Některé rastrové algoritmy pro vykreslování 2D objektů

Pavel Strachota

FJFI ČVUT v Praze

2. listopadu 2012

Obsah

Obsah

Rasterizace

- objekty popsány matematicky - geometrické útvary, text
- cíl - zobrazení na výstupním zařízení
  - **plotter** - přímý tisk vektorové grafiky
  - **obrazovka, tiskárna** - reprezentace objektu v diskrétní mřížce obrazových bodů - **rasterizace** (*scan conversion*)
- co nejjpřesnější vystižení spojitých útvarů na diskrétní mřížce
- **ořezávání** (*clipping*) - omezení zobrazovaných útvarů na výřez daný plochou obrazovky, resp. na daný výřez na obrazovce (*viewport* - obdélník, *clip region* - obecný tvar)

Rasterizace

Nízkoúrovňové operace

- vykreslení jednoho bodu - zápis do grafické paměti (frame bufferu)
  - obsah frame bufferu kontinuálně vykreslován na obrazovku
  - mapování frame bufferu do adresového prostoru CPU
  - příprava obrazu (více obrazů) „offline“ a blokové zkopírování do paměti
- hardwarový RIP (*Raster Image Processor*)
  - interpret PostScriptu na tiskárně
  - 2D akcelerátor

Ořezávání

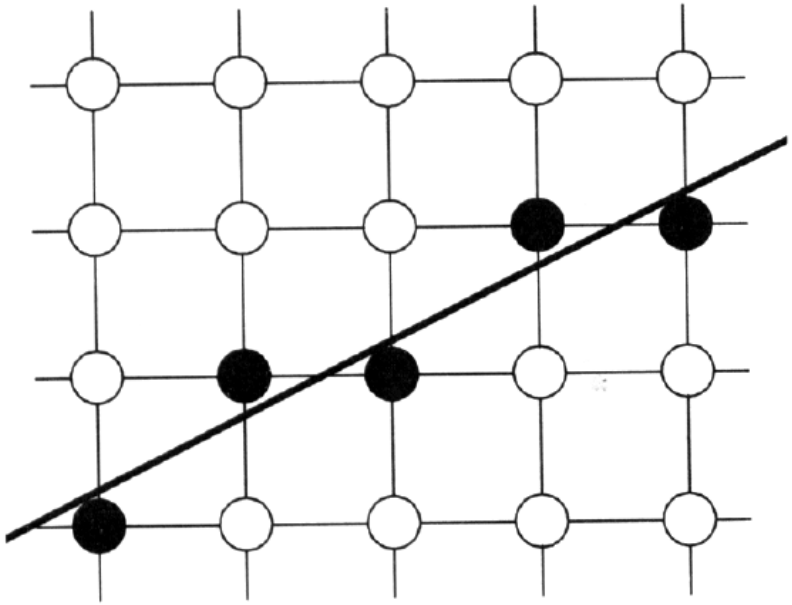
- ořezání před rasterizací - výpočtem určíme, které části objektu se zobrazí do výřezu a vykreslujeme jen je
  - nejefektivnější pro objekty viditelné jen z malé části
  - budeme se jimi zabývat později
- *scissoring* - před zápisem každého pixelu testujeme, zda leží ve výřezu
  - nejjednodušší a nejnákladnější metoda
  - umožňuje obecné tvary výřezů
  - pokud je efektivně implementována (např. v hardwaru), může být rychlejší než 1. metoda
- generování celého obrazu na „off-line“ plátno (do RAM). Následně se do framebufferu přenese je obsah výřezu.

Obsah

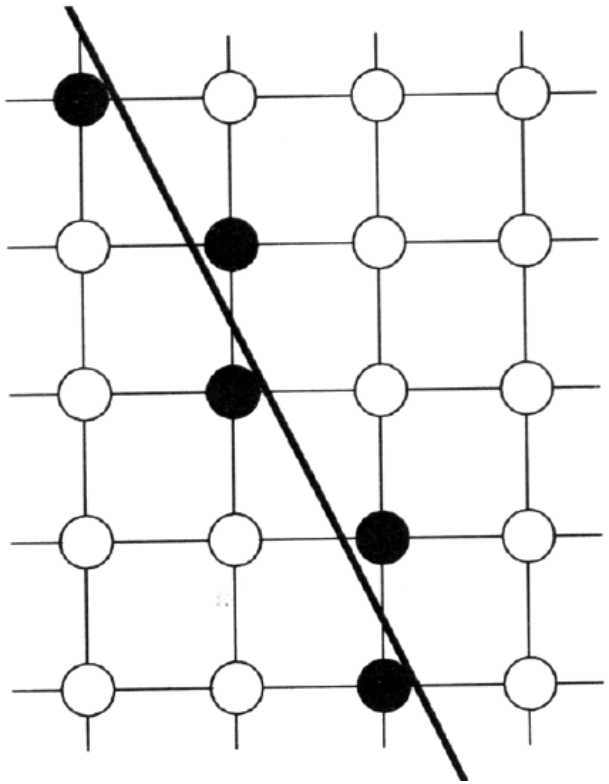
Rasterizace úsečky

- určení pixelů ležících na, resp. blízko ideální nekonečně tenké úsečky, případně přímky
- aproximace - čára 1 pixel široká
  - sklon do $\pm 45°$ $\Longrightarrow$ 1 pixel v každém sloupci
  - ostatní sklony $\Longrightarrow$ 1 pixel v každém řádku
- tlustší čáry
  - styl čáry
  - styl pera
  - další atributy (např. navazování čar)
- ideální **geometrie pixelu** - kruh se středem na mřížce s celočíselnými souřadnicemi uzlů

Rasterizace úsečky



Sklon $\leq 45°$, řídící osa x



Sklon $> 45°$, řídící osa y

Rasterizace úsečky

Nejjednodušší algoritmus - tzv. DDA

- směrnice přímky $m = \Delta y / \Delta x \leq 1$ (sklon $\leq 45°$), rozsah na ose x dán souřadnicemi x_i, kde $i \in \mathbb{N} \cap [x_{start}, x_{end}]$
- výpočet souřadnice y_i bodu ve sloupci x_i:

$$y_i = mx_i + B$$

- vykreslíme pak bod $[x_i, \text{round}(y_i)]$, kde

$$\text{round}(y_i) = \text{floor}\left(y_i + \frac{1}{2}\right) = \left\lfloor y_i + \frac{1}{2} \right\rfloor$$

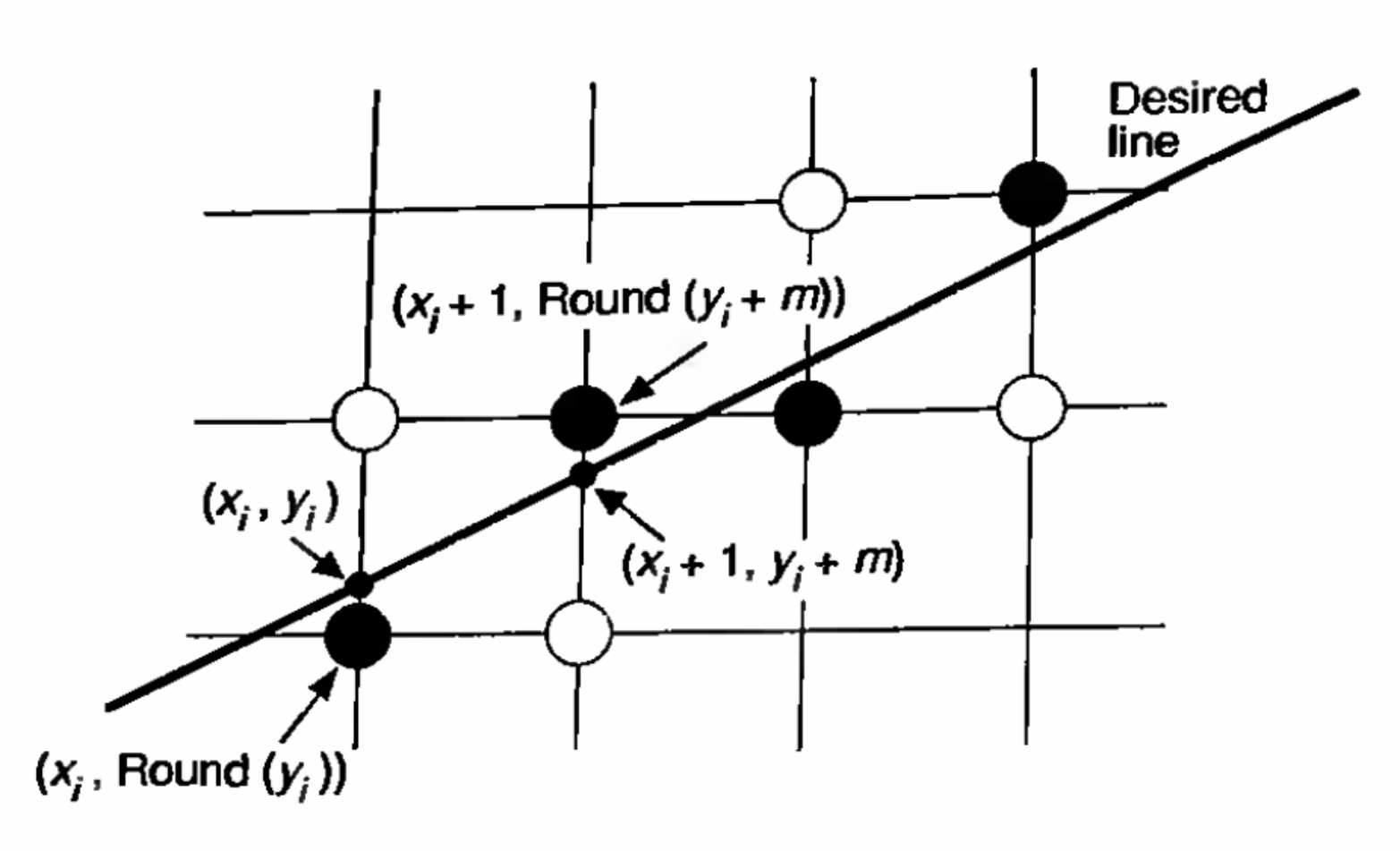
- lze se **zbavit násobení** $\Longrightarrow$ DDA

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

obvykle $\Delta x = 1 \Longrightarrow y_{i+1} = y_i + m$

Rasterizace úsečky

Nejjednodušší algoritmus - tzv. DDA



- pokud $m > 1 \implies$ v algoritmu zaměníme x a y

Rasterizace úsečky

Vlastnosti DDA

- **DDA** - *Digital Differential Analyzer*
- Differential Analyzer - mechanický počítač pro řešení diferenciálních rovnic
- DDA - digitální verze DA, numerická integrace

$$\int_a^b y'(x)\,\mathrm{d}x \approx y(a) + \sum_i y'(x_i)\,\Delta x_i$$

- zde $y'(x) = \frac{\mathrm{d}y}{\mathrm{d}x} = m$
- do kategorie DDA v širším slova smyslu spadají i další (následující) algoritmy

Rasterizace úsečky

Vlastnosti DDA

nevýhody **tohoto** DDA:

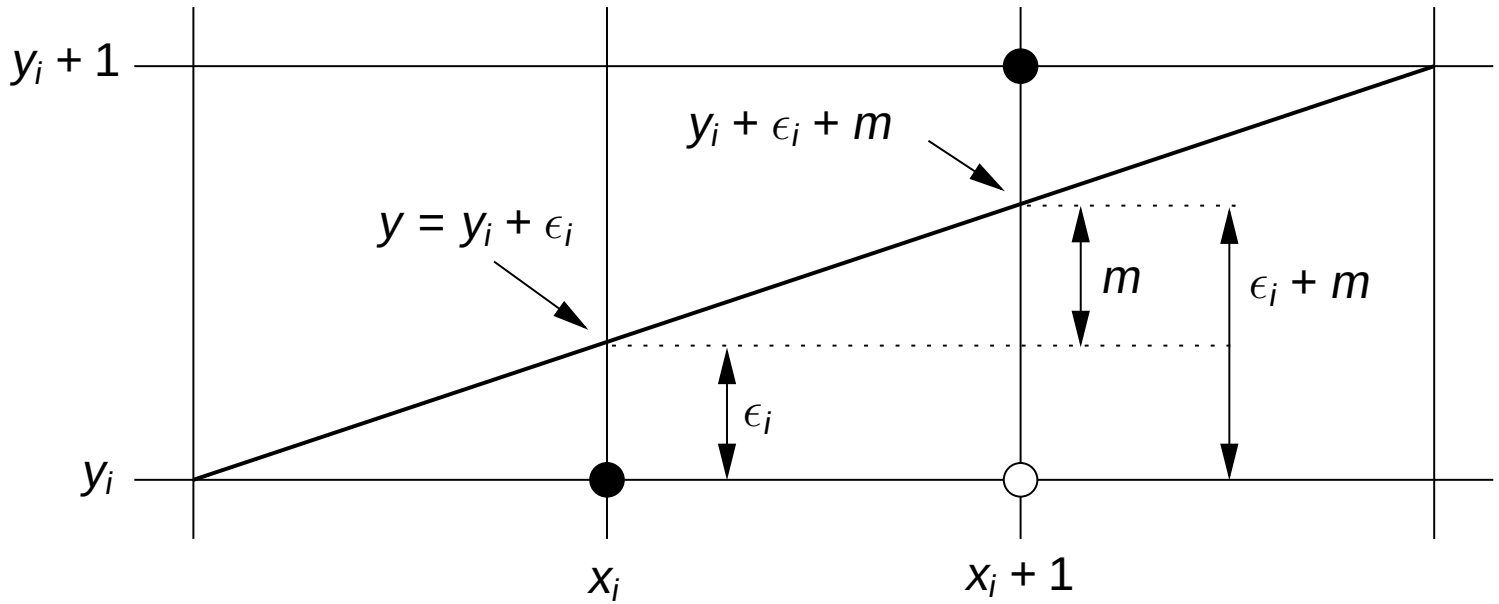
- akumulace chyby při inkrementálním přičítání - nepředstavuje velký problém pro krátké úsečky (max. rozměru monitoru)
- m je neceločíselné - nutno použít floating point aritmetiku
- funkce `floor()` zdržuje

$\Longrightarrow$ snaha o omezení na celočíselnou aritmetiku

Rasterizace úsečky

Bresenhamův algoritmus 1/5

- nechť $\Delta = 1$, rovnice skutečné přímky nechť je $y = y(x)$, **celočíselná** souřadnice pixelu y_i
- DDA pro sklon $0 < m < 1$: současný *pixel* $[x_i, y_i]$, další bude buď $[x_i + 1, y_i]$, nebo $[x_i + 1, y_i + 1]$
- označme chybu v souřadnici y jako $\epsilon_i = y(x_i) - y_i$



Rasterizace úsečky

Bresenhamův algoritmus 2/5

Rozhodovací test pro bod na x_{i+1}

- $\epsilon_i + m \leq \frac{1}{2} \implies$ vykreslíme bod $[x_i + 1, y_i]$, tj.

$$\begin{aligned} y_{i+1} &= y_i \\ \epsilon_{i+1} &= y(x_{i+1}) - y_{i+1} = (y(x_i) + m) - y_i = \epsilon_i + m \end{aligned}$$

- $\epsilon_i + m > \frac{1}{2} \implies$ vykreslíme bod $[x_i + 1, y_i + 1]$, tj.

$$\begin{aligned} y_{i+1} &= y_i + 1 \\ \epsilon_{i+1} &= y(x_{i+1}) - y_{i+1} = (y(x_i) + m) - (y_i + 1) = \epsilon_i + m - 1 \end{aligned}$$

- podmínku $\epsilon_i + m \gtreqless \frac{1}{2}$ lze napsat ve tvaru $y(x_{i+1}) \gtreqless y_i + \frac{1}{2}$, tj. zda tzv. *střední bod* $y_i + \frac{1}{2}$ leží pod nebo nad přímkou (*midpoint line algorithm*)

Rasterizace úsečky

Bresenhamův algoritmus 3/5

- předpokládejme počáteční a koncový bod úsečky ve středech pixelů (celočíselné souřadnice) $\Longrightarrow \epsilon_0 = 0$
- $m = \Delta y / \Delta x = (y_{end} - y_{start}) / (x_{end} - x_{start})$
- definujeme $\tilde{\epsilon}_i = \Delta x \cdot \epsilon_i$, potom dostaneme

Rozhodovací test v celočíselné aritmetice

- $2(\tilde{\epsilon}_i + \Delta y) \leq \Delta x \Longrightarrow$ vykreslíme bod $[x_i + 1, y_i]$, tj.

$$\begin{aligned} y_{i+1} &= y_i \\ \tilde{\epsilon}_{i+1} &= \tilde{\epsilon}_i + \Delta y \end{aligned}$$

- $2(\tilde{\epsilon}_i + \Delta y) > \Delta x \Longrightarrow$ vykreslíme bod $[x_i + 1, y_i + 1]$, tj.

$$\begin{aligned} y_{i+1} &= y_i + 1 \\ \tilde{\epsilon}_{i+1} &= \tilde{\epsilon}_i + \Delta y - \Delta x \end{aligned}$$

Rasterizace úsečky

Bresenhamův algoritmus 4/5

Algoritmus pro úsečku $[x1, y1] - [x2, y2]$, $x2 > x1$ a $y2 > y1$

```
void line(int x1, int y1, int x2, int y2, c) {
 int x,eps=0,y=y1;
 int dx=x2-x1, dy=y2-y1;
 for(x=x1;x<x2;x++) {
  PutPixel(x,y,c);
  if(2*(eps+dy) <= dx)
   eps += dy;
  else {
   y++; eps += dy-dx;
  }
 }
}
```

Rasterizace úsečky

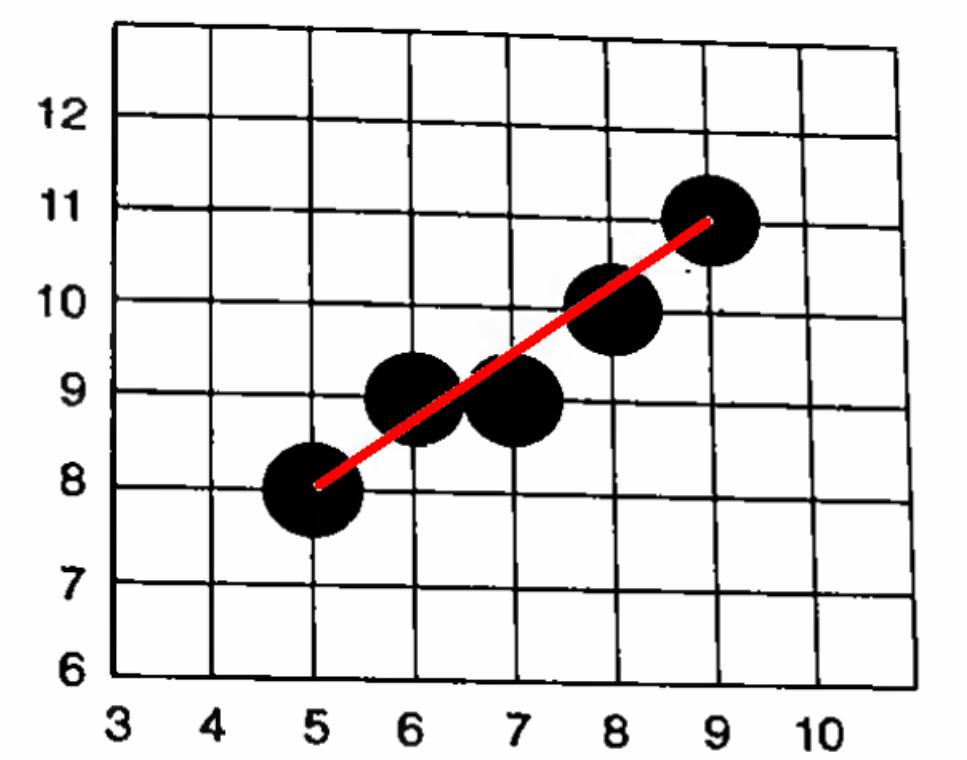
Bresenhamův algoritmus 5/5 - vlastnosti

- pouze celočíselná aritmetika, místo `2*(eps+dy)` lze napsat `(eps+dy) << 1`
- je třeba ošetřit
  - negativní sklon
  - sklon $m > 1$ (nad 45°) $\Longrightarrow$ řídící osa bude y
  - svislé a vodorovné úsečky
- každá (původní, ideální) úsečka je symetrická podle středu $\Longrightarrow$ lze kreslit 2 pixely najednou

Rasterizace úsečky

Závislost na pořadí koncových bodů

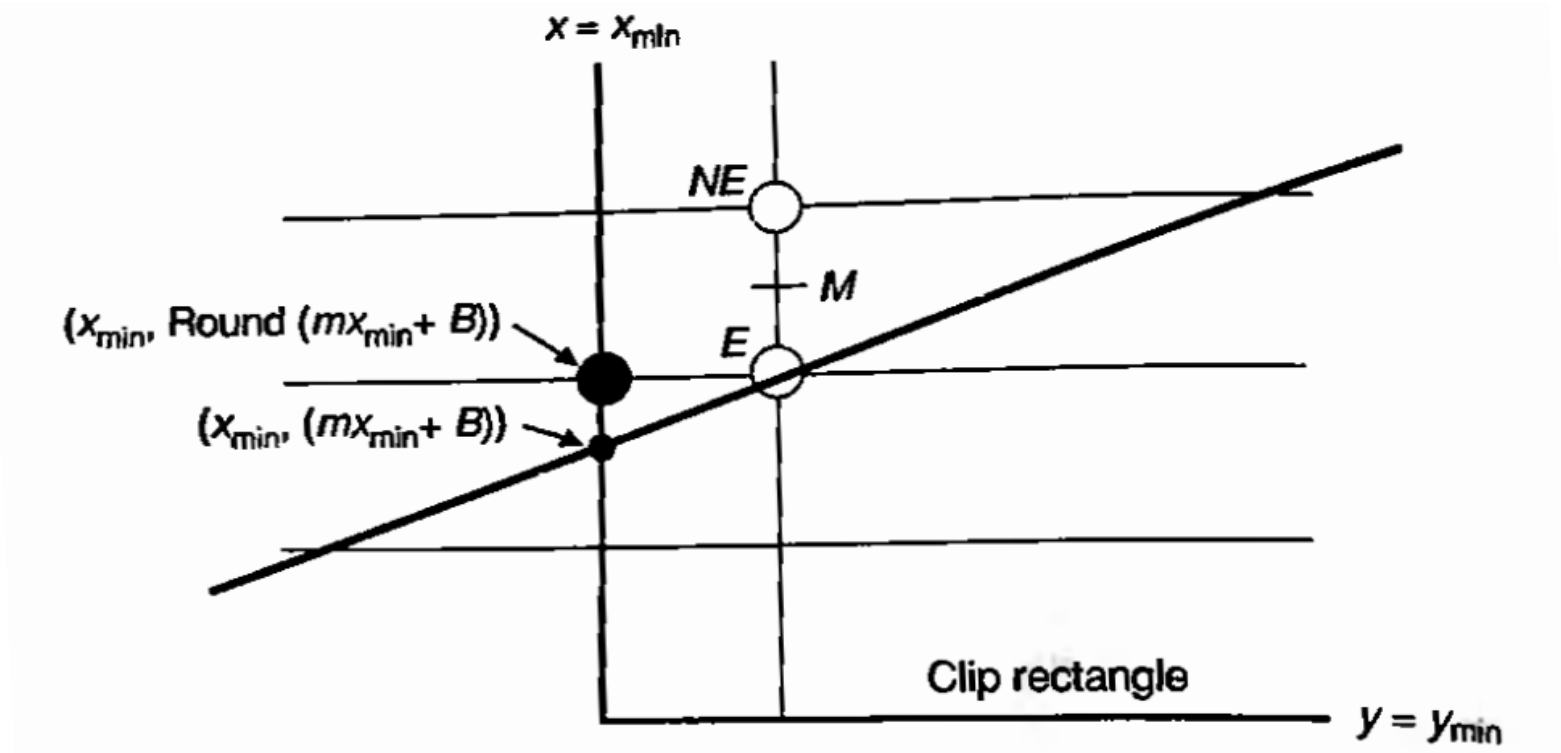
- úsečka $[x1, y1] - [x2, y2]$ může po nakreslení vypadat jinak než úsečka $[x2, y2] - [x1, y1]$
- řešení: seřadit body aby např. vždy $x2 > x1$.
- u přerušovaných čar složitější



Rasterizace úsečky

Ořezávání úsečky: výchozí bod na hraně ořezávacího okna (*viewport*)

- jedna ze souřadnic bodu na hranici není celočíselná

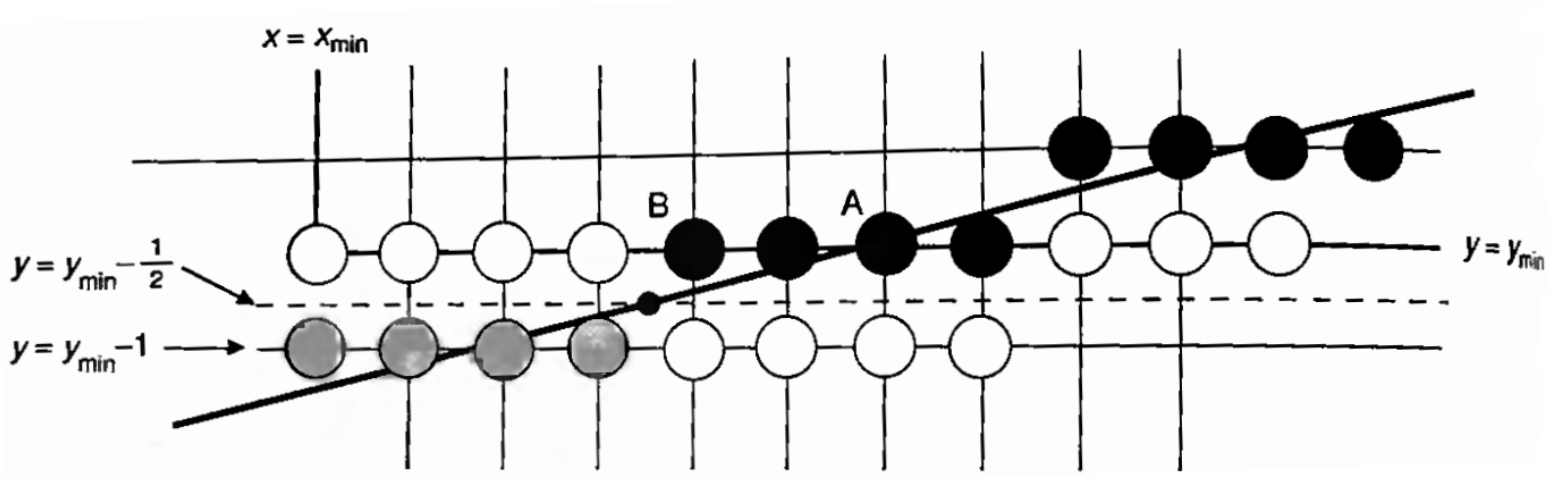


- řešení: nastavit správně počáteční chybu $\tilde{\epsilon}_0 \neq 0$, jinak získáme úsečku s jiným sklonem

