Primitive grafice

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Spațiul culorilor

Vârfuri

Primitive grafice - generalități

Algoritmi de rasterizare

Preliminarii, notații

Algoritmii

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:
  - în OpenGL "vechi" folosind funcția

    ```
    glColor* (  );
    ```
    – "deprecated"

    Sufixul * poate indica:

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:
  - în OpenGL "vechi" folosind funcția

    ```
    glColor* (  );
    ```
    – "deprecated"

    Sufixul * poate indica:
    - dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul "alpha", legat de opacitate/transparență.

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:
  - în OpenGL "vechi" folosind funcția

    `glColor* (  );` – "deprecated"

    Sufixul * poate indica:
    - dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul "alpha", legat de opacitate/transparență.
    - tipul de date utilizat, care poate fi:
      i (integer) ($\mathtt{i} \in \{0, 1, \ldots, 255\}$)
      f (float)
      d (double) ($\mathtt{f}, \mathtt{d} \in [0.0, 1.0]$)

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:
  - în OpenGL "vechi" folosind funcția

    ```
    glColor* (  );
    ```
    – "deprecated"

    Sufixul * poate indica:
    - dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul "alpha", legat de opacitate/transparență.
    - tipul de date utilizat, care poate fi:
      `i` (integer) ($\mathtt{i} \in \{0, 1, \ldots, 255\}$)
      `f` (float)
      `d` (double) ($\mathtt{f}, \mathtt{d} \in [0.0, 1.0]$)
    - (opțional) posibila formă vectorială, indicată prin sufixul `v`.

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:
  - în OpenGL "vechi" folosind funcția

    ```
    glColor* (  );
    ```
    – "deprecated"

    Sufixul * poate indica:
    - dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul "alpha", legat de opacitate/transparență.
    - tipul de date utilizat, care poate fi:
      `i` (integer) ($\mathtt{i} \in \{0, 1, \ldots, 255\}$)
      `f` (float)
      `d` (double) ($\mathtt{f}, \mathtt{d} \in [0.0, 1.0]$)
    - (opțional) posibila formă vectorială, indicată prin sufixul `v`.
  - în OpenGL "nou": culorile sunt manevrate într-un mod asemănător vârfurilor (folosind VBO), odată cu acestea (atribute ale vârfurilor).

Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – Cubul RGB.
- În OpenGL o culoare este indicată prin:
  - în OpenGL "vechi" folosind funcția

    ```
    glColor* (  );
    ```
    – "deprecated"

    Sufixul * poate indica:
    - dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul "alpha", legat de opacitate/transparență.
    - tipul de date utilizat, care poate fi:
      `i` (integer) ($\mathtt{i} \in \{0, 1, \ldots, 255\}$)
      `f` (float)
      `d` (double) ($\mathtt{f}, \mathtt{d} \in [0.0, 1.0]$)
    - (opțional) posibila formă vectorială, indicată prin sufixul `v`.
  - în OpenGL "nou": culorile sunt manevrate într-un mod asemănător vârfurilor (folosind VBO), odată cu acestea (atribute ale vârfurilor).
  - elementul comun este faptul că, în ambele cazuri, culorile conțin informații legate de canalele R, G, B, precum și de canalul A (factorul α, legat de opacitate).

Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

- în OpenGL "vechi" cu ajutorul funcției

```
glVertex* (  );
```

– "deprecated"

Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

- în OpenGL “vechi” cu ajutorul funcției

```
glVertex* (  );
```

— “deprecated”

- în OpenGL “nou”:
  vârfurile sunt stocate în matrice;
  împachetate în VAO (Vertex Array Objects);
  trimise plăcii grafice sub formă de VBO (Vertex Buffer Objects).

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:
  - coordonate (fac parte din definiție),

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:
  - coordonate (fac parte din definiție),
  - o culoare (v. secțiunea Culori...),

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:
    - coordonate (fac parte din definiție),
    - o culoare (v. secțiunea Culori...),
    - o normală (legată de funcții de iluminare),

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:
  - coordonate (fac parte din definiție),
  - o culoare (v. secțiunea Culori...),
  - o normală (legată de funcții de iluminare),
  - coordonate de texturare

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:
  - coordonate (fac parte din definiție),
  - o culoare (v. secțiunea Culori...),
  - o normală (legată de funcții de iluminare),
  - coordonate de texturare
- În OpenGL "vechi": pentru o anumită caracteristică, este considerată valoarea *curentă* a respectivei caracteristici. Altfel spus, ea trebuie indicată în codul sursă *înaintea* *vârfului*.

