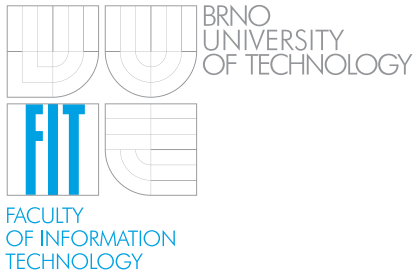
IZG 2010/11, cvičení 2: Generování základních objektů v rastru

Marek Šolony

(s využitím materiálů Přemysla Krška, Michala Španěla, Ondřeje Šilera, Bronislava Přibyla)

Ústav počítačové grafiky a multimédií
Fakulta informačních technologií

Vysoké učení technické v Brně

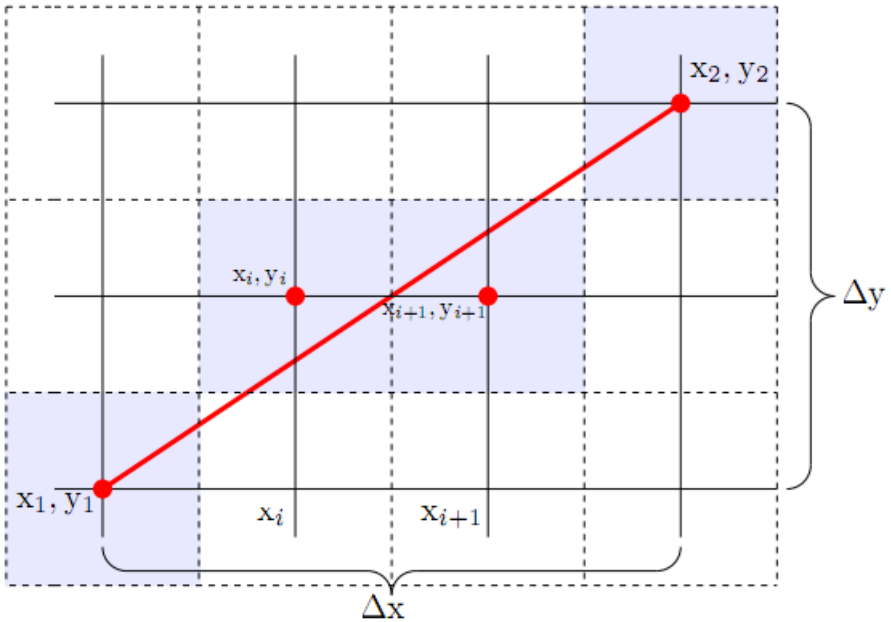


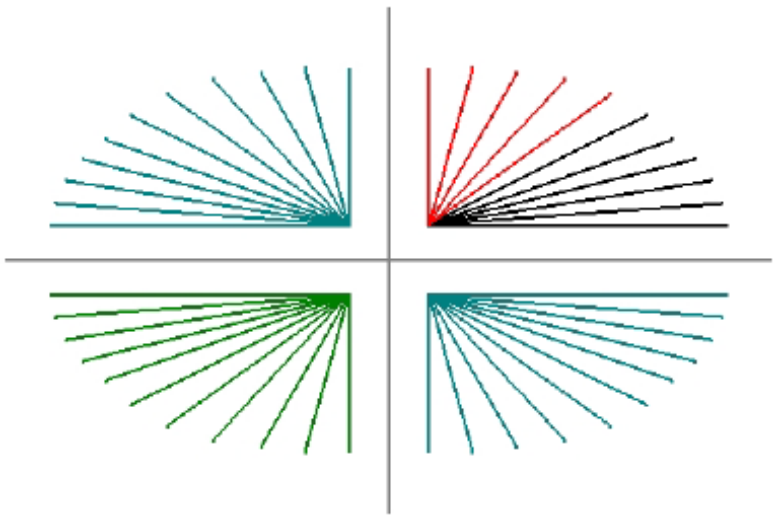
Předpoklady užití raterizačních algoritmů

Rasterizujeme úsečku mezi body $[x_1, y_1]$ a $[x_2, y_2]$.

- $x_1, x_2, y_1, y_2 \geq 0$ – úsečka leží v 1. kvadrantu
- $x_1 < x_2, y_1 \leq y_2$ – je neklesající od poč. bodu ke koncovému
- $\Delta x \geq \Delta y$ – roste rychleji ve směru osy X