Rasterizace úsečky

Ořezávání úsečky: výchozí bod na hraně ořezávacího okna (*viewport*)

- jedna ze souřadnic bodu na hranici není celočíselná



- řešení: najít průsečík s přímkou $y = y_{min} - \frac{1}{2}$, zaokrouhlit x nahoru. Jinak bude přímka vykreslena až od pixelu A, nikoliv od B

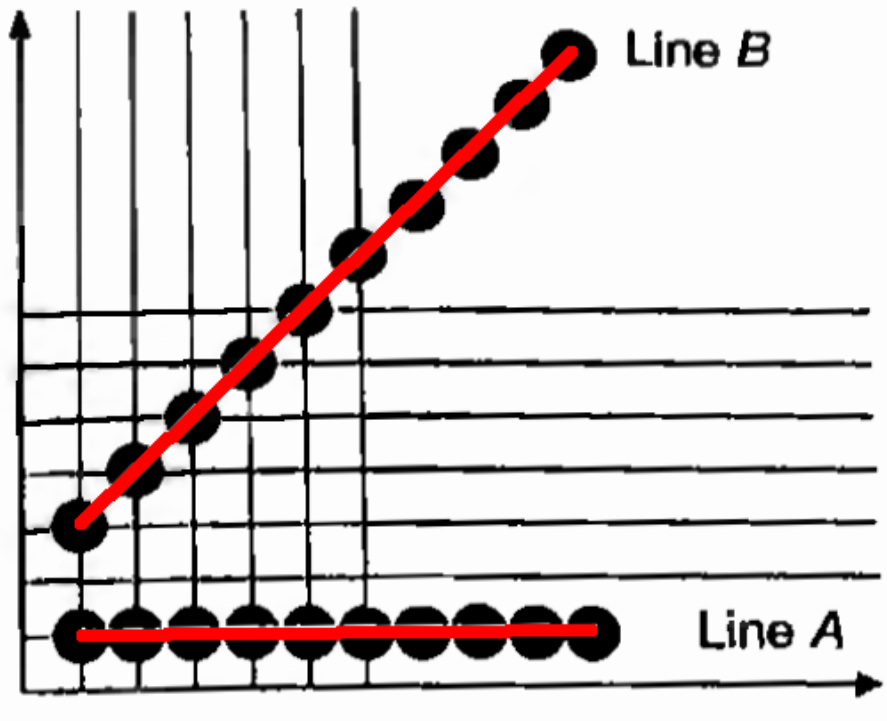
Rasterizace úsečky

Závislost intenzity barvy na sklonu úsečky

- při stejném počtu pixelů je úsečka se sklonem 45° $\sqrt{2}$-krát delší než vodorovná nebo svislá úsečka

$\Longrightarrow$ např. černá úsečka na bílém pozadí $\sqrt{2}$-krát světlejší

- řešení:
  - určit jinou barvu (sytost) bodů v závislosti na sklonu
  - použít některou z metod antialiasingu (viz dále)



Kresba přerušované úsečky

- bitová maska `M` o rozsahu 32 bitů, resp. pole
- řádek

```
PutPixel(x,y,c);
```

nahradit v případě bitové masky

```
if(M & (1 << (x%32)) )
 PutPixel(x,y,c);
```

resp. v případě pole délky `l`

```
if(M[x % l])
 PutPixel(x,y,c);
```

- délka segmentů závisí na sklonu, lepší je rozdělit úsečku podle euklidovské vzdálenosti, rasterizovat jednotlivé segmenty
- posunout tak, aby se nezačínalo ani nekončilo mezerou

Rasterizace kružnice

- střed $[x_S, y_S]$, poloměr R:

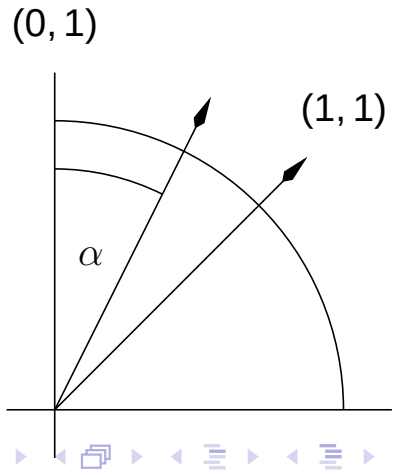
$$x^2 + y^2 = R^2$$

BÚNO předpokládejme $[x_S, y_S] = [0,0]$ (jen posunutí)

- středová symetrie $\Longrightarrow$ počítáme jen 1 **oktant**
  - řídící osa x, jeden pixel ve sloupci pro každé x_i
  - napočítáváme souřadnice y_i v oktantu vymezeném vektory $(0,1)$ a $(1,1)$ (sever až severo-východ)
  - ostatních 7 pixelů lze zrcadlit (resp. 3 pixely na souř. osách)

neefektivní metody:

- přímý výpočet $y_i = \sqrt{R^2 - x_i^2}$
- výpočet $x_i = R\sin\alpha_i$, $y_i = R\cos\alpha_i$, $\alpha_i \in \left[0, \frac{\pi}{4}\right]$



Rasterizace kružnice

Metoda středového bodu 1/4

- *midpoint algorithm*
- zobecněný Bresenhamův algoritmus, lze aplikovat na všechny kuželosečky
- označme

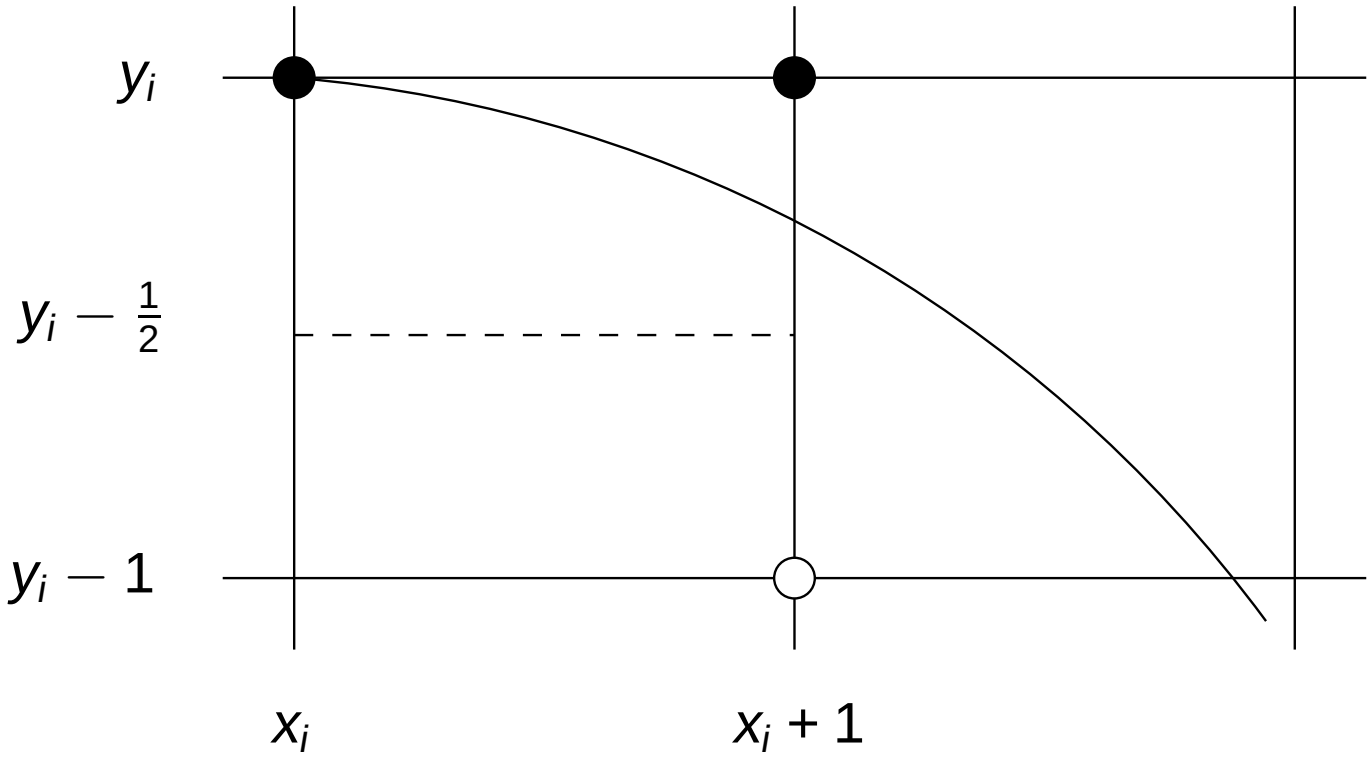
$$F(x,y) = x^2 + y^2 - R^2,$$

potom $F(x,y) < 0$, resp. $= 0$, resp. $> 0 \implies [x,y]$ leží **uvnitř**, resp. **na**, resp. **vně** kruhu

- současný *pixel* $[x_i, y_i]$, další bude buď $[x_i + 1, y_i]$, nebo $[x_i + 1, y_i - 1]$
- vypočítáme tzv. predikci $d_i = F\left(x_i + 1, y_i - \frac{1}{2}\right)$

Rasterizace kružnice

Metoda středového bodu 2/4



- pokud bod $\left[x_i + 1, y_i - \frac{1}{2}\right]$ leží uvnitř kružnice ($d_i < 0$), nakreslíme $[x_i + 1, y_i]$, jinak $[x_i + 1, y_i - 1]$

Rasterizace kružnice

Metoda středového bodu 3/4

Rozhodovací test metody středového bodu pro kružnici

- $d_i \leq 0 \implies$ vykreslíme bod $[x_i + 1, y_i]$, tj.

$$\begin{aligned} y_{i+1} &= y_i \\ d_{i+1} &= F\left(x_{i+1} + 1, y_{i+1} - \frac{1}{2}\right) = F\left(x_i + 2, y_i - \frac{1}{2}\right) \\ &= d_i + 2x_i + 3 \end{aligned}$$

- $d_i > 0 \implies$ vykreslíme bod $[x_i + 1, y_i - 1]$, tj.

$$\begin{aligned} y_{i+1} &= y_i - 1 \\ d_{i+1} &= F\left(x_{i+1} + 1, y_{i+1} - \frac{1}{2}\right) = F\left(x_i + 2, y_i - \frac{3}{2}\right) \\ &= d_i + 2x_i - 2y_i + 3 \end{aligned}$$

Rasterizace kružnice

Metoda středového bodu 4/4

- shrnutí:

$$d_{i+1} = \begin{cases} d_i + 2x_i + 3 & d_i \leq 0 \\ d_i + 2x_i - 2y_i + 3 & d_i > 0 \end{cases}$$

- všechny operace by byly celočíselné, kdyby $d_0 \in \mathbb{Z}$. Bohužel

$$d_0 = F\left(1, R - \frac{1}{2}\right) = \frac{5}{4} - R$$

- definujeme $h_i = d_i - \frac{1}{4}$. Potom už $(\forall i)\,(h_i \in \mathbb{Z})$ a test přejde v

$$h_i \gtreqless -\frac{1}{4}$$

Protože ale $h_i \in \mathbb{Z}$, tento test je ekvivalentní s

$$h_i \gtreqless 0$$

Rasterizace elipsy

- rovnice elipsy se středem v bodě $[0,0]$ a poloosami a, b rovnoběžnými s osami x, y

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

resp.

$$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

- obecná elipsa je složitější
- vykreslování opět pomocí metody středového bodu
- vykreslování v celém kvadrantu (řídící osa se obecně **nemění** na hranici oktantů - na přímce $y = x$ jako u kružnice)

Rasterizace elipsy

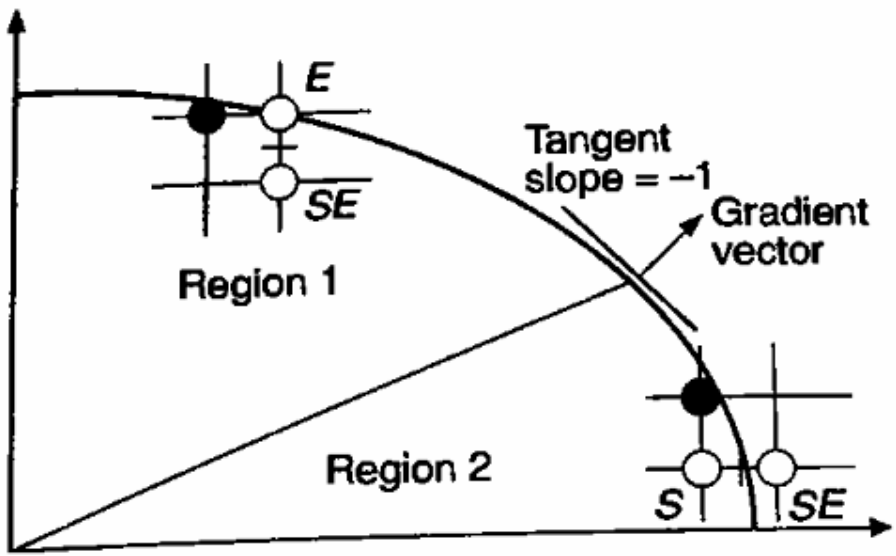
Bod změny řídící osy

- určíme gradient F

$$\nabla F(x,y) = \begin{pmatrix} \partial_x F(x,y) \\ \partial_y F(x,y) \end{pmatrix} = \begin{pmatrix} 2b^2 x \\ 2a^2 y \end{pmatrix}$$

- v oblasti, kde $2b^2 x \leq 2a^2 y$, je řídící osou osa x, v oblasti $2b^2 x > 2a^2 y$ je řídící osa y
- začneme s řídící osou x, přepneme se na y, když pro následující střední bod na souřadnici $x_{i+1} = x_i + 1$ platí

$$a^2\left(y_i - \frac{1}{2}\right) \leq b^2 (x_i + 1)$$



Rasterizace elipsy

Metoda středového bodu 1/2

- současný pixel $[x_i, y_i]$, predikce opět $d_i = F\left(x_i + 1, y_i - \frac{1}{2}\right)$

Rozhodovací test metody střed. bodu pro elipsu, řídící osa x

- $d_i \leq 0 \implies$ vykreslíme bod $[x_i + 1, y_i]$, tj. $y_{i+1} = y_i$ a

$$\begin{aligned} d_{i+1} &= F\left(x_{i+1} + 1, y_{i+1} - \frac{1}{2}\right) = F\left(x_i + 2, y_i - \frac{1}{2}\right) \\ &= d_i + b^2\,(2x_i + 3) \end{aligned}$$

- $d_i > 0 \implies$ vykreslíme bod $[x_i + 1, y_i - 1]$, tj. $y_{i+1} = y_i - 1$ a

$$\begin{aligned} d_{i+1} &= F\left(x_{i+1} + 1, y_{i+1} - \frac{1}{2}\right) = F\left(x_i + 2, y_i - \frac{3}{2}\right) \\ &= d_i + b^2\,(2x_i + 3) + a^2\,(-2y_i + 2) \end{aligned}$$

Rasterizace elipsy

Metoda středového bodu 2/2

- v každé iteraci je kromě d_i třeba testovat přepnutí do oblasti s řídící osou y
- potom provedeme záměnu x a y, pohybujeme se stále směrem **dolů a doprava**
  - $y_{i+1} = y_i - 1$
  - predikce bude mít tvar

$$d_i = F\left(x_i + \frac{1}{2}, y_i - 1\right)$$

  - pro y_{i+1} kreslíme bod buď $[x_i + 1, y_i - 1]$ pro $d_i \leq 0$, nebo $[x_i, y_i - 1]$ pro $d > 0$
  - aktualizace d_i se odvodí analogicky dosazením do $F(\cdot, \cdot)$
- jestliže $a, b \in \mathbb{N}$, potom lze algoritmus opět formulovat v celočíselné aritmetice

Obsah

Kresba silné čáry 1/2

- místo jednoho pixelu kreslit t pixelů **nad** sebou (sklon $\leq 45°$), resp. vedle sebe (sklon $> 45°$)
- t... tloušťka, *thickness*
- skutečná tloušťka závisí na sklonu
- zakončení čar je nepřirozené (vždy svislé, resp. vodorovné pro sklon $> 45°$)
- mezery při napojování úseček s různými řídícími osami
- výhoda: rychlý algoritmus

Kresba silné čáry 2/2

Vyplňování plochy mezi hranicemi

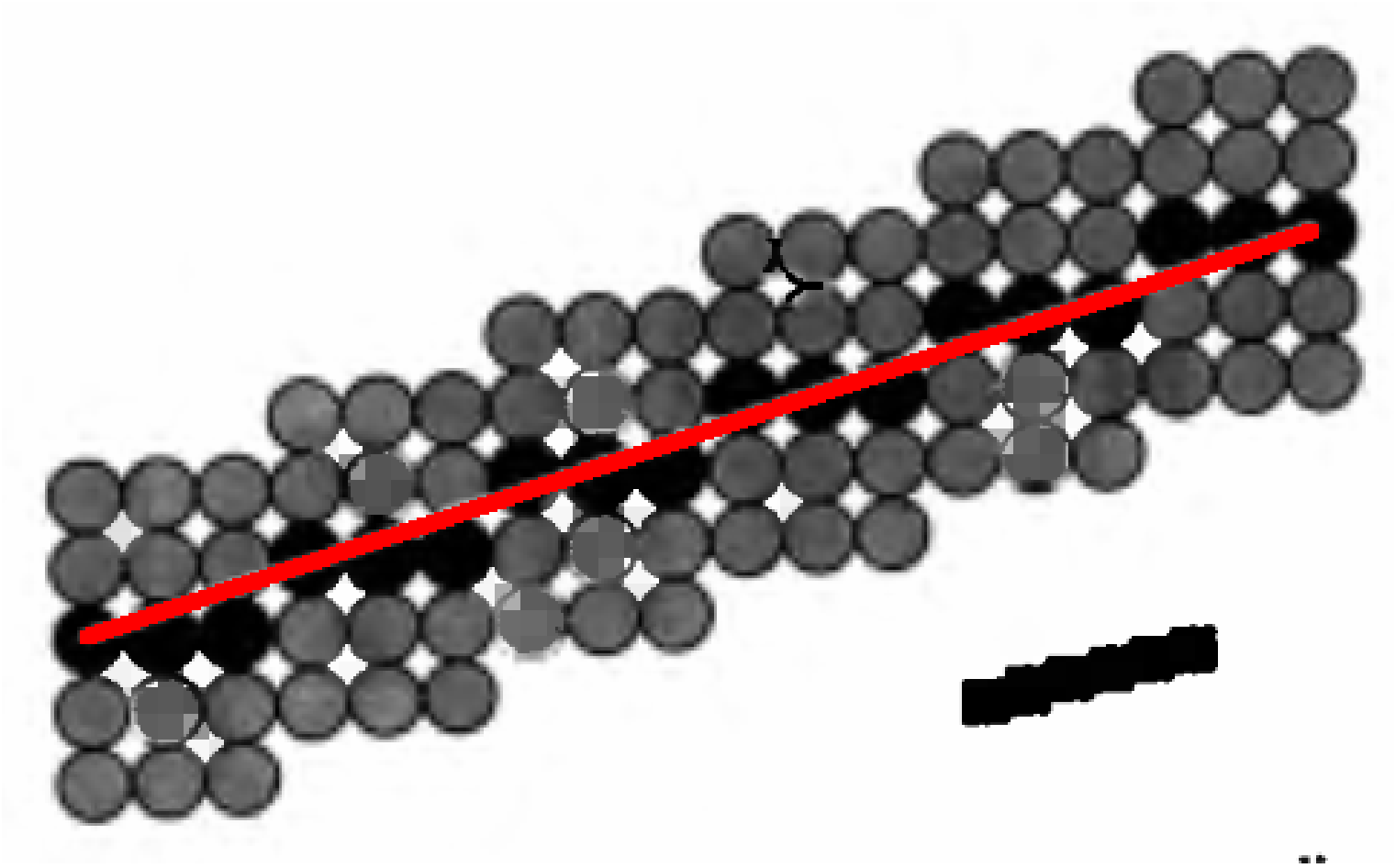
- silná čára jako obdélník nebo jiný uzavřený útvar, v závislosti na zakončení čáry
- kružnice - vyplnění mezikruží (lze i u elips)

Kresba pomocí obdélníkového pera

- místo 1 pixelu kreslíme obdélník
- v každé pozici kreslíme jen body, které jsme ještě nenakreslili