Funcții pentru primitive

Vârfurile sunt utilizate pentru trasarea *primitivelor grafice*. Funcțiile folosite sunt diferite pentru cele două moduri de randare:

- în OpenGL "vechi", o funcție de tipul `glVertex ( )` poate fi apelată într-un cadru de tip

```
glBegin (*);

glEnd;
```

– "deprecated"

(unde * reprezintă tipul de primitivă generat);

Funcții pentru primitive

Vârfurile sunt utilizate pentru trasarea *primitivelor grafice*. Funcțiile folosite sunt diferite pentru cele două moduri de randare:

- în OpenGL "vechi", o funcție de tipul `glVertex ( )` poate fi apelată într-un cadru de tip

```
glBegin (*);

glEnd;
```

— "deprecated"

  (unde * reprezintă tipul de primitivă generat);
- în OpenGL "nou" este utilizată o funcție de tipul

```
glDrawArrays (GLenum mode, GLint first, GLint count);
```

  unde

  `mode`: tipul primitivei;

  `first`: primul vârf;

  `count`: câte vârfuri se iau în considerare.

Tipuri de primitive

- Puncte: `GL_POINTS`

Tipuri de primitive

- Puncte: `GL_POINTS`
- Segmente de dreaptă: `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`

Tipuri de primitive

- Puncte: GL_POINTS
- Segmente de dreaptă: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
- Triunghiuri: GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN

Tipuri de primitive

- Puncte: `GL_POINTS`
- Segmente de dreaptă: `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`
- Triunghiuri: `GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`
- Dreptunghiuri: `GL_QUADS`, `GL_QUAD_STRIP`

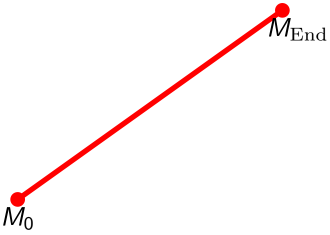
Tipuri de primitive

- Puncte: `GL_POINTS`
- Segmente de dreaptă: `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`
- Triunghiuri: `GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`
- Dreptunghiuri: `GL_QUADS`, `GL_QUAD_STRIP`
- Poligoane (convexe!): `GL_POLYGON`

Motivație și problematizare

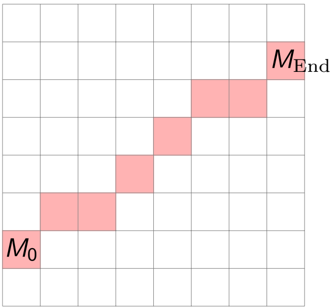
“Continuu”

“Grafica vectorială”



“Discret”

“Grafica rasterială”



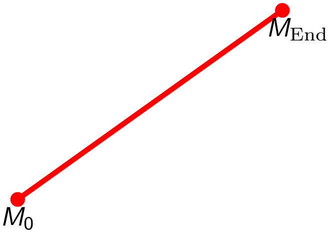
$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}})$$
$$x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{R}$$

$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}})$$
$$x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{N}(\mathbb{Z})$$

Motivație și problematizare

“Continuu”

“Grafica vectorială”

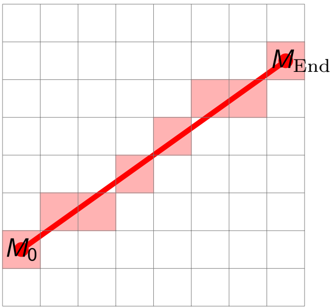


$$M_0 = (x_0, y_0), M_{\mathrm{End}} = (x_{\mathrm{End}}, y_{\mathrm{End}})$$

$$x_0, y_0, x_{\mathrm{End}}, y_{\mathrm{End}} \in \mathbb{R}$$

“Discret”

“Grafica rasterială”



$$M_0 = (x_0, y_0), M_{\mathrm{End}} = (x_{\mathrm{End}}, y_{\mathrm{End}})$$

$$x_0, y_0, x_{\mathrm{End}}, y_{\mathrm{End}} \in \mathbb{N}(\mathbb{Z})$$

Ce este un algoritm de rasterizare?

