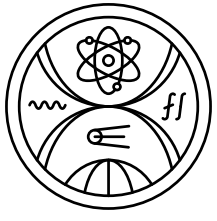
Rasterizácia a vypĺňanie oblastí

Róbert Bohdal

robert.bohdal@fmph.uniba.sk
https:\\flurry.dg.fmph.uniba.sk\webog\bohdal-vyucba

Katedra algebry a geometrie
Fakulta matematiky fyziky a informatiky
Univerzita Komenského v Bratislave

Počítačová grafika (1)
prednáška č. 9



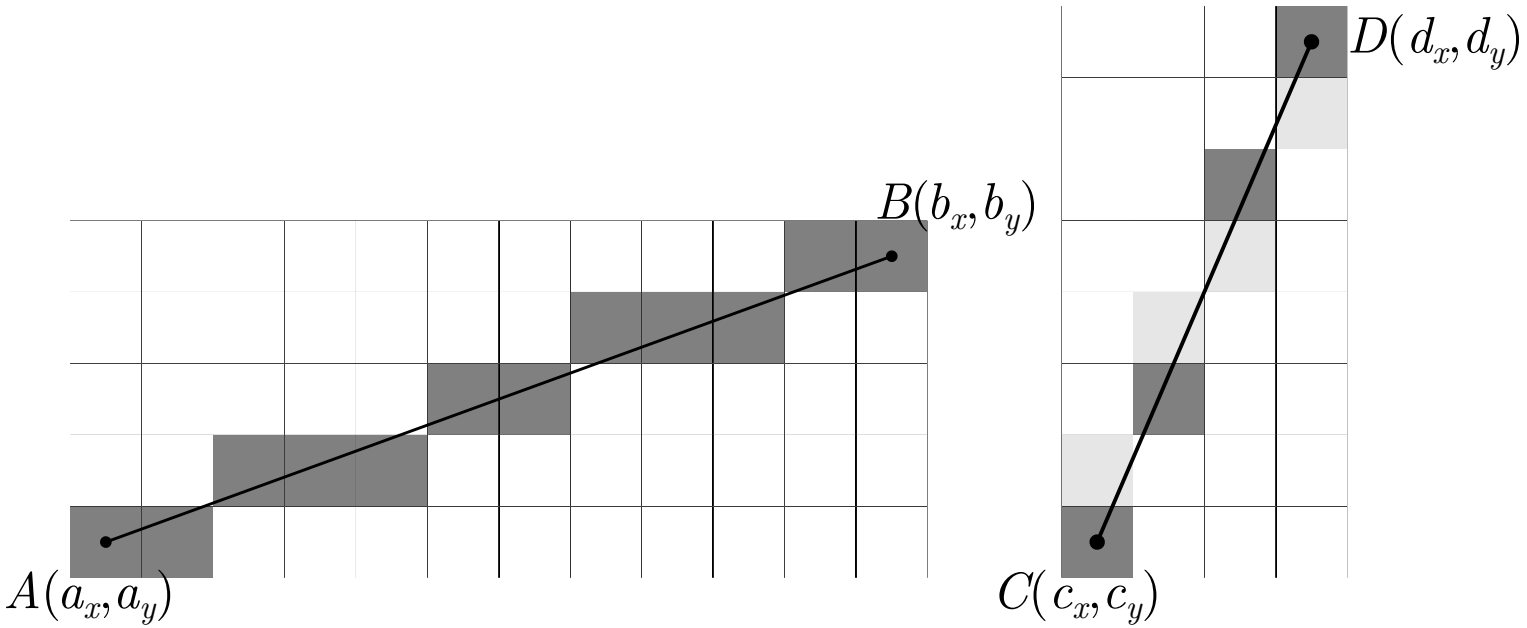
Obsah

Rasterizácia

- Grafické displeje zobrazujú objekty diskrétne pomocou pixlov, ktoré sú uložené v bitmape (*framebuffer*)
- Pixle sú reprezentované jednotkovými štvorčekmi, ktorých celočíselná hodnota či vektor (r,g,b,α) predstavuje farbu
- Poloha pixlov v bitmape (rastri, štvorcovej sieti) je daná celočíselnými súradnicami (x, y), veľkosť bitmapy je daná počtom riadkov a stĺpcov
- Nutnosť konvertovať (rasterizovať) spojité elementy (úsečky, krivky, mnohouholníky, ...) na postupnosť celočíselných bodov v štvorčekovom rastri
- Rasterizácia musí byť veľmi efektívna, aby bolo možné zobraziť mnoho grafických elementov za krátky čas
- Je vykonávaná grafickým hardvérom počítača a výpočet rasterizácie nezamestnáva hlavný procesor počítača (CPU)
- Pre vykreslenie pixla nám bude slúžiť funkcia `drawpixel(x,y)` a pre načítanie farebnej hodnoty pixla `readpixel(x,y)`

Rasterizácia úsečky

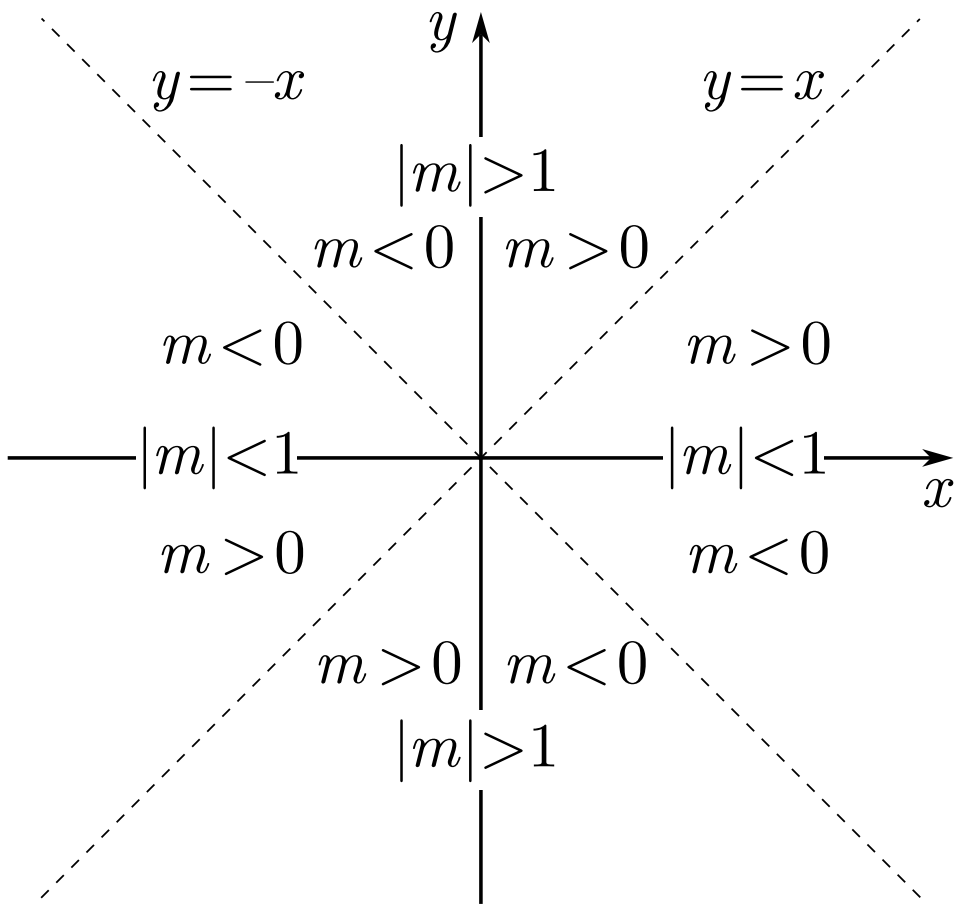
- Priamka (rôznobežná s osou y) určená dvomi rôznymi bodmi $A(a_x, a_y)$, $B(b_x, b_y)$ s celočíselnými súradnicami, je zvyčajne vyjadrená pomocou rovnice $y = a_y + \frac{b_y - a_y}{b_x - a_x}(x - a_x) = mx + b$, kde $m = \frac{b_y - a_y}{b_x - a_x} = \frac{\Delta y}{\Delta x}$ a $b = a_y - ma_x$
- Pre rasterizáciu úsečky je dobré používať celočíselnú aritmetiku, ktorá je menej náročná ako aritmetika pohyblivej desatinnej čiarky (*floatpoint*)



Digitálny diferenciálny analyzátor (DDA)

- Predstavuje algoritmus pre výpočet celočíselných polôh pixlov úsečky na základe rovnice $\Delta y = m\Delta x \Rightarrow dy = mdx$, kde $dy = y_{k+1} - y_k$ a $dx = x_{k+1} - x_k$ so začiatočným bodom $(x_0, y_0) = (a_x, a_y)$
- Vo výpočte pixlov úsečky sa volí jednotkový krok (± 1 pixel) dx alebo dy v smere „vhodnej“ súradnicovej osi, aby body vykresľovanej úsečky ležali v celočíselnom rastri
- Aby pri tomto postupe nevznikali medzery na zobrazených úsečkách, nesmie zobrazovaná úsečka prudko stúpať/klesať vzhľadom k osi, na ktorej je zvolený jednotkový krok. Ak tento prípad nastane, je treba vykonávať jednotkový krok na inej súradnicovej osi
- Jednotkový krok sa zvolí podľa toho, v akej oblasti vykresľovaná úsečka leží, pričom hranice oblastí predstavujú priamky $y = x$ a $y = -x$, v ktorých sa mení absolútna hodnota sklonu úsečiek z hodnoty $|m| < 1$ na $|m| > 1$
- Znamienko jednotkového kroku „+“ / „–“ sa zasa zvolí podľa polohy druhého koncového bodu, resp. podľa znamienka Δx či Δy
- Zvislé úsečky, ktoré nemajú definovaný sklon m, treba riešiť samostatne

Digitálny diferenciálny analyzátor (DDA)



Algoritmus DDA

- Ak je $\Delta x > 0$ a $|m| <= 1$, zvolíme krok na osi x s hodnotou $dx = 1$. Keďže $dy = mdx = m$, dostávame pre výpočet bodov úsečky vzťah:

