# Laborator 6

## 1 Obiective

Acest laborator evidențiază algoritmii Bresenham folosiți pentru rasterizarea unor primitive grafice pe un ecran de computer. Această lucrare începe cu prezentarea unor informații generice despre algoritmi și apoi îi exemplifică pentru rasterizarea liniilor și cercurilor.

## 2 Fundament teoretic

Algoritmul Bresenham pentru trasarea liniilor într-un spațiu bi-dimensional (precum afișajul unui computer) este o metodă fundamentală utilizată în disciplina graficii computerizate. Eficiența algoritmului îl face una dintre cele mai utilizate metode pentru trasarea liniilor continue, a cercurilor sau a altor primitive grafice. Acest proces se numește rasterizare.

Fiecare linie, cerc sau alte primitive grafice vor fi reprezentate folosind pixeli. Fiecare pixel este descris de o poziție *(x, y)* fixă în spațiul bi-dimensional XOY. Algoritmul aproximează linia reală prin calculul poziției fiecărui pixel din linie. Deoarece pixelii sunt cele mai mici elemente adresabile într-un dispozitiv de afișare, aproximarea algoritmului este suficient de bună pentru a "convinge" ochiul uman și pentru a obține iluzia unei linii reale. Figura 1a și Figura 1b prezintă linia reală și linia aproximată trasată peste grila pixelilor.

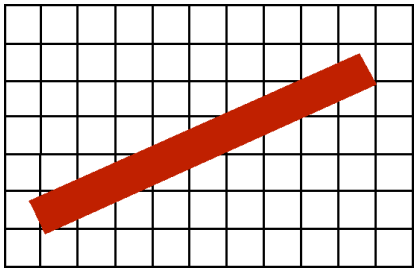

**Figura 1a.** Linia reală

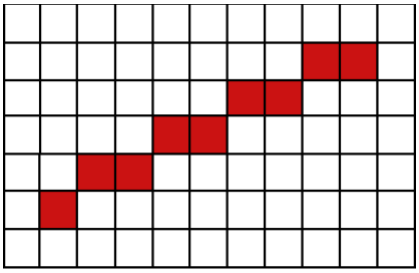

**Figura 1b.** Aproximarea liniei

Înainte de a merge mai departe, merită menționat faptul că linia (1) și cercul (2) pot fi descrise matematic folosind următoarele ecuații:

$$y = m \cdot x + c \quad (1)$$

$$\text{with } m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$(x-a)^2 + (y-b)^2 = R^2 \quad (2)$$

unde:

- $m$ este panta liniei;
- $(x_1, x_2)$, $(y_1, y_2)$ sunt cele două capete ale segmentului de linie;

- $(a, b)$ reprezintă coordonatele centrului cercului;

## 2.1 Algoritmul Bresenham pentru trasarea liniilor

Pentru simplitate vom lua în considerare un segment de linie cu panta între 0 și 1. Să presupunem că cele două puncte finale ale liniei sunt $A(x_1, y_1)$ și $B(x_2, y_2)$. În acest moment trebuie să alegem un punct inițial pentru a porni algoritmul. Putem alege ca acest punct ( $P(x_i, y_i)$ ) să fie A sau B. Pe baza poziției de plecare, avem opt opțiuni posibile pentru a trasa următorul pixel al liniei. Acest lucru se datorează faptului că fiecare pixel este înconjurat de 8 pixeli adiacenți.

Exemplul nostru va lua în considerare numai cazul în care avem două alternative pentru următoarea poziție a pixelului (cu alte cuvinte acest exemplu va funcționa numai pentru primul octant al cercului trigonometric). De exemplu, pentru punctul curent P avem următoarele posibilități de trasare: $T(x_{i+1}, y_i)$ sau $S(x_{i+1}, y_{i+1})$.

Criteriul de decizie (Figura 2) pentru algoritmul lui Bresenham se bazează pe distanța dintre punctul curent, P și segmentul de linie reală. Se va alege cel mai apropiat punct *(T sau S)* pentru segmentul de linie.