Un algoritm de rasterizare are ca efect reprezentarea grafică a unei primitive într-un sistem de reprezentare de tip grilă (monitor), care este format dintr-o structură discretă de pixeli. Pentru un segment, un algoritm de rasterizare are ca:

Input: Coordonatele $x_0, y_0, x_{\mathrm{End}}, y_{\mathrm{End}} \in \mathbb{N}$ $(\in \mathbb{Z})$ ale extremităților segmentului care urmează să fie reprezentat - altfel spus, pixelii inițial $M_0 = (x_0, y_0)$, respectiv final $M_{\mathrm{End}} = (x_{\mathrm{End}}, y_{\mathrm{End}})$.

Ce este un algoritm de rasterizare?

Un algoritm de rasterizare are ca efect reprezentarea grafică a unei primitive într-un sistem de reprezentare de tip grilă (monitor), care este format dintr-o structură discretă de pixeli. Pentru un segment, un algoritm de rasterizare are ca:

Input: Coordonatele $x_0, y_0, x_{\mathrm{End}}, y_{\mathrm{End}} \in \mathbb{N}$ ($\in \mathbb{Z}$) ale extremităților segmentului care urmează să fie reprezentat - altfel spus, pixelii inițial $M_0 = (x_0, y_0)$, respectiv final $M_{\mathrm{End}} = (x_{\mathrm{End}}, y_{\mathrm{End}})$.

Output: Pixelii selectați pentru a trasa segmentul de la M_0 la M_{End}

Ipoteze / restricții

Raționamentele se pot adapta la restul situațiilor/cazurilor. Ipoteze (simplificatoare) făcute:

Ip. 1 Intuitiv: “deplasarea se face înspre dreapta/sus”

$$x_{\mathrm{End}} > x_0, \quad y_{\mathrm{End}} > y_0 \Leftrightarrow$$
$$\Leftrightarrow \Delta x > 0, \quad \Delta y > 0.$$

Ipoteze / restricții

Raționamentele se pot adapta la restul situațiilor/cazurilor. Ipoteze (simplificatoare) făcute:

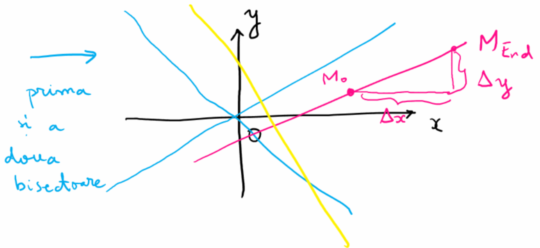
Ip. 1 Intuitiv: “deplasarea se face înspre dreapta/sus”

$$x_{\mathrm{End}} > x_0, \quad y_{\mathrm{End}} > y_0 \Leftrightarrow$$

$$\Leftrightarrow \Delta x > 0, \quad \Delta y > 0.$$

Ip. 2 Intuitiv: “dreapta trasată prin origine și paralelă cu dreapta suport este situată sub prima bisectoare”

$$\Leftrightarrow \Delta x > \Delta y > 0.$$



Ecuația dreptei

Ecuația dreptei care unește punctele M_0 și M_{End}

$$y = mx + n. \tag{1}$$

Ecuația dreptei

Ecuația dreptei care unește punctele M_0 și M_{End}

$$y = mx + n. \tag{1}$$

Elemente importante: panta m și coeficientul n, care poate fi exprimat în funcție de pantă și de coordonatele lui M_0

$$\boxed{m = \frac{\Delta y}{\Delta x} \text{ (panta)}}$$

$$y_0 = m x_0 + n \Rightarrow n = y_0 - m x_0 \Rightarrow \boxed{n = y_0 - \frac{\Delta y}{\Delta x} \cdot x_0}$$

Ecuația dreptei

Ecuația dreptei care unește punctele M_0 și M_{End}

$$y = mx + n. \tag{1}$$

Elemente importante: panta m și coeficientul n, care poate fi exprimat în funcție de pantă și de coordonatele lui M_0

$$\boxed{m = \frac{\Delta y}{\Delta x}\ (\text{panta})}$$

$$y_0 = m x_0 + n \Rightarrow n = y_0 - m x_0 \Rightarrow \boxed{n = y_0 - \frac{\Delta y}{\Delta x} \cdot x_0}$$