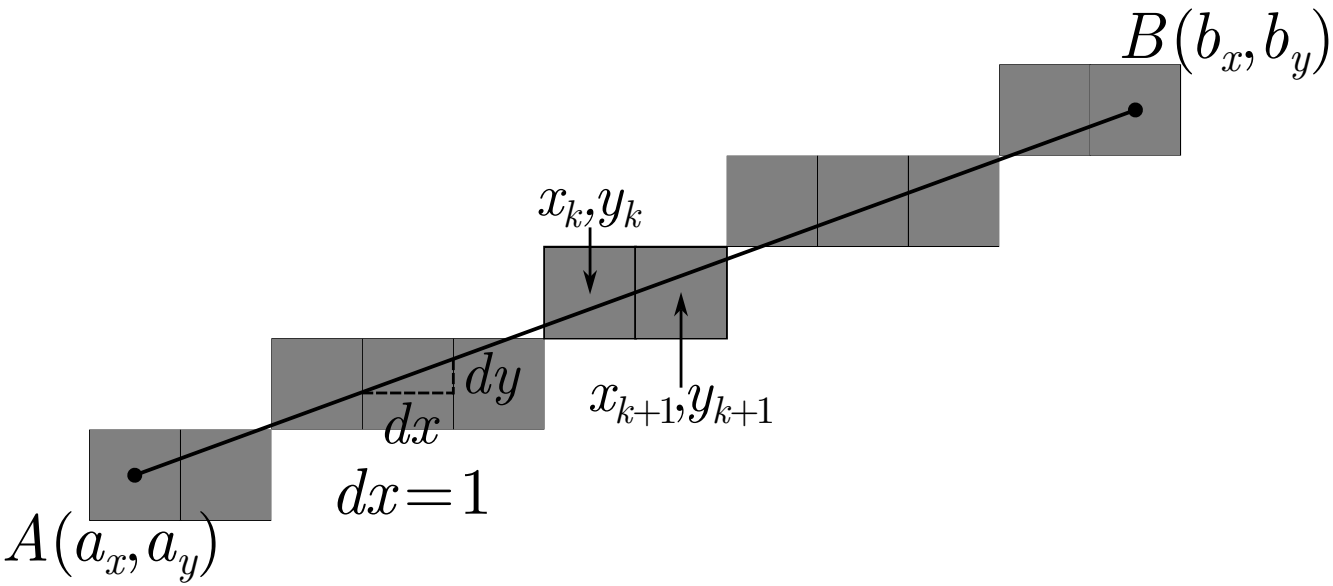
$$x_{k+1} = x_k + dx = x_k + 1$$

$$y_{k+1} = y_k + dy = y_k + mdx = y_k + m$$

- Ak je $\Delta x < 0$ a $|m| <= 1$, zvolíme krok na osi x s hodnotou $dx = -1$. Keďže $dy = mdx = -m$, dostávame pre výpočet bodov úsečky vzťah:

$$x_{k+1} = x_k + dx = x_k - 1$$

$$y_{k+1} = y_k + dy = y_k + mdx = y_k - m$$



Algoritmus DDA

- Ak je $\Delta y > 0$ a $|m| > 1$, zvolíme krok na osi y s hodnotou $dy = 1$. Keďže $dx = dy/m = 1/m$, dostávame pre výpočet bodov úsečky vzťah:

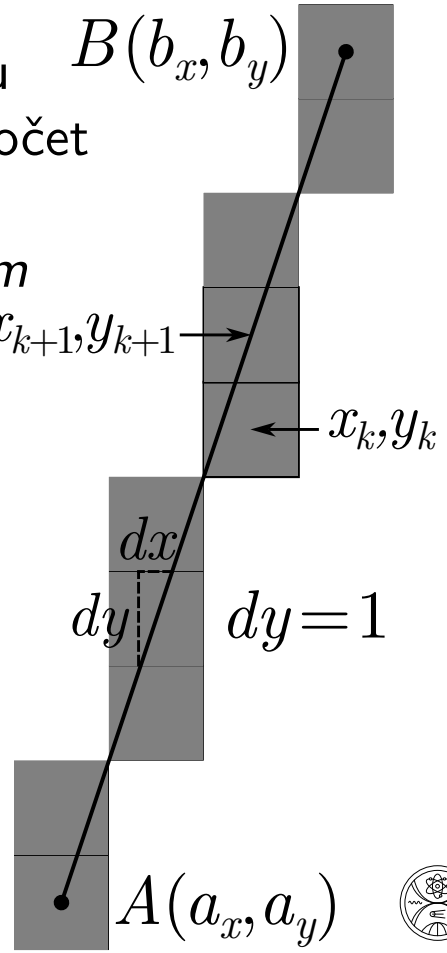
$$x_{k+1} = x_k + dx = x_k + dy/m = x_k + 1/m$$

$$y_{k+1} = y_k + dy = y_k + 1$$

- Ak je $\Delta y < 0$ a $|m| > 1$, zvolíme krok na osi y s hodnotou $dy = -1$. Keďže $dx = dy/m = -1/m$, dostávame pre výpočet bodov úsečky vzťah:

$$x_{k+1} = x_k + dx = x_k + dy/m = x_k - 1/m$$

$$y_{k+1} = y_k + dy = y_k - 1$$



Algoritmus DDA

- Súradnice výsledného vypočítaného bodu úsečky (x_k, y_k) je ešte treba pred vykreslením **zaokrúhliť**
- DDA algoritmus je efektívnejšia metóda pre výpočet polôh pixlov ako priame použitie rovnice priamky $y = mx + b$, keďže namiesto násobenia používa iba operácie sčítania a odčítania
- Výpočty sú ale menej efektívne ako v nasledujúcom Bresenhamovom algoritme, keďže používajú aritmetiku pohyblivej desatinnej čiarky a zaokruhľovacie operácie `round()`

Algoritmus DDA – príklad implementácie

```
procedure dda(ax, ay, bx, by: integer);
deltax, deltay, steps, k: integer;
xinc, yinc, x, y: real;
begin
  deltax := bx - ax;
  deltay := by - ay;
  if abs(deltax) > abs(deltay) then steps := abs(deltax)
  else steps := abs(deltay);
  xinc := deltax / steps;
  yinc := deltay / steps;
  x := ax; y := ay;
  drawpixel(round(x), round(y));
  for k := 1 to steps do begin
    x := x + xinc;
    y := y + yinc;
    drawpixel(round(x), round(y));
  end
end
```

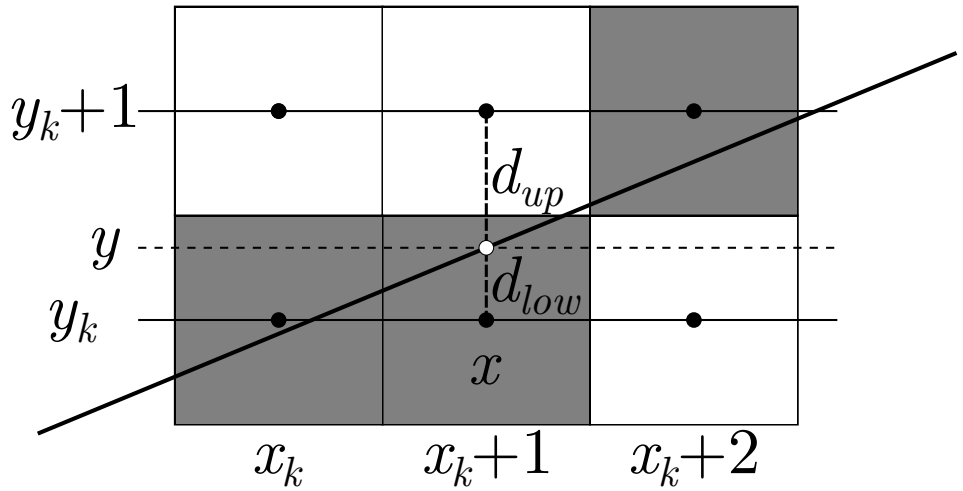
Bresenhamov algoritmus

- Algoritmus počíta celočíselné body úsečky, ktoré sa nachádzajú najbližšie ku „geometrickému“ obrazu úsečky v zvislom smere (y-ovej súradnice)
- Výpočty sú uskutočňované výlučne pomocou **celočíselnej** aritmetiky, preto je tento algoritmus veľmi efektívny a často implementovaný
- Budeme predpokladať, že vykresľovaná úsečka leží v prvom oktante, t.j. $\Delta x > 0$ a $0 < m < 1$, v inom prípade využijeme symetriu, t.j. súmernosť podľa priamok $y = x$ a $y = -x$:
  - pre 2. oktant $x' = +y$, $y' = +x$
  - pre 3. oktant $x' = +y$, $y' = -x$
  - pre 4. oktant $x' = -x$, $y' = +y$
  - pre 5. oktant $x' = -x$, $y' = -y$
  - pre 6. oktant $x' = -y$, $y' = -x$
  - pre 7. oktant $x' = -y$, $y' = +x$
  - pre 8. oktant $x' = +x$, $y' = -y$
- Keďže pre vykresľovanú úsečku platí, že $\Delta x > 0$ a $0 < m < 1$, zvolí sa prírastok na osi x s hodnotou $dx = 1$. Z toho dostaneme $x_{k+1} = x_k + dx = x_k + 1$
- Hodnota prírastku $dy = mdx = m = \Delta y / \Delta x$ nie je celočíselná, preto výpočet novej súradnice $y_{k+1} = y_k + dy$ budeme realizovať inak

Bresenhamov algoritmus

- Po vykreslenom bode úsečky so súradnicami (x_k, y_k) je vzhľadom na jednotkový prírastok v smere osi x vykreslený, buď bod $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k + 1)$, podľa toho, ktorý je bližšie ku geometrickému priesečníku danej úsečky so spojnicou stredov týchto pixlov

1. Vypočítame reálnu hodnotu súradnice y pre $x_{k+1} = x_k + 1$:
$$y = mx + b = mx_{k+1} + b = m(x_k + 1) + b = mx_k + m + b$$
2. Z toho dostaneme $d_{low} = y - y_k = mx_k + m + b - y_k$ a $d_{up} = y_k + 1 - y = y_k + 1 - (mx_k + m + b)$
3. Vyčíslime rozdiel $\Delta d = d_{low} - d_{up} = (mx_k + m + b - y_k) - (y_k + 1 - (mx_k + m + b)) = 2m(x_k + 1) - 2y_k + 2b - 1$



Bresenhamov algoritmus

- Ak hodnota $\Delta d < 0$ ($d_{low} < d_{up}$), nová y súradnica bude $y_{k+1} = y_k$
- Ak hodnota $\Delta d > 0$ ($d_{low} > d_{up}$), nová y súradnica bude $y_{k+1} = y_k + 1$
- Pre určenie, ktorá z dvoch y_{k+1} to bude, stačí určiť znamienko Δd, ktoré ešte vynásobíme kladnou hodnotou Δx, čím dostaneme $\Delta d \Delta x = 2m\Delta x(x_k + 1) - 2\Delta x y_k + \Delta x(2b - 1) = 2\Delta y(x_k + 1) - 2\Delta x y_k + \Delta x(2b - 1)$
- Označme $\Delta d \Delta x$ ako p_k, pričom znamienko p_k bude určovať, ktorý bod vykresliť