Kresba silné čáry

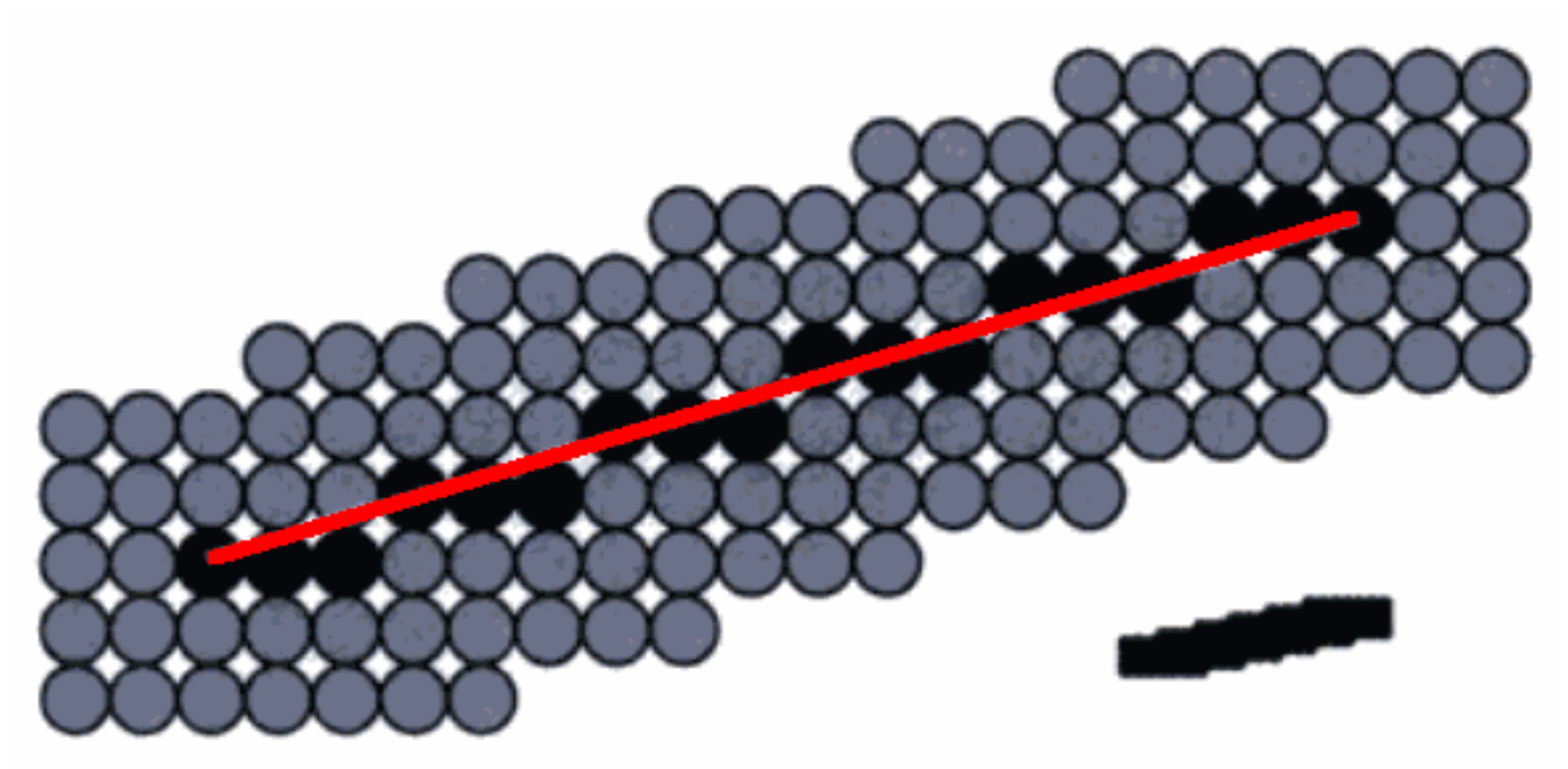
Srovnání



Kreslení čáry opakováním ve sloupci

Kresba silné čáry

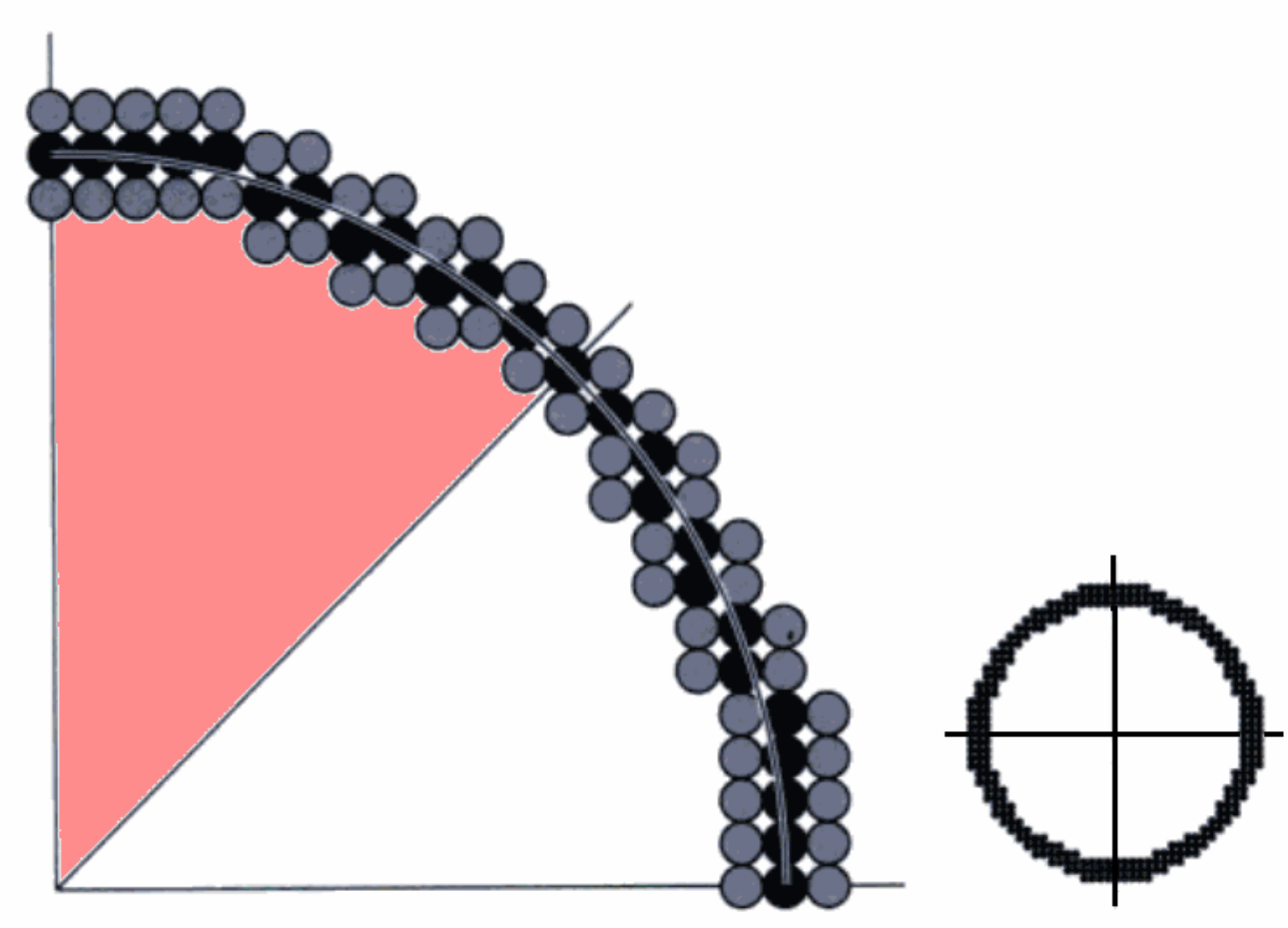
Srovnání



Kreslení čáry obdélníkovým perem

Kresba silné čáry

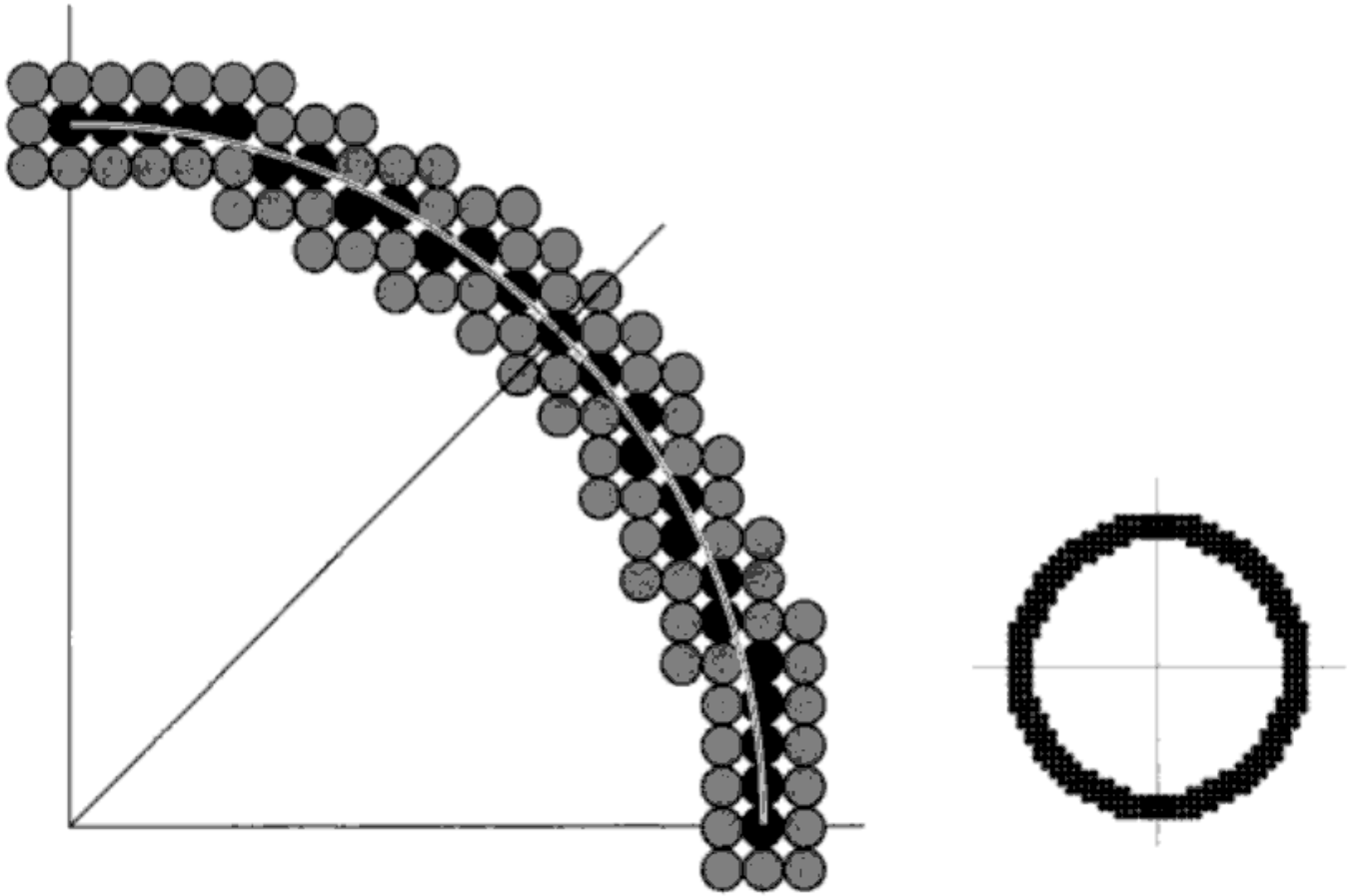
Srovnání



Keslení kružnice opakováním ve sloupci (v 1 oktantu)

Kresba silné čáry

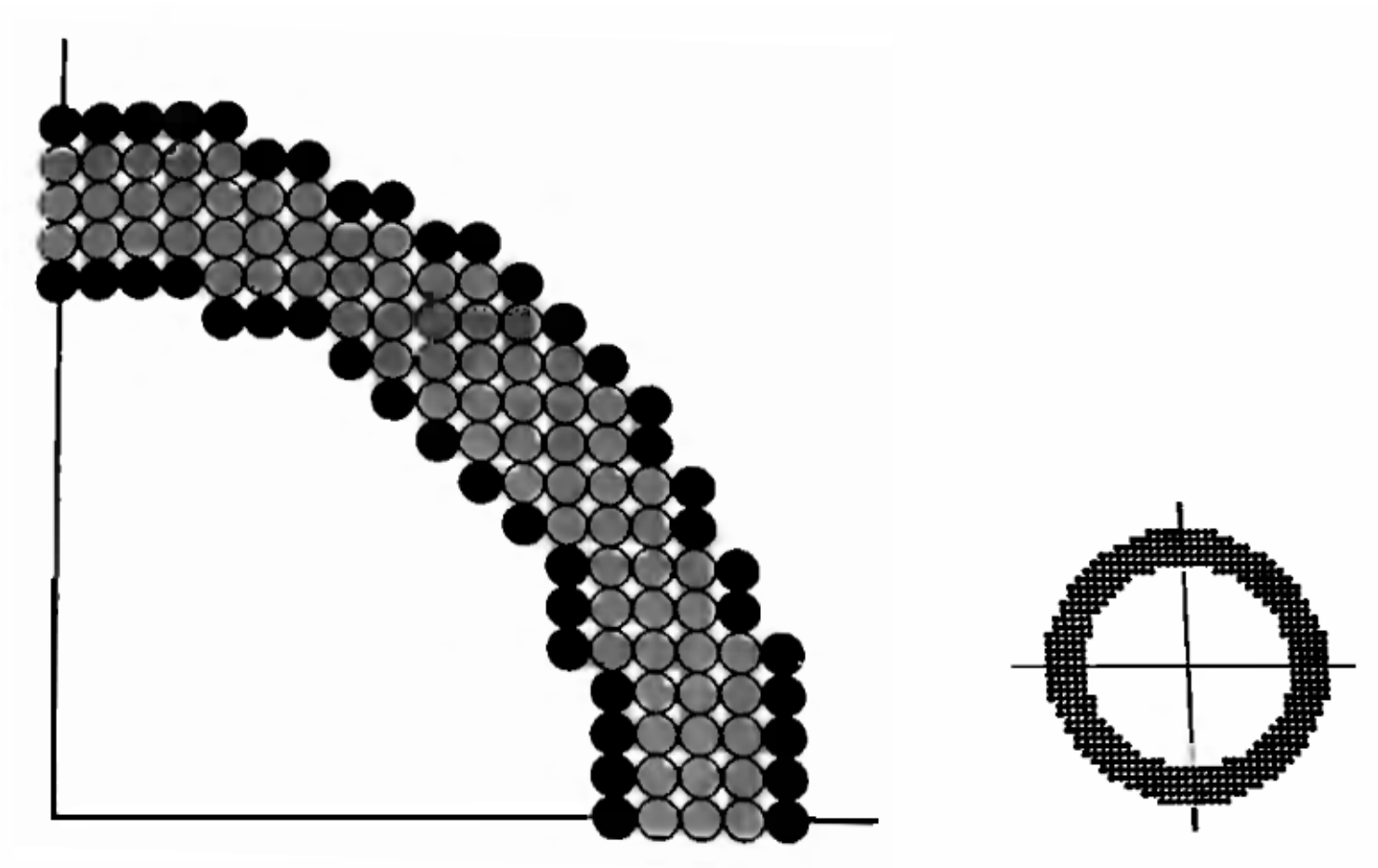
Srovnání



Kreslení kružnice obdélníkovým perem

Kresba silné čáry

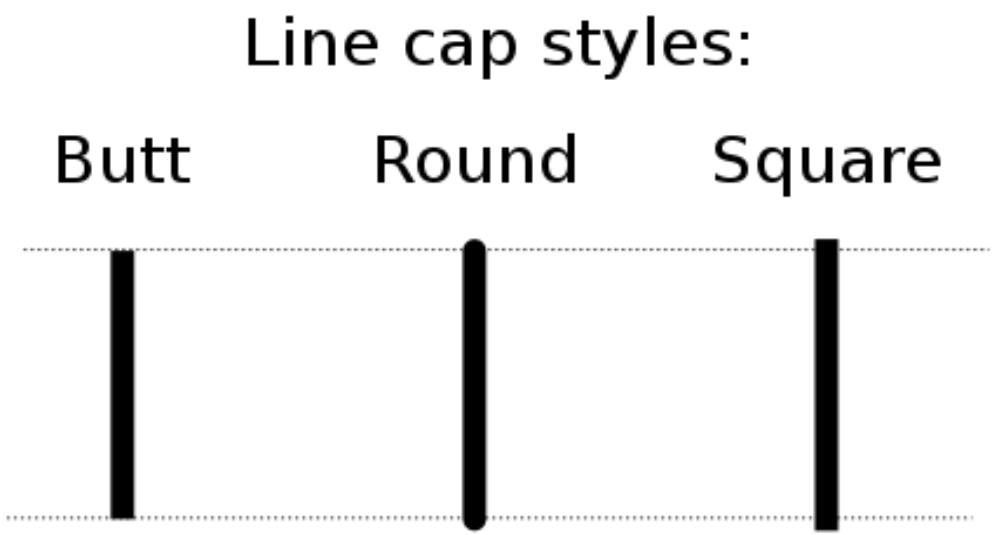
Srovnání



Kreslení kružnice vyplněním mezikruží

Kresba silné čáry

Zakončení silné čáry



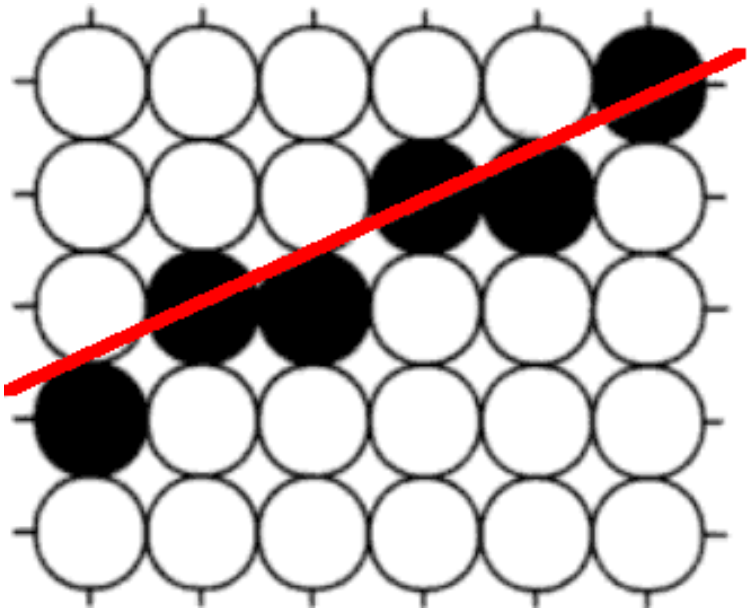
Antialiasing

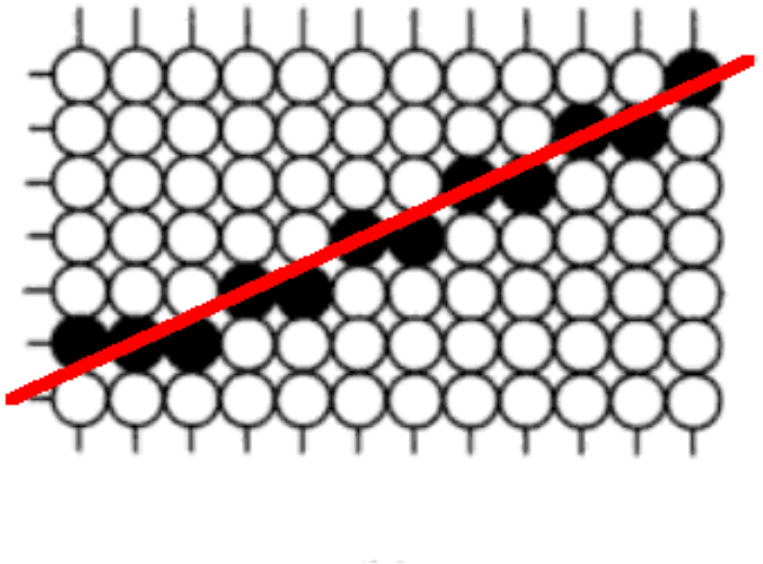
- dosud - úsečka jako množina pixelů - pixel buď vykreslíme, nebo ne
  $\Longrightarrow$ jsou vidět „schody“ (*jaggies*, *staircasing*)
- jde o **alias**(**ing**) - nežádoucí a nereálný efekt vzniklý nedostatečnou vzorkovací frekvencí (časovou, prostorovou) signálu
- v našem případě
  - **signál**: obraz, konkrétně úsečka. Je nekonečně tenká, její spektrum obsahuje všechny frekvence
  - **vzorkovací frekvence**: rozlišení (počet pixelů na jednotku vzdálenosti)
  - viz letní semestr - teorie signálu
- **antialiasing** - procedura vedoucí k potlačení aliasu

Antialiasing

Zvýšení rozlišení

- zuby se zmenší, ale máme větší nároky na paměť, dobu vykreslování

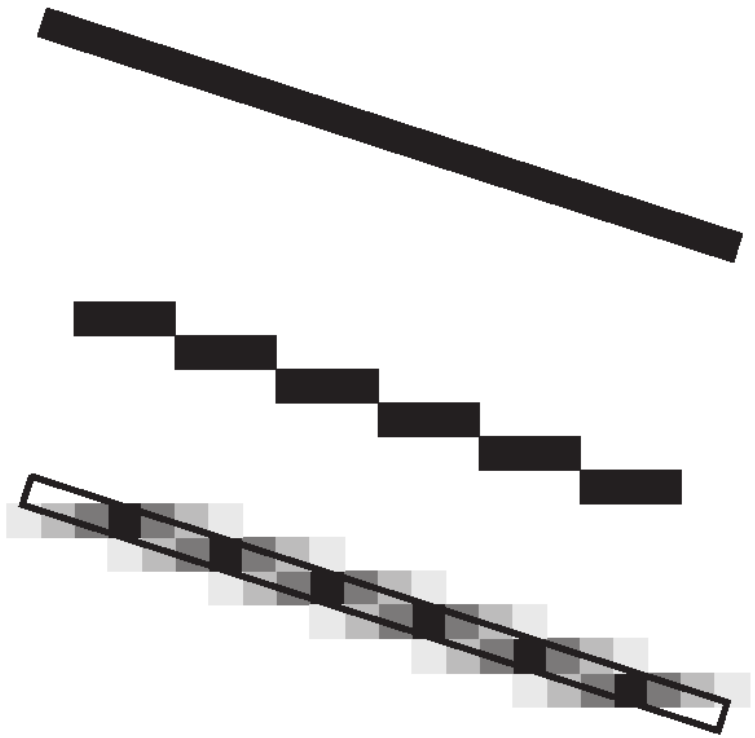




Antialiasing

Unweighted area sampling

- vodorovná nebo svislá čára má tloušťku 1 pixel
- místo nekonečně tenké čáry tedy uvažujeme čáru o konečné tloušťce 1 pixel
- pixel považujeme za čtverec



černá čára na bílém pozadí

- vykreslíme **všechny pixely, které čára alespoň částečně překrývá**, s intenzitou (resp. tmavostí) úměrnou podílu překryté plochy pixelu
- kreslení na jiné pozadí - průhlednost, alfa míchání (*alpha blending*) - viz transformace obrazu

Antialiasing

Weighted area sampling 1/2

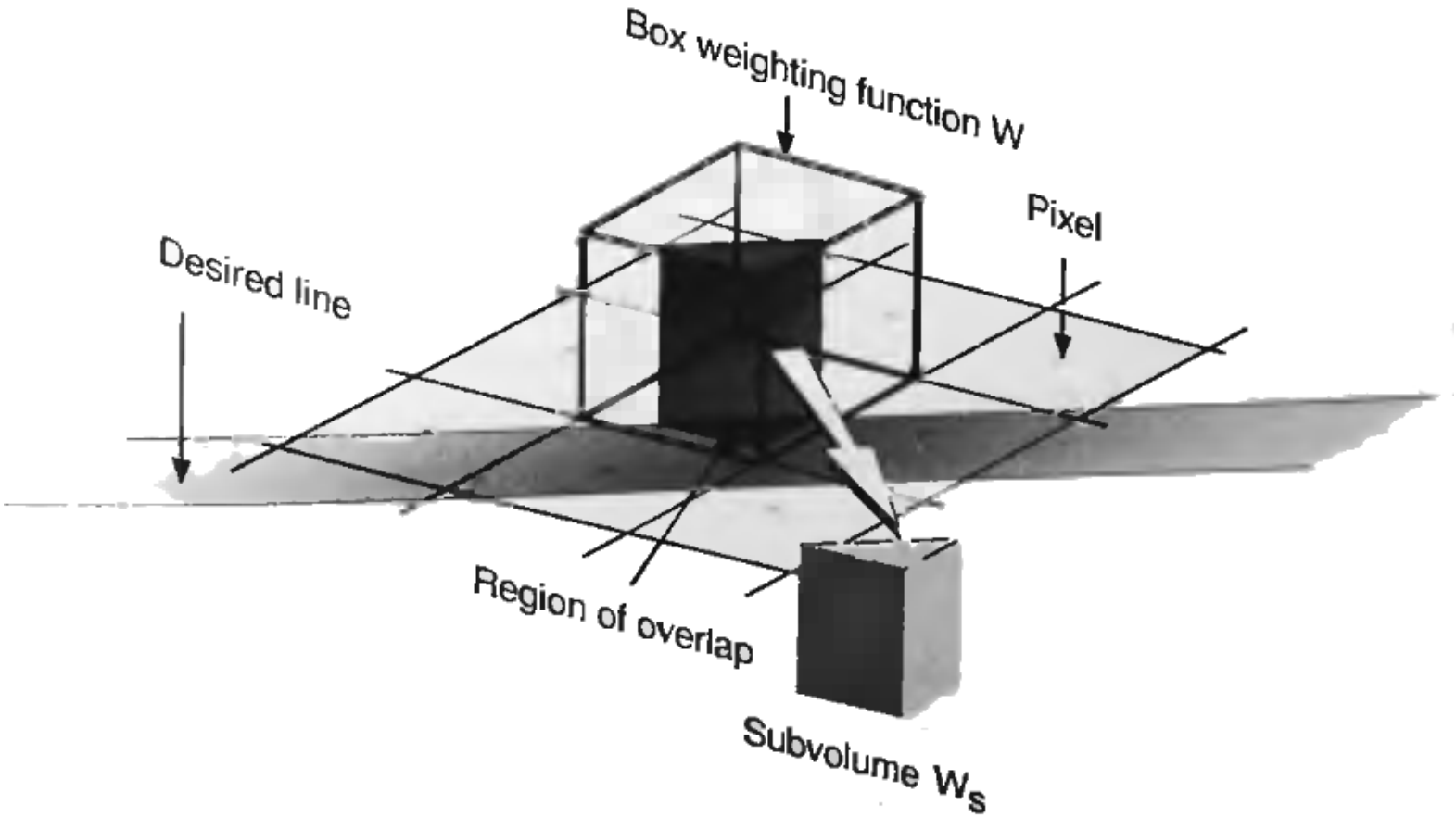
- dosud byla barva závislá na míře překrytí, ale nikoliv na pozici (překrytí přes střed, přes okraj pixelu)
- ještě lepší výsledky, když míra ovlivnění barvy pixelu klesá se vzdálenosti překryté oblasti od středu
  $\Longrightarrow$ pixel $[i,j]$: **vážení** (*weighting*) vhodně zvolenou funkcí $w_{i,j}(x,y)$ s maximem v $[i,j]$
- intenzita objektu I_o, charakteristická funkce objektu (čáry) $\chi(x,y)$ $\Longrightarrow$ intenzita pixelu $[i,j]$ bude

$$I(i,j) = I_o \int_{j-\frac{1}{2}}^{j+\frac{1}{2}} \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} w_{i,j}(x,y)\chi(x,y)\,\mathrm{d}x\mathrm{d}y$$

- unweighted area sampling: $w_{i,j}(x,y) = 1$

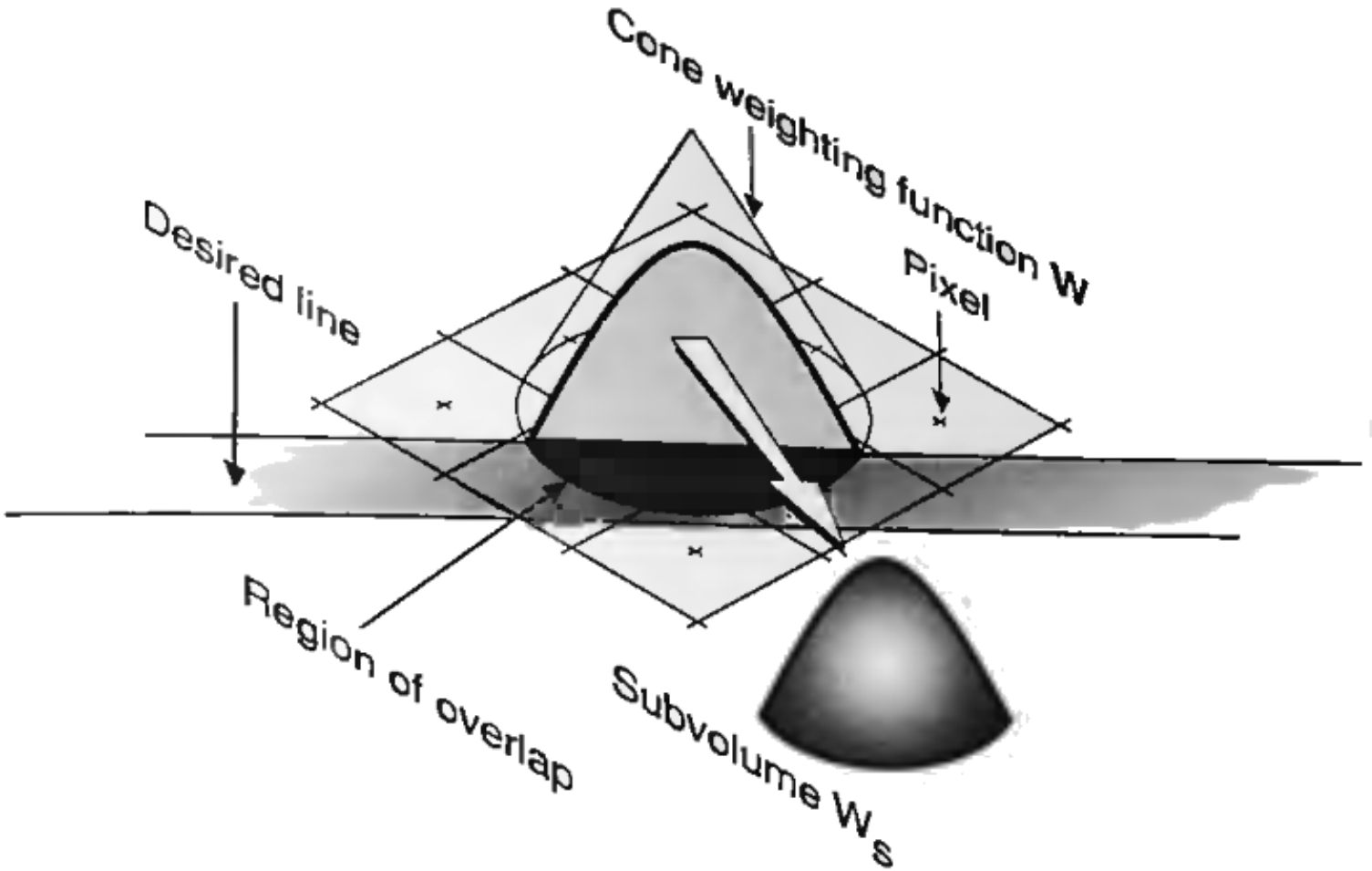
Antialiasing

Weighted area sampling 2/2



Antialiasing

Weighted area sampling 2/2



Antialiasing

Jednoduchý antialiasing pro Bresenhamův algoritmus

- předpokládejme řídící osu x
- využijeme chybu ϵ z Bresenhamova algoritmu
- kreslíme vždy 2 pixely nad sebou: vypočítaný pixel a pixel **pod**, resp. **nad** ním, podle toho, zda je $\epsilon > \frac{1}{2}$, resp. $\epsilon < \frac{1}{2}$
- pixel blíže k přímce kreslíme s intenzitou úměrnou $1 - \epsilon$, druhý s intenzitou ϵ (součet dá 1)

