de intersecţie $G(8, 10)$ şi $(10, 10)$ şi vom proceda ca mai sus.

ii. dreapta de scanare $y = 6$ intersectează muchiile HI, BC, DE în $(1, 6)$, $C(8, 6)$, $D(10, 6)$. Remarcăm că vârful C este doar vârf de tip $y_{\max}$. Eliminăm acest vârf şi vom obţine doar punctele de intersecţie $(1, 6)$, şi $D(10, 6)$ şi vom proceda ca mai sus.

Regula pe care am aplicat-o în acest caz este conformă cu convenţia stabilită în secţiunea 2.4, pag. 34.

În cazul unor poligoane cu unghiuri foarte ascuţite (de exemplu triunghiul din figura 2.8, pag. 39) se recomandă şi afişarea pixelilor de pe muchii sau chiar din exteriorul poligonului dar cu intensităţi care variază în funcţie de distanţa dintre pixel şi cea mai apropiată muchie.

Algoritmul 10 Afişare poligon

`Date de intrare` : Un poligon specificat prin mulţimea muchiilor sale.

`Date de ieşire` : $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, mulţimea pixelilor interiori poligonului.

`Metodă` : Se aplică procedura *initializareET* pentru iniţializarea unei tabele de muchii, o structură de date ce va conţine, la finalul algoritmului, pentru fiecare dreaptă de scanare, punctele de intersecţie ale acestei drepte cu muchiile neorizontale ale poligonului. Procedura *calculssm* calculează incremental, plecând de la tabela de muchii, segmentele de scanare maximale, pentru fiecare dreaptă de scanare.

```
type
  VARF = struct S1 { int x, y; };
  MUCHIE = struct S2 { VARF vi, vf;};
  POLIGON = set of MUCHIE;
  INTERSECTIE = struct S3 { int ymax; double xmin; double ratia} ;
  INTERSECTII = list of INTERSECTIE;
```