Bresenhamov algoritmus

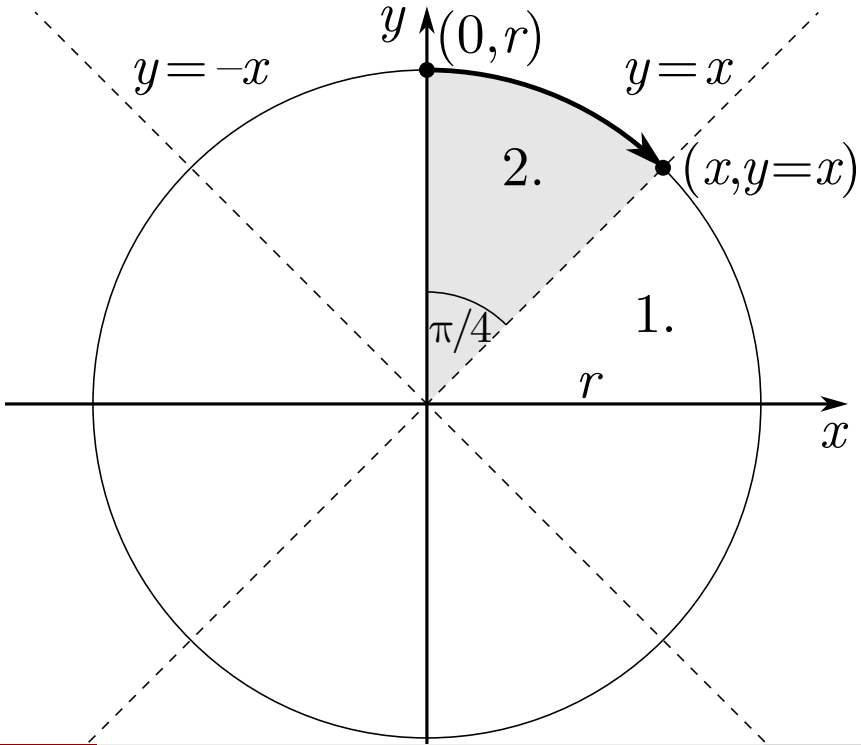
- Pre určenie prírastku p_k vypočítame rozdiel $p_{k+1} - p_k$:
  1. $p_k = 2\Delta y x_k - 2\Delta x y_k + C$, kde $C = 2\Delta y + \Delta x(2b - 1)$
  2. $p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C = 2\Delta y x_k - 2\Delta x y_{k+1} + 2\Delta y + C$
  3. $p_{k+1} - p_k = (2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C) - (2\Delta y x_k - 2\Delta x y_k + C) = -2\Delta x(y_{k+1} - y_k) + 2\Delta y$, keďže $x_{k+1} = x_k + 1$
  4. $\Rightarrow p_{k+1} = p_k - 2\Delta x(y_{k+1} - y_k) + 2\Delta y$
- Pre prvú hodnotu p_0 dostaneme $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2b - 1) = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2(y_0 - m x_0) - 1) = 2\Delta y - \Delta x$, keďže $-2m\Delta x x_0 = -2\Delta y x_0 \Leftarrow m\Delta x = \Delta y$
- Na základe znamienka hodnoty p_k vieme určiť, či sa bude vykresľovať bod $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k + 1)$:
  - ak $p_k < 0$ ($d_{low} < d_{up}$), potom $y_{k+1} = y_k$ a $p_{k+1} = p_k + 2\Delta y$
  - ak $p_k \geq 0$ ($d_{low} \geq d_{up}$), potom $y_{k+1} = y_k + 1$ a $p_{k+1} = p_k + 2(\Delta y - \Delta x)$
- **Ostatné** (jednoduché) prípady, keď $\Delta x = 0$ (zvislá úsečka) alebo $\Delta y = 0$ (vodorovná úsečka) alebo $\Delta x = \Delta y$ (diagonálna úsečka) sa riešia **osobitne**

Bresenhamov algoritmus pre 1. oktant

```
procedure bresenham(ax, ay, bx, by: integer);
p, c1, c2, deltax, deltay, x, y: integer;
begin
  deltax := bx - ax;
  deltay := by - ay;
  c1 := 2 * deltay;
  c2 := 2 * (deltay - deltax);
  p := 2 * deltay - deltax;
  x := ax; y := ay;
  drawpixel(x, y);
  while x <= bx do begin
    x := x + 1;
    if p >= 0 then begin
      p := p + c2;
      y := y + 1;
      end
    else
      p := p + c1
    drawpixel(x, y);
  end
end
```

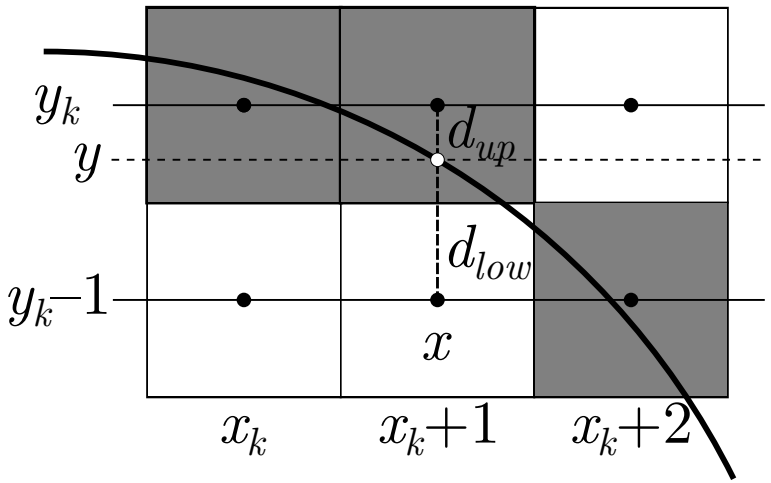
Bresenhamov algoritmus pre kreslenie kružnice

- Výpočet pixlov kružnice využíva rovnakú myšlienku ako pre vykreslenie úsečky, z dvoch pixlov algoritmus vyberie ten, ktorý je bližšie k danej kružnici
- Na vykreslenie kružnice $\mathcal{K}$: $x^2 + y^2 - r^2 = 0$ sa počíta iba časť v 2. oktante
- Ostatné segmenty sa kreslia s použitím symetrie ($y = x$ a $y = -x$)
- Stred kružnice bude v $(0, 0)$, ak v ňom nie je, použije sa posunutie
- Vykresľovanie začína v bode $(0, r)$ a končí v bode, pre ktorý je $x = y$, jednotkový krok pre vyčísľovanie bodov bude $dx = 1$, t.j. $x_{k+1} = x_k + 1$



Bresenhamov algoritmus pre kreslenie kružnice

- Po vykreslenom bode (x_k, y_k) nasleduje, buď bod $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k - 1)$, podľa toho, ktorý je bližšie ku priesečníku danej kružnice so spojnicou stredov týchto pixlov
  1. Vypočítame reálnu hodnotu súradnice y^2 pre $x_{k+1} = x_k + 1$:
  $$y^2 = r^2 - x_{k+1}^2 = r^2 - (x_k + 1)^2$$
  2. Z toho dostaneme $d_{low} = y^2 - (y_k - 1)^2 = r^2 - (x_k + 1)^2 - (y_k - 1)^2$ a $d_{up} = y_k^2 - y^2 = y_k^2 - r^2 + (x_k + 1)^2$
  3. Vyčíslime rozdiel $\Delta d = d_{up} - d_{low} = 2(x_k + 1)^2 + y_k^2 + (y_k - 1)^2 - 2r^2$
- Ak hodnota $\Delta d < 0$ $(d_{up} < d_{low})$, nová y súradnica bude $y_{k+1} = y_k$
- Ak hodnota $\Delta d \geq 0$ $(d_{up} \geq d_{low})$, nová y súradnica bude $y_{k+1} = y_k - 1$
- Pre určenie, ktorá z dvoch hodnôt y_{k+1} to bude, stačí určiť znamienko Δd



Bresenhamov algoritmus pre kreslenie kružnice

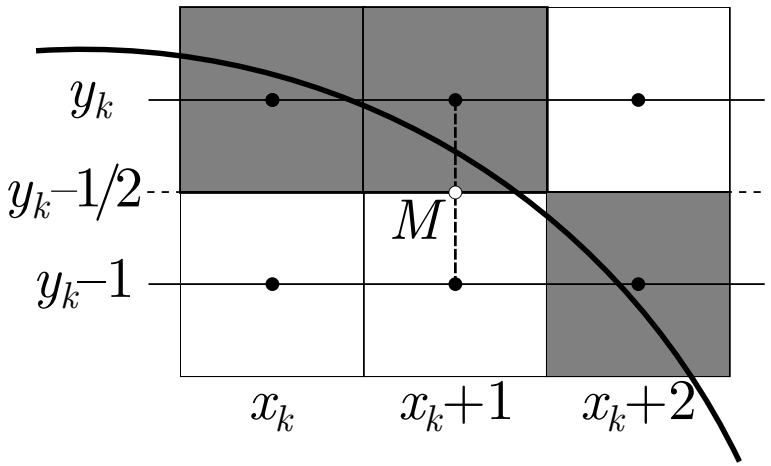
- Označme Δd ako p_k, pričom znamienko p_k bude určovať, ktorý bod vykresliť
- Pre určenie prírastku p_k vypočítame rozdiel $p_{k+1} - p_k$:
  1. $p_k = 2(x_k + 1)^2 + y_k^2 + (y_k - 1)^2 - 2r^2$
  2. $p_{k+1} = 2(x_{k+1} + 1)^2 + y_{k+1}^2 + (y_{k+1} - 1)^2 - 2r^2$
  3. $p_{k+1} - p_k = 4x_k + 6 + 2(y_{k+1}^2 - y_k^2) - 2(y_{k+1} - y_k)$, keďže $x_{k+1} = x_k + 1$
  4. $\Rightarrow p_{k+1} = p_k + 4x_k + 6 + 2(y_{k+1}^2 - y_k^2) - 2(y_{k+1} - y_k)$
- Pre prvú hodnotu p_0 dostaneme $p_0 = 2(x_0 + 1)^2 + y_0^2 + (y_0 - 1)^2 - 2r^2 = 2(0 + 1)^2 + r^2 + (r - 1)^2 - 2r^2 = 3 - 2r$, keďže $(x_0, y_0) = (0, r)$
- Na základe znamienka hodnoty p_k vieme určiť, či sa bude vykresľovať bod $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k - 1)$:
  - ak $p_k < 0$ $(d_{up} < d_{low})$, potom $y_{k+1} = y_k$ a $p_{k+1} = p_k + 4x_k + 6$
  - ak $p_k \geq 0$ $(d_{up} \geq d_{low})$, potom $y_{k+1} = y_k - 1$ a $p_{k+1} = p_k + 4(x_k - y_k) + 10$

Bresenhamov algoritmus pre kreslenie kružnice

```
procedure bresenham_cirle(xc, yc, r: integer);
p, x, y: integer;
procedure draw_circle_points();
begin
  drawpixel(xc + x, yc + y); drawpixel(xc - x, yc + y);
  drawpixel(xc + x, yc - y); drawpixel(xc - x, yc - y);
  drawpixel(xc + y, yc + x); drawpixel(xc - y, yc + x);
  drawpixel(xc + y, yc - x); drawpixel(xc - y, yc - x);
end;
begin
  x := 0;
  y := r;
  p := 3 - 2*r;
  while x < y do begin
    draw_circle_points();
    if p < 0 then p := p + 4*x + 6
    else begin
      p := p + 4*(x - y) + 10;
      y := y - 1;
    end;
    x := x + 1;
  end
  if x == y then draw_circle_points();
end
```

Midpoint algoritmus pre kreslenie kružnice

- Predstavuje úpravu Bresenhamovho algoritmu, v ktorej rozhoduje o výbere ďalšieho vykresľovaného bodu poloha tzv. „stredného“ bodu $M = (x_{k+1}, y_k - 1/2)$, ktorý leží v prostriedku medzi dvoma možnými kandidátmi (x_{k+1}, y_k) a $(x_{k+1}, y_k - 1)$
- Ak je M vnútri danej kružnice, potom nasledujúci bod bude (x_{k+1}, y_k)
- Ak je M mimo danej kružnice, potom nasledujúci bod bude $(x_{k+1}, y_k - 1)$
- Pre určenie či bod leží v kružnici alebo mimo, použijeme implicitnú rovnicu kružnice $f(x, y) : x^2 + y^2 - r^2$, kde $f(x, y) = 0$
- Bod (x, y) leží v kružnici, ak $f(x, y) < 0$, leží mimo kružnice, ak $f(x, y) > 0$



