

Obr. 1.8 Základná štruktúra interaktívnej grafickej aplikácie