Obsah

Ořezávání bodů a úseček

- omezíme se na ořezávání obdélníkovou oblastí (viewport)
- úsečka - stačí zkoumat polohy **koncových bodů** vůči obdélníku

Ořezávání bodu o souřadnicích $[x, y]$

Bod $[x, y]$ vykreslíme $\Longleftrightarrow x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max}$

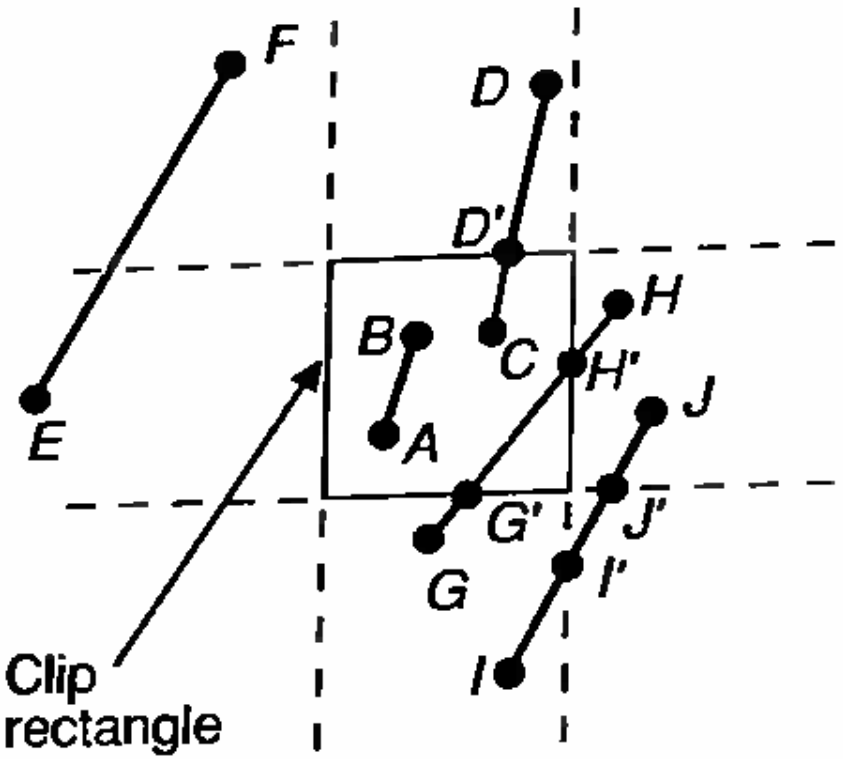
Algoritmy pro ořezávání úsečky:

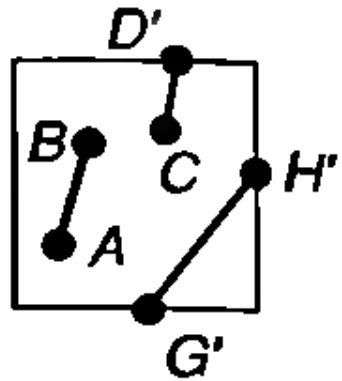
- ořezávání hrubou silou
- algoritmus Cohenův-Sutherlandův
- algoritmus Cyruse a Becka
- algoritmus Lianga a Barského
- Nicholl-Lee-Nichollův algoritmus
- atd.

Ořezávání úseček

Ořezávání hrubou silou

- najdeme průsečíky úsečky s přímkami ohraničujícími ořezávací okno
- testujeme, zda průsečíky leží na hranici okna (D', G', H'), nebo mimo (I', J')

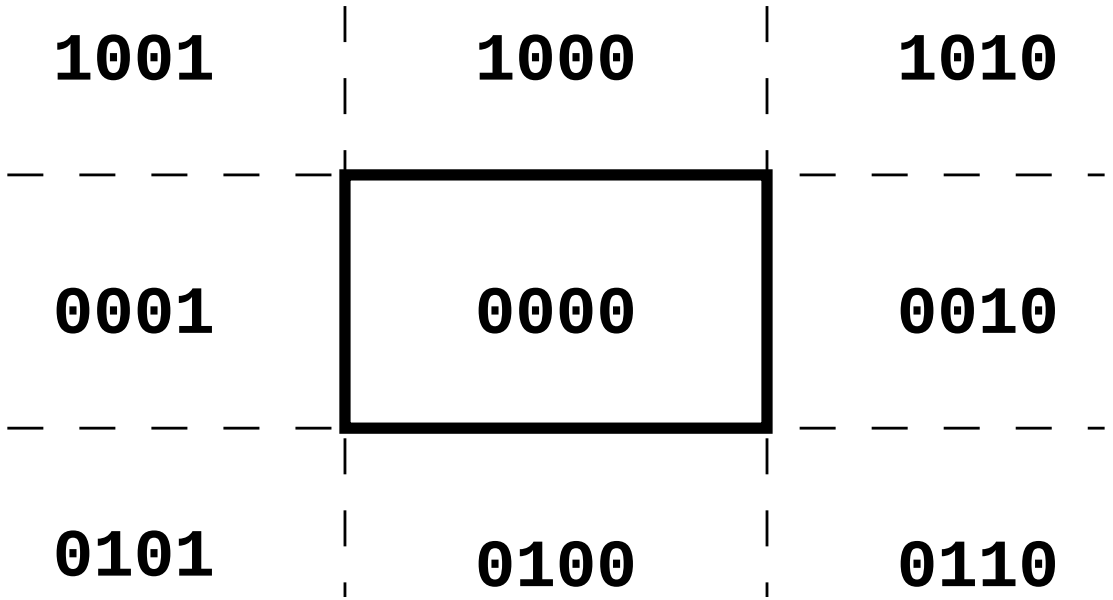




Ořezávání úseček

Cohenův-Sutherlandův algoritmus 1/4

- oblasti roviny oddělené stranami ořezávacího okna ohodnoceny 4-bitovými kódy
- `ABRL` - Above, Below, Right, Left



Ořezávání úseček

Cohenův-Sutherlandův algoritmus 2/4

- koncovým bodům úsečky $\overline{P_0P_1}$ se přiřadí bitové kódy p_0, p_1 podle pozice vzhledem k oknu
- efektivní přiřazení: znaménkový bit rozdílů (po řadě `ABRL`)

$$y_{max} - y,\ y - y_{min},\ x_{max} - x,\ x - x_{min}$$

- potom platí:
  1. p_0 or $p_1 = 0 \implies$ úsečka je **celá v okně** (*triviální přijetí*)
  2. p_0 and $p_1 \neq 0 \implies$ úsečka je **celá mimo okno** (*triviální zamítnutí*)
  3. jinak $\implies$ úsečka prochází několika oblastmi
- v posledním případě zkoumáme dál - úsečku zkrátíme

Ořezávání úseček

Cohenův-Sutherlandův algoritmus 3/4

Zkrácení úsečky:

1. vybereme koncový bod P_{out} ležící mimo okno (mohou být oba $\Longrightarrow$ zvolíme jeden)
2. podle bitového kódu p_{out} příslušného P_{out} zvolíme přímku definující okno, která dělí bod P_{out} od vnitřku okna

Příklad

- $p_{out} = 1001$ (= nahoře vlevo)
- vypočítáme průsečík s horní nebo levou přímkou (přímkou $y = y_{max}$ nebo $x = x_{min}$), nikoliv spodní nebo pravou

3. bod P_{out} nahradíme získaným průsečíkem

$\Longrightarrow$ alespoň 1 souřadnice nového bodu již leží ve správném rozsahu (na hranici okna). **Opakujeme algoritmus**, dokud úsečka není **celá v okně**.

Ořezávání úseček

Cohenův-Sutherlandův algoritmus 4/4

- snadno rozšiřitelné do 3D - ořezávání kvádrem
- průsečík přímky $y = ax + b$ s obecnou osou $x = x_0$:

$$[x_0, ax_0 + b]$$

- průsečík přímky $y = ax + b$, $a \neq 0$ s obecnou osou $y = y_0$:

$$\left[\frac{y_0 - b}{a}, y_0\right]$$

- (zbytečně) vypočítáváme souřadnice i těch koncových bodů, které v dané iteraci stále leží mimo okno

Ořezávání úseček

Algoritmus Cyruse a Becka 1/4

- rychlejší než Cohenův-Sutherlandův algoritmus pokud většinu přímek nelze ošetřit triviálním přijmutím nebo zamítnutím
- **parametrický** algoritmus
- parametrická rovnice přímky P dané koncovými body P_0, P_1

$$P(t) = P_0 + (P_1 - P_0)\,t$$

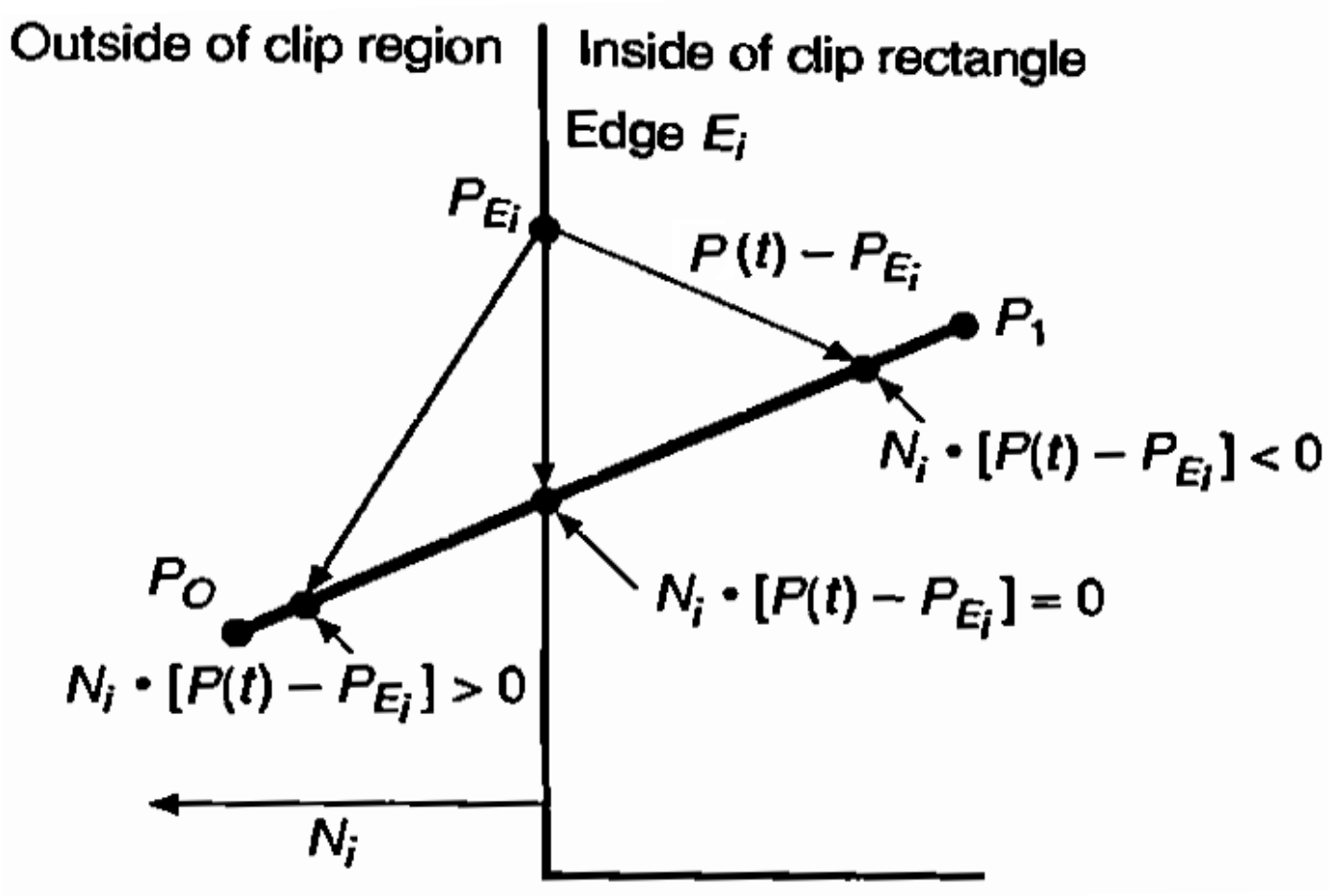
- nevypočítává průsečíky, ale pouze parametr t odpovídající průsečíkům se 4 osami
- ořezávací okno: obecný konvexní polygon, definovaný průnikem polorovin (přímek s orientovanou normálou)
- dále uvažujeme opět pouze obdélník
- i-tá přímka E_i má normálu $\boldsymbol{N}_i$ směřující **ven** z okna

Ořezávání úseček

Algoritmus Cyruse a Becka 2/4

určení hodnoty t_i průsečíku $P \cup E_i$, tj. tak, aby $P(t_i) \in E_i$, $\forall i$:

- zvolíme libovolný bod $P_{E_i} \in E_i$



Ořezávání úseček

Algoritmus Cyruse a Becka 2/4

určení hodnoty t_i průsečíku $P \cup E_i$, tj. tak, aby $P(t_i) \in E_i$, $\forall i$:

- zvolíme libovolný bod $P_{E_i} \in E_i$
- vyřešíme rovnici

$$\begin{aligned} \boldsymbol{N}_i \cdot \left(P(t_i) - P_{E_i}\right) &= 0 \\ \boldsymbol{N}_i \cdot \left(P_0 + (P_1 - P_0)\, t_i - P_{E_i}\right) &= 0 \\ \boldsymbol{N}_i \cdot \left(P_0 - P_{E_i}\right) + \boldsymbol{N}_i \cdot (P_1 - P_0)\, t_i &= 0 \\ \frac{\boldsymbol{N}_i \cdot \left(P_0 - P_{E_i}\right)}{-\boldsymbol{N}_i \cdot \boldsymbol{D}} &= t_i \end{aligned}$$

  kde $\boldsymbol{D} = (P_1 - P_0)$
- kontrolujeme, zda $\boldsymbol{N}_i \cdot \boldsymbol{D} \neq 0$, tj. zda neplatí $E_i \parallel P$. V opačném případě průsečík není a jdeme na další i.

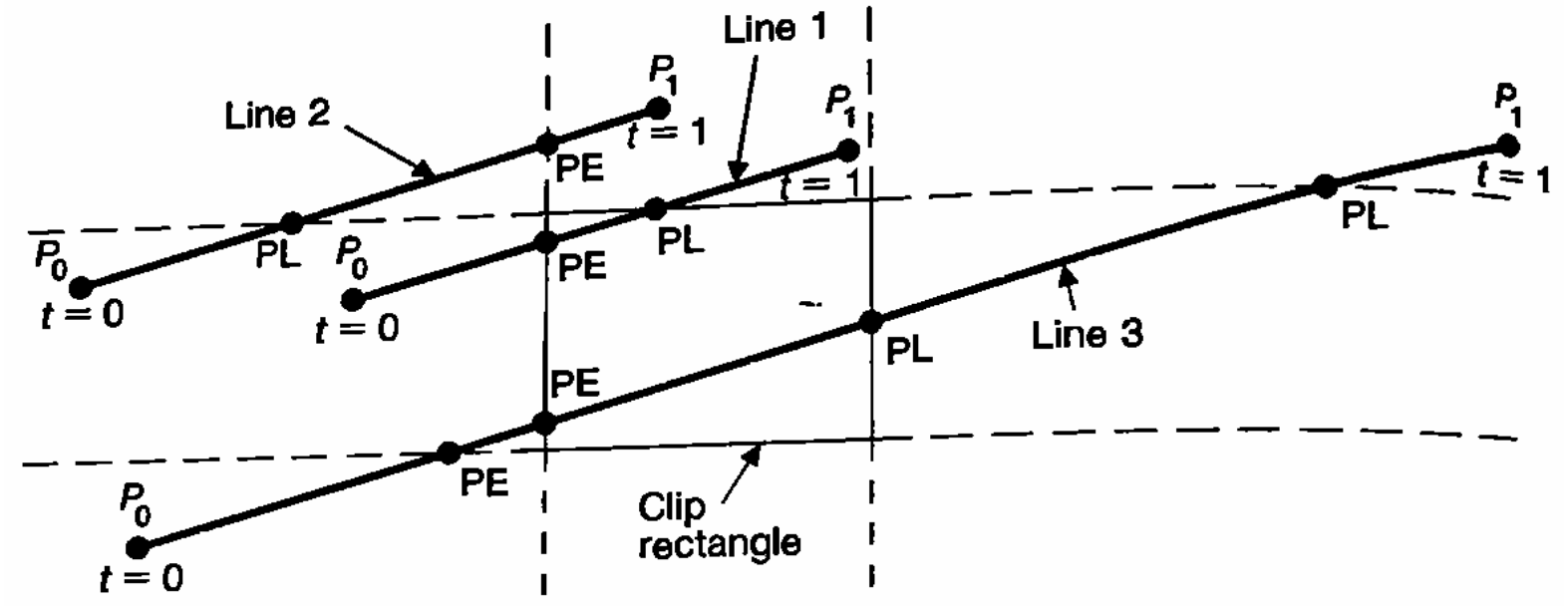
Ořezávání úseček

Algoritmus Cyruse a Becka 3/4

- nalezené hodnoty parametrů t_i průsečíků seřadíme vzestupně (jak jdou na přímce od P_0 k P_1)
- pokud ve směru od P_0 k P_1 procházíme přes průsečík s E_i:
  - do vnitřní poloroviny dané $E_i \implies$ průsečík označen jako **potenciálně vstupující** (PE, *potentially entering*) do ořezávacího okna
  - do vnější poloroviny dané $E_i \implies$ průsečík označen jako **potenciálně vystupující** (PL, *potentially leaving*)
- směr PE nebo PL rozlišíme podle úhlu vektoru $\boldsymbol{D} = P_1 - P_0$ s normálou $\boldsymbol{N}_i$
  - PE $\iff \measuredangle \boldsymbol{D}\boldsymbol{N}_i > 90° \iff \boldsymbol{N}_i \cdot \boldsymbol{D} < 0$
    (skalární součin počítáme při výpočtu t_i)
  - PL $\iff \measuredangle \boldsymbol{D}\boldsymbol{N}_i < 90° \iff \boldsymbol{N}_i \cdot \boldsymbol{D} > 0$

Ořezávání úseček

Algoritmus Cyruse a Becka 4/4



- ozn. t_E jako **největší** t_i typu PE a t_L **nejmenší** t_i typu PL
- pokud $t_E > t_L$, úsečka oknem neprochází vůbec
  - právě toto platí pouze pro **konvexní ořezávací okno**
- v opačném případě úsek v okně je, a to pro interval parametrů $[\max(t_E, 0), \min(t_L, 1)]$
  (máme $t_i \in \mathbb{R}$, ale **původní úsečka** měla jen $t \in [0, 1]$)

Ořezávání úseček

Algoritmus Lianga a Barského

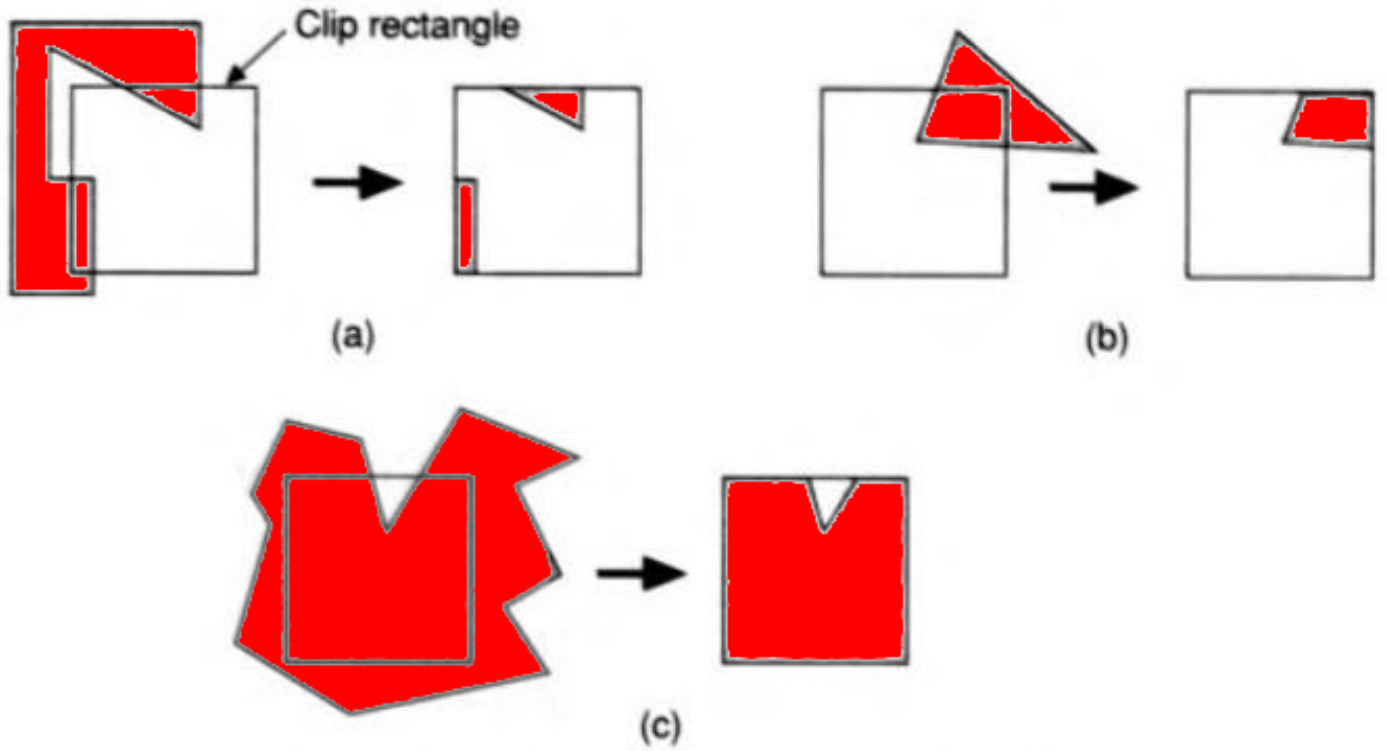
- modifikovaný algoritmus Cyruse a Becka
- obsahuje test na triviální zamítnutí v průběhu výpočtů t_i
  - leží-li úsečka zcela mimo, není nutné počítat všechna t_i
- oba algoritmy **lze zobecnit do 3D**
  - ořezávání konvexním mnohostěnem s poloprostory určenými rovinami (s danými normálami)
- pro případ **ořezávání obdélníkem orientovaným ve směru souř. os** mnoho **zjednodušení**
  - vždy jedna složka $\boldsymbol{N}_i$ rovna nule
  - není třeba dopočítávat 2 souřadnice P_{E_i}, stačí jedna
  - atd.