Midpoint algoritmus pre kreslenie kružnice

- Označme hodnotu $f(M) = f(x_k + 1, y_k - 1/2)$ ako p_k, pričom znamienko p_k bude určovať, ktorý nasledujúci bod kružnice vykresliť
- Pre určenie prírastku p_k vypočítame rozdiel $p_{k+1} - p_k$:
  1. $p_k = (x_k + 1)^2 + (y_k - 1/2)^2 - r^2$
  2. $p_{k+1} = (x_{k+1} + 1)^2 + (y_{k+1} - 1/2)^2 - r^2$
  3. $p_{k+1} - p_k = (x_k + 2)^2 + (y_{k+1} - 1/2)^2 - (x_k + 1)^2 - (y_k - 1/2)^2 = 2x_k + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 3$
  4. $\Rightarrow p_{k+1} = p_k + 2x_k + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 3$
- Pre prvú hodnotu p_0 dostaneme $p_0 = (x_0 + 1)^2 + (y_0 - 1/2)^2 - r^2 = (0 + 1)^2 + (r - 1/2)^2 - r^2 = 5/4 - r \approx 1 - r$, keďže $(x_0, y_0) = (0, r)$
- Na základe znamienka hodnoty p_k vieme určiť, či sa bude vykresľovať bod $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k - 1)$:
  - ak $p_k < 0$ ($f(M) < 0$), potom $y_{k+1} = y_k$ a $p_{k+1} = p_k + 2x_k + 3$
  - ak $p_k \geq 0$ ($f(M) \geq 0$), potom $y_{k+1} = y_k - 1$ a $p_{k+1} = p_k + 2(x_k - y_k) + 5$

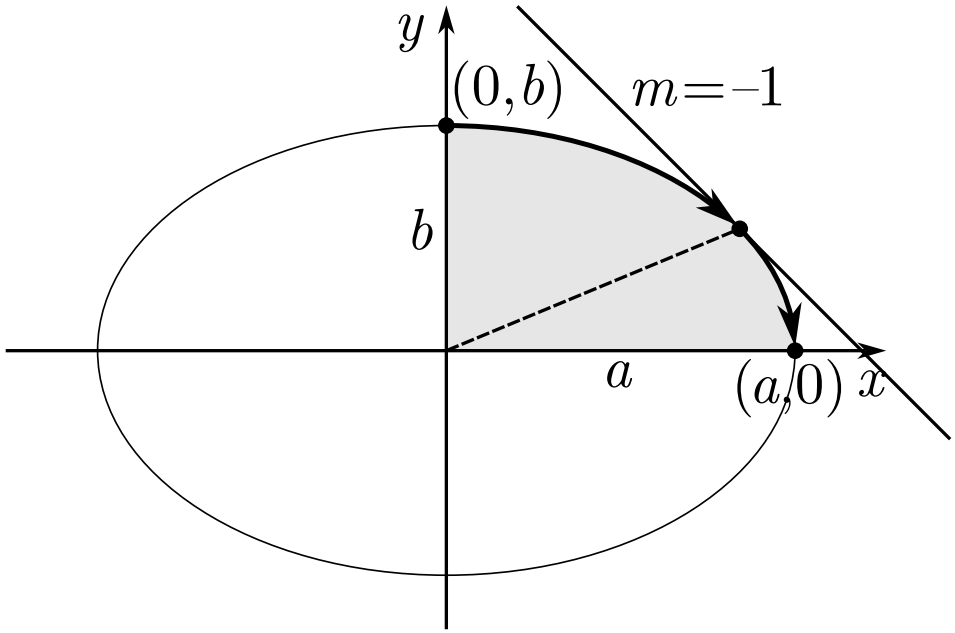
Midpoint algoritmus pre kreslenie kružnice

```
procedure midpoint_cirle(xc, yc, r: integer);
p, x, y: integer;
procedure draw_circle_points();
begin
  drawpixel(xc + x, yc + y); drawpixel(xc - x, yc + y);
  drawpixel(xc + x, yc - y); drawpixel(xc - x, yc - y);
  drawpixel(xc + y, yc + x); drawpixel(xc - y, yc + x);
  drawpixel(xc + y, yc - x); drawpixel(xc - y, yc - x);
end;
begin
  x := 0;
  y := r;
  p := 1 - r;
  while x < y do begin
    draw_circle_points();
    if p < 0 then p := p + 2*x + 3
    else begin
      p := p + 2*(x - y) + 5;
      y := y - 1;
    end;
    x := x + 1;
  end
  if x == y then draw_circle_points();
end
```

Midpoint algoritmus pre kreslenie elipsy

- Postup je podobný ako u midpoint algoritmu pre kreslenie kružnice
- Použijeme implicitnú rovnicu elipsy $f(x, y)$: $b^2x^2 + a^2y^2 - a^2b^2$, kde $f(x, y) = 0$
- Pre elipsu, ktorej osi nie sú rovnobežné so súradnicovými osami, je treba aplikovať rotáciu
- Pre elipsu so stredom inde ako v začiatku $(0, 0)$ použijeme posunutie
- Algoritmus vykresľuje elipsu v 1. kvadrante postupne v **dvoch** oblastiach, ktoré rozdeľuje bod elipsy, v ktorom jej dotyčnica má smernicu $m = -1$
- Ostatné segmenty elipsy sa vykresľujú s použitím symetrie ($x = 0$ a $y = 0$)
- Vykresľovanie začína v bode $(0, b)$ elipsy s jednotkovým krokom v smere osi x v prvej oblasti, až kým sa vykreslí bod na hranici medzi oblasťami
- Potom sa vykresľujú body v druhej oblasti s jednotkovým krokom v smere zápornej osi y, až kým nie je vykreslený celý 1. kvadrant

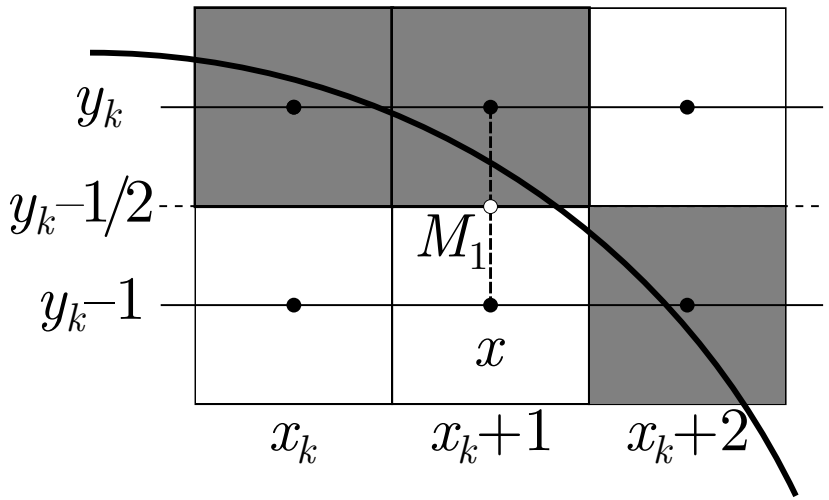
Midpoint algoritmus pre kreslenie elipsy

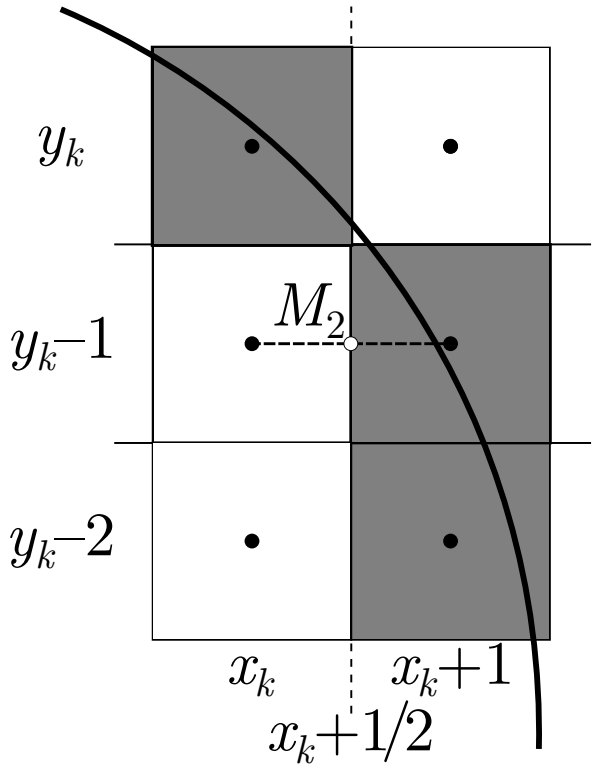


- Smer dotyčnice elipsy v bode (x, y) je **kolmý** na normálu $\left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right)$ a má vyjadrenie $\left(\frac{\partial f(x,y)}{\partial y}, -\frac{\partial f(x,y)}{\partial x}\right) = (2a^2y, -2b^2x) = (dx, dy)$
- Pre smernicu m dotyčnice v bode hranice elipsy medzi dvoma oblasťami platí: $m = \frac{dy}{dx} = -1 = \frac{-2b^2x}{2a^2y} = -\frac{b^2x}{a^2y}$

Midpoint algoritmus pre kreslenie elipsy

- V prvej oblasti pre smernicu m platí $-1 \leq -\frac{b^2 x}{a^2 y} \leq 0 \Rightarrow b^2 x \leq a^2 y$ a nasledujúci vykresľovaný bod je buď $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k - 1)$. Za stredný bod sa zvolí $M_1 = (x_{k+1}, y_k - 1/2)$, pričom $x_{k+1} = x_k + 1$
- V druhej oblasti pre smernicu m platí $-\infty < -\frac{b^2 x}{a^2 y} < -1 \Rightarrow b^2 x > a^2 y$ a nasledujúci vykresľovaný bod je buď $(x_k, y_k - 1)$ alebo $(x_k + 1, y_k - 1)$. Za stredný bod sa zvolí $M_2 = (x_k + 1/2, y_{k+1})$, pričom $y_{k+1} = y_k - 1$