Concluzie: În cazul continuu avem

$$y = mx + n \overset{NOT}{=} f(x).$$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- `Iniţializare` $x_0, y_0 = f(x_0), m, n$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Iniţializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \to k+1 \ (k \geq 0)$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Iniţializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \to k+1$ $(k \geq 0)$

  $x_{k+1} \leftarrow x_k + 1$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Iniţializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \to k+1$ $(k \geq 0)$

  $x_{k+1} \leftarrow x_k + 1$

  $f(x_{k+1}) \leftarrow m \cdot x_{k+1} + n$ // formula (1)

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Iniţializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \to k+1$ $(k \geq 0)$

$x_{k+1} \leftarrow x_k + 1$
$f(x_{k+1}) \leftarrow m \cdot x_{k+1} + n$ // formula (1)
$y_{k+1} \leftarrow \texttt{round}(f(x_{k+1}))$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Iniţializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \to k+1$ $(k \geq 0)$

$x_{k+1} \leftarrow x_k + 1$
$f(x_{k+1}) \leftarrow m \cdot x_{k+1} + n$ // formula (1)
$y_{k+1} \leftarrow \mathtt{round}(f(x_{k+1}))$

Calcule...

ptr. $f(x_{k+1})$

$$\underline{f(x_{k+1})} = m \cdot \underbrace{x_{k+1}}_{x_k + 1} + n = m(x_k + 1) + n =$$

$$= m \cdot x_k + m + n =$$

$$= \underbrace{m \cdot x_k + n}_{f(x_k)} + m = \underline{f(x_k) + m}$$

Varianta 2 - algoritmul Digital Differential Analyzer (DDA)

Algoritm

- Iniţializare $x_0, y_0 = f(x_0), m$
- Pasul $k \to k+1$ $(k \geq 0)$

  $x_{k+1} \leftarrow x_k + 1$

  $\underline{f(x_{k+1}) \leftarrow f(x_k) + m}$ // formula (1), calculele anterioare

  $y_{k+1} \leftarrow \mathtt{round}(f(x_{k+1}))$

Exemplu

$$f(x_{k+1}) = f(x_k) + m$$

$$M_0 = (10, 20), M_{\text{End}} = (20, 28)$$

$$\Delta x = 10 \; ; \quad \Delta y = 8 \; ; \quad m = \frac{\Delta y}{\Delta x} = \frac{8}{10} = 0.8$$

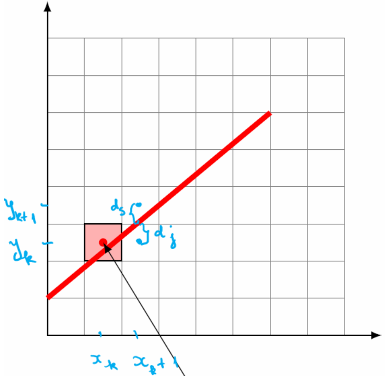
k	0	1	2	3	4	5	6	7	8	9	10
x_k	10	11	12	13	14	15	16	17	18	19	20
$f(x_k)$	20	20.8	21.6	22.4	23.2	24	24.8	25.6	26.4	27.2	28
y_k	20	21	22	22	23	24	25	26	26	27	28

$\Downarrow$ $\Downarrow$ $\Downarrow$

(11,21) (12,22) (13,22)

Varianta 3 - algoritmul lui Bresenham. Idei fundamentale

Conceptul de "pixeli candidați"



Presupunem că pixelul (x_k, y_k) a fost selectat. Care este pixelul următor?

Doi "candidați" pentru pixelul următor: (⚠ ipoteza privind panta dreptei)

$j : (x_k + 1, y_k)$

$s : (x_k + 1, y_k + 1)$

Ideea: se calculează distanța de la punctul de pe dreaptă corespunzător abscisei $x_k + 1$ la centrele pixelilor candidați:

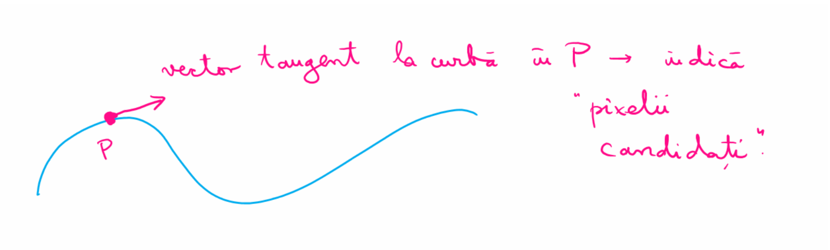
$$d_s = y_k + 1 - f(x_k + 1)$$

$$d_j = f(x_k + 1) - y_k$$

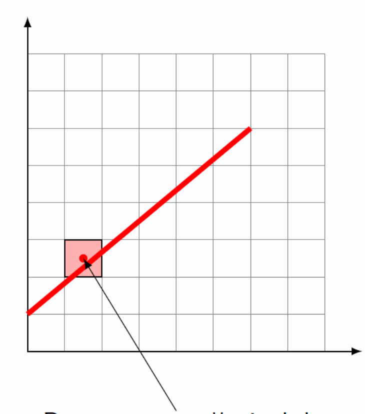
(sunt diferențe de ordonate!)

Algoritmul lui Bresenham. Observație

Varianta originală (1965) avea în vedere în special segmentele de dreaptă. Folosind conceptul de vector tangent, algoritmul poate fi adaptat și aplicat și în cazul altor curbe (v. art 1977)!



Algoritmul lui Bresenham. Factorul de decizie



Presupunem că pixelul (x_k, y_k) a fost selectat. Care este pixelul următor?

$$d_s = \underbrace{y_k + 1}_{\text{ordonata centrului pixelului de sus}} - \underbrace{f(x_k + 1)}_{\text{ordonata pct. de pe dreaptă}}$$

$$d_j = f(x_k + 1) - y_k$$

$$d_s = y_k + 1 - m \cdot x_k - m - n$$

$$d_j = m x_k + m + n - y_k$$

$$d_s \;\begin{matrix} > \\ = \\ < \end{matrix}\; d_j \;?$$

Algoritmul lui Bresenham. Semnul factorului de decizie

Ne interesează semnul numărului $d_j - d_s \in \mathbb{Q}$

$$d_j - d_s = 2\underset{\frac{\Delta y}{\Delta x}}{\underset{\shortparallel}{m}}(x_k + 1) - 2y_k + 2m - 1 =$$

$$= \frac{2\Delta y(x_k+1) - 2y_k \cdot \Delta x + 2m\Delta x - \Delta x}{\Delta x} \qquad \Delta x > 0$$

$$\boxed{\gamma = 2\Delta y + 2\Delta x m - \Delta x}$$

Deci: semnul lui $d_j - d_s$ este semnul lui p_k

$$\boxed{p_k \overset{DEF}{=} 2\Delta y x_k - 2\Delta x y_k + \gamma}$$

$p_k < 0 \Leftrightarrow d_j - d_s < 0 \Leftrightarrow d_j < d_s \Rightarrow$ se alege (x_k+1, y_k)

$p_k \geq 0 \Leftrightarrow d_j - d_s \geq 0 \Leftrightarrow d_j \geq d_s \Rightarrow$ — " — (x_k+1, y_k+1)

Algoritmul lui Bresenham. Parametrul de decizie

Semnul lui $d_j - d_s$ este semnul **parametrului de decizie**

$$p_k \overset{DEF}{=} 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + \gamma$$

Dacă $p_k < 0 \Rightarrow d_j < d_s \Rightarrow$ se alege pixelul $(x_k + 1, y_k)$

Dacă $p_k \geq 0 \Rightarrow d_j \geq d_s \Rightarrow$ se alege pixelul $(x_k + 1, y_k + 1)$

Algoritmul lui Bresenham. Parametrul de decizie - rescriere

Evaluăm $p_{k+1} - p_k =$

$$= 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + \cancel{b} - $$
$$- 2\Delta y \cdot x_k + 2\Delta x \cdot y_k - \cancel{b}$$

$\| \; x_{k+1} = x_k + 1$

$2\Delta y$

$$y_{k+1} - y_k = \begin{cases} 0, & p_k < 0 \\ 1, & p_k \geq 0 \end{cases}$$

$$= \begin{cases} 2\Delta y, & \text{dacă } p_k < 0 \text{ (adică } y_{k+1} = y_k) \\ 2\Delta y - 2\Delta x, & \text{dacă } p_k \geq 0 \text{ (adică } y_{k+1} = y_k + 1) \end{cases}$$