Ořezávání kružnice, elipsy

- testujeme průsečík **obalového čtverce** s ořezávacím oknem
- buď triviálně přijmeme, zamítneme, nebo rozdělíme na kvadranty
  - opět průsečík čtverce, v případě triviálního přijetí kreslíme 1/4 kružnice
- dále lze rozdělit na oktanty atd.
- v každém kvadrantu (oktantu), který nebyl triviálně vyřešen
  - analyticky vypočítáme průsečíky a kreslíme výsledné oblouky
  - nebo rovnou kreslíme a testujeme každý pixel
- **elipsa** - podobně (obalový obdélník)
- **vyplněný kruh** - kreslíme vodorovné čáry po řádcích, najdeme průsečíky těchto čar s oknem

Ořezávání polygonů

- mnoho různých případů
  - každou hranu nutno ořezávat zvlášť
  - nekonvexní polygon se může rozpadnout na více částí
  - je třeba odstraňovat, zachovávat, přidávat hrany a vrcholy



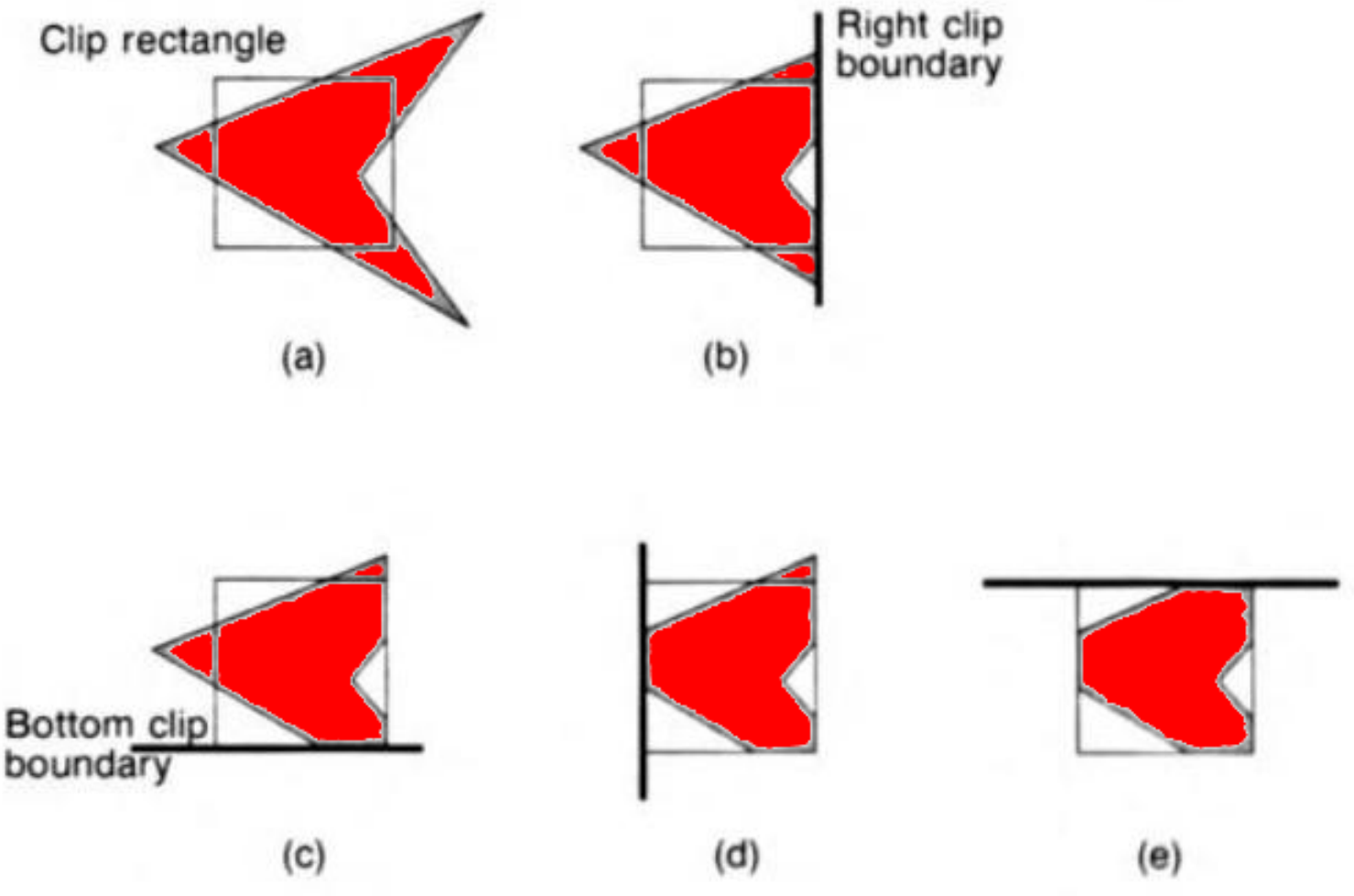
Ořezávání polygonů

Sutherlandův-Hodgmanův algoritmus 1/6

- **rozděl a panuj**
- zpracovává i nekonvexní polygony
- místo ořezávání obdélníkem ořezává postupně každou přímkou (polorovinou), která ho definuje
- obdélníkové okno lze zobecnit na **konvexní polygon**
- lze zobecnit do **3D** na nekonvexní mnohostěny (*polyhedra*), ořezávací objem je konvexní mnohostěn

Ořezávání polygonů

Sutherlandův-Hodgmanův algoritmus 2/6



Postupné ořezávání polorovinami

Ořezávání polygonů

Sutherlandův-Hodgmanův algoritmus 3/6

- vstup každé iterace algoritmu: uspořádaná n-tice vrcholů

$$v_1, v_2, \ldots, v_n$$

  hrany jsou mezi $v_k, v_{k+1}\ \forall k \in \{1, 2, \ldots, n-1\}$ a v_n, v_1
- výstup: uspořádaná m-tice vrcholů (obecně $m \neq n$)

$$w_1, w_2, \ldots, w_m$$

- polygon nemá díry, hrany se navíc nesmí křížit

Problém

- z definice je výsledný polygon formálně **opět jeden uzavřený obrazec**
- pokud ořezáním ve skutečnosti vznikne **více samostatných částí**, vedou mezi nimi hrany
- to obvykle nevadí nebo lze dodatečně odstranit (netriviální)

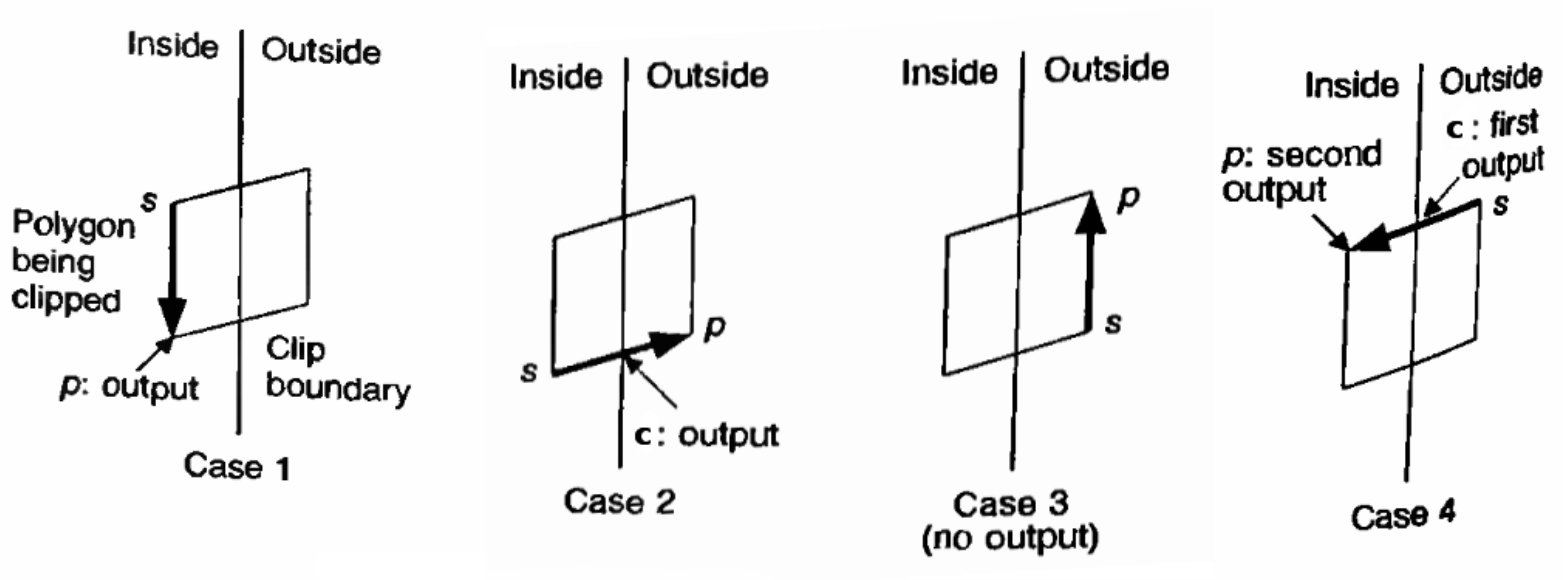
Ořezávání polygonů

Sutherlandův-Hodgmanův algoritmus 4/6

- bereme vrcholy po řadě okolo polygonu: $i = n, 1, 2, \ldots, n$
- označme $s = v_i, p = v_{i+1}$, c průsečík hrany (s, p) s ořez. přímkou, w_j následující výstupní vrchol

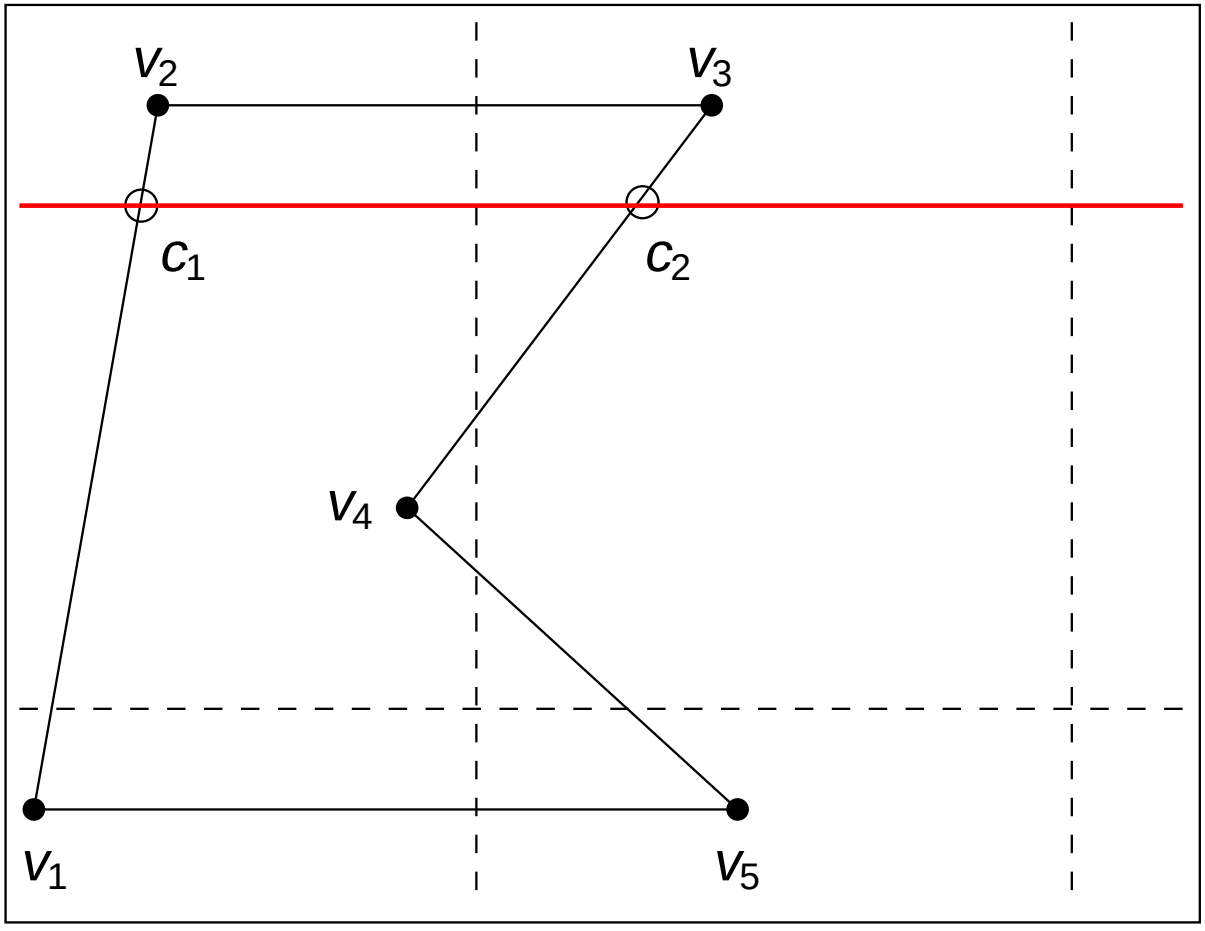
4 možné případy:

1. hrana je celá ve **vnitřní polorovině**: $p \to w_j$, $j + 1 \to j$
2. hrana jde z vnitřku ven: $c \to w_j$, $j + 1 \to j$
3. hrana je celá venku: žádný výstup
4. hrana jde z venku dovnitř: $c \to w_j$, $p \to w_{j+1}$, $j + 2 \to j$



Ořezávání polygonů

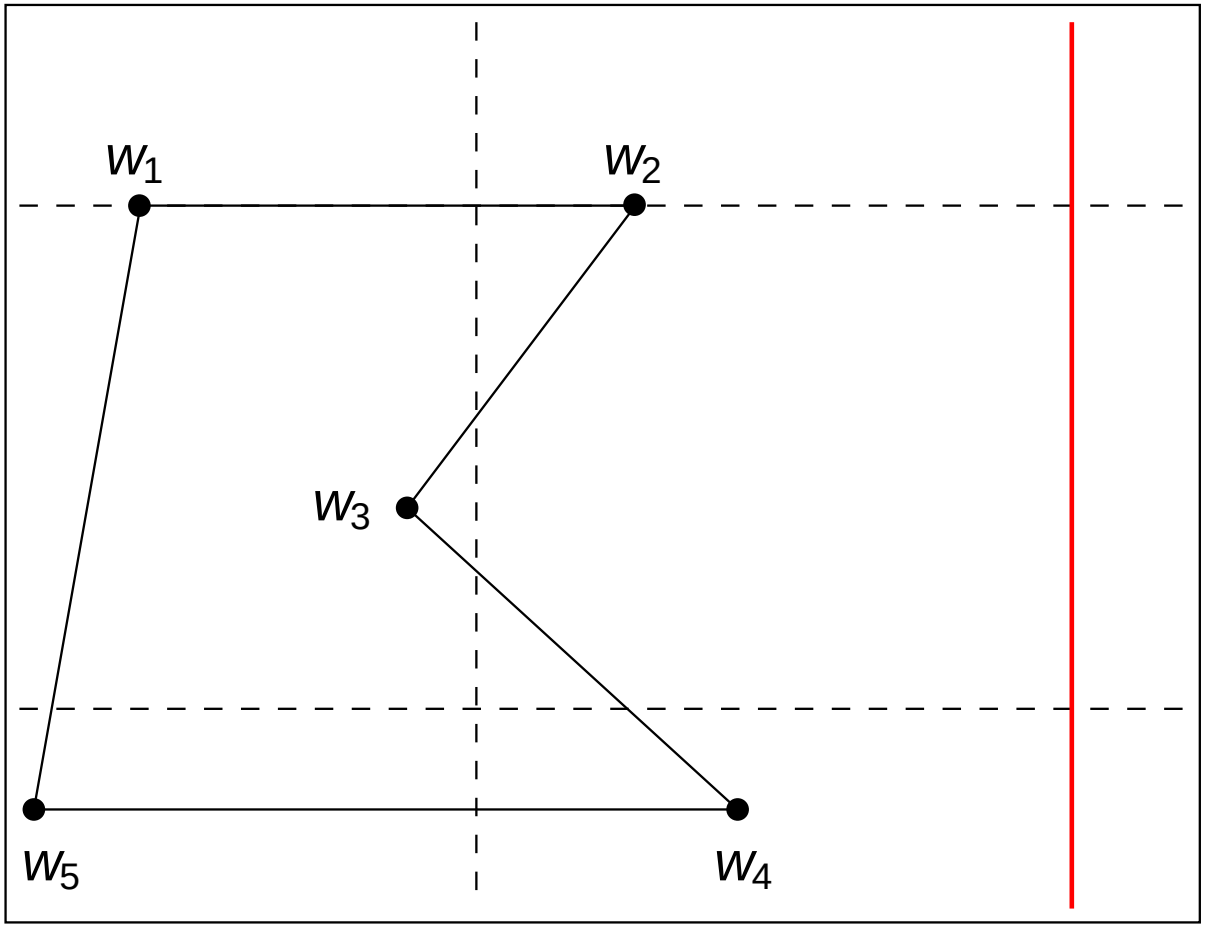
Sutherlandův-Hodgmanův algoritmus 5/6



Ukázka práce algoritmu

Ořezávání polygonů

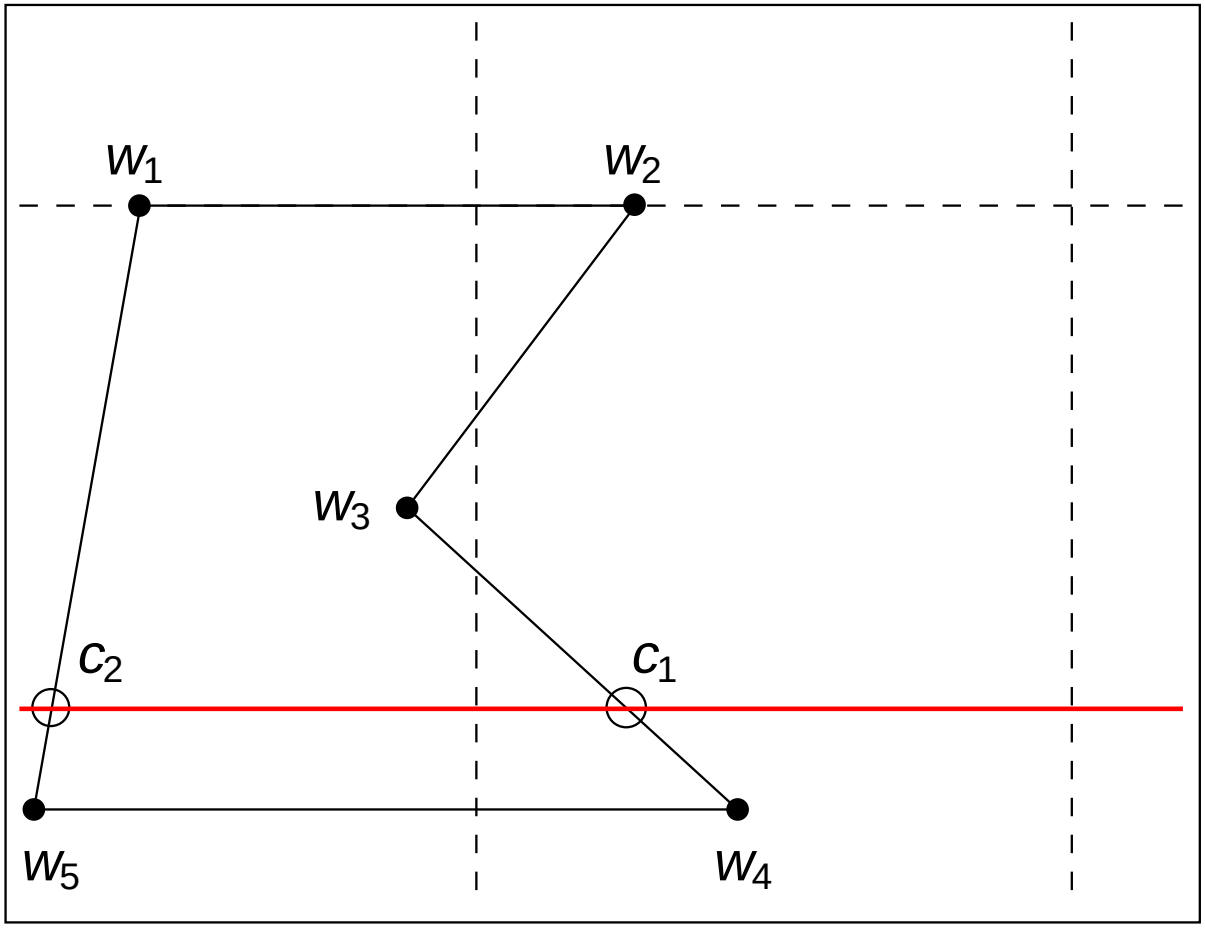
Sutherlandův-Hodgmanův algoritmus 5/6



Ukázka práce algoritmu

Ořezávání polygonů

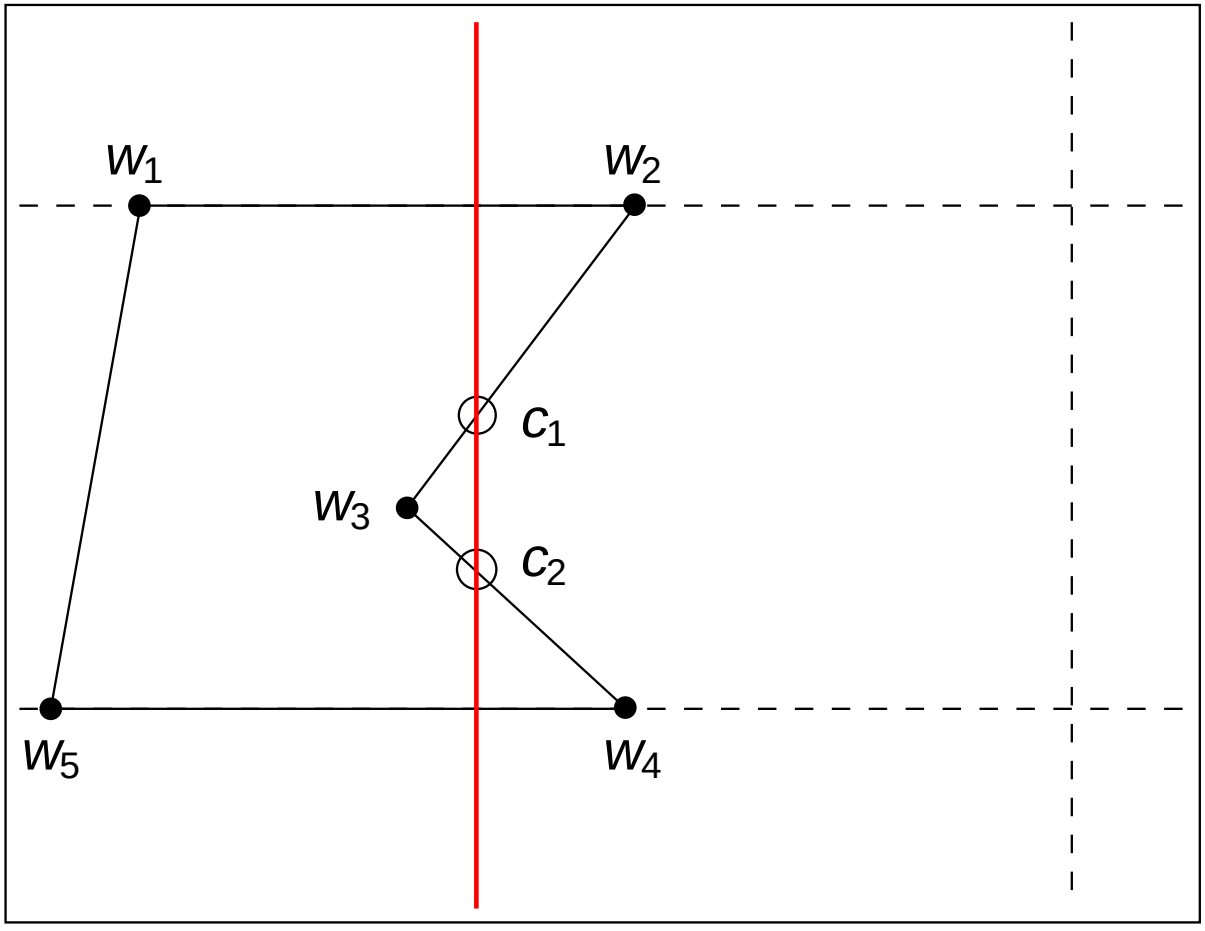
Sutherlandův-Hodgmanův algoritmus 5/6



Ukázka práce algoritmu

Ořezávání polygonů

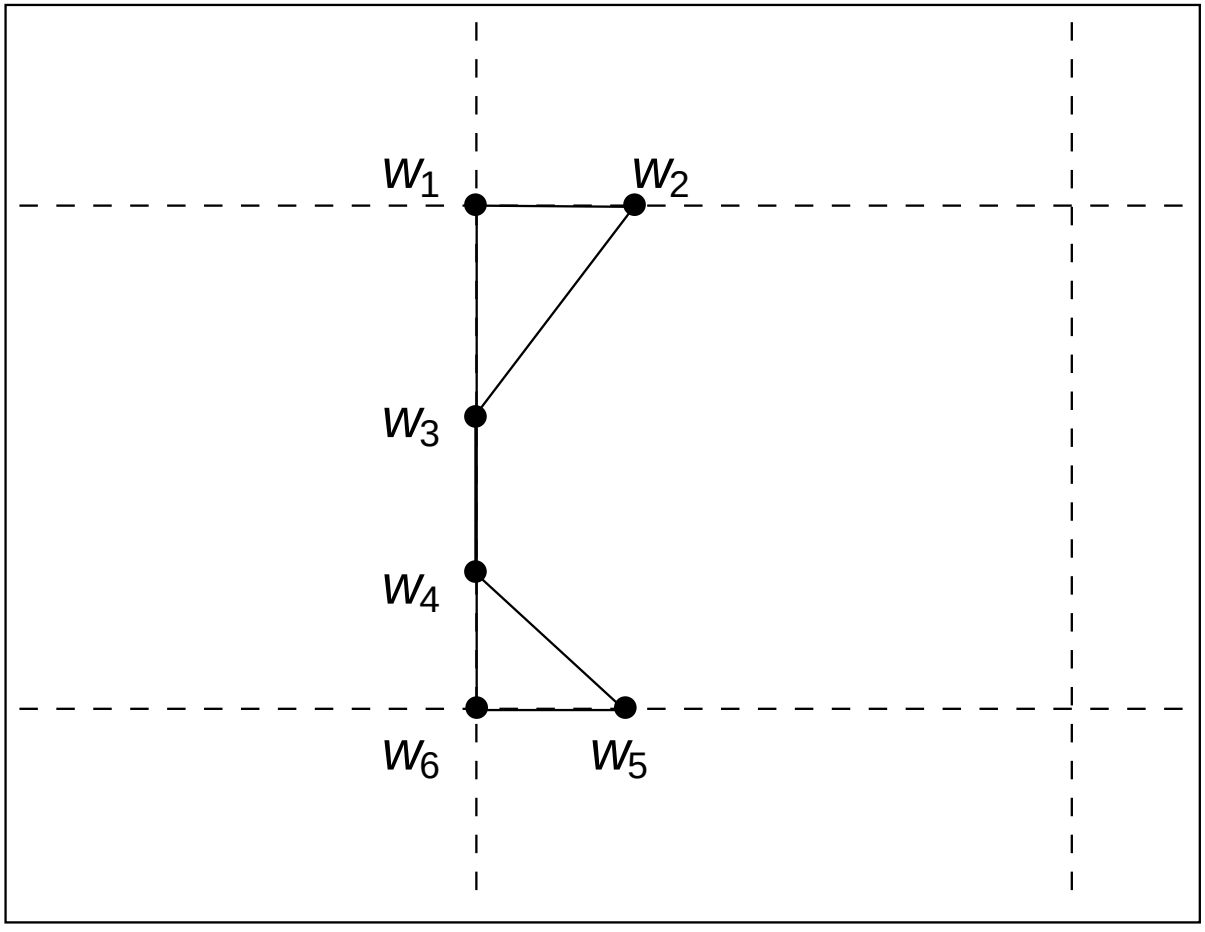
Sutherlandův-Hodgmanův algoritmus 5/6



Ukázka práce algoritmu

Ořezávání polygonů

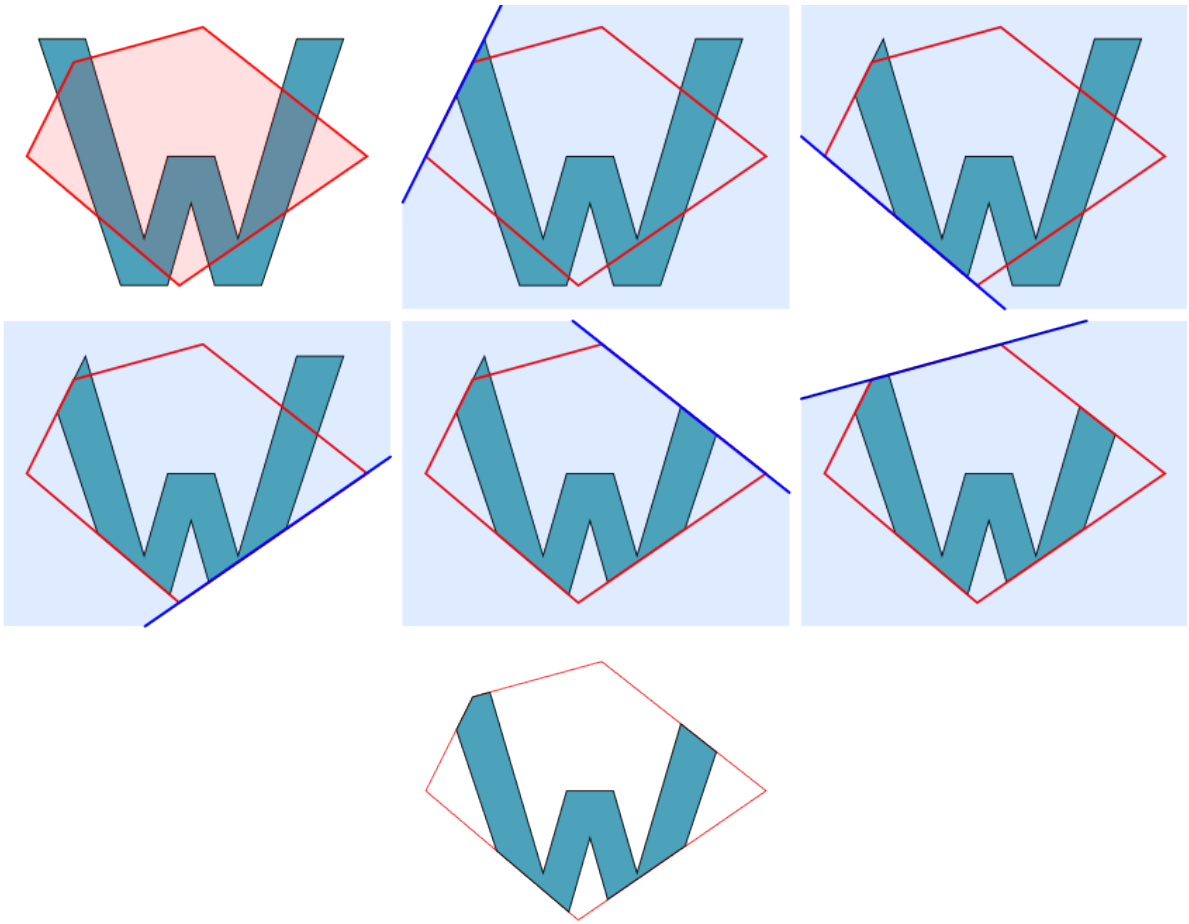
Sutherlandův-Hodgmanův algoritmus 5/6



Ukázka práce algoritmu

Ořezávání polygonů

Sutherlandův-Hodgmanův algoritmus 5/6



Obecné konvexní ořezávací okno

Ořezávání polygonů

Sutherlandův-Hodgmanův algoritmus 6/6

- algoritmus lze naprogramovat jako **reentrantní** - na výstupní vrcholy w_j okamžitě zavolat stejnou proceduru (*clipper*) pro další ořezávací polorovinu