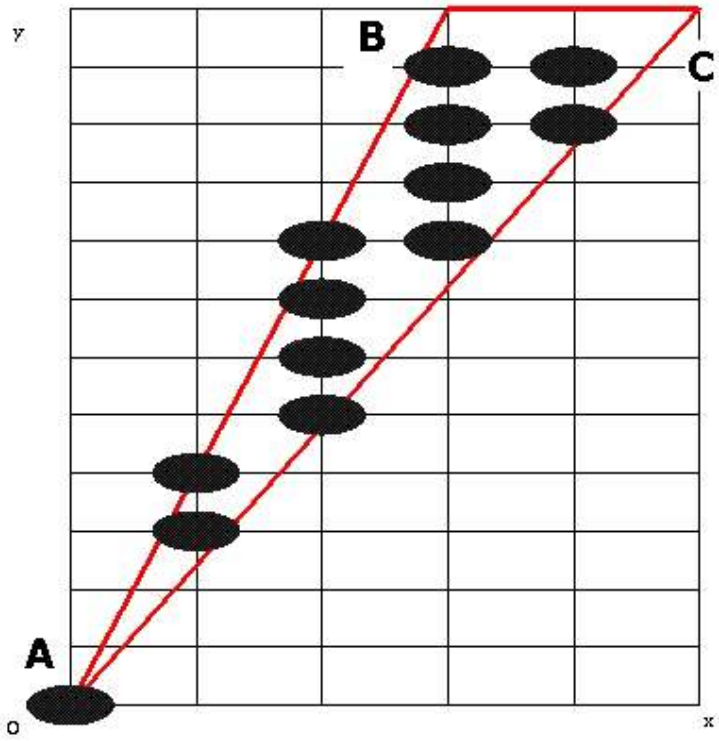

Figura 2.8: Pixelii interiori triunghiului ABC

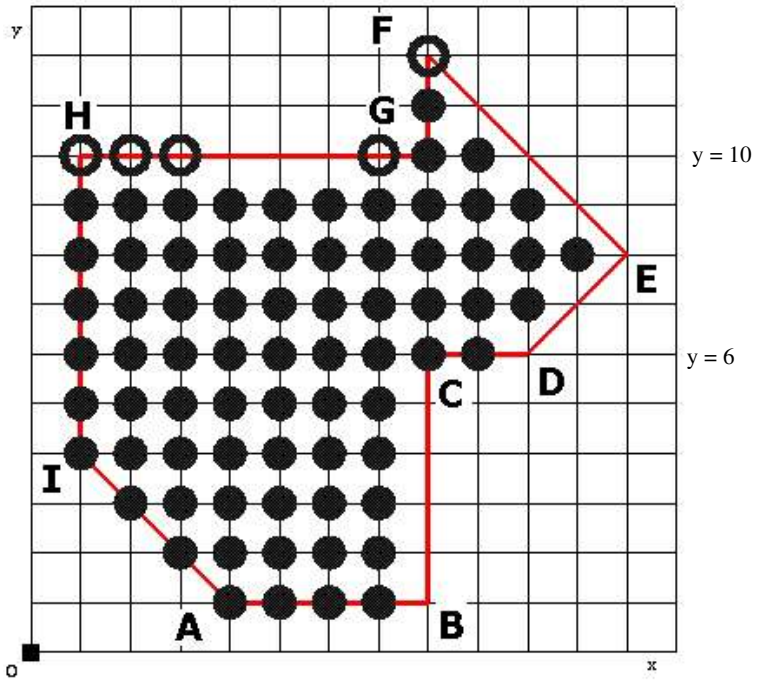

Figura 2.9: Pixelii interiori poligonului $ABCDEFGHI$

```
  DOM_SCAN = 0 .. 100;
  ET = DOM_SCAN -> INTERSECTII;

procedure initializareET(p : POLIGON, var et : ET)
{
  int xm,ym,xM,yM;
  bool change;

  for (i in DOM_SCAN) et(i) = [ ];
  for (m in p)
  {
    // pentru fiecare muchie din poligon ...
    if (m.vi.y != m.vf.y)
    {
      // ... care nu este orizontala
      ym = min (m.vi.y, m.vf.y);
      yM = max (m.vi.y, m.vf.y);
      xm = (ym == m.vi.y) ? m.vi.x : m.vf.x;
      xM = (yM == m.vi.y) ? m.vi.x : m.vf.x;
      et(ym) = et(ym) . [(yM, xm, (xm - xM)/(ym - yM))];
    }
  }
  /*
     sortarea in ordine crescatoare conform cu
     xm a intersectiilor dreptei de scanare cu
     muchiile poligonului
  */
  for (i in DOM_SCAN)
  {
    do
    {
      change = false;
      if (|et(i)| == 0) break;
      for (j = 1; j < |et(i)|; j++)
      {
        if (et(i)(j).xmin > et(i)(j+1).xmin)
        {
          interschimba et(i)(j) <-> et(i)(j+1);
          change = true;
        }
      }
    } while (change)
  }
}

procedure calculssm(p : POLIGON, et : ET, var finalET : ET)
{
  INTERSECTII activeSSM;
```

```
int y, k;

for (i in DOM_SCAN) finalET(i) = [ ];

// Se determina, in ordine crescatoare, care este
// prima dreapta de scanare care intersecteaza
// poligonul
// lui y i se atribuie o valoare care nu este din DOM_SCAN
y = -1;
// se determina y = min {y' | et(y') != \emptyset}
for (i in DOM_SCAN)
{
  if (! et(i).empty())
  {
    y = i;
    break;
  }
}
if (! y in DOM_SCAN) return;

activeSSM = [ ];
do
{
  activeSSM = activeSSM . et(y);

  // eliminarea varfurilor cu ymax == y
  for (i = 1; i <= |activeSSM|; i++)
  {
    if (activeSSM(i).ymax == y)
      activeSSM.delete(activeSSM(i));
  }

  // sortarea activeSSM cf. cheii xmin
  k = |activeSSM|;
  while (k >= 2)
  {
    for (i = 1; i < k; i++)
    {
      if (activeSSM(i).xmin > activeSSM(i+1).xmin)
        interschimba activeSSM(i) <-> activeSSM(i+1);
    }
    k--;
  }

  finalET(y) = activeSSM;

  y++;
```

```
    // reactualizarea punctelor de intersectie pentru noua dreapta de scanare
    for (i = 1; i <= |activeSSM|; i++)
    {
      if (activeSSM(i).ratia != 0)
        activeSSM(i).xmin += activeSSM(i).ratia;
    }
  } while (!activeSSM.empty() || !et(y).empty())
}

procedure coloreaza(ssms : ET)
{
  // Se aplica regulile discutate anterior
}

procedure main()
{
  POLIGON p;
  ET et, ssms;

  read(p);
  initializareET(p, et);
  calculssm(p, et, ssms);
  coloreaza(ssms);
}
```