Midpoint algoritmus pre kreslenie elipsy 1. oblasť

- Označme hodnotu $f(M_1) = f(x_k + 1, y_k - 1/2)$ ako ${}^1p_k$, pričom znamienko ${}^1p_k$ bude určovať, ktorý nasledujúci bod elipsy vykresliť
- Pre určenie prírastku ${}^1p_k$ vypočítame rozdiel ${}^1p_{k+1} - {}^1p_k$:
  1. ${}^1p_k = b^2(x_k + 1)^2 + a^2(y_k - 1/2)^2 - a^2b^2$
  2. ${}^1p_{k+1} = b^2(x_{k+1} + 1)^2 + a^2(y_{k+1} - 1/2)^2 - a^2b^2$
  3. ${}^1p_{k+1} - {}^1p_k = b^2(x_k + 2)^2 + a^2(y_{k+1} - 1/2)^2 - b^2(x_k + 1)^2 - a^2(y_k - 1/2)^2 = 2b^2x_k + a^2((y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)) + 3b^2$
  4. $\Rightarrow {}^1p_{k+1} = {}^1p_k + 2b^2x_k + a^2((y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)) + 3b^2$
- Pre prvú hodnotu ${}^1p_0$ dostaneme ${}^1p_0 = b^2(x_0 + 1)^2 + a^2(y_0 - 1/2)^2 - a^2b^2 = b^2(0 + 1)^2 + a^2(b - 1/2)^2 - a^2b^2 = b^2 - a^2b + a^2/4$, keďže $(x_0, y_0) = (0, b)$
- Na základe znamienka hodnoty ${}^1p_k$ vieme určiť, či sa bude vykresľovať bod $(x_k + 1, y_k)$ alebo $(x_k + 1, y_k - 1)$:
  - ${}^1p_k < 0$ $(f(M_1) < 0) \Rightarrow y_{k+1} = y_k$, ${}^1p_{k+1} = {}^1p_k + 2b^2x_k + 3b^2$
  - ${}^1p_k \geq 0$ $(f(M_1) \geq 0) \Rightarrow y_{k+1} = y_k - 1$, ${}^1p_{k+1} = {}^1p_k + 2(b^2x_k - a^2y_k) + 2a^2 + 3b^2$

Midpoint algoritmus pre kreslenie elipsy 2. oblasť

- Označme hodnotu $f(M_2) = f(x_k + 1/2, y_k - 1)$ ako ${}^2p_k$, pričom znamienko ${}^2p_k$ bude určovať, ktorý nasledujúci bod elipsy vykresliť
- Pre určenie prírastku ${}^2p_k$ vypočítame rozdiel ${}^2p_{k+1} - {}^2p_k$:
  1. ${}^2p_k = b^2(x_k + 1/2)^2 + a^2(y_k - 1)^2 - a^2b^2$
  2. ${}^2p_{k+1} = b^2(x_{k+1} + 1/2)^2 + a^2(y_k - 2)^2 - a^2b^2$
  3. ${}^2p_{k+1} - {}^2p_k = b^2(x_{k+1} + 1/2)^2 + a^2(y_k - 2)^2 - b^2(x_k + 1/2)^2 - a^2(y_k - 1)^2 = -2a^2y_k + b^2((x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)) + 3a^2$
  4. $\Rightarrow {}^2p_{k+1} = {}^2p_k - 2a^2y_k + b^2((x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)) + 3a^2$
- Pre prvú hodnotu ${}^2p_0$ dostaneme ${}^2p_0 = b^2(x_k + 1)^2 + a^2(y_k - 1/2)^2 - a^2b^2 = a^2(y_k^2 - y_k) + b^2(x_k + 1)^2 - a^2b^2 + a^2/4$, pričom sme použili posledný bod (x_k, y_k) z 1. oblasti
- Na základe znamienka hodnoty ${}^2p_k$ vieme určiť, či sa bude vykresľovať bod $(x_k, y_k - 1)$ alebo $(x_k + 1, y_k - 1)$:
  - ${}^2p_k < 0$ $(f(M_2) < 0) \Rightarrow x_{k+1} = x_k$, ${}^2p_{k+1} = {}^2p_k - 2a^2y_k + 3a^2$
  - ${}^2p_k \geq 0$ $(f(M_2) \geq 0) \Rightarrow x_{k+1} = x_k + 1$, ${}^2p_{k+1} = {}^2p_k + 2(b^2x_k - a^2y_k) + 2b^2 + 3a^2$

Midpoint algoritmus pre kreslenie elipsy

```
procedure midpoint_ellipse(xc, yc, a, b: integer);
p, x, y: integer;
procedure draw_ellipse_points();
begin
  drawpixel(xc + x, yc + y); drawpixel(xc - x, yc + y);
  drawpixel(xc + x, yc - y); drawpixel(xc - x, yc - y);
end
begin
  x := 0; y := b;
  p := round(b*b - a*a*b + a*a/4);
  while b*b*x < a*a*y do begin
    draw_ellipse_points();
    if p < 0 then p := p + 2*b*b*x + 3*b*b
    else begin
      p := p + 2*(b*b*x - a*a*y) + 2*a*a + 3*b*b;
      y := y - 1;
    end
    x := x + 1;
  end
  p := round(a*a*(y*y - y) + b*b*(x + 1)*(x + 1) - a*a*b*b + a*a/4);
  while y > 0 do begin
    draw_ellipse_points();
    if p < 0 then p := p - 2*a*a*y + 3*a*a
    else begin
      p := p + 2*(b*b*x - a*a*y) + 2*b*b + 3*a*a;
      x := x - 1;
    end
    y := y - 1;
  end
```

Rasterizácia mnohouholníka

- Pri rasterizácii mnohouholníka budeme rozumieť vykreslenie nielen hranice mnohouholníka ale aj jeho vnútro
- Prístup, v ktorom sa pomocou Bresenhamovho algoritmu vykreslia hraničné úsečky mnohouholníka a potom sa vyplní jeho vnútro je neefektívny, preto sa namiesto toho používa tzv. algoritmus zametacej (skenovacej – *scanline*) vodorovnej či zvislej priamky

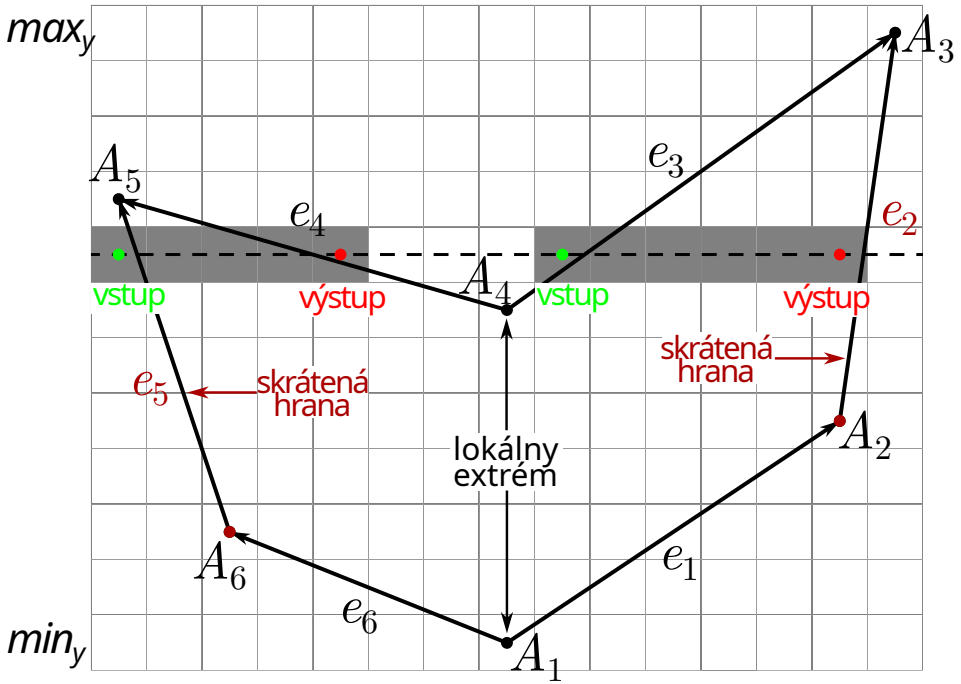
Scanline algoritmus

- Algoritmus využíva princíp koherentnosti, t.j. pri vykresľovaní nasledujúceho riadku pixlov sa situácia „veľmi“ nezmení a využijú sa informácie z predošlého kroku, čo o.i. znamená, že susedné body majú vzhľadom na mnohouholník rovnakú polohu (vnútri/vonku) ako v predošlom riadku
- Skenovacia priamka bude vodorovná s celočíselnými y-súradnicami, pričom jej rovnica bude $y = y_k$, kde y_k nadobúda celočíselné hodnoty od najmenšej min_y po najväčšiu max_y súradnicu vrcholov daného mnohouholníka
- Algoritmus funguje tak, že v každom kroku zistí prieniky skenovacej priamky so stranami mnohouholníka, usporiada ich do dvojíc a úseky medzi dvojicami vykreslí farbou vnútra

Základný scanline algoritmus (bez koherencie, neefektívny)

Pre každý skenovací riadok od min_y do max_y:

1. nájdi priesečníky hrán mnohouholníka so skenovacou priamkou:
   ak sa na skenovacej priamke nachádza rovnobežná hrana, vykresli ju celú
   ak má skenovacia priamka prienik s hranou mnohouholníka vo vrchole:
      ak tento vrchol nie je v „lokálnom extréme“, vytvor jeden priesečník
      ak tento vrchol tvorí „lokálny extrém“, vytvor dva priesečníky
   ak skenovacia priamka nemá prienik s hranou mnohouholníka vo vrchole:
      vytvor jeden priesečník
2. usporiadaj priesečníky podľa x-ovej súradnice
3. vykresli pixle, ktoré sú vnútri podľa princípu parity (vstup/výstup)