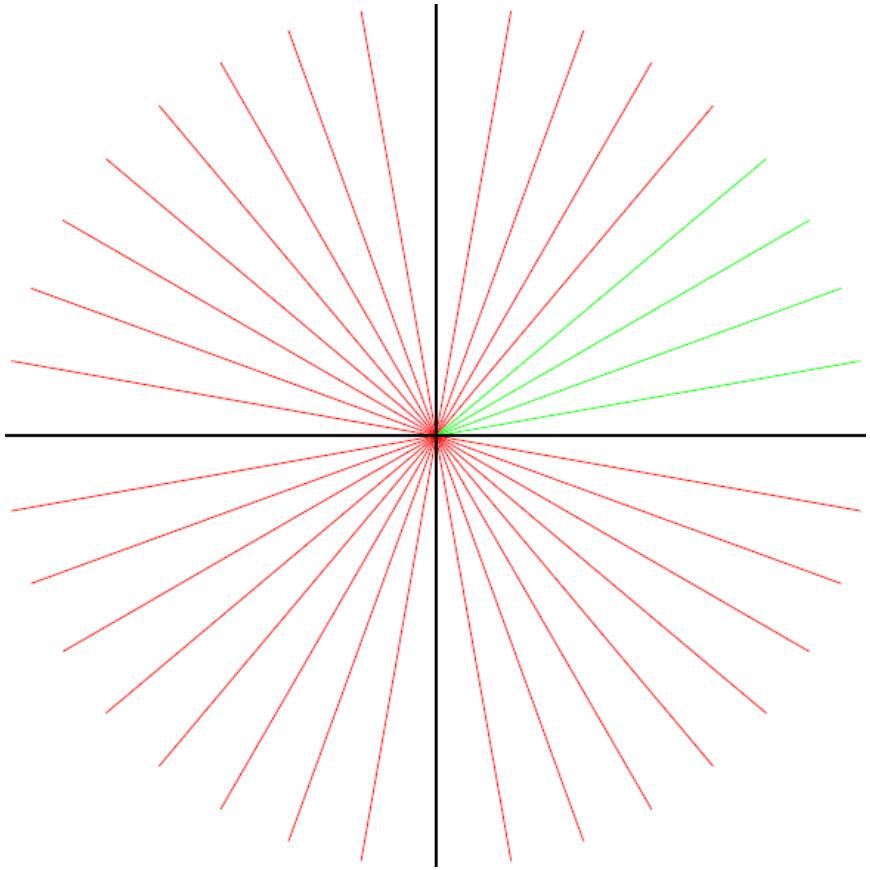
Ostatní polohy úsečky je třeba převést na tento případ (prohození souřadnic, os apod.)





Obecná poloha úsečky

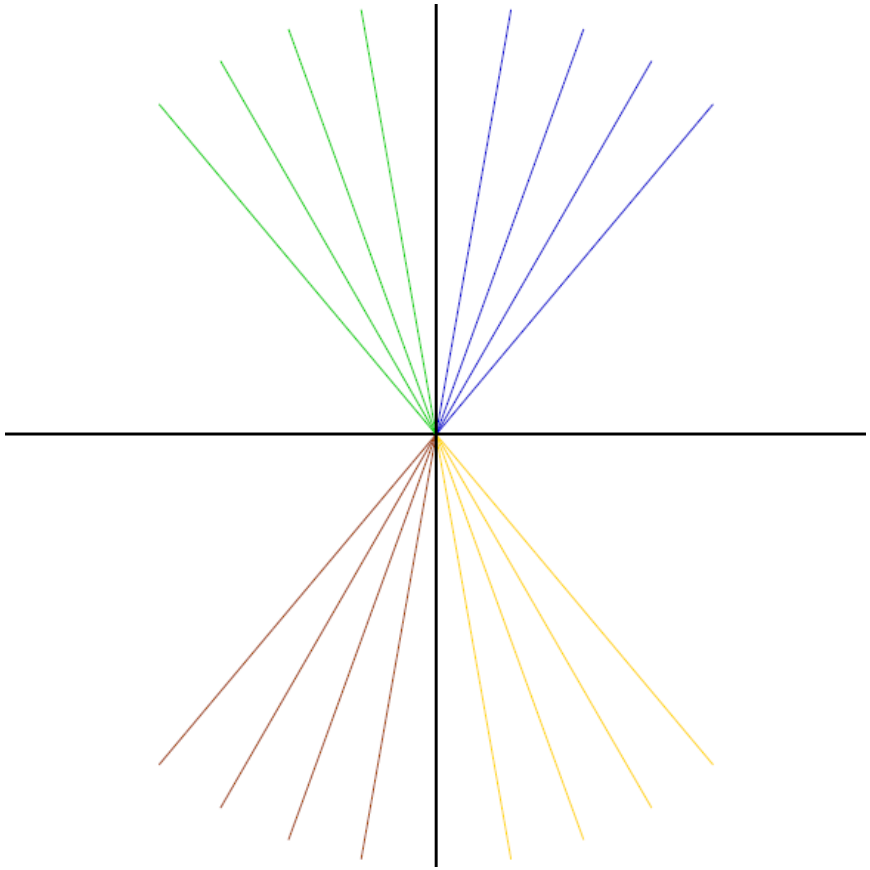
Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:



Obecná poloha úsečky

Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:

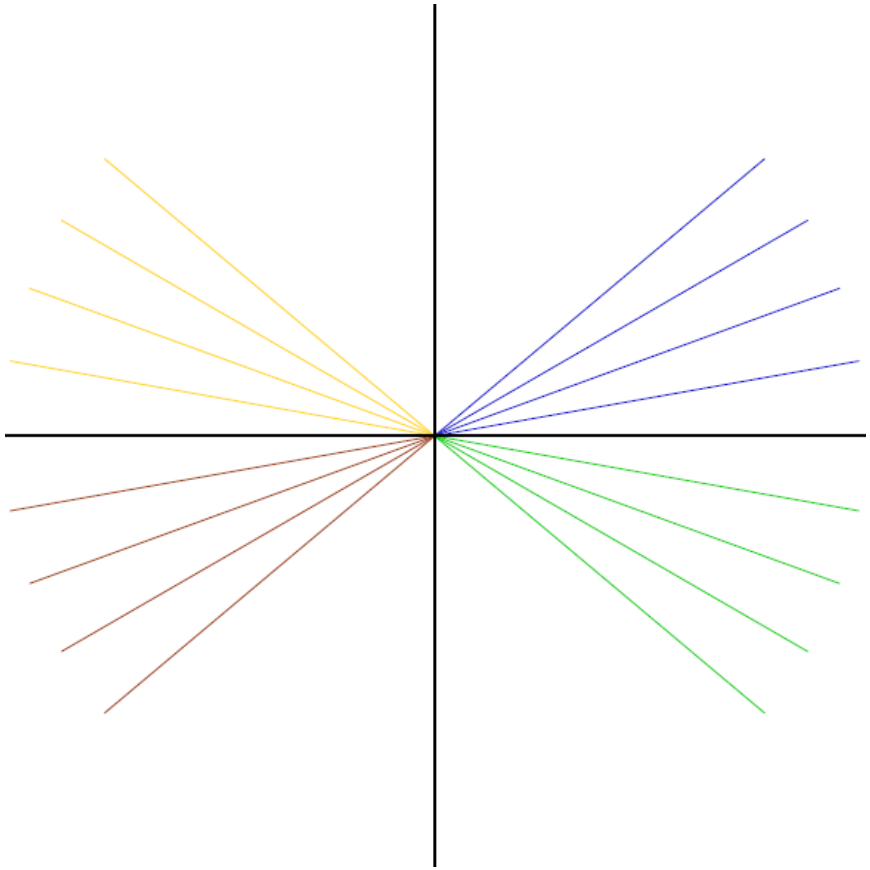
- **Příliš rychle roste/klesá**



Obecná poloha úsečky

Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:

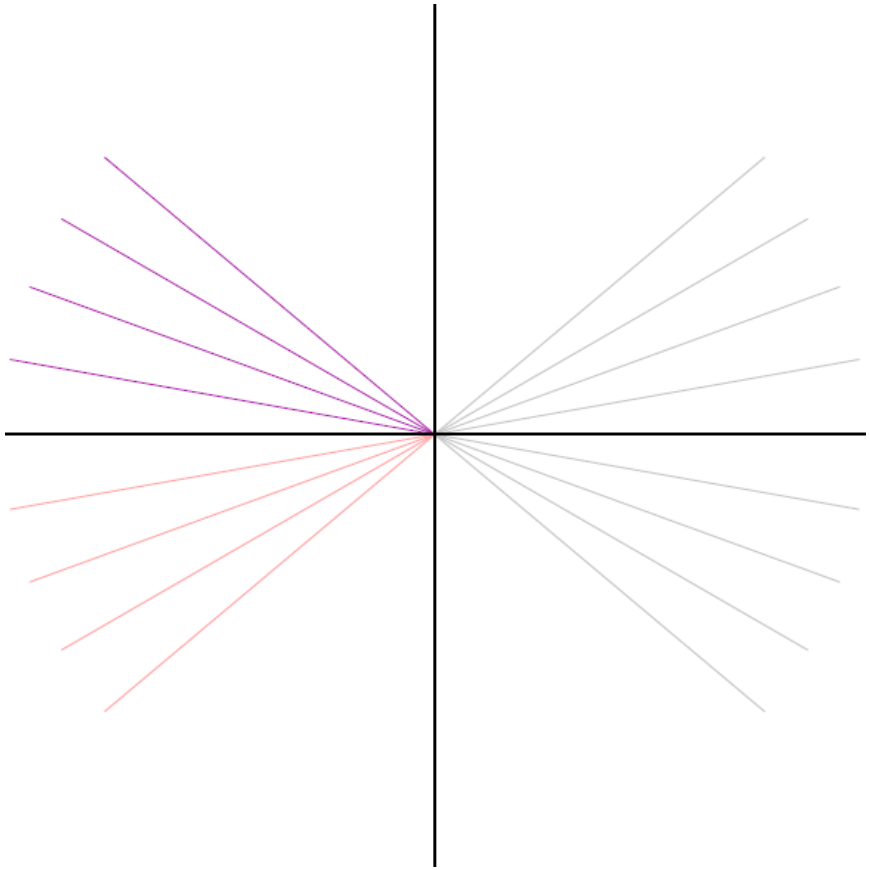
- **Příliš rychle roste/klesá**
  - záměna souřadnic X a Y
  - souřadnice nutno znovu zaměnit při zápisu pixelu do framebufferu!



Obecná poloha úsečky

Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:

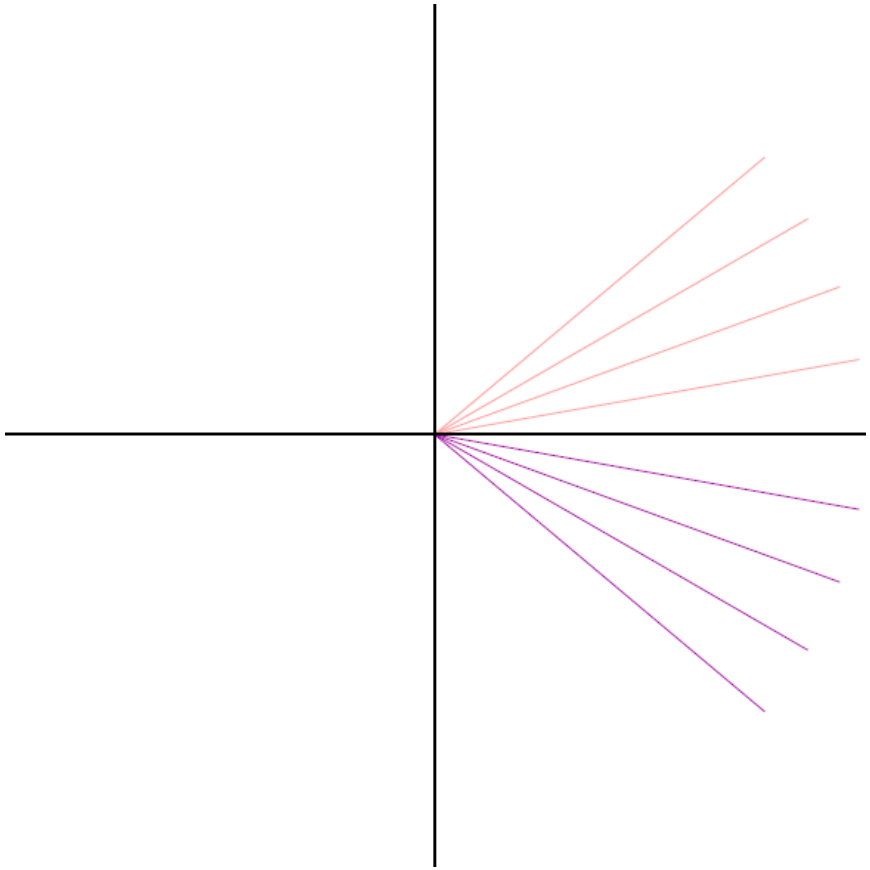
- **Příliš rychle roste/klesá**
  - záměna souřadnic X a Y
  - souřadnice nutno znovu zaměnit při zápisu pixelu do framebufferu!
- **Zprava doleva**



Obecná poloha úsečky

Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:

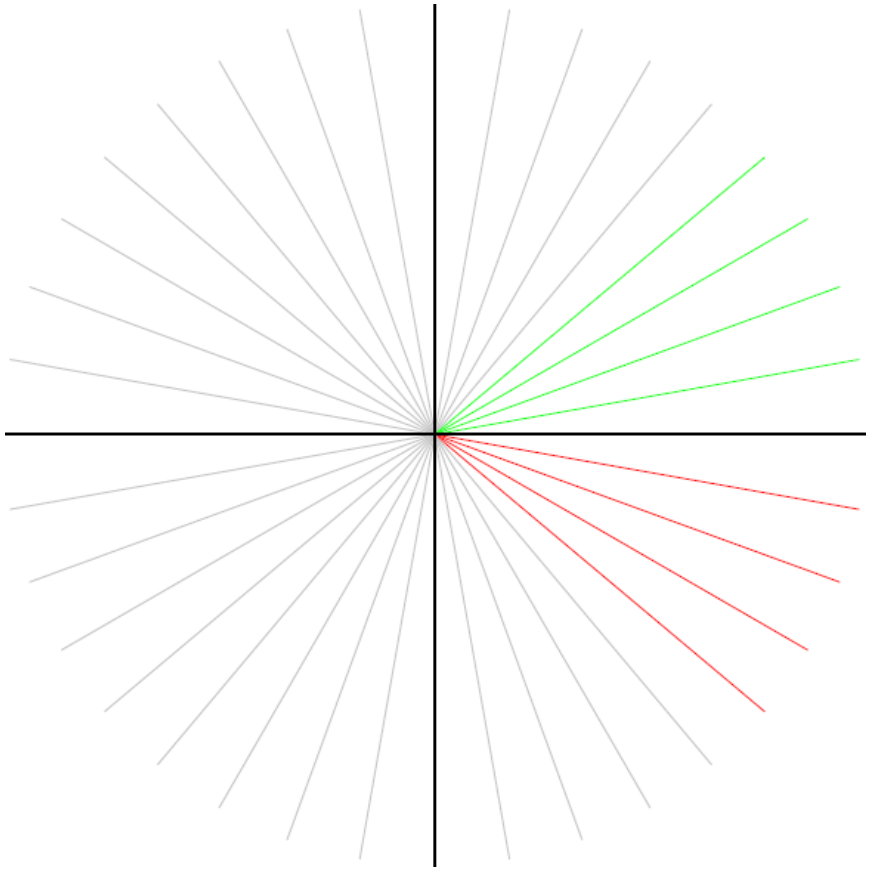
- **Příliš rychle roste/klesá**
  - záměna souřadnic X a Y
  - souřadnice nutno znovu zaměnit při zápisu pixelu do framebufferu!
- **Zprava doleva**
  - prohození koncových bodů



Obecná poloha úsečky

Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:

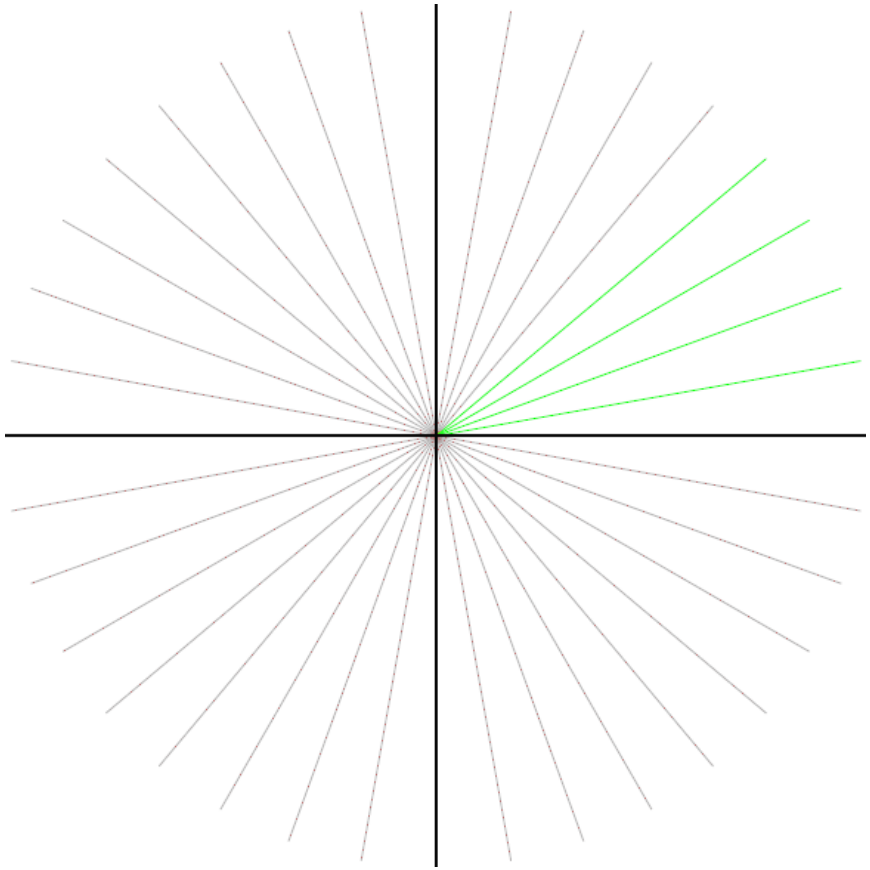
- **Příliš rychle roste/klesá**
  - záměna souřadnic X a Y
  - souřadnice nutno znovu zaměnit při zápisu pixelu do framebufferu!
- **Zprava doleva**
  - prohození koncových bodů
- **Shora dolů**



Obecná poloha úsečky

Chceme rasterizovat úsečku, která vede zleva zespodu a doprava nahoru, ale neroste příliš rychle. Co ale dělat, když:

- **Příliš rychle roste/klesá**
  - záměna souřadnic X a Y
  - souřadnice nutno znovu zaměnit při zápisu pixelu do framebufferu!
- **Zprava doleva**
  - prohození koncových bodů
- **Shora dolů**
  - posun v Y v opačném směru



Algoritmy rasterizace úsečky

- DDA
- DDA s hlídáním chyby
- Bresenhamův algoritmus

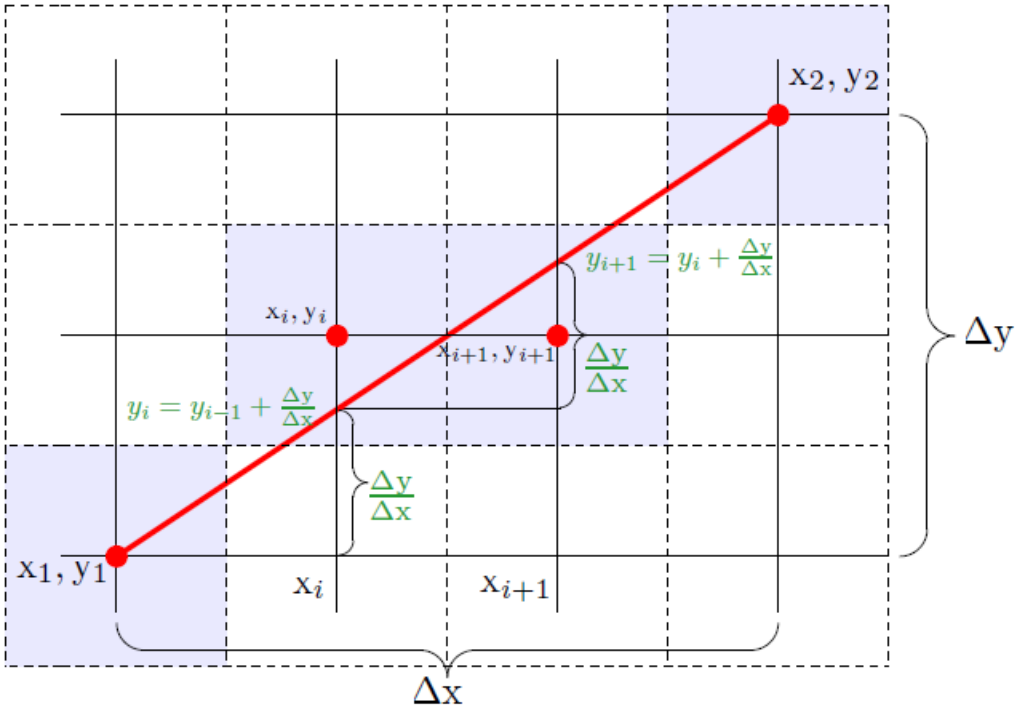
Vlastnosti

- Jeden z prvních a nejjednodušších algoritmů.
- Používá aritmetiku čísel v **plovoucí řádové čárce**.
- Tudíž **neefektivní** a náročné implementovat v HW.