$\Longrightarrow$ tzv. *clipper pipeline* - není třeba udržovat dočasnou paměť pro vrcholy $w_1, w_2 \ldots, w_m$

- clipper pipeline lze snadno implementovat hardwarově

Ořezávání polygonů

Weilerův-Athertonův algoritmus

- ořezávání libovolného neprotínajícího se polygonu **libovolným polygonem**
- ořezávaný polygon může mít i díry
- vznikne popis jednoho nebo více polygonů ve stejném tvaru jako vstupní data
- **neobsahuje přebytečné hrany** po rozpadu polygonu na více částí $\Longrightarrow$ lze použít např. při výpočtu stínů, kde přebytečné hrany vadí
- komplikovanější, výpočetně náročnější

Obsah

Vyplňování obecného polygonu

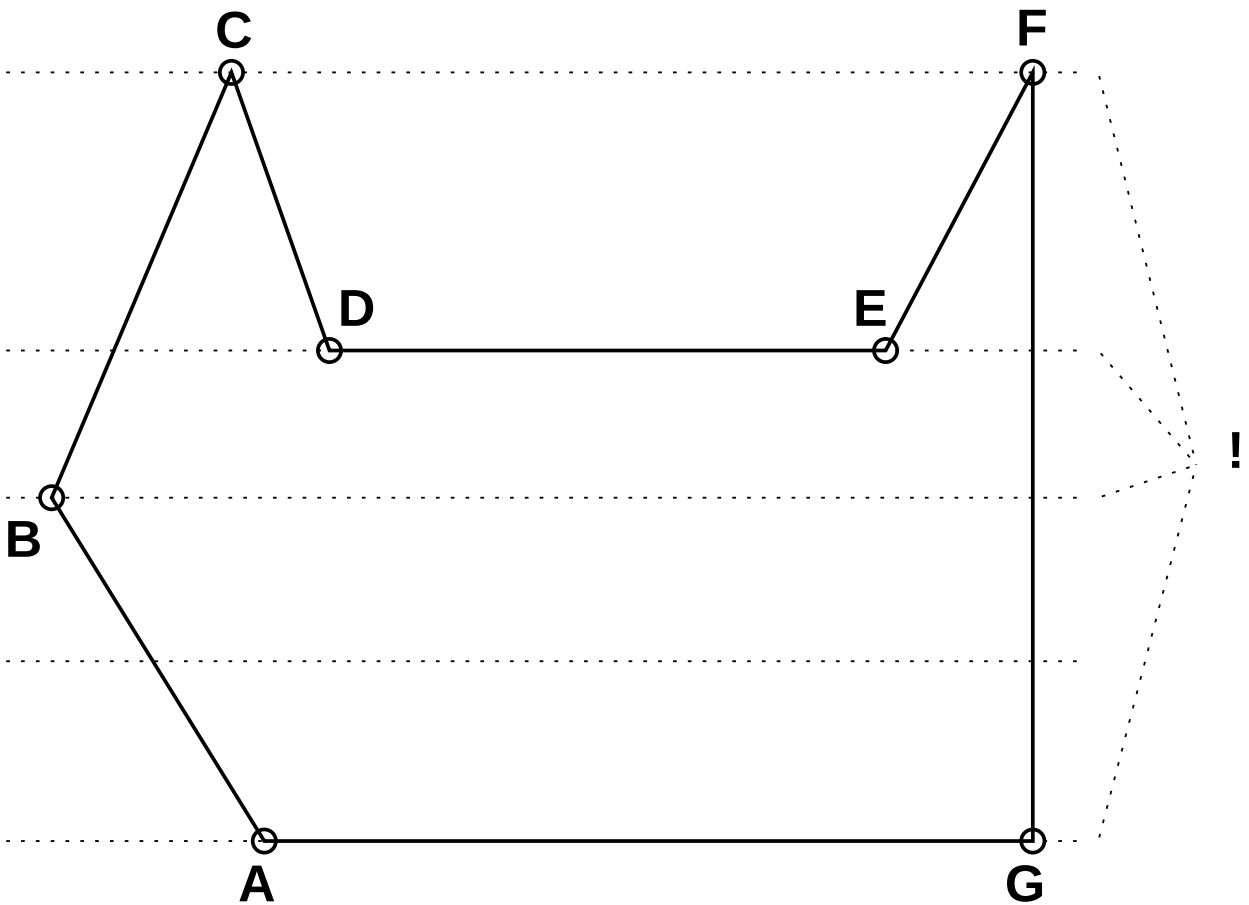
Klasický algoritmus podle Foley et al. - *Computer Graphics*

- vyplňujeme **po řádcích**
- určíme rozsah polygonu na ose *y*
- na každém řádku určíme průsečíky vodorovné přímky vedoucí středem řádku (*scan-line*) se všemi hranami polygonu
- seřadíme zleva doprava
- vyplňujeme úsek mezi každou následující dvojicí průsečíků (1.-2., 3.-4, atd.)
  - **pravidlo liché parity**: průchod každým průsečíkem přepne bit parity, kreslíme, pokud je parita lichá
  - na začátku je parita sudá
- je třeba ošetřit několik speciálních případů

Vyplňování obecného polygonu

Speciální případy

- průsečíky na celočíselných souřadnicích



- B, C, F - průsečík se 2 hranami $\Longrightarrow$ dva průsečíky

řešení: všechny dolní (nebo všechny horní) koncové vrcholy hran nezahrnovat do výpočtu průsečíku

- $\overline{AG}, \overline{DG}$ - vodorovné hrany

řešení: vůbec je nezahrnovat do výpočtu průsečíku

Vyplňování obecného polygonu

Výpočet průsečíků

- výpočet průsečíků hrubou silou je pomalý
- **hranová koherence** - řádek y protne většinu hran protnutých řádkem $y-1 \implies$ budeme si je **pamatovat**
- **AET** (*Active Edge Table*) - tabulka aktivních hran na řádku
- pro hrany v AET lze počítat průsečík x_y s řádkem y **inkrementálně**

$$x_y = x_{y-1} + 1/m$$

kde m je směrnice $m = \Delta y / \Delta x$

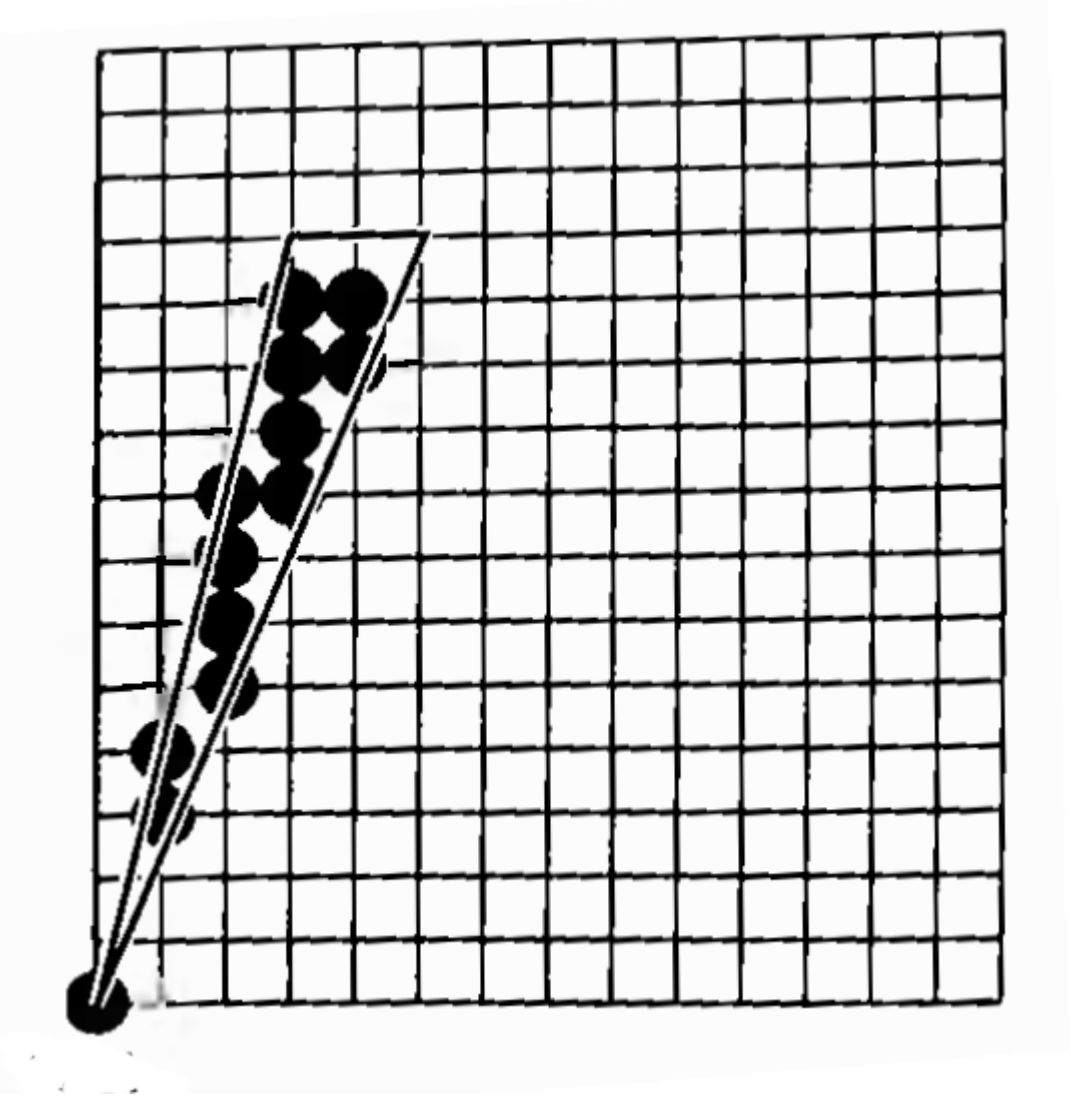
  - lze přejít do celočíselné aritmetiky

- pixely **nejbližší** průsečíkům lze najít Bresenhamovým algoritmem
- problém: nejsou vždy nutně **uvnitř** polygonu
  - může vadit při navazování různobarevných polygonů $\implies$ nutná modifikace

Vyplňování obecného polygonu

„Štěrbiny“

- další komplikace - průsečíky s hranami vzálené od sebe < 1 pixel
- lze aproximovat 1 pixelem
  $\Longrightarrow$ aliasing
- lze aplikovat antialiasing jako na úsečky (změna intenzity pixelů v závislosti na šířce štěrbiny)



Vyplňování kruhu a elipsy

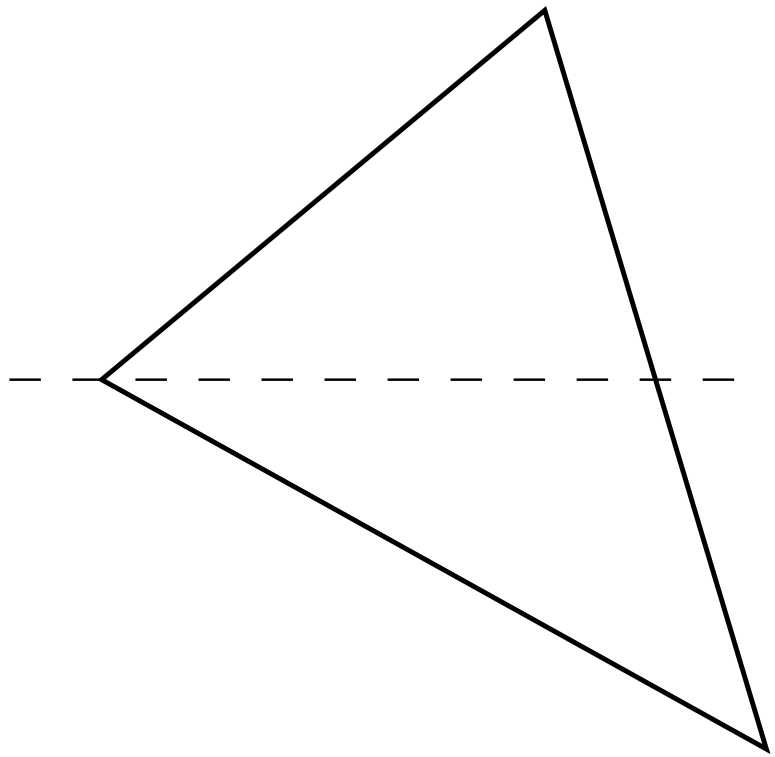
- nerovnice kruhu/vyplněné elipsy

$$F(x, y) \leq 0$$

- analogický postup - vyplnění **po řádcích**
- opět není nutné počítat rozsah hodnot x pro každý řádek y
- **inkrementální** postup
  - lze rozhodnout, zda bod $[x, y+1]$ leží uvnitř, resp. vně kruhu/elipsy podle znaménka funkce $F(x, y+1)$
  - podle toho se hranice vyplněného řádku posune doleva, resp. doprava
  - funkci F lze také vyčíslovat inkrementálně
- lze využít symetrie

Vyplňování trojúhelníku

- velmi častý úkol v grafice (sítě trojúhelníků,...)
- i každý obecný polygon lze převést na síť △ (**triangulace**) a pak vyplňovat po △
- trojúhelník lze vyplnit velmi **jednoduše a efektivně**
- rozdělíme △ vodorovně na 2 části
- každá část omezena jen dvěma hranami
- lze vyplňovat po řádcích, inkrementálně měnit rozsah vyplňovaného řádku na ose x



Vyplňování obecného polygonu s antialiasingem

Úvod

- u obvyklých algoritmů: antialiasing = práce navíc, kvalita na úkor rychlosti
  - naivní antialiasing: rasterizace polygonu ve vyšším rozlišení, poté podvzorkování
- potřeba algoritmu s kvalitním zobrazováním dostatečně rychlým i na zařízeních s nižším výkonem (*embedded* platformy, mobily, atd.)
- finský Ph.D. student Kiia Kallio: *Scanline edge-flag algorithm with antialiasing* (**2007 !**)
- modifikace a spojení již existujících algoritmů s využitím vlastností architektury procesoru

Vyplňování obecného polygonu s antialiasingem

Základní *Scanline edge-flag* algoritmus

- obdobný princip jako klasický algoritmus z knihy Foley - *Computer Graphics*
- vyplňování po řádcích
- vyplní se úseky mezi průsečíky s hranami
- pomocný obrazový buffer odpovídající velikosti (aby se do něj vešel polygon), 1 bit na 1 pixel

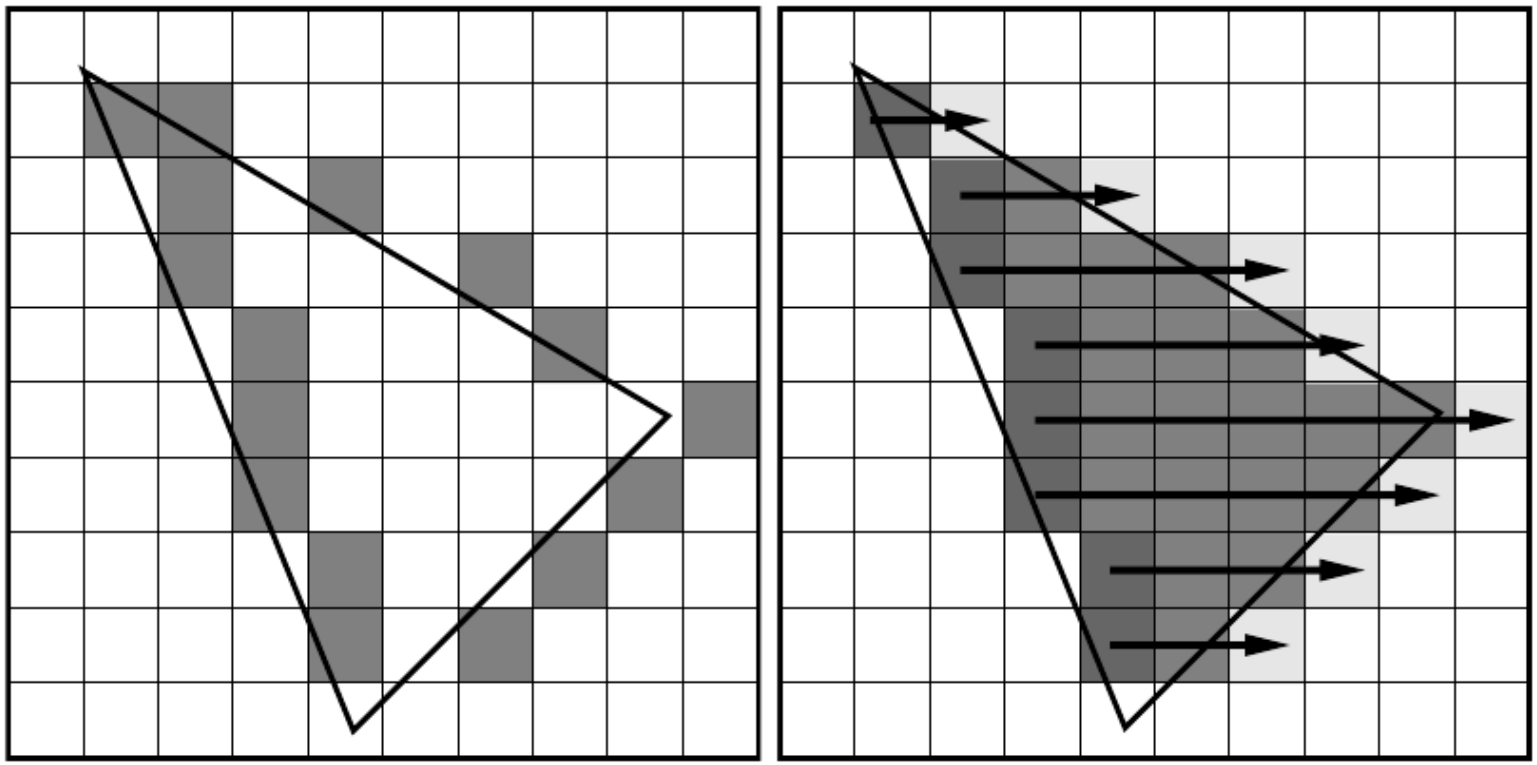
Algoritmus

1. do bufferu nakreslíme hrany inkrementálním algoritmem **vždy** s řídící osou y:
   - kde je hrana, tam dáme 1, jinde 0 ($\Longrightarrow$ *edge-flag* buffer)
2. po řádcích vyplníme úseky mezi sousedními jedničkami

Rozdíl: nemusíme řadit hrany, ale je potřeba paměť navíc

Vyplňování obecného polygonu s antialiasingem

Základní *Scanline edge-flag* algoritmus - schéma



1. Edge-flag buffer

2. Vyplňování

Vyplňování obecného polygonu s antialiasingem

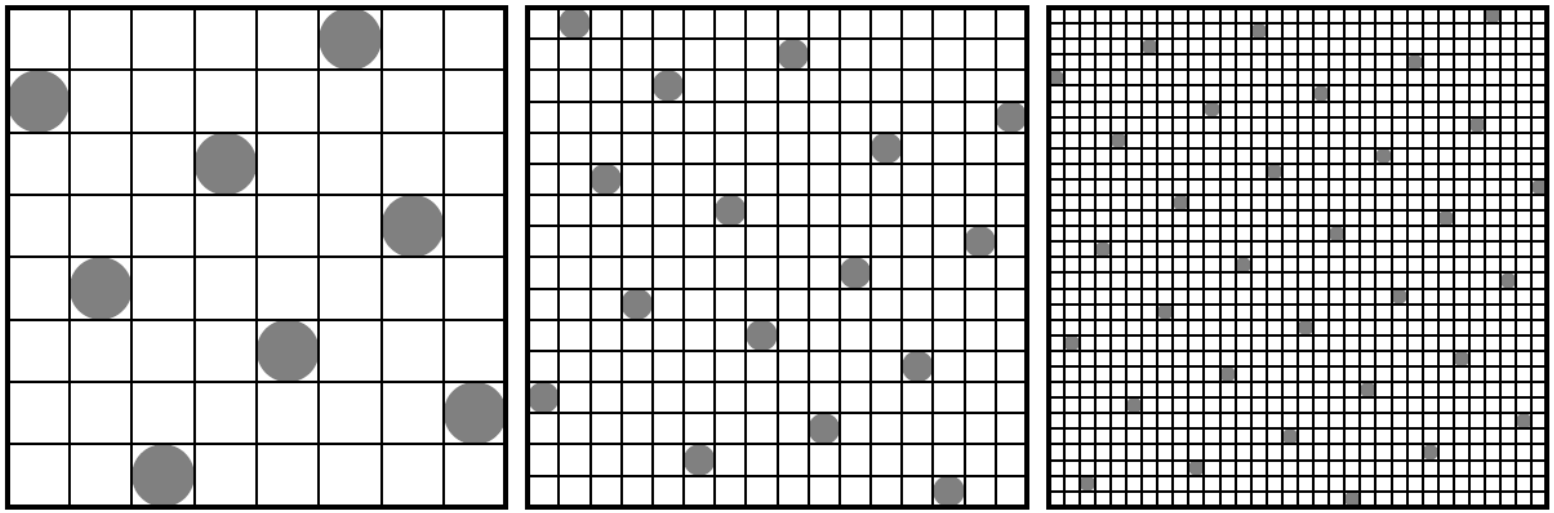
Antialiasing

- 1 bit na 1 pixel je efektivní paměťově, ale ne výpočetně (nutné operace bitového posuvu atp.)
- počítač dobře zpracovává celé byty (8 bitů) najednou, resp. celá slova (*word*): (16, 32 nebo 64 bitů podle typu procesoru)
- $\Longrightarrow$ provedeme tzv. řídké nadvzorkování (*sparse supersampling*): pixel nahradíme čtvercem $n \times n$ subpixelů, v němž vybereme n vzorků
- $n \in \{8, 16, 32\}$
- v edge-flag bufferu nyní 1 byte (resp. n-bitové slovo) na 1 pixel, v každém n bitů pro n vzorků
- **význam:** počet bitů nastavených na 1 $\approx$ intenzita barvy (resp. neprůhlednost - *alpha*) výsledného pixelu

Vyplňování obecného polygonu s antialiasingem

Volba vzorků

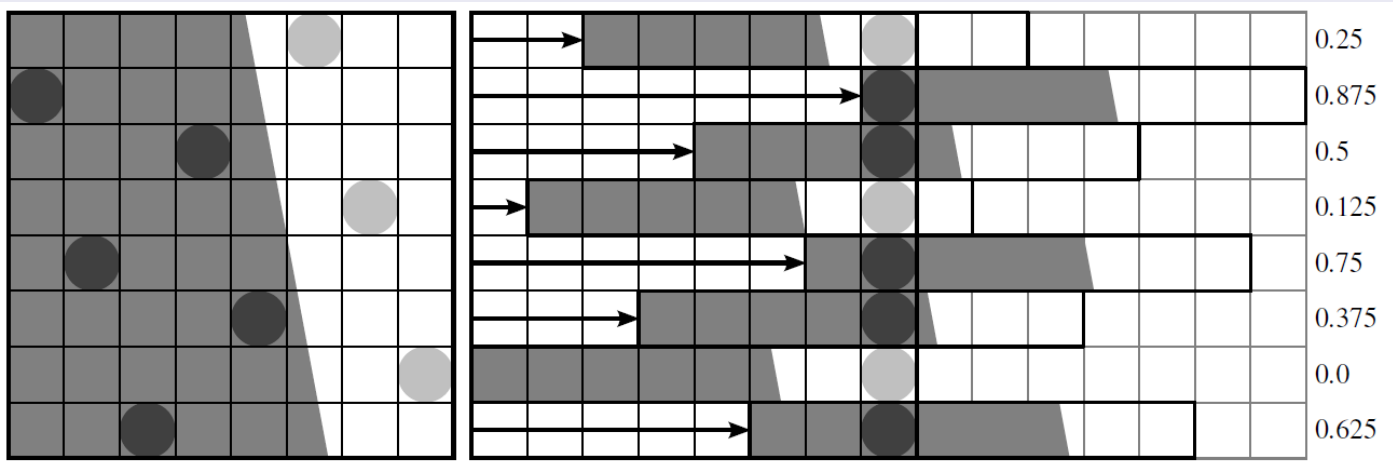
- snaha o co nejrovnoměrnější rozmístění n vzorků do rastru $n \times n$
- problém n věží (*n-rooks pattern*): rozmístění n šachových věží do šachovnice $n \times n$ tak, aby se navzájem neohrožovaly
  - právě 1 vzorek v každém sloupci i řádku
  - mnoho možných kombinací: minimální součet kvadrátů vzdáleností nejbližších vzorků $\Longrightarrow$ rovnoměrnost