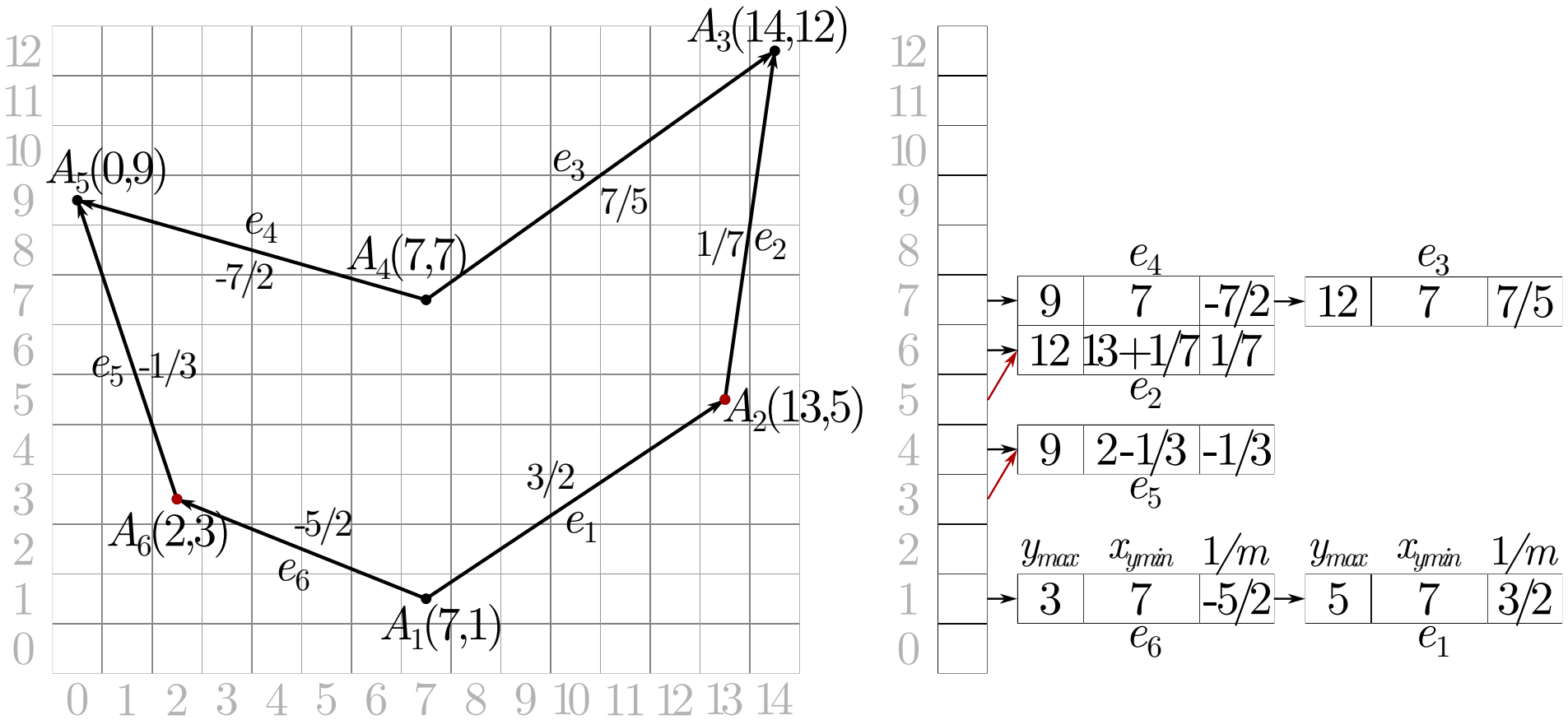
Scanline algoritmus

- Tento algoritmus bude využívať smernicu m jednotlivých hrán (úsečiek) mnohouholníka, ktoré môžeme zapísať v tvare $y = mx + b$, pričom $m = \frac{\Delta y}{\Delta x} = \frac{dy}{dx}$
- Keďže skenovacia priamka postupuje po jednom riadku (prírastok $dy = 1$) od y_{min} do y_{max}, bude algoritmus využívať podobný postup ako bol uvedený v algoritme DDA, t.j. $y_{k+1} = y_k + dy = y_k + 1$ a keďže $dx = 1/m$, potom $x_{k+1} = x_k + dx = x_k + 1/m$
- Algoritmus využíva dve dátové polia linkovaných zoznamov, tzv. **tabuľku hrán (TH)** a **tabuľku aktívnych hrán (TAH)** s položkami, ktoré obsahujú:
  1. **maximálnu y-ovú súradnicu hrany** (y_{max}), ktorá určuje dokedy treba testovať prienik skenovacej priamky s touto hranou
  2. **x-ovú súradnicu hrany s minimálnou y-ovou súradnicou** (x_{ymin}), ktorá predstavuje x-ovú súradnicu bodu hrany, v ktorom skenovacia priamka prvýkrát pretne danú hranu
  3. **hodnotu $1/m$**, ktorá predstavuje prevrátenú hodnotu smernice priamky, na ktorej hrana leží
- Hrany v zoznamoch sú **naorientované** podľa minimálnej y-ovej súradnice

Scanline algoritmus – TH, TAH

- Pri prieniku skenovacej priamky s hranami mnohouholníka potrebujeme získať iba správny počet prienikov, musíme preto zo zoznamu hrán mnohouholníka vylúčiť vodorovné hrany a zdola (o 1 pixel) skrátiť hrany v neextremálnych vrcholoch
- Tabuľka hrán sa vytvára pri inicializácii, má veľkosť $max_y - min_y + 1$ a má neprázdny obsah iba na tých riadkoch, kde dané hrany majú minimálnu y-ovú súradnicu, t.j. kde sa nachádza dolný vrchol hrany, keďže všetky hrany sú usporiadané podľa minimálnej y-ovej súradnice
- Ak je vrchol hrany s menšou y-ovou súradnicou neextremálny, bude hrana zapísaná až v ďalšom riadku TH (pretože je zdola skrátená), navyše k druhej hodnote záznamu x_{ymin} pripočítame zlomok $1/m$
- Tabuľka aktívnych hrán je zoznam hrán, s ktorými má aktuálna skenovacia priamka prienik a vytvára sa počas behu algoritmu vyberaním relevantných záznamov hrán z TH
- V TAH sa druhá položka záznamu x_{ymin} každej hrany aktualizuje tak, aby vždy mala hodnotu x-ovej súradnice prieniku hrany a skenovacej priamky

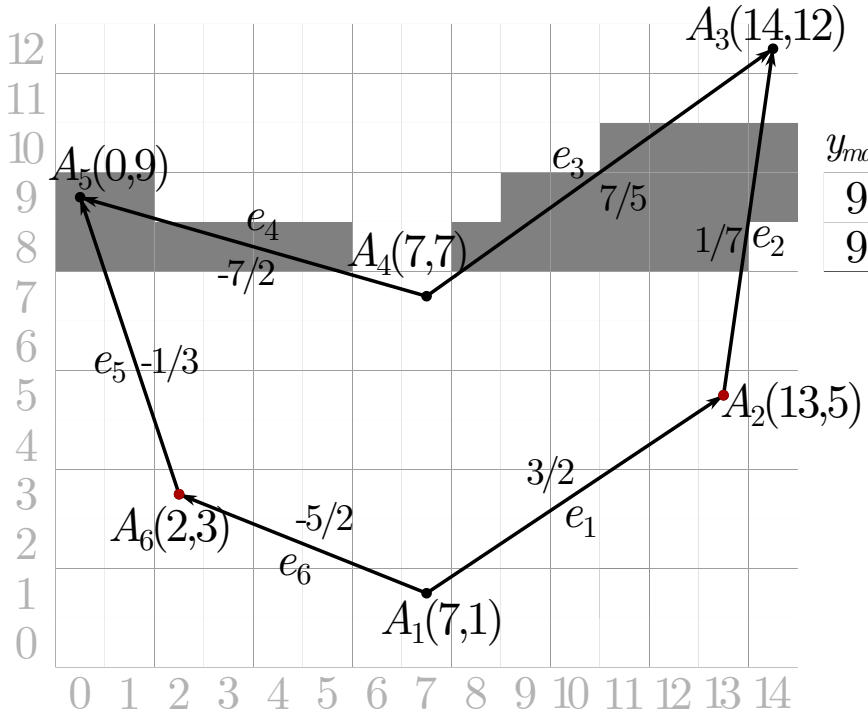
Scanline algoritmus – TH



Scanline algoritmus

1. Vyraď všetky vodorovné hrany mnohouholníka z ďalšieho uvažovania
2. Nájdi všetky neextremálne vrcholy a skráť ich príslušné hrany zdola o 1 pixel
3. Vytvor TH, inicializuj TAH ako prázdnu
4. Pre všetky y od min_y do max_y:
5. skopíruj hrany z TH do TAH pre aktuálny riadok y, položka x_{ymin} bude x // TAH obsahuje všetky hrany, s ktorými sa skenovacia priamka pretína
6. usporiadaj hrany v TAH podľa x-ovej súradnice (druhá položka zoznamu)
7. vytvor dvojice súradníc podľa princípu parity (vstup/výstup)
8. vykresli pixle vo vnútorných častiach pozdĺž skenovacej priamky
9. vymaž tie hrany v TAH, pre ktoré je $y = y_{max}$ // hrany sa už celé použili
10. zmeň položku x v TAH na hodnotu $x + 1/m$
11. $y = y + 1$ // prejdi na nasledujúci skenovací riadok
12. Vykresli všetky hrany mnohouholníka // ak sa pri vypĺňaní vodorovných úsekov kreslilo iba vnútro

Scanline algoritmus – TAH



y_{max}	x_{ymin}	$1/m$
9	0	-1/3
9	1/3	-1/3

e_5

y_{max}	x_{ymin}	$1/m$
9	0	-7/2
9	7/2	-7/2

e_4

y_{max}	x_{ymin}	$1/m$
12	56/5	7/5
12	49/5	7/5
12	42/5	7/5

e_3

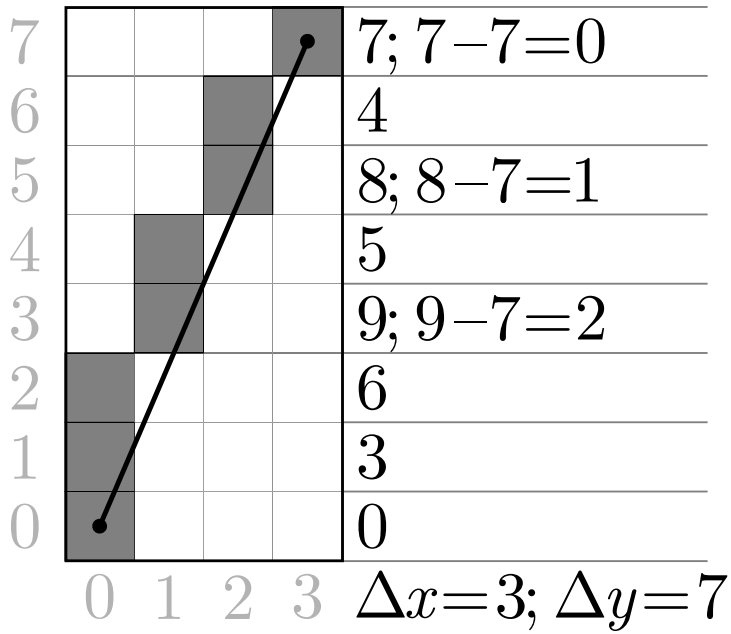
y_{max}	x_{ymin}	$1/m$
12	96/7	1/7
12	95/7	1/7
12	94/7	1/7