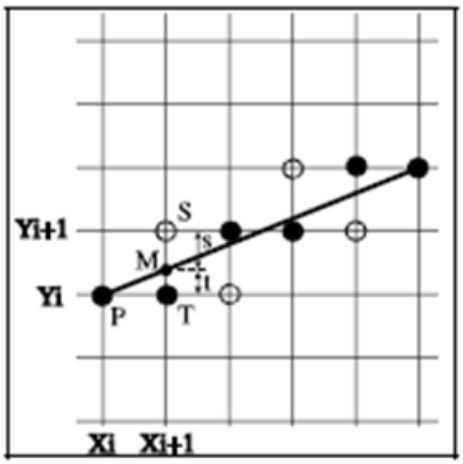


**Figura 2.** Criteriul de decizie pentru alegerea următorului pixel al liniei

Următoarele paragrafe vor descrie pașii generali ai algoritmului lui Bresenham în limbajul natural, mai degrabă decât unul de programare, pentru că este mai ușor de înțeles.

a. Să presupunem că trebuie să trasăm un segment de linie cu punctele finale reprezentate de $A(x_1, y_1)$ și $B(x_2, y_2)$. Efectuăm o translație a segmentului de linie cu $(-x_1, -y_1)$ pentru a o plasa în originea sistemului de coordinate XOY.

b. Fie $dx = x_2 - x_1, dy = y_2 - y_1$. Linia care trebuie trasată poate fii descrisă ca $y = \frac{dx}{dy} x$ .

c. În acest pas intenționăm să calculăm următorul pixel al liniei utilizând criteriul menționat mai sus. Din Figura 2, putem deduce faptul că punctul cel mai apropiat de valoarea liniei reale este $T(x_{i+1}, y_i)$. Pe baza acestei observații, putem spune că $M(x_{i+1}, x_{i+1} \cdot \frac{dy}{dx})$ (3).

Cu alte cuvinte $\begin{cases} t = y_m - y_i \\ s = y_{i+1} - y_m \end{cases} \Rightarrow t - s = 2 \cdot y_m - 2 \cdot y_i - 1$ (4).

Având în vedere (3) și (4) obținem $dx \cdot (t-s) = 2 \cdot dy \cdot x_{i+1} - 2 \cdot dx \cdot y_i - dx$. În cazul în care coordonatele $x$ ale capetelor segmentului de linie sunt în relația $x_1 < x_2$, atunci semnul lui $t-s$ va coincide cu semnul lui $dx \cdot (t-s)$.

d. Putem obține următoarea relație de recurență: $d_{i+1} = d_i + 2 \cdot dy - 2 \cdot dx \cdot (y_i - y_{i+1})$ dacă luăm în considerare faptul că $dx \cdot (t-s) = d_{i+1}$.

Valoarea inițială a $d_i = 2 \cdot dy - dx$ se obține pentru $x_0 = 0$ și $y_0 = 0$. Putem concluziona că:

- Dacă $d_i \geq 0 \Rightarrow (t-s) \geq 0$, iar cel mai apropiat punct de linia reală este $S(x_{i+1}, y_{i+1})$. Pe baza acestor observații concluzionăm ca valoarea lui $d_i$ se poate calcula ca $d_{i+1} = d_i + 2(dy - dx)$.
- Dacă $d_i < 0 \Rightarrow (t-s) < 0$, iar cel mai apropiat punct de linia reală este $T(x_{i+1}, y_i)$. În acest caz, formula de recurență pentru calculu valorii lui $d_i$ este $d_{i+1} = d_i + 2dy$ .

Pseudo-codul pentru algoritmul Bresenham este descris în următorul paragraf și se bazează pe observațiile matematice menționate mai sus.

```
//Traseaza o linie in primul octant
Algorithm Bresenham_line()
{
   //Initializeaza incrementurile
   dx = abs(x2-x1);
   dy = abs(y2-y1);
   d = 2*dy-dx;
   inc1 = 2*dy;
   inc2 = 2*(dy-dx);

   //Seteaza punctul de pornire, punctul de stop si punctul curent
   startX = x1;
   startY = y1;
   endX   = x2;
   endY   = y2;
   currentX = x1;
   currentY = y1;


   //Traseaza fiecare pixel al liniei
   while (currentX < endX) {
```

```
   //Traseaza pixelul curent
   DrawPixel(currentX, currentY);
   incrementeaza currentX;

      if (d < 0) then {
         incrementeaza d folosind inc1;
      }
      else {
         incrementeaza currentY;
         incrementeaza d folosind inc2;
      }
   }
}
```