■

În procedura *coloreaza* din algoritmul 10, pag. 38 segmentele de scanare maximale sunt colorate utilizând regulile 3a, pag. 37 şi 3b, pag. 37.

2.6 Colorarea uniformă a cercurilor şi a elipselor

Atât în cazul cercurilor cât şi în cazul elipselor vom ţine cont de simetria acestor obiecte geometrice. În cazul unui cerc este suficient să colorăm pixelii din cel de-al doilea octant şi apoi, utilizând simetria faţă de axele Ox, Oy, faţă de originea O cât şi faţă de prima bisectoare, putem colora pixelii interiori cercului. În cazul unei elipse este suficient să colorăm pixelii din primul cadran şi apoi, utilizând simetria faţă de axele Ox, Oy, şi faţă de originea O, putem colora pixelii interiori elipsei.

Fără a restrânge generalitatea vom presupune că cercurile sunt centrate în origine şi au raza R iar elipsele sunt centrate în origine şi au semiaxele a, b.

În ambele cazuri algoritmii propuşi calculează, în mod incremental, extremităţile segmentelor de scanare maximale din cerc, respectiv elipsă. Atât în cazul unui cerc cât şi în cazul unei elipse vom utiliza o structură de date care să ne permită să memorăm una din extremităţile unui segment de scanare maximal. Cealaltă extremitate se obţine, în cazul cercului, prin intersecţia dreptelor de scanare $y = 0$, $y = 1$, ..., $y = R$ cu dreapta $x = 0$ iar în cazul elipsei, prin intersecţia dreptelor de scanare $y = 0$, $y = 1$, ..., $y = b$ cu dreapta $x = 0$.

Algoritmii sunt similari (sunt incrementali şi utilizează doar aritmetica întregilor) celor de conversie prin scanare prezentaţi în secţiunile anterioare.

În cazul cercului, să presupunem că pixelul curent este pixelul $P(x_P, y_P)$ şi deci extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$ este P. În momentul în care se va trece la un nou pixel curent va trebui să luăm două decizii :

1. dacă să reactualizăm sau nu extremitatea segmentului de scanare maximal a cărui extremitate este $P(x_P, y_P)$ (sau, mai bine zis, dacă pixelul E poate deveni extremitate în locul lui P),

2. să reactualizăm extremitatea segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$.

Ecuaţia cercului centrat în origine, de rază R este $F(x, y) = 0$, unde $F(x, y) = x^2 + y^2 - R^2$. Deoarece punctul P este extremitatea unui segment de scanare maximal atunci el este în interiorul sau pe cerc, deci $F(P) \leq 0$, unde $F(P) \doteqdot F(x_P, y_P)$. Putem calcula valorile $F(E)$ şi $F(SE)$ plecând de la valoarea lui $F(P)$: $F(E) \doteqdot F(x_P+1, y_P) = (x_P+1)^2 + y_P^2 - R^2 = F(P) + 2 \cdot x_P + 1$ şi $F(SE) \doteqdot F(x_P+1, y_P-1) = (x_P+1)^2 + (y_P-1)^2 - R^2 = F(P) + 2 \cdot x_P - 2 \cdot y_P + 2$.