Princip

- Vykreslujeme pixel po pixelu od $[x_1, y_1]$ k $[x_2, y_2]$.
- V ose X postupujeme s přírůstkem 1.
- V ose Y postupujeme s přírůstkem daným velikostí **směrnice** úsečky $k = \frac{\Delta y}{\Delta x}$.
- Y-ová souřadnice se zaokrouhluje na nejbližší celé číslo.

DDA – pseudokód




```
LineDDA(int x1, int y1, int x2, int y2) {
  const double k = (y2-y1) / (x2-x1);
  double y = y1;
  for (int x = x1; x <= x2; x++) {
    DrawPixel(x, round(y));
    y += k;
  }
}
```

DDA s hlídáním chyby

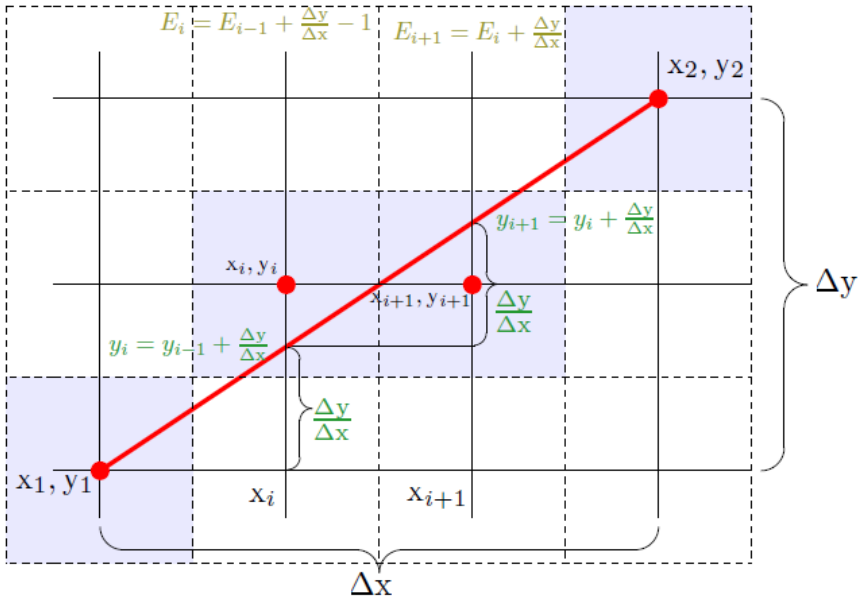
Vlastnosti

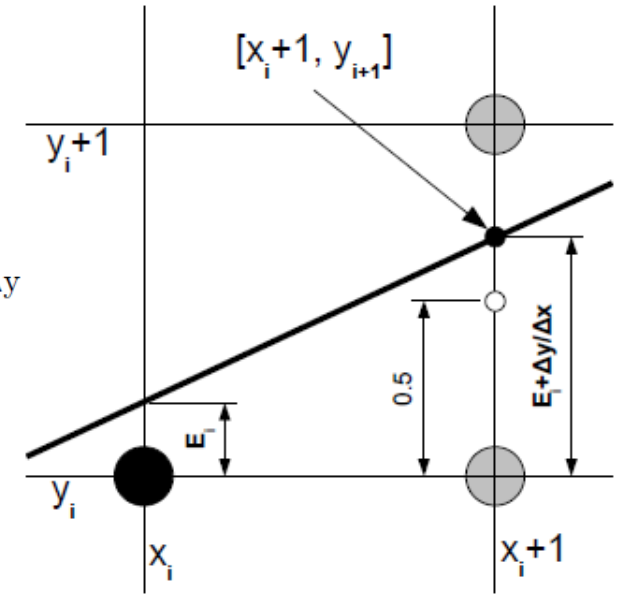
- Modifikace DDA algoritmu.
- Také používá aritmetiku čísel v **plovoucí řádové čárce**.
- Tudíž **neefektivní** a náročné implementovat v HW.

Princip

- Vykreslujeme pixel po pixelu od $[x_1, y_1]$ k $[x_2, y_2]$.
- V ose X postupujeme s přírůstkem 1.
- V ose Y počítáme aktuální **odchylku** (chybu) E přesné souřadnice od celočíselné souřadnice.
- Pokud $E \geq 0,5$, v Y posun na další řádek a $E = E - 1$.

DDA s hlídáním chyby – pseudokód






```
LineECDDA(int x1, int y1, int x2, int y2) {
  const double k = (y2-y1) / (x2-x1);
  double E = 0.0;
  int    y = y1;
  for (int x = x1; x <= x2; x++) {
    DrawPixel(x, y);
    E += k;
    if (E >= 0.5) { y++; E -= 1; }
  }
}
```