e_2

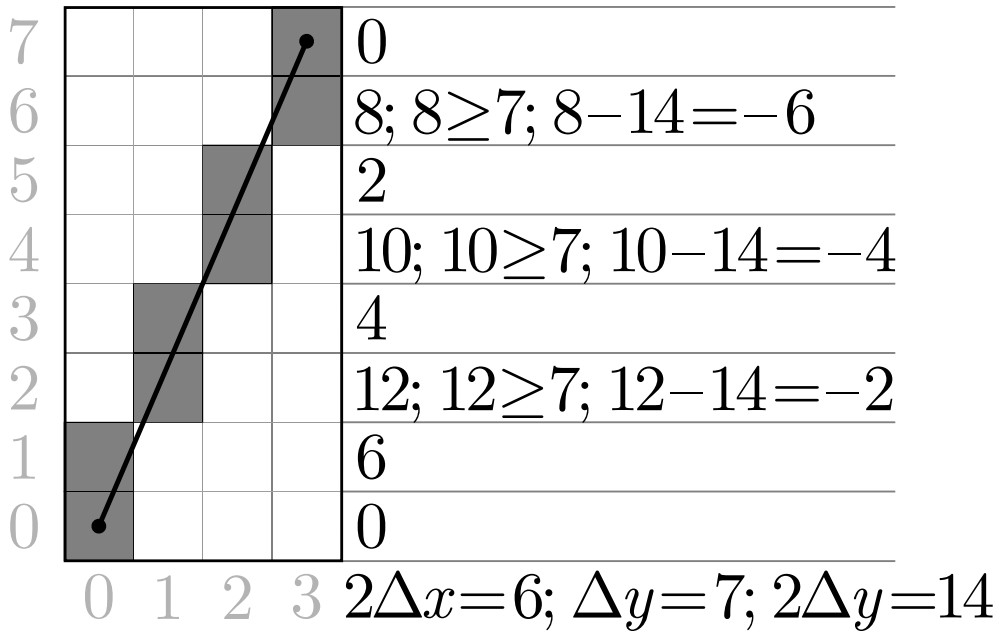
Scanline algoritmus – úpravy

- Pri mnohouholníkoch bez samopriesekov netreba vždy usporiadavať hrany v TAH po každej zmene hodnoty x
- Namiesto vypĺňania riadok po riadku, je možné použiť vodorovné šrafovanie, ak bude prírastok $dy = k$ a $dx = k/m$
- Algoritmus pracuje s reálnočíselným prírastkom $dx = 1/m$ a pri určovaní výslednej hodnoty pixlov sa používa zaokrúhľovanie
- Algoritmus sa dá upraviť na celočíselnú aritmetiku, napr. s využitím **počítadla** s prírastkom Δx, ktoré využíva vzťah $x_{k+1} = x_k + 1/m = x_k + \Delta x/\Delta y$:
  1. Počítadlo pri prechode od y_k ku y_{k+1} zväčšíme o hodnotu Δx a ak dosiahne hodnotu $\geq \Delta y$, zväčšíme x-ovú položku v TAH o 1 a súčasne odpočítame z počítadla hodnotu Δy
  2. Keďže postup č. 1 nevyužíva zaokrúhľovanie zlomku $\Delta x/\Delta y$, je nutné ho upraviť: $x_{k+1} = x_k + \Delta x/\Delta y \Rightarrow x_{k+1} = x_k + (\Delta x/\Delta y + 0.5) \Rightarrow$ $x_{k+1} = x_k + 2\Delta x/2\Delta y + \Delta y/2\Delta y$
     Teraz pri prechode skenovacej priamky na nový riadok zväčšíme počítadlo o hodnotu $2\Delta x$, porovnáme ju s hodnotou Δy (to plynie z porovnania čitateľa s menovateľom: $2\Delta x + \Delta y \geq 2\Delta y$), a ak je hodnota počítadla väčšia ako Δy, tak odpočítame z počítadla hodnotu $2\Delta y$ a zväčšíme hodnotu x o 1

Scanline algoritmus – úpravy



1. počítadlo s prírastkom Δx

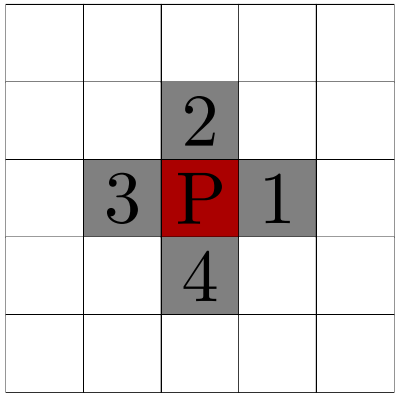


2. počítadlo s prírastkom $2\Delta x$

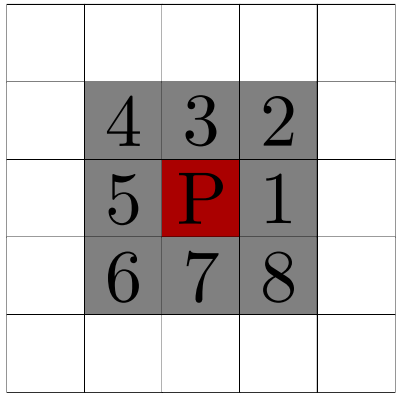
Susednosť, oblasť

- V bitovej mape (rastri, štvorcovej sieti) má každý vnútorný pixel ôsmich susedov, ktorý sú zvyčajne očíslovaný proti smeru hodinových ručičiek
- Susedov, ktorý sa nachádzajú v rastri nad ním, pod ním, vpravo a vľavo od neho nazývame **priami susedia**, alebo **4-susedia**. Platí, že každý dvaja 4-susedia majú spoločnú stranu
- Susedov, ktorý sa nachádzajú na najbližších štyroch diagonálnych štvorcoch rastra nazývame **nepriami susedia**
- Všetkých najbližších okolitých susedov, ktorý nazývame **8-susedia**. Platí, že každý dvaja 8-susedia majú spoločný aspoň jeden vrchol
- Oblasť je množina prvkov (pixlov) bitovej mapy, ktoré sú **súvislo** pospájané
- Podľa typu súvislosti rozdeľujeme oblasti na *4-súvislé* a *8-súvislé*:
  - **4-súvislá** oblasť je taká, ktorej každé dva body (pixle) môžeme spojiť postupnosťou 4-susedov
  - **8-súvislá** oblasť je taká, ktorej každé dva body (pixle) môžeme spojiť postupnosťou 8-susedov
- Každá 4-súvislá oblasť je aj 8-súvislá, ale neplatí to naopak

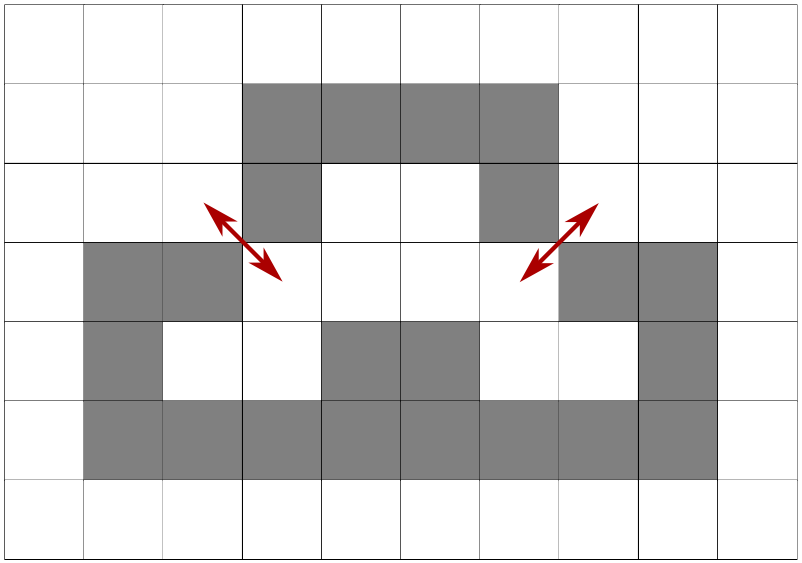
Susednosť, oblasť



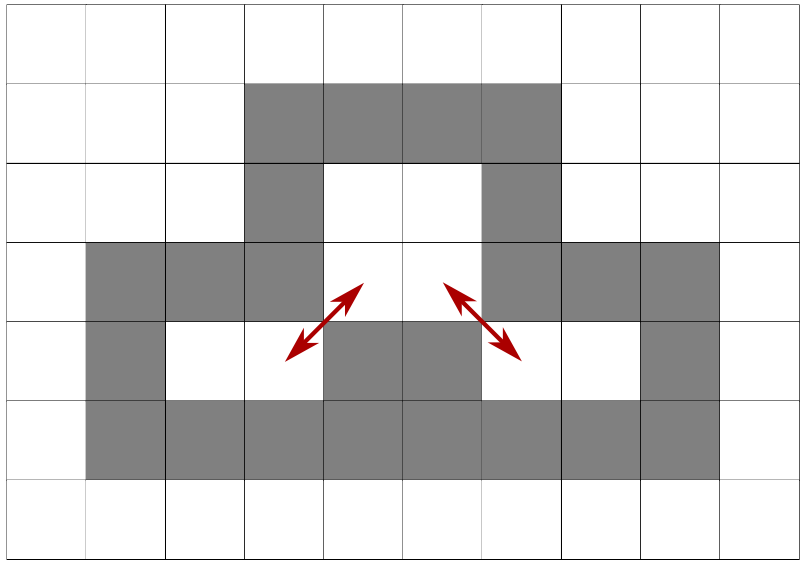
4-susednosť



8-susednosť



4-súvislá oblasť s 8-súvislou hranicou



8-súvislá oblasť s 4-súvislou hranicou

Susednosť, oblasť

- Každá uzavretá krivka v rovine rozdeľuje rovinu na dve súvislé oblasti: vnútro a vonkajšok
- Ak je hraničná krivka 8-súvislá, tak vo všeobecnosti neplatí, že rozdeľuje raster do dvoch (vnútornej a vonkajšej) 8-súvislých oblastí, preto ak požadujeme:
  - aby hraničná krivka bola 8-súvislá, je potrebné, aby ňou definovaná oblasť bola 4-súvislá,
  - aby hraničná krivka bola 4-súvislá, je potrebné, aby ňou definovaná oblasť bola 8-súvislá
- Oblasti môžu byť *vnútorne* definované alebo *hranične* definované
- **Vnútorne** definovaná oblasť je zadaná súvislou množinou bodov (vnútrom) rovnakej farby
- **Hranične** definovaná oblasť je zadaná súvislou postupnosťou bodov (hranicou) rovnakej farby

Vypĺňanie oblastí

- Ak je oblasť zadaná vrcholmi svojho hraničného mnohouholníka, používame na jej vyplnenie scanline algoritmus, t.j. rozklad mnohouholníka do rastra. Ak je oblasť zadaná množinou pixlov, používame na jej vyplnenie algoritmy semienkového vypĺňania
- Pri vypĺňaní oblasti býva požiadavka na 4-súvislosť často vhodnejšia ako požiadavka na 8-súvislosť, keďže bráni prepojeniu dvoch oblastí, ktoré sa dotýkajú v rohových pixloch
- Každý algoritmus, ktorý je navrhnutý pre 4-súvislé oblasti možno jednoducho prepísať, aby fungoval aj pre 8-súvislé
- Pri vypĺňaní vnútorne definovanej oblasti, keď každá iná farba od farby vnútra predstavuje hranicu oblasti, používame tzv. záplavový/vlnový algoritmus (*floodfill*)
- Pri vypĺňaní hranične definovanej oblasti, keď každá iná farba od hranice predstavuje vnútro alebo vonkajšok oblasti, používame algoritmus vypĺňania po hranicu (*boundfill*)

Vypĺňanie oblastí semienkovými algoritmami

- Pri hľadaní pixlov a následnom vypĺňaní oblastí vychádzame z daného vnútorného bodu – semienka a klasifikujeme vnútorné body oblasti takto:
  - testovaný bod je vnútorný, ak má farbu, ktorá je rôzna od farby hranice,
  - testovaný bod je vnútorný, ak má rovnakú farbu ako je farba bodov vnútra
- Pri určovaní rovnakej či rôznej farby môžeme požadovať striktnú alebo iba približnú rovnosť/rôznosť farieb. V prípade približnej rovnosti či rôznosti hovoríme o tzv. vypĺňaní s danou prahovou (*treshold*) hodnotou

Algoritmus floodfill

- Algoritmus sa rekurzívne snaží priradiť bodom 4-súvislej vnútorne definovanej oblasti určenej farbou `old_color` novú farbu `new_color` (ktorou sa má oblasť vyplniť), pričom každá iná farba ako `old_color` predstavuje farbu možnej hranice
- Algoritmus zisťuje, či už bol testovaný vnútorný bod (x, y). Ak nie, t.j. ak má hodnotu `old_color`, tak ju zmení na `new_color` a rekurzívne vyšetrí jeho 4-susedov. Ak bod (x, y) nemá hodnotu `old_color`, tak nerobí nič
- Algoritmus možno modifikovať na 8-súvislé oblasti tak, že sa vyšetria všetci 8-susedia bodu (x, y)