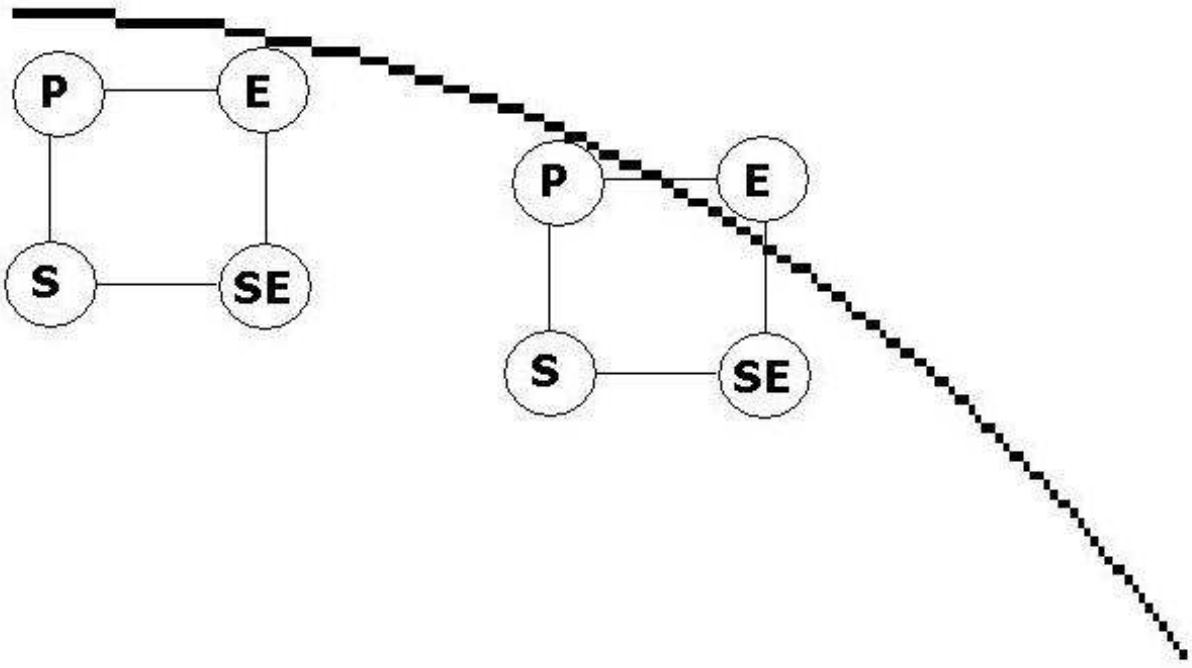

Figura 2.10: Cazuri posibile în al doilea octant al unui cerc

Ţinând cont de cele de mai sus :

1. Dacă $F(E) \leq 0$ (deci pixelul E este interior, a se vedea figura 2.10, pag. 43) atunci reactualizăm segmentul de scanare maximal corespunzător dreptei de scanare $y = y_P$ cu extremitatea E în loc de P. În acest caz pixelul E devine pixelul curent P.

2. Dacă $F(SE) \leq 0$ atunci actualizăm segmentul de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$ cu extremitatea SE (pixelul SE este interior, a se vedea figura 2.10, pag. 43). În acest caz pixelul SE devine pixelul curent P.

3. Altfel (i.e., dacă $F(E) > 0$ şi $F(SE) > 0$) înseamnă că atât E cât şi SE sunt în exteriorul cercului şi deci vom alege S în loc de P (deoarece este evident că dacă P este în interiorul cercului atunci pixelul S este tot în interiorul cercului : avem $F(P) \leq 0$, deci $x_P^2 + y_P^2 - R^2 \leq 0$, $F(S) \doteqdot F(x_P, y_P - 1) = x_P^2 + (y_P - 1)^2 - R^2 = F(P) - 2 \cdot y_P + 1$ şi cum $1 - 2 \cdot y_P < 0$ deoarece $y_P \geq 1 > \frac{1}{2}$ avem deci $F(S) \leq 0$).

În cazul elipsei problema este similară dar puţin mai complicată deoarece în acest caz raţionamentul trebuie făcut pentru tot cadranul întâi pentru cele două regiuni identificate la algoritmul de conversie prin scanare al unei elipse.

Ecuaţia elipsei de semiaxe a, b centrate în origine este $F(x, y) = 0$, unde $F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2$. Regiunea 1 este acea regiune din primul cadran caracterizată de inecuaţia $a^2 \cdot y > b^2 \cdot x$ iar regiunea 2 de inecuaţia $a^2 \cdot y < b^2 \cdot x$. Un punct $A(x, y)$ este interior elipsei ddacă $F(A) < 0$.