Bresenhamův algoritmus

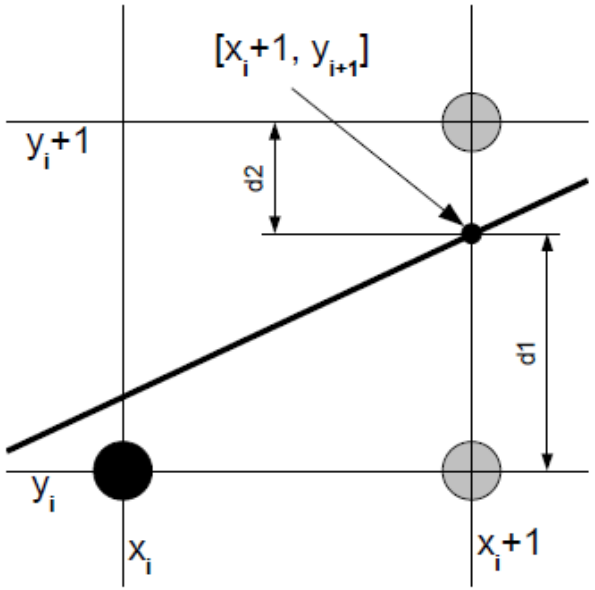
Vlastnosti

- Nejpoužívanější a nejefektivnější algoritmus rasterizace úsečky.
- Používá **celočíselnou aritmetiku** – sčítání, posuny, porovnání.
- Velmi efektivní, ideální pro **HW implementaci**.

Princip

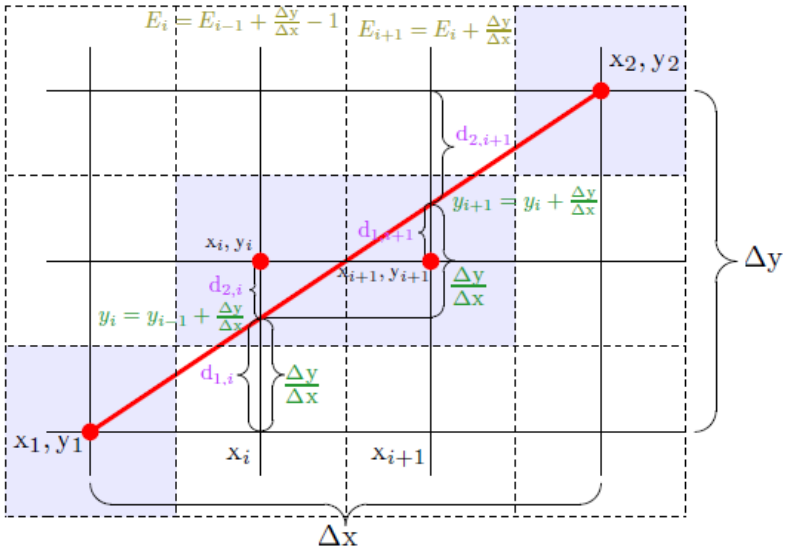
- Vykreslujeme pixel po pixelu od $[x_1, y_1]$ k $[x_2, y_2]$.
- V ose X postupujeme s přírůstkem 1.
- O posunu v ose Y rozhoduje **znaménko** tzv. **prediktoru**.

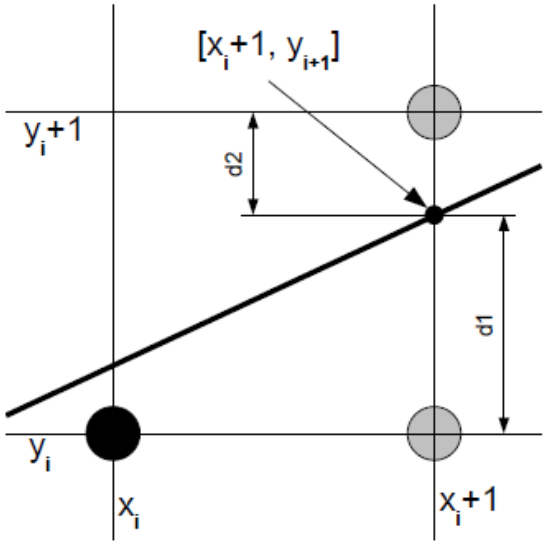
Bresenhamův algoritmus – prediktor



$$d_2 > d_1 \quad \Rightarrow \quad P < 0 \quad \Rightarrow \quad y_{i+1} = y_i$$
$$d_2 \leq d_1 \quad \Rightarrow \quad P \geq 0 \quad \Rightarrow \quad y_{i+1} = y_i + 1$$

Bresenhamův algoritmus – pseudokód






```
LineBres(int x1, int y1, int x2, int y2) {
  const int dx = x2-x1, dy = y2-y1;
  const int P1 = 2*dy,  P2 = P1 - 2*dx;
  int P = 2*dy - dx;
  int y = y1;
  for (int x = x1; x <= x2; x++) {
    DrawPixel(x, y);
    if (P >= 0) { P += P2; y++; }
    else        { P += P1; }
  }
}
```

Úkol

- Doplnit tělo funkce `DrawLine()` v souboru *student.cpp*
- Vykreslit úsečku Bresenhamovým algoritmem

Ovládání

- Úsečka: Natáhnout myší z počátečního do koncového bodu.
- 'c' – (clean) přepíše framebuffer bílou barvou.
- 'Esc'/'q' – ukončí program.

Nástroje

- Funkce `void PutPixel(int x, int y, S_RGBA color)`.
- Makro `SWAP(a,b)` v souboru *main.h*.