Vyplňování obecného polygonu s antialiasingem

Implementace

1. **rasterizace hran**
   - analogie vykreslování pixelů: v každém **sub-řádku** rozhodování, který vzorek je nejblíže hraně
     $\Longleftrightarrow$ v kterém slově zapnout který bit
   - `offset[]`: pozice 1. až n-tého vzorku v rámci pixelu **zprava**
     čára prochází(x, y) $\Longrightarrow$ bit na pozici `round(`x + `offset[`y `mod` n`])`



   - zápis do bufferu pomocí bitového posuvu a operace **`XOR`**
     $\Longrightarrow$ 2 hrany přes sebe se vyruší

Vyplňování obecného polygonu s antialiasingem

Implementace

1. rasterizace hran
2. **vyplňování** úseků mezi hranami v každém **řádku**
   - již není třeba znát pozici vzorků
   - postupuje se po celých pixelech (*n*-ticích vzorků) najednou
   - buffer vyplňujeme *n*-bitovou maskou, kterou získáme operací **XOR**:

```
mask = 0;
for each j on line i {
  mask = mask XOR buffer[i,j];
  buffer[i,j] = mask;
}
```

Vyplňování obecného polygonu s antialiasingem

Implementace

1. **rasterizace hran**
2. **vyplňování** úseků mezi hranami v každém **řádku**
3. **převod vzorků na barvy pixelů**
   - nejjednodušší možnost: počet vyplněných vzorků (bitů rovných 1) $\approx$ neprůhlednost pixelu

Vyplňování obecného polygonu s antialiasingem

Optimalizace a rozšíření 1/2

- není nutný buffer pro celý polygon, možno postupovat jen po řádcích
- **tabulka všech hran** (*edge table*, ET) - pole ukazatelů
  - pro každý řádek *y* ukazatel na seznam hran začínajících na tomto řádku
  - tyto hrany se při dosažení daného *y* přidají do ...
- ... **tabulky aktivních hran** (*active edge table*, AET) na daném řádku
  - vhodná implementace AET: spojový seznam
  - (opět) pořadí hran v AET může být libovolné

1. vykreslení průniků hran v AET do bufferu o výšce 1 pixel
2. vyplnění 1 řádku

Vyplňování obecného polygonu s antialiasingem

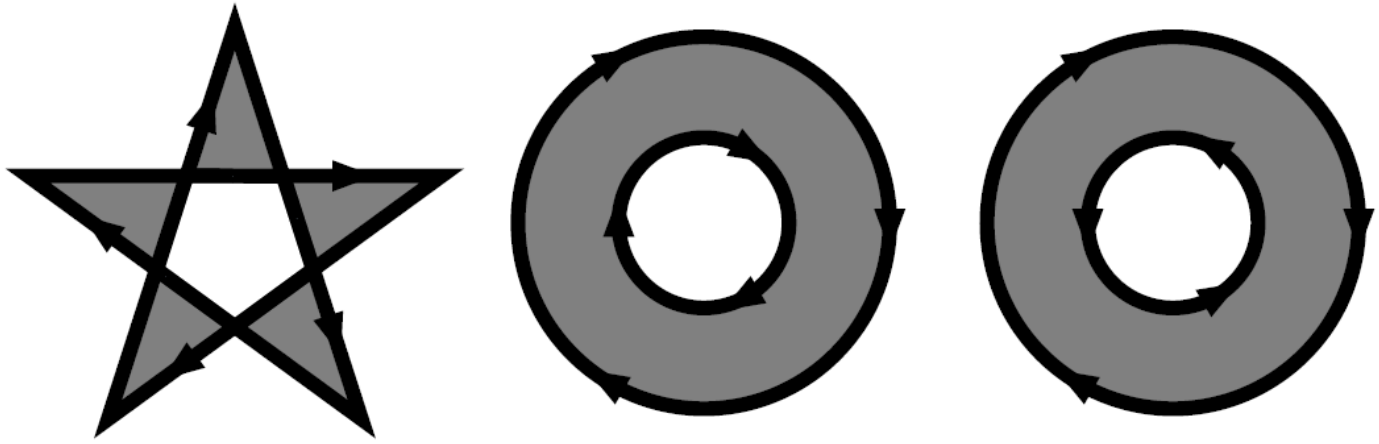
Optimalizace a rozšíření 2/2 - vyplňování sebeprotínajících polygonů

- místo vyplňování s lichou paritou (*even-odd fill rule*) lze vyplňovat sebeprotínající uzavřené lomené čáry mezi jejich krajními hranami (*nonzero winding rule*)
- *winding number*, resp. *polygon density* - počet otáček, které vytváří křivka kolem daného bodu proti směru hod. ručiček
  - v komplexní analýze tzv. *index* bodu
  - může být kladné i záporné
  - na každém řádku se mění spolu s protnutím hrany
  - vyplňujeme jen tam, kde je nenulové
- modifikace algoritmu: místo 1 bitu pro každý vzorek nutno v bufferu uložit winding number
- stále zajímavé „paralelní“ zpracování: 64 bitový systém $\Longrightarrow$ 8 vzorků po 8 bytech v 1 proměnné typu **`int`**

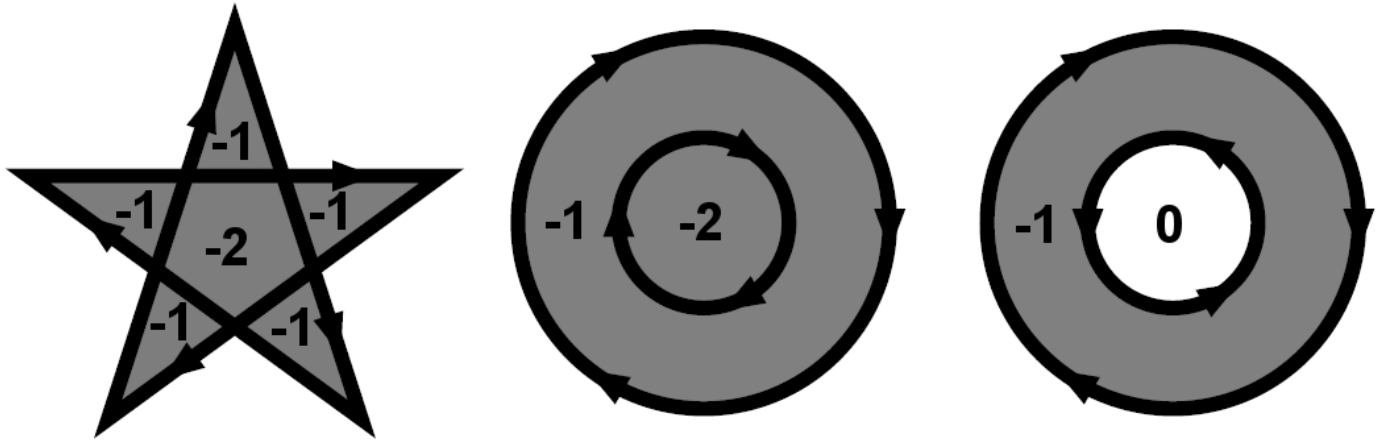
Vyplňování obecného polygonu s antialiasingem

Pravidla pro vyplňování

even-odd fill rule

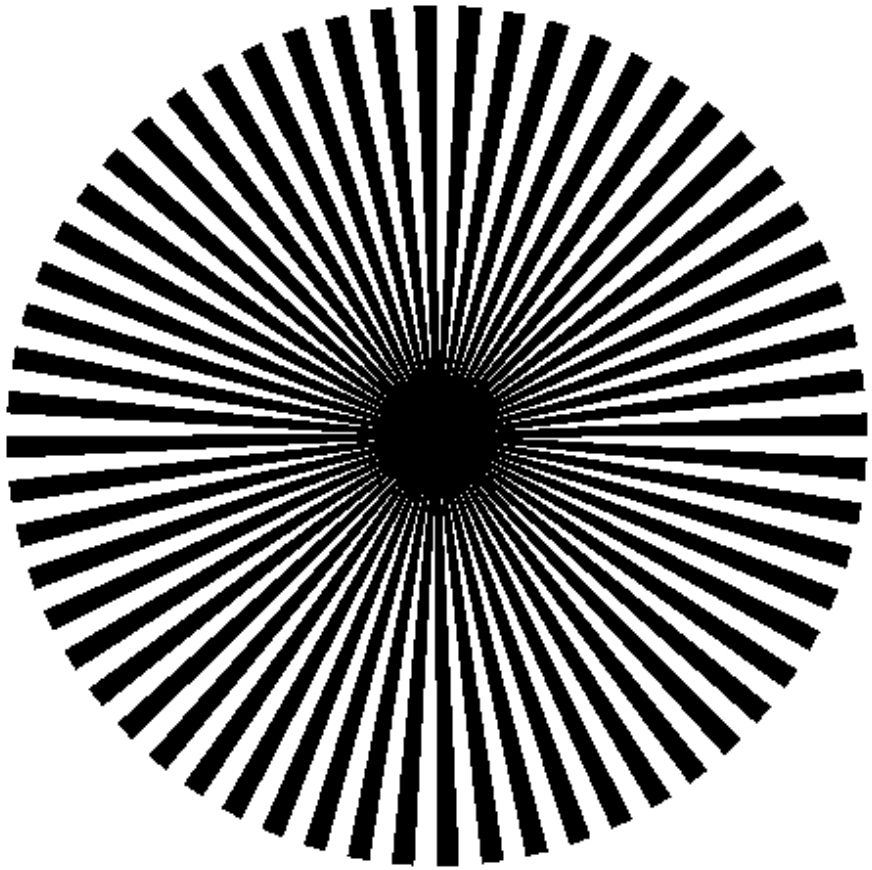


nonzero winding rule



Vyplňování obecného polygonu s antialiasingem

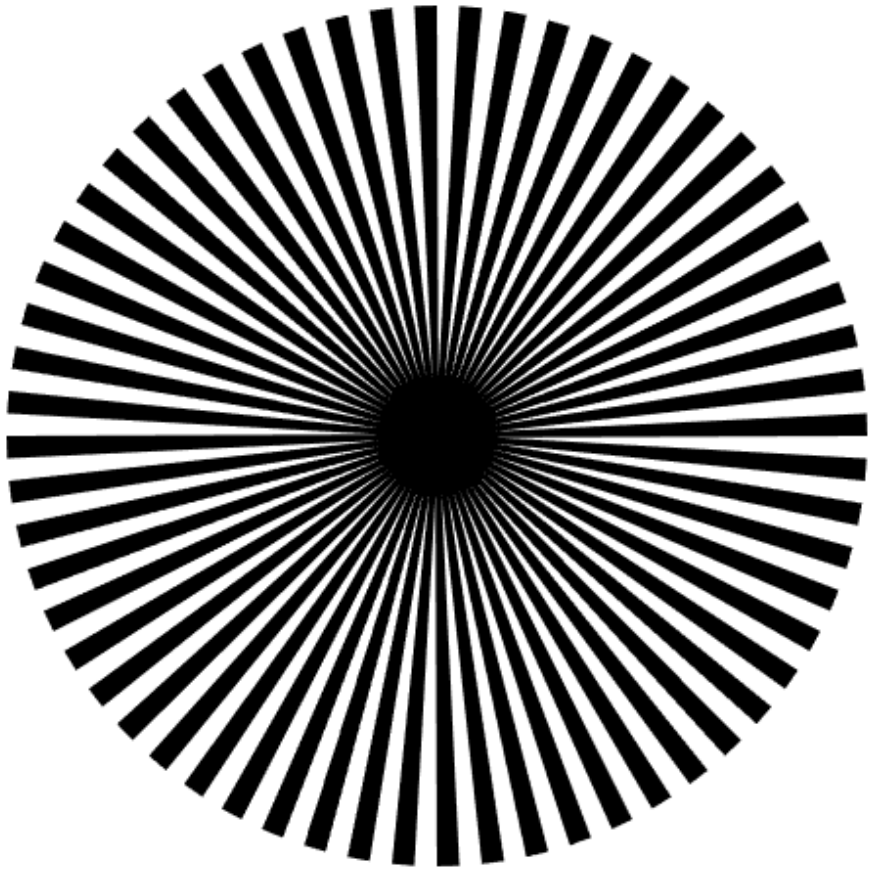
Srovnání různých úrovní antialiasingu



1 vzorek / pixel

Vyplňování obecného polygonu s antialiasingem

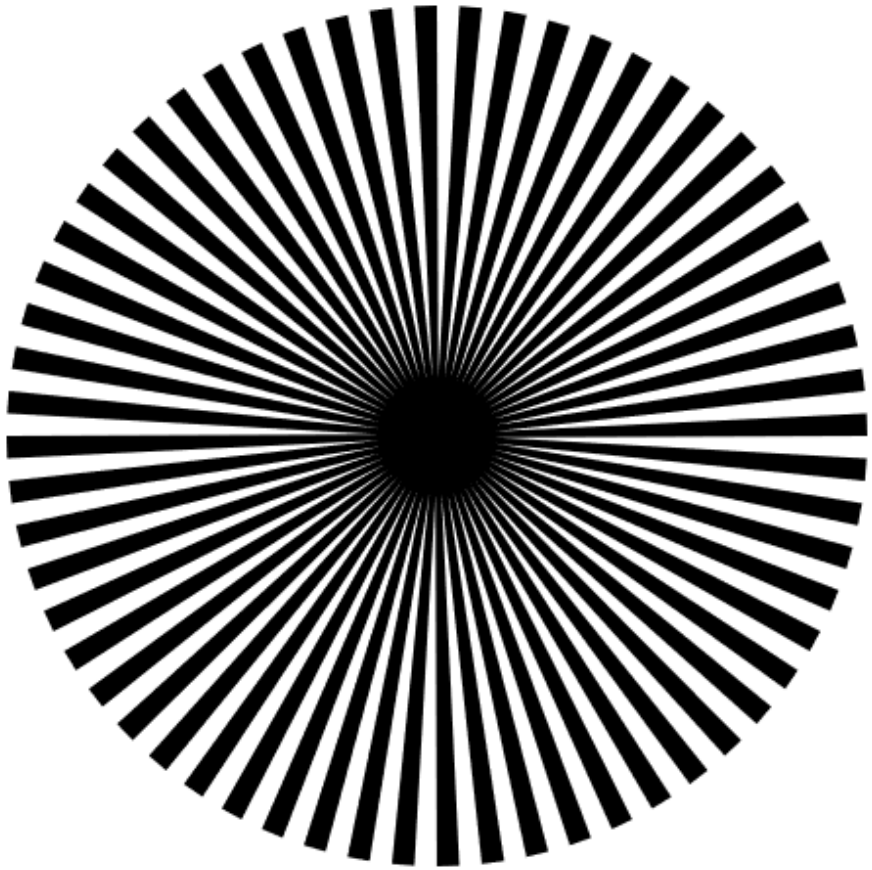
Srovnání různých úrovní antialiasingu



8 vzorků / pixel

Vyplňování obecného polygonu s antialiasingem

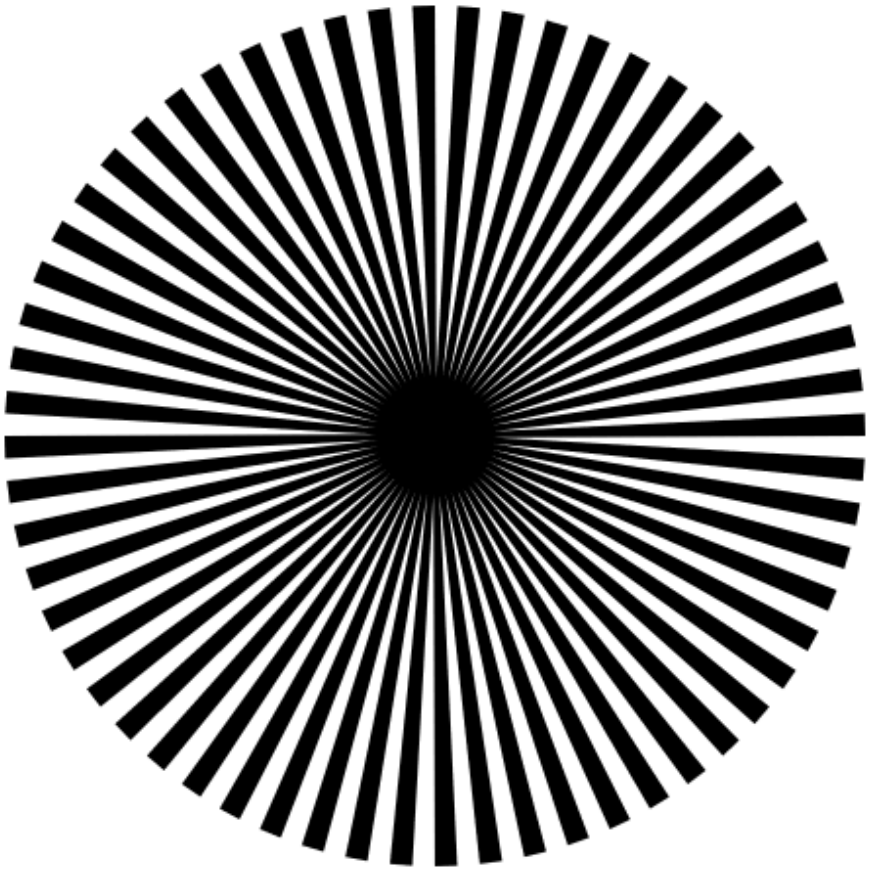
Srovnání různých úrovní antialiasingu



16 vzorků / pixel

Vyplňování obecného polygonu s antialiasingem

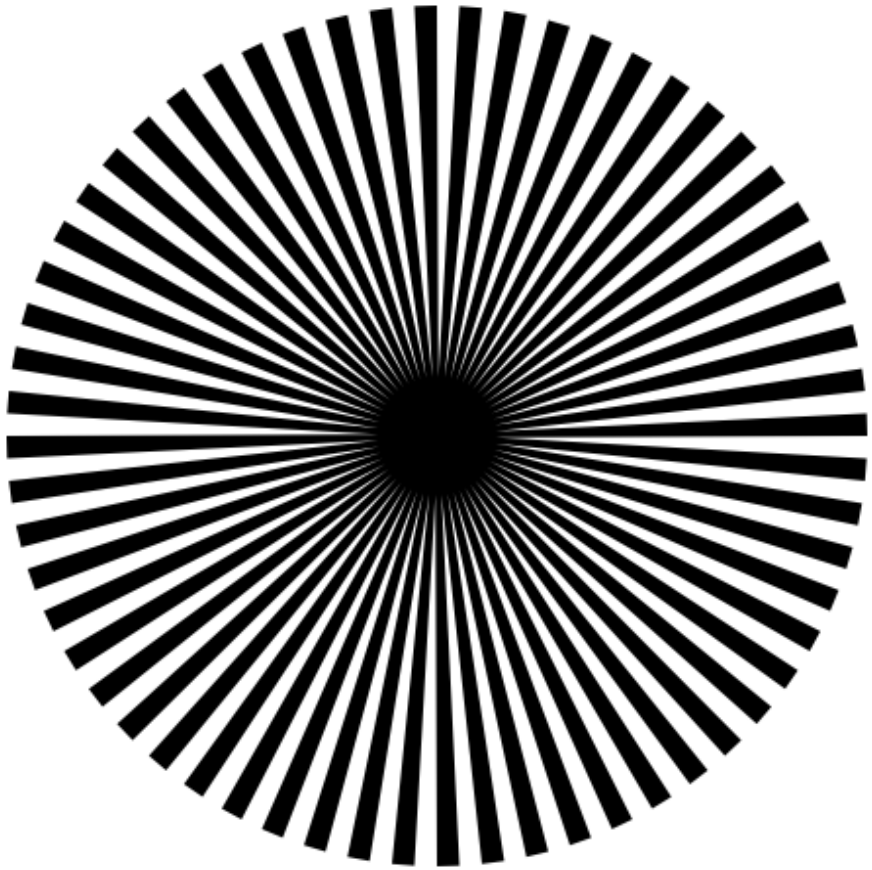
Srovnání různých úrovní antialiasingu



32 vzorků / pixel

Vyplňování obecného polygonu s antialiasingem

Srovnání různých úrovní antialiasingu

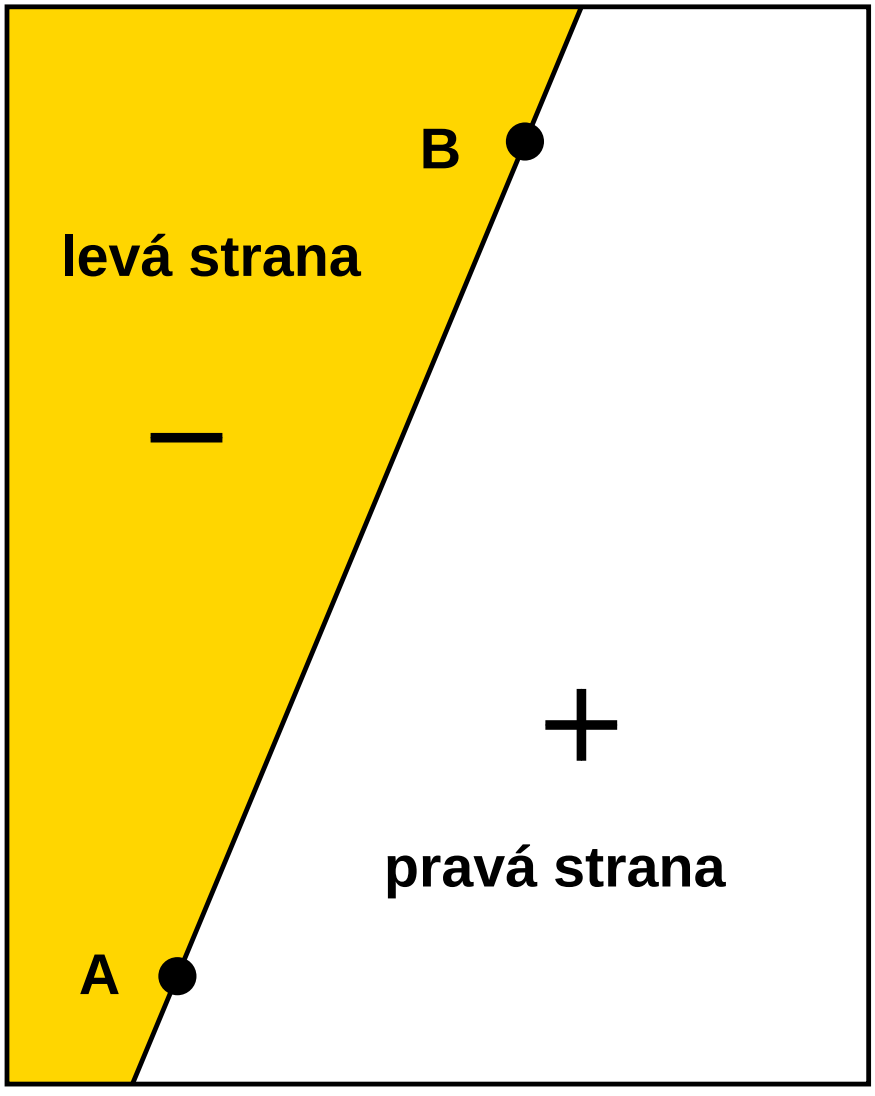


antialiasing s analytickým výpočtem pokrytí pixelů

Vyplňování konvexního polygonu

Pinedův algoritmus 1/4

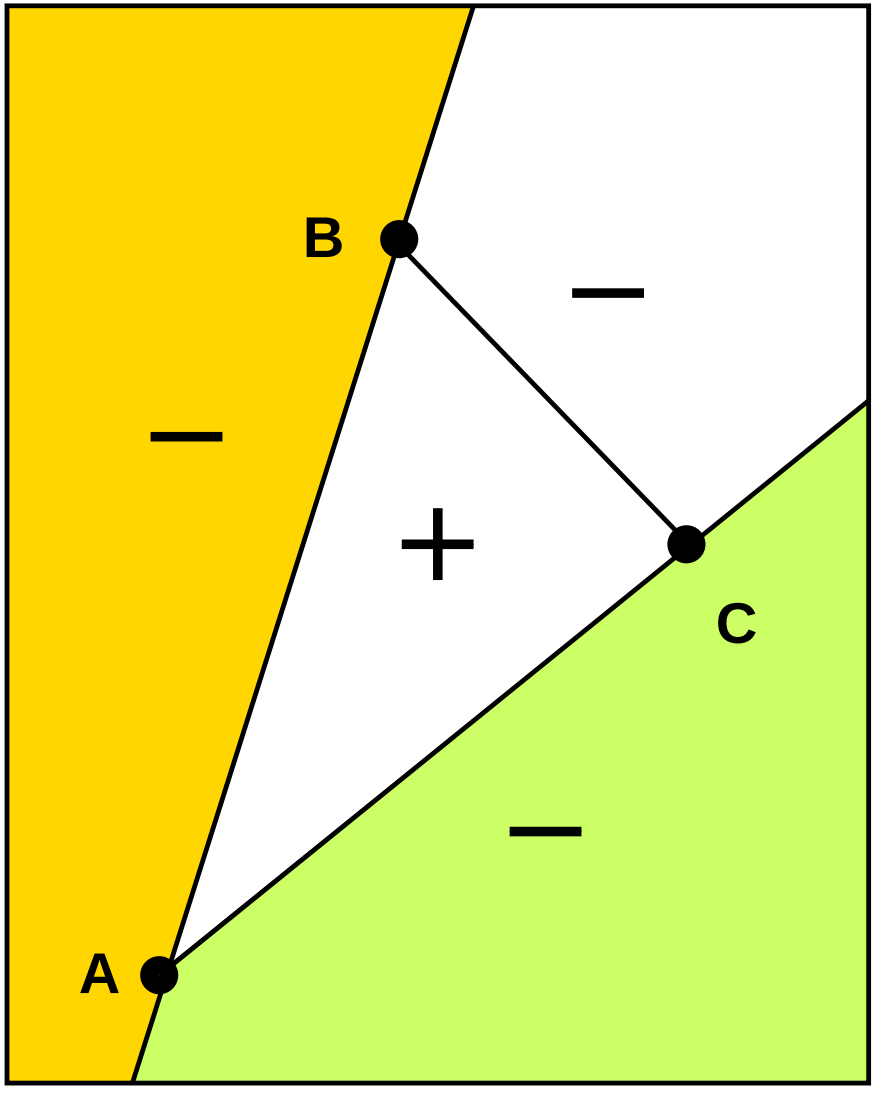
- **konvexní** polygon popsán jako průnik polorovin, daných jednotlivými hranami
- polygon zadán **orientovanými** hranami (po směru hodinových ručiček)
- **hranová fukce** $E_i(x, y)$ je kladná, pokud bod $[x, y]$ leží v pravé polorovině od i-té hrany (ve směru orientace), jinak záporná
- polygon definován oblastí, kde je $E_i(x, y) > 0 \; \forall i$