În prima regiune să presupunem că pixelul curent este pixelul $P(x_P, y_P)$ şi deci extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$ este P. Avem mai multe cazuri :

1. Dacă pixelul $E(x_P + 1, y_P)$ este interior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$. Ea devine E. În acest caz pixelul E devine următorul pixel curent.

2. Altfel, dacă pixelul $SE(x_P+1, y_P-1)$ este interior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine SE. În acest caz pixelul SE devine următorul pixel curent.

3. Altfel, dacă pixelul $SE(x_P + 1, y_P - 1)$ este exterior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine S. În acest caz pixelul S devine următorul pixel curent.

În a doua regiune să presupunem că pixelul curent este pixelul $P(x_P, y_P)$ şi deci extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$ este P. Avem mai multe cazuri :

1. Dacă pixelul $SE(x_P + 1, y_P - 1)$ este interior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine SE. În acest caz pixelul SE devine următorul pixel curent.

2. Altfel, dacă pixelul $SE(x_P + 1, y_P - 1)$ este exterior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine S (este evident că dacă P este în interiorul elipsei atunci S este şi el în interiorul elipsei deoarece presupunând că $F(P) \leq 0$ atunci cum $F(S) = F(P) + a^2 \cdot (-2 \cdot y_P + 1)$ şi cum $y_P \geq 1$ avem deci $F(S) \leq F(P) \leq 0$). În acest caz pixelul S devine următorul pixel curent.

Valorile $F(E)$, $F(SE)$, $F(S)$ sunt calculate în funcţie de $F(P)$ astfel : în regiunea 1 $F(E) = F(P) + b^2 \cdot (2 \cdot x_P + 1)$ iar $F(SE) = F(P) + b^2 \cdot (2 \cdot x_P + 1) + a^2 \cdot (-2 \cdot y_P + 1)$. Valori similare se pot deduce şi pentru a doua regiune.

Algoritmul care realizează umplerea unei elipse este :

Algoritmul 11 UmplereElipsa

`Date de intrare` : O elipsă centrată în origine de semiaxe întregi $a, b \in \mathbb{Z}$.

`Date de ieşire` : $M \in$ `list of` $(\mathbb{Z} \times \mathbb{Z})$, mulţimea pixelilor interiori elipsei.

`Metodă` : O procedură C/C++ care implementează algoritmul descris mai sus este următoarea :

```
void umplereelipsa(int x0, int y0, int a, int b, double val,
                   void (*sablon) (int x, int y, double val,
                                   CGrilaCarteziana &cc))
{
  int xi = 0, x = 0, y = b;
  double fxpyp = 0.0;
  double deltaE, deltaSE, deltaS;

  ssm.vidare();
  ssm.adauga(y+y0,xi+x0,x+x0);

  // regiunea 1
  while (a*a*(y-0.5) > b*b*(x+1))
  {
    deltaE = b * b * ( 2 * x + 1 );
    deltaSE = b * b * ( 2 * x + 1 ) + a * a * (-2 * y + 1);
    if (fxpyp + deltaE <= 0.0)
    {
      // E este in interior
      fxpyp += deltaE;
      x++;
      ssm.setare(y+y0,xi+x0, x+x0);
    }
    else if (fxpyp + deltaSE <= 0.0)
    {
      // SE este in interior
      fxpyp += deltaSE;
      x++;y--;
      ssm.adauga(y+y0,xi+x0, x+x0);
    }
  }

  // regiunea 2
  while (y > 0)
  {
    deltaSE = b * b * ( 2 * x + 1 ) + a * a * (-2 * y + 1);
    deltaS = a*a*(-2 * y + 1);
    if (fxpyp + deltaSE <= 0.0)
    {
      // SE este in interior
      fxpyp += deltaSE;
      x++;y--;
```

```
    }
    else
    {
      // S este in interior
      fxpyp += deltaS;
      y--;
    }
    ssm.adauga(y+y0,xi+x0, x+x0);
  }
  ssm.desenare(val, *this, sablon);
}
```

■

După ce am determinat pixelii interiori unui cerc sau unei elipse în cadranul întâi, prin simetrie, putem determina pixelii interiori din celelalte cadrane.