Kružnice

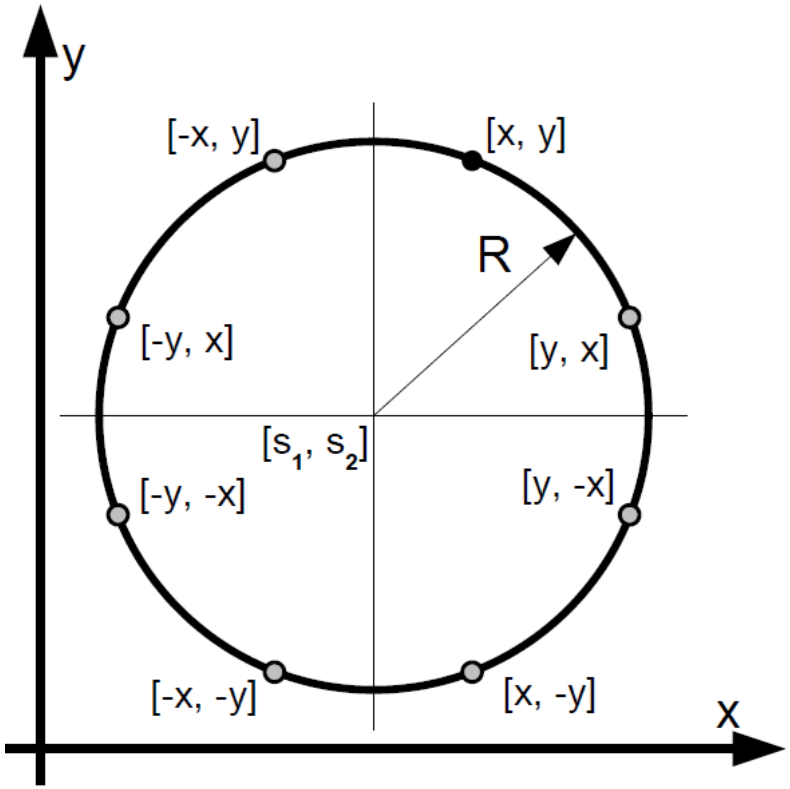
Definice

- Souřadnice středu $[s_1, s_2]$
- Poloměr R

$$\{[x, y] : (x - s_1)^2 + (y - s_2)^2 = R^2\}$$

Vlastnosti

- Je 8× symetrická
- Provádíme výpočet pro 1/8 bodů (v polovině 1. kvadrantu)
- Zbylé body získáme záměnou souřadnic
- Algoritmy jsou odvozeny pro kružnici se středem v počátku



Midpoint algoritmus pro kružnici

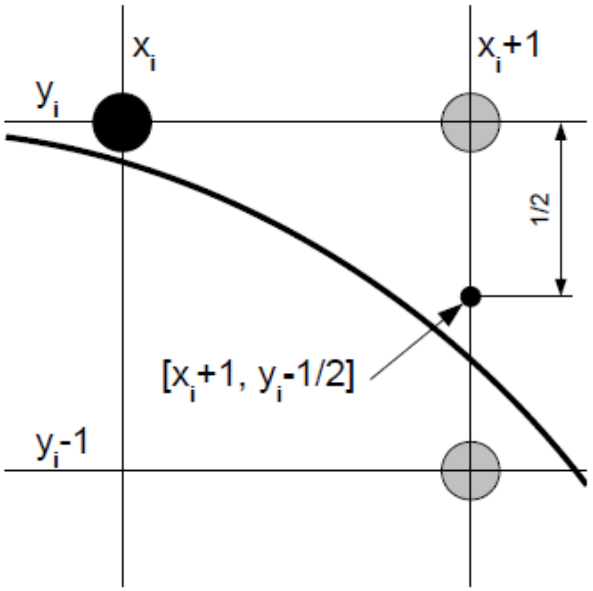
Vlastnosti

- Variace na Bresenhamův algoritmus. Stejný přístup.
- Testování polohy "midpointu" vůči kružnici (uvnitř/vně).
- Používá **celočíselnou aritmetiku** – sčítání, porovnání.
- Velmi efektivní, ideální pro **HW implementaci**.

Princip

- Vykreslujeme pixel po pixelu od bodu $[0, R]$, dokud $x < y$.
- V ose X postupujeme s přírůstkem 1.
- O posunu v ose Y rozhoduje **znaménko prediktoru**.

Midpoint algoritmus – prediktor



$$\textit{midpoint uvnitr kruznice} \Rightarrow P < 0 \Rightarrow y_{i+1} = y_i$$
$$\textit{midpoint vne kruznice} \Rightarrow P \geq 0 \Rightarrow y_{i+1} = y_i - 1$$

Midpoint algoritmus – pseudokód

Následjící algoritmus platí pro kružnici se středem v počátku!

```
CircleMidpoint(int s1, int s2, int R) {
  int x = 0, y = R;
  int P = 1-R, X2 = 3, Y2 = 2*R-2;
  while (x < y) {
    DrawPixel(x, y);
    if (P >= 0) {
      P += -Y2;
      Y2 -= 2;
      y--;
    }
    P += X2;
    X2 += 2;
    x++;
  }
}
```

Bodovaný úkol

Úkol

- Doplnit tělo funkce `DrawCircle()` v souboru *student.cpp*
- Vykreslit kružnici Midpoint algoritmem
- Hodnocení: 0 – 3 body

Ovládání

- Kružnice: Nakreslit pravým tlačítkem
  (nastavit kurzor na střed, ze středu "natáhnout" poloměr)
- 'c' – (clean) přepíše framebuffer bílou barvou.
- 'Esc'/'q' – ukončí program.

Nástroje

- Funkce `void PutPixel(int x, int y, S_RGBA color)`.
- Makro `SWAP(a,b)` v souboru *main.h*.