Vyplňování konvexního polygonu

Pinedův algoritmus 1/4

- **konvexní** polygon popsán jako průnik polorovin, daných jednotlivými hranami
- polygon zadán **orientovanými** hranami (po směru hodinových ručiček)
- **hranová fukce** $E_i(x, y)$ je kladná, pokud bod $[x, y]$ leží v pravé polorovině od i-té hrany (ve směru orientace), jinak záporná
- polygon definován oblastí, kde je $E_i(x, y) > 0 \; \forall i$



Vyplňování konvexního polygonu

Pinedův algoritmus 2/4

- polygon definován vrcholy $v_i = [x_i, y_i]$
- hrana $e_i = (v_i, v_{i+1})$ definována vrcholem v_i a směrem

$$\begin{pmatrix} \Delta x_i \\ \Delta y_i \end{pmatrix} = \begin{pmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{pmatrix}$$

- normálový vektor hrany e_i je tedy

$$\boldsymbol{n}_i = \begin{pmatrix} \Delta y_i \\ -\Delta x_i \end{pmatrix}$$

- i-tá hranová funkce je (bi)lineární funkce tvaru

$$E_i(x,y) = \boldsymbol{n}_i \cdot \begin{pmatrix} x - x_i \\ y - y_i \end{pmatrix} = (x - x_i)\,\Delta y_i - (x - x_i)\,\Delta x_i$$

- je zřejmé, že $E_i(x,y) = 0$ je rovnice **přímky** e_i a pro $[x,y] \notin e_i$ znaménko E_i odpovídá našemu požadavku

Vyplňování konvexního polygonu

Pinedův algoritmus 3/4

- hranové funkce lze pro sousední pixely počítat opět inkrementálně

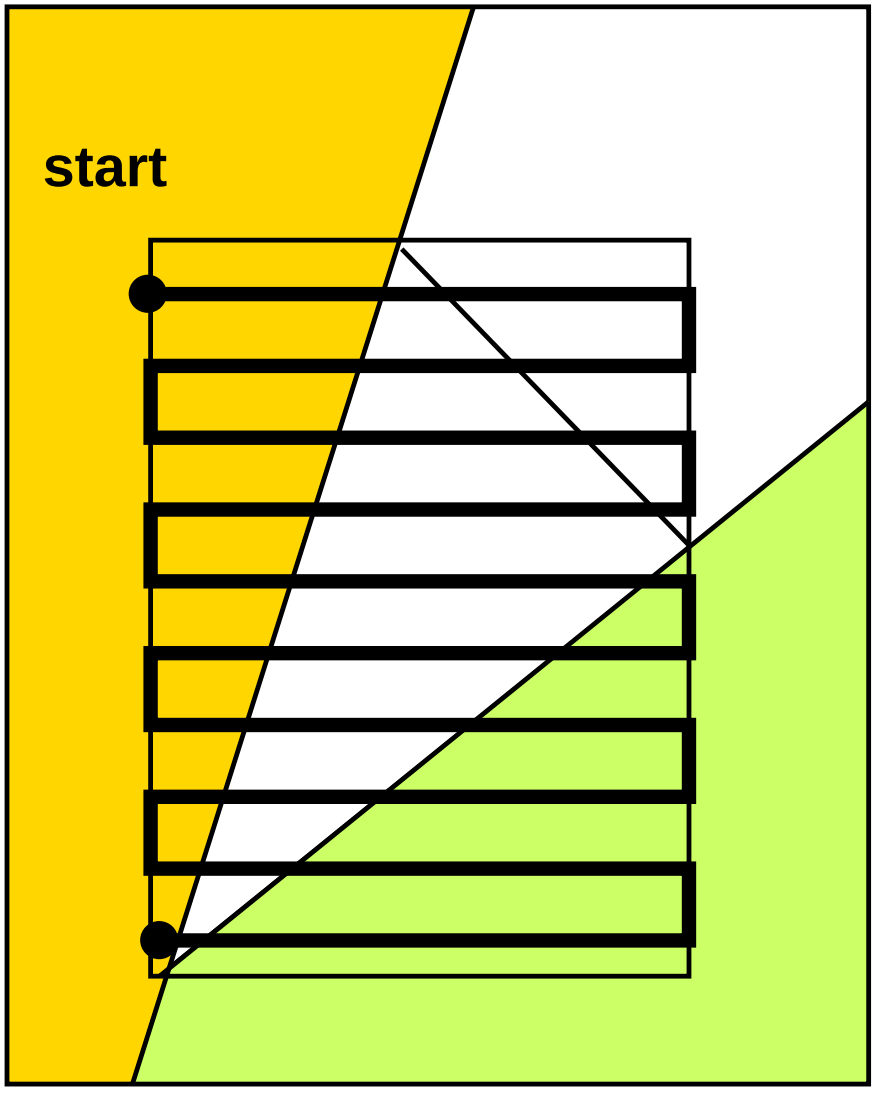
$$\begin{aligned} E_i(x+1,y) &= E_i(x,y)+\Delta y_i \\ E_i(x-1,y) &= E_i(x,y)-\Delta y_i \\ E_i(x,y+1) &= E_i(x,y)-\Delta x_i \\ E_i(x,y-1) &= E_i(x,y)+\Delta x_i \end{aligned}$$

- po pixelech $[x,y]$ procházíme oblast, kde leží polygon, a vyčíslujeme $E_i(x,y)\ \forall i$, podle toho vykreslíme pixel nebo ne

Vyplňování konvexního polygonu

Pinedův algoritmus 4/4

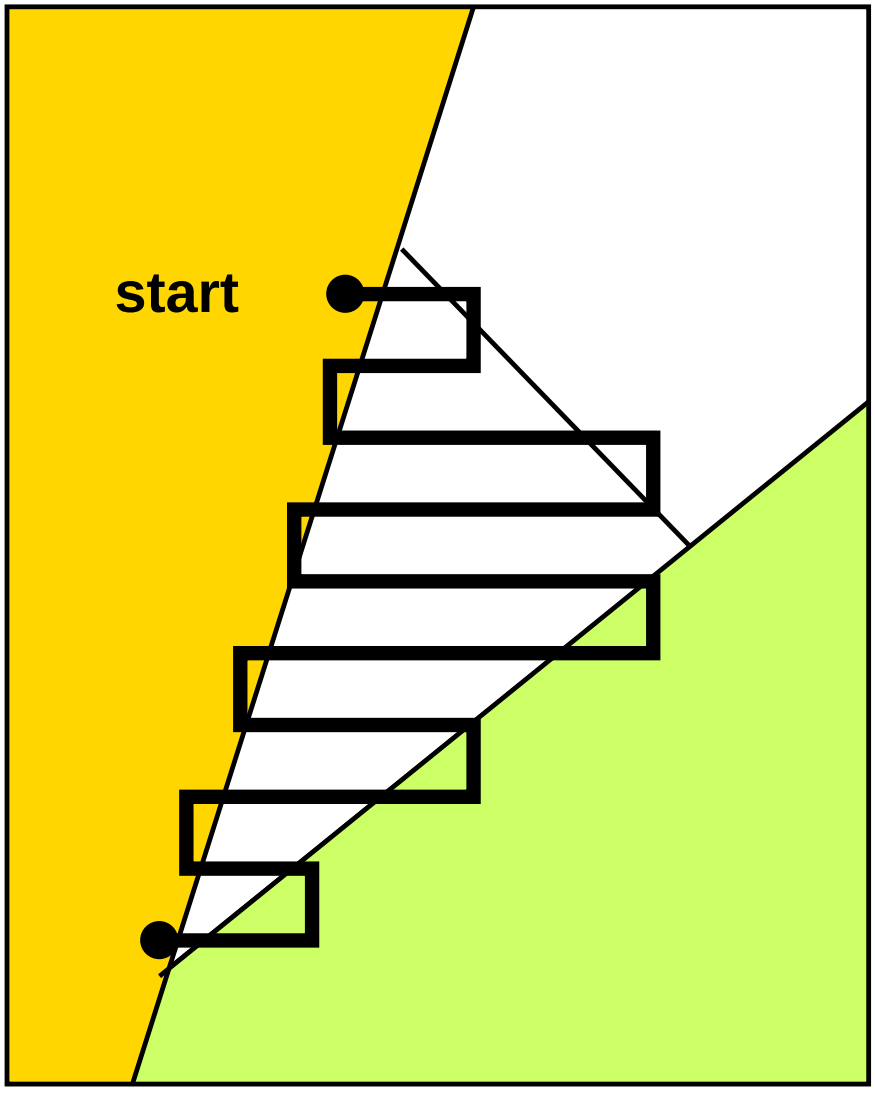
- základní algoritmus prochází po řádcích obalový obdélník
  - snadné ořezávání obdélníkem
- lepší algoritmus se zastaví na hraně a postoupí na další řádek
  - na následujícím řádku obecně nestačí obrátit směr
  - nutnost hledat začátek vyplňované oblasti podle znaménka E_i
- řádek také můžeme procházet od „prostředku“



Vyplňování konvexního polygonu

Pinedův algoritmus 4/4

- základní algoritmus prochází po řádcích obalový obdélník
  - snadné ořezávání obdélníkem
- lepší algoritmus se zastaví na hraně a postoupí na další řádek
  - na následujícím řádku obecně nestačí obrátit směr
  - nutnost hledat začátek vyplňované oblasti podle znaménka E_i
- řádek také můžeme procházet od „prostředku“



Vyplňování vzorem

- vzor určen jako bitová mapa `M` o rozměru $m \times n$
- může být upevněn
  - k referenčnímu bodu objektu (polygonu) - „plovoucí vzor“
  - k obrazovce (absolutní souřadnice, při posunu objektu se vzor nepohne)
- příkaz

```
PutPixel(x,y,c);
```

nahradit v případě upevnění k referenčnímu bodu $[x_R, y_R]$

```
PutPixel(x,y,M[(x-xR) % m][(y-yR) % n]);
```

resp. v případě upevnění k obrazovce

```
PutPixel(x,y,M[x % m][y % n]);
```

Flood fill

- také tzv. **semínkové vyplňování** (*seed fill*, *boundary fill*)
- vyplnění plochy uzavřené hranicí určenou v rastru
  - vyplnění ploch v obrázku ohraničených jistou barvou
  - vyplnění ploch se stejnou, resp. podobnou barvou jako daná oblast
  - odkrytí pole ve hře „**miny**“
- je třeba určit výchozí bod uvnitř oblasti (*seed*), ze kterého algoritmus vychází
- algoritmus je rekurzivní, resp. používá zásobník nebo frontu pixelů
- oblast může být:
  - **4-spojitá**: mezi 2 body oblasti existuje cesta **pouze ve směru souř. os** (4 směry)
  - **8-spojitá**: mezi 2 body oblasti existuje cesta ve směru souř. os **nebo diagonálně** (8 směrů)

Flood fill

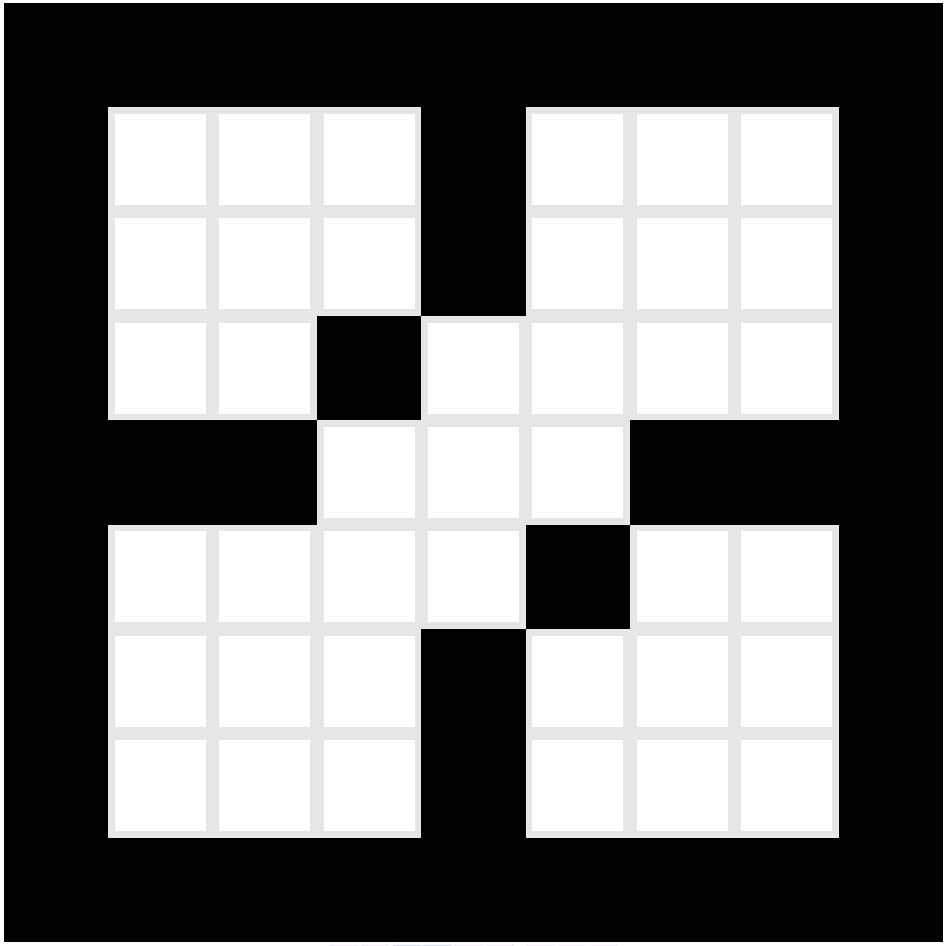
Kód rekurzivního algoritmu

- pro 4-spojitou oblast

```
void FloodFill4(int x, int y, int c_old, int c_new)
{
 if(GetPixel(x,y)==c_old) {
  PutPixel(x,y,c_new);
  FloodFill4(x-1,y,c_old,c_new);
  FloodFill4(x+1,y,c_old,c_new);
  FloodFill4(x,y-1,c_old,c_new);
  FloodFill4(x,y+1,c_old,c_new);
 }
}
```

Flood fill

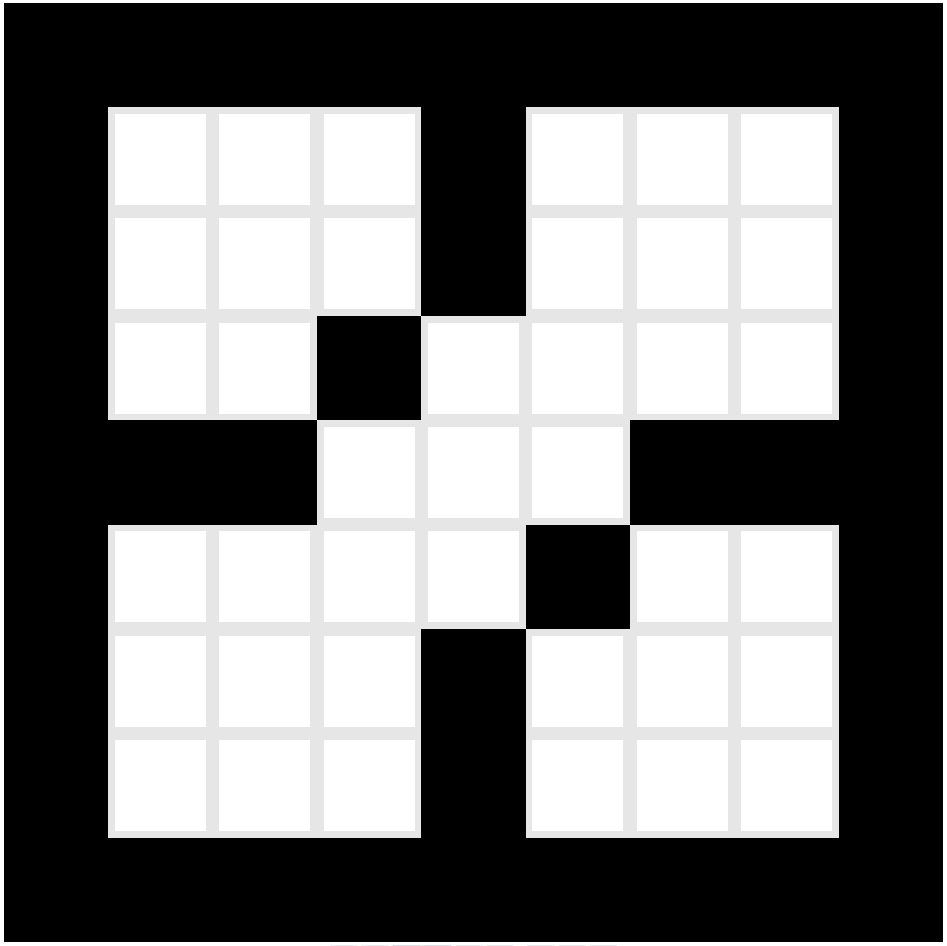
Příklad 1



4-spojitá oblast (8-spojitá hranice)

Flood fill

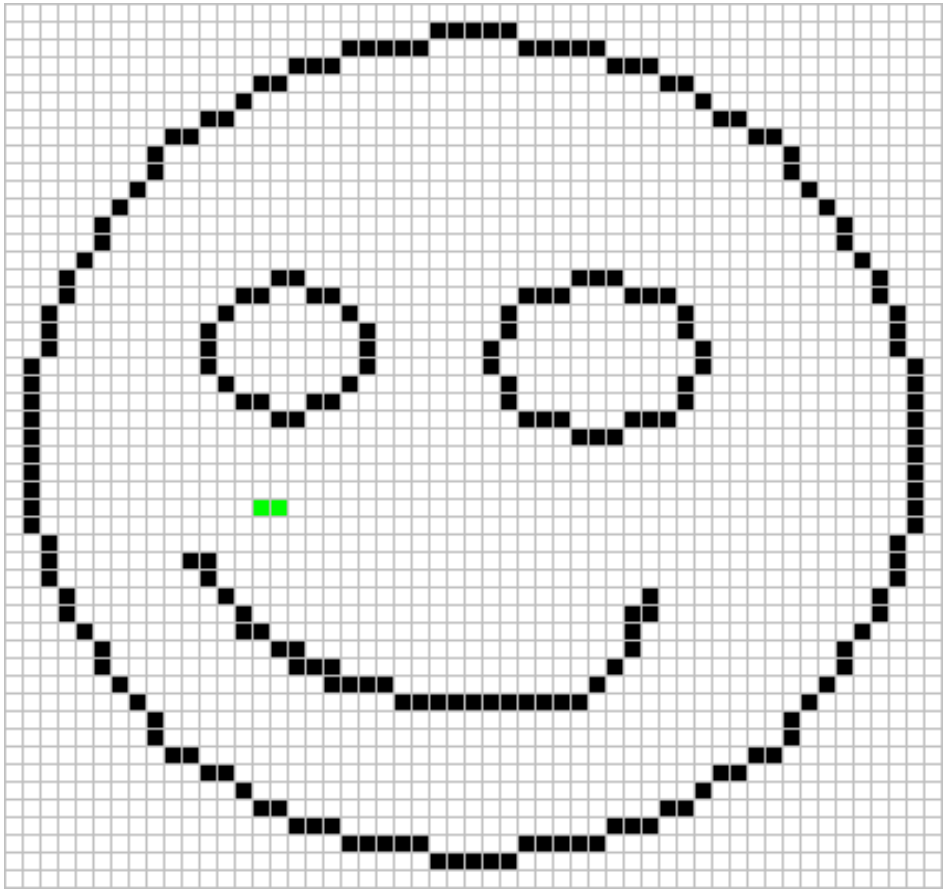
Příklad 2



8-spojitá oblast (4-spojitá hranice)

Flood fill

Příklad 3

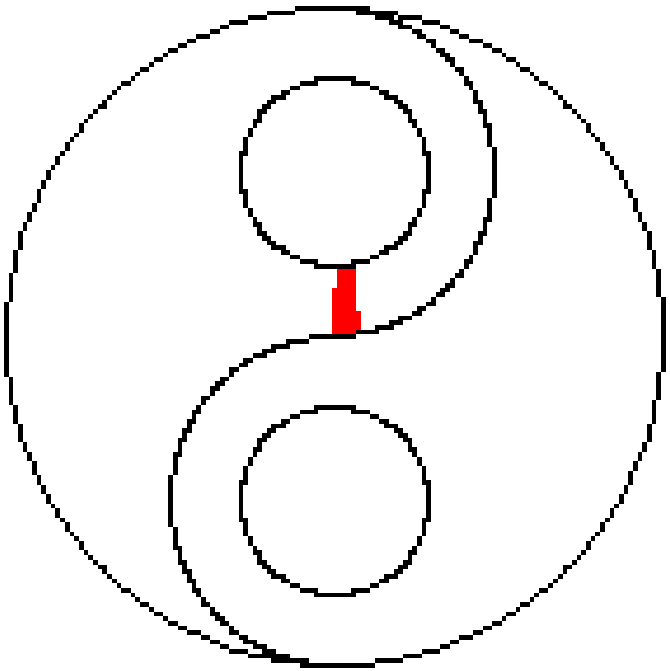


řádkové vyplňování (*scan-line fill*)

- vyplníme najednou celý řádek y, na kterém je semínko, kam jen to jde
- na zásobník uložíme semínko pro každý nevyplněný úsek v řádku $y-1$ a $y+1$
  - úseky odděleny barvou hranice či jinou nevyplnitelnou barvou
  - semínko např. vždy na prvním pixelu úseku
- nutno ošetřit střet 2 směrů vyplnění proti sobě
  - zdola i shora

Flood fill

Příklad 4

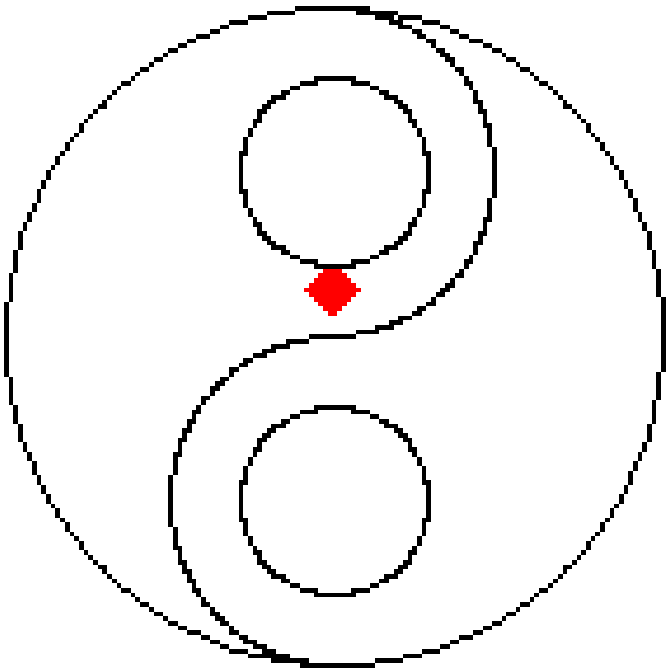


Flood fill s použitím zásobníku

- LIFO (Last In, First Out)
- nejprve se postupuje v jednom směru, hloubka rekurze roste
- ostatní směry se dostanou na řadu, až když v prvním směru není kam dál
- ekvivalentní s předchozím rekurzivním algoritmem

Flood fill

Příklad 5



Flood fill s použitím fronty

- FIFO (First In, First Out)
- okolní body se přidají do fronty dozadu
- z fronty se vybírá odpředu
- nejprve se vyplní všechny směry, potom se pokračuje dál

Literatura 1/2

J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes: *Computer Graphics: Principles and Practice*, Addison Wesley, 1997.

M. Cyrus, J. Beck: *Generalized two- and three-dimensional clipping*, Computers and Graphics, 3(1), 1978, 23-28.

S. Gupta, R. E. Sproull: *Filtering Edges for Gray-scale Displays*, SIGGRAPH 81, 1-5.

K. Kallio: Scanline Edge-Flag Algorithm for Antialiasing. In proc. TP CG 07, I. S. Lim, D. Duce (eds.), 2007. Available online: `http://mlab.uiah.fi/~kkallio/antialiasing/`

Y-D. Liang, B. Barsky: *A new concept and method for line clipping*, ACM Trans. Graphics, 3(1), 1984, 1-22.

Literatura 2/2

- M. D. McIlroy: *Getting raster ellipses right*, ACM Transactions on Graphics, Vol. 11, No. 3 (1992) 259-275.
- J. Pineda: *A parallel algorithm for polygon rasterization*, Computer Graphics, 22(4), 1988, 17-20.
- I. E. Sutherland, G. W. Hodgman: *Reentrant polygon clipping*, CACM, 17(1), 1974, 32-42.
- K. Weiler, P. Atherton: *Hidden surface removal using polygon area sorting*, ACM SIGGRAPH 11(2), 1977, 214-222.
- T. Whitted: *Anti-aliased line drawing using brush extrusion*, SIGGRAPH 83, 151-156.