Algoritmul lui Bresenham. Parametrul de decizie - rescriere

Evaluăm

$$p_0 = 2\Delta y \cdot x_0 - 2\Delta x \cdot y_0 + \gamma =$$

$$= 2\Delta y \cdot x_0 - 2\Delta x \cdot (m \cdot x_0 + n) + 2\Delta y + 2\Delta x \cdot n - \Delta x \Rightarrow$$

$$p_0 = 2\Delta y - \Delta x.$$

Algoritmul lui Bresenham

Algoritm

`- Calcul preliminar` $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$

Algoritmul lui Bresenham

Algoritm

- `Calcul preliminar` $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- `Pasul` $k \to k+1$ $(k \geq 0)$. `// Pp. că avem` $x_k, y_k, p_k \in \mathbb{Z}$

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \to k+1$ $(k \geq 0)$. // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$

  Dacă $p_k < 0$ (‘‘stagnare pe orizontală’’)

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \to k+1$ $(k \geq 0)$. // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$

  Dacă $p_k < 0$ (''stagnare pe orizontală'')

  $x_{k+1} \leftarrow x_k + 1$

  $y_{k+1} \leftarrow y_k$

  $p_{k+1} \leftarrow p_k + 2\Delta y$

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \to k+1$ $(k \geq 0)$. // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$

  Dacă $p_k < 0$ (``stagnare pe orizontală'')

  $x_{k+1} \leftarrow x_k + 1$
  $y_{k+1} \leftarrow y_k$
  $p_{k+1} \leftarrow p_k + 2\Delta y$

  Dacă $p_k \geq 0$ (``în sus'')

Algoritmul lui Bresenham

Algoritm

- `Calcul preliminar` $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- `Pasul` $k \to k+1$ $(k \geq 0)$. `// Pp. că avem` $x_k, y_k, p_k \in \mathbb{Z}$

  `Dacă` $p_k < 0$ `(‘‘stagnare pe orizontală’’)`

  $x_{k+1} \leftarrow x_k + 1$
  $y_{k+1} \leftarrow y_k$
  $p_{k+1} \leftarrow p_k + 2\Delta y$

  `Dacă` $p_k \geq 0$ `(‘‘în sus’’)`

  $x_{k+1} \leftarrow x_k + 1$
  $y_{k+1} \leftarrow y_k + 1$
  $p_{k+1} \leftarrow p_k + 2\Delta y - 2\Delta x$

Algoritmul lui Bresenham. Exemplu

$M_0 = (10, 20), M_{\mathrm{End}} = (20, 28)$

$$x_0 = 10,\ y_0 = 20,\ x_{\mathrm{End}} = 20,\ y_{\mathrm{End}} = 28.$$

$$\Delta x = 10,\ \Delta y = 8$$

$$p_0 = 2\Delta y - \Delta x = 6$$

$$2\Delta y = 16$$

$$2\Delta y - 2\Delta x = -4$$

k	0	1	2	3	4	5	6	7	8	9	10
x_k	10	11	12	13	14	15	16	17	18	19	20
y_k	20	21	22	22	23	24	25	26	26	27	28
p_k	6	2	−2	14	10	6	2	−2	14	10	6

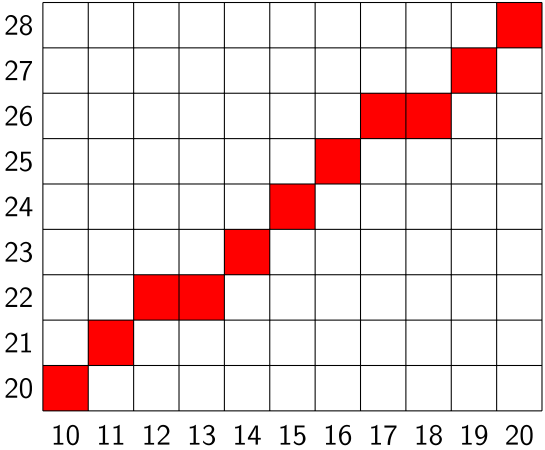


Figura: **Algoritmul DDA / algoritmul lui Bresenham.** Pixelii selectați pentru a uni punctele $(10, 20)$ și $(20, 28)$ sunt colorați cu roșu.