Algoritmus floodfill

```
procedure flood_fill_4(x, y, old_color, new_color: integer);
begin
  if readpixel(x, y) == old_color then
    begin
      drawpixel(x, y, new_colour);
      flood_fill_4(x, y-1, old_color);
      flood_fill_4(x, y+1, old_color);
      flood_fill_4(x-1, y, old_color);
      flood_fill_4(x+1, y, old_color);
    end
end
```

Algoritmus boundfill

- Myšlienka tohto algoritmu je podobná algoritmu floodfill, avšak nestačí iba testovať, či bod (x, y) patrí oblasti, ale sú potrebné až dva testy:
  1. či je aktuálne testovaný bod vo vnútri oblasti
  2. či testovanému bodu ešte nebola pridelená nová farba
- Algoritmus sa rekurzívne snaží priradiť bodom 4-súvislej hraničnej určenej oblasti farbu výplne `new_color`, pričom farba `bound_color` predstavuje farbu hranice oblasti
- Algoritmus zisťuje, či už bol testovaný (vyfarbený) vnútorný bod (x, y) a či súčasne nemá farbu hranice `bound_color`. Ak nie, tak ju zmení na `new_color` a rekurzívne vyšetrí jeho 4-susedov. V inom prípade nerobí nič
- Aj tento algoritmus možno modifikovať na 8-súvislé oblasti tak, že sa vyšetria všetci 8-susedia bodu (x, y)

Algoritmus boundfill

```
procedure bound_fill_4(x, y, bound_color, new_colour: integer);
begin
  if readpixel(x, y) ≠ bound_color and readpixel(x, y) ≠ new_color then
  begin
    drawpixel(x, y, new_color);
    bound_fill_4(x, y-1, old_color, new_color);
    bound_fill_4(x, y+1, old_color, new_color);
    bound_fill_4(x-1, y, old_color, new_color);
    bound_fill_4(x+1, y, old_color, new_color);
  end
end
```

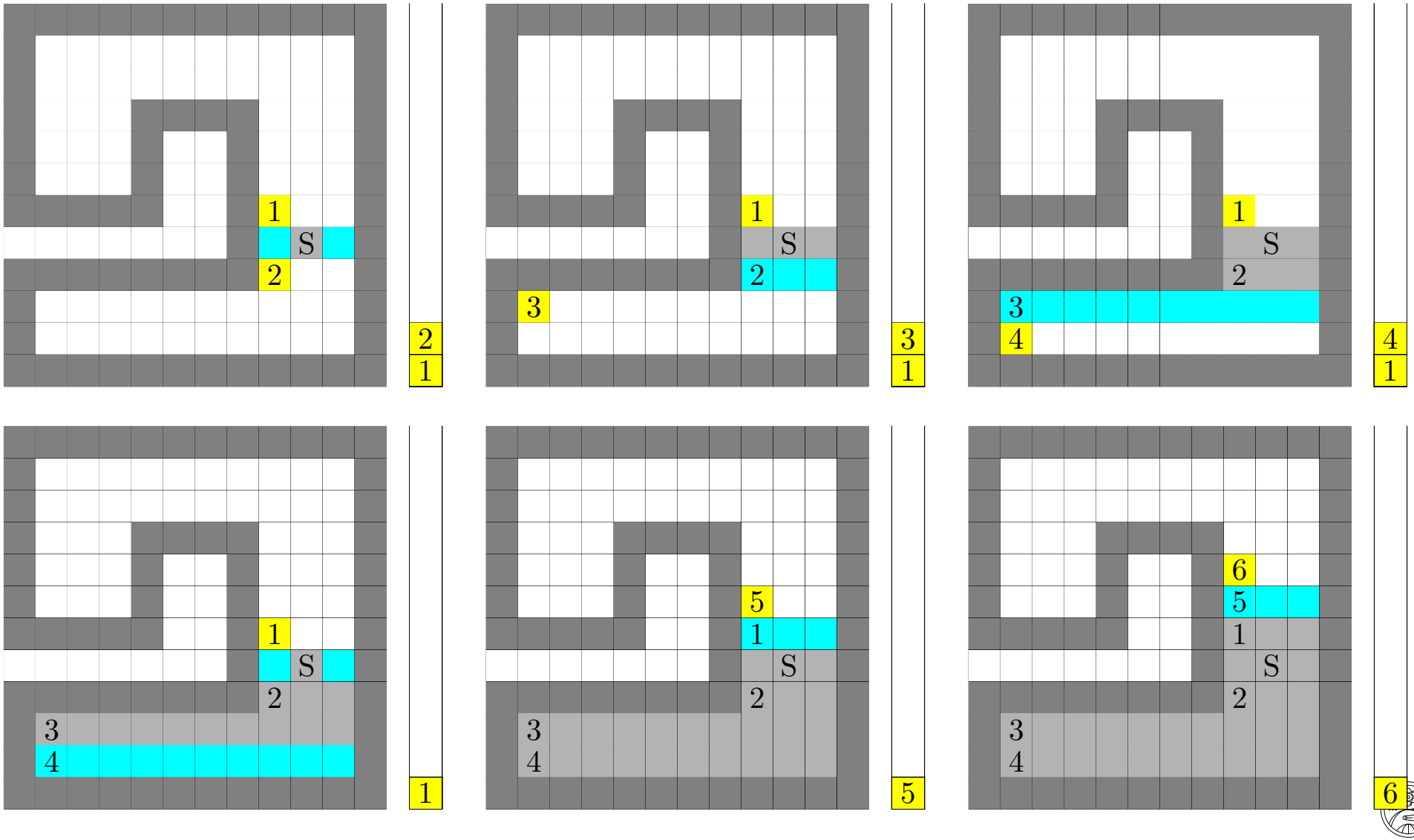
Algoritmy semienkového vypĺňania

- Tieto rekurzívne algoritmy sú veľmi jednoduché, avšak nie celkom vhodné pre implementáciu
- Oba opísané algoritmy musia pri implementácii obsahovať aj test, či aktuálne testovaný bod leží v danom okne, t.j. či $x_{min} \leq x \leq x_{max}$ a $y_{min} \leq y \leq y_{max}$
- Majú veľké pamäťové nároky, kvôli rekurzívnemu volaniu a navyše pracujú neefektívne, keďže v každom kroku vypĺňajú iba štyri susedné body
- Pre implementáciu semienkového vypĺňania sú vhodnejšie rekurzívne algoritmy, ktoré pracujú po riadkoch a vypĺňajú postupne všetky body napravo a naľavo, až pokým nájdu v aktuálne vypĺňanom riadku body hranice. Sú známe ako riadkové semienkové vypĺňanie (scanline/span filling)

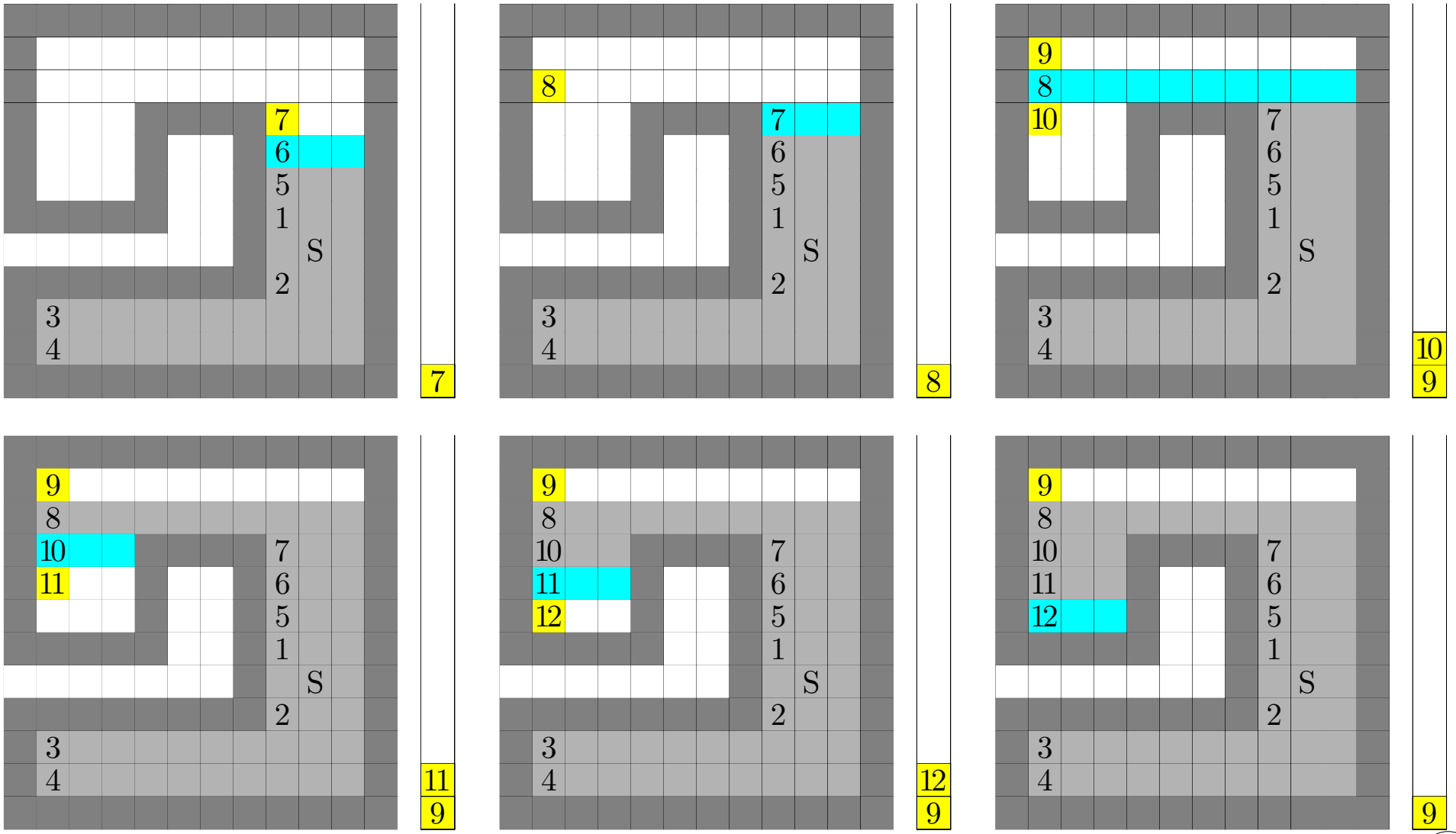
Algoritmy riadkového vypĺňania – scanline/span filling

- Podľa toho, či je oblasť určená hranicou alebo vnútrom, môže sa na efektívne vyplnenie oblastí použiť buď *span flood fill* alebo *span bound fill* algoritmus
- Oba algoritmy vypĺňajú súvislý riadok naraz a do zásobníka sa ukladajú iba „najľavejšie“ vnútorné oblasti riadku nad a pod aktuálne vypĺňaným (vykresľovaným) riadkom. Rekurzia sa potom volá iba na riadok vyššie a riadok nižšie

Algoritmy riadkového vypĺňania – scanline/span filling



Algoritmy riadkového vypĺňania – scanline/span filling



Algoritmy riadkového vypĺňania – scanline/span filling