## 2.2 Algoritmul Bresenham pentru trasarea cercurilor

Să presupunem că vrem să trasăm un cerc centrat la $(0,0)$ cu o rază R exprimată folosind un număr întreg (Figura 3). Vom vedea că ideile pe care le-am folosit anterior pentru trasarea liniei pot fi folosite și pentru această sarcină. Mai întâi de toate, observați faptul că interiorul cercului este caracterizat de inegalitatea $D(x,y): x^2 + y^2 - R^2 < 0$ .

Folosim $D(x,y)$ pentru a determina variabila noastră de decizie.

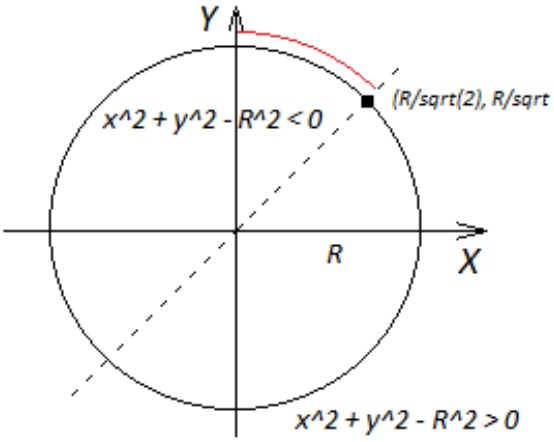


**Figura 3.** Un cerc și descrierea zonelor de interior/exterior folosind inegalități pătratice.

Urmând aceeași abordare ca în cazul reprezentării segmentului de linie, vom prezenta mai întâi algoritmul lui Bresenham pentru cerc în limbaj natural, descriind pentru fiecare pas ideile generale din spatele acestuia.

a. Mai întâi, să ne gândim cum să trasăm pixeli apropiați de 1/8 din cercul marcat cu roșu în Figura 3. Intervalul de coordonate pentru astfel de pixeli este de la 0 la $R\sqrt{2}$. Vom trece linii de scanare verticale prin centrele pixelilor și, pentru fiecare astfel de linie de scanare, vom calcula pixelul de pe acea linie care este cel mai apropiat de punctul de intersecție al cercului cu linia de scanare (punctele negre din Figura 4). Toți acești pixeli vor fi trasați de către procedura noastră.

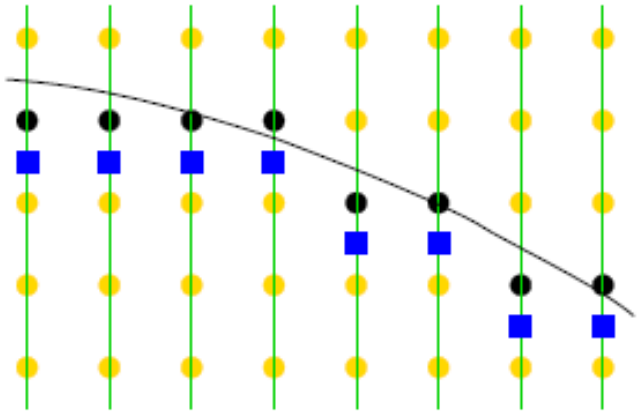

**Figure 4.** A circle and the description of its interior and exterior as two quadratic inequalities

b. Observați că de fiecare dată când trecem la următoarea linie de scanare, coordonata y a punctului trasat rămâne aceeași sau scade cu 1 (panta cercului este între -1 și 0). Pentru a decide ce trebuie făcut, vom folosi variabila de decizie, care va fi valoarea lui $D(x, y)$ evaluată la pătratul albastru (de exemplu, punctul intermediar între pixelul planificat și pixelul imediat inferior).

c. Primul pixel trasat este $(0, R)$, prin urmare valoarea inițială a variabilei de decizie trebuie să fie

$$D(0, R-0.5)=(R-0.5)^2-R^2=0.25-R$$

Variabila $y$, care reține coordonatele secundare ale pixelilor trasați se initializează cu *R*. Să ne gândim ce se întâmplă după ce un punct $(x, y)$ este trasat. În primul rând, să presupunem că am avea nevoie să mutăm punctul trasat la dreapta (fără schimbare de $y$) și să verificăm dacă valoarea variabilei de decizie rămâne negativă (nu vrem niciun pătrat albastru în exteriorul cercului). Dacă $(x, y)$ este ultimul punct trasat, variabila de decizie este $D(x, y-0.5)$. După ce ne mișcăm la dreapta, valoarea devine $D(x+1, y-0.5)$. Aritmetica simplă ne arată că valoarea crește cu $D(x+1, y-0.5)$ - $D(x, y-0.5) = 2x+1$. Dacă această creștere o face pozitivă, trebuie să mergem în jos cu 1 pixel. Pătratul albastru ajunge la $(x+1, y-1.5)$, asta însemnând că trebuie să creștem valoarea variabilei de decizie cu $2x+1$ plus $D(x+1, y-1.5)-D(x+1, y-0.5)=2-2y$.

d. Ca să putem reprezenta variabila de decizie folosind valori intregi, trebuie să o scalăm cu un factor de 4. Pentru a trasa întregul cerc, trebuie să ne folosim de proprietățile de simetrie ale octanților.

Pseudo-codul algoritmului lui Bresenham pentru cerc este descris mai jos, pe baza observațiilor făcute anterior.

```
Algorithm Bresenham_circle ()
{
    currentY = R;
    d = 1/4 – R;

    //Traseaza primul octant al cercului
    for x = 0 to ceil(R/sqrt(2)) do {

        plot_points(currentX,currentY);
        incrementeaza variabila de decizie cu 2x + 1;

        if (d > 0) then
        {
             incrementeaza variabila de decizie cu 2 – 2y;
             decrementeaza currentY;
        }
    }
}
```

Definiția funcției **plot_points** se găsește mai jos:

```
Function plot_points (x, y)
{
    DrawPixel (x,y);
    DrawPixel (x,–y);
    DrawPixel (–x,y);
    DrawPixel (–x,–y);
    DrawPixel (y,x);
    DrawPixel (–y,x);
    DrawPixel (y,–x);
    DrawPixel (–y,–x);
}
```

# 3 Rasterizarea geometriei folosind API-ul SDL

Biblioteca SDL 2.0 oferă un API de redare accelerată hardware pentru forme de bază, cum ar fi dreptunghiuri, linii sau puncte.

Pentru a utiliza redarea accelerată hardware, trebuie să creați un nou SDL_Renderer pentru fereastra SDL.

```
//The window renderer
SDL_Renderer* renderer = NULL;
...
//Create renderer for window
renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
```

Pentru a selecta culoarea de rasterizare și pentru a reseta conținutul ferestrei putem sa folosim **SDL_SetRenderDrawColor** și **SDL_RenderClear**.

```
//Clear screen
SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
SDL_RenderClear(renderer);
```

Ori de câte ori este rasterizată o primitivă, culoarea sa este culoarea de rasterizare selectată în prezent. Pentru a rasteriza puncte (pixeli) putem folosi **SDL_RenderDrawPoint**.

```
//Draw current point
SDL_RenderDrawPoint(renderer, tmpCurrentX, tmpCurrentY);
```

# 4 Temă

- Explorați implementarea algoritmului lui Bresenham pentru desenarea liniilor (furnizată în folderul de resurse al laboratorului).
- Extindeți implementarea algoritmului lui Bresenham pentru desenarea liniilor astfel încât acesta să funcționeze în toate octantele cercului trigonometric. Algoritmul prezentat în acest document, precum și exemplul de cod acoperă doar primul octant. Notă: Aveți grijă atunci când implementați algoritmul lui Bresenham. Sistemul de coordonate SDL este diferit de sistemul de coordonate cartezian.
- Implementați funcția de trasareva cercurilor utilizând algoritmul lui Bresenham din exemplul de cod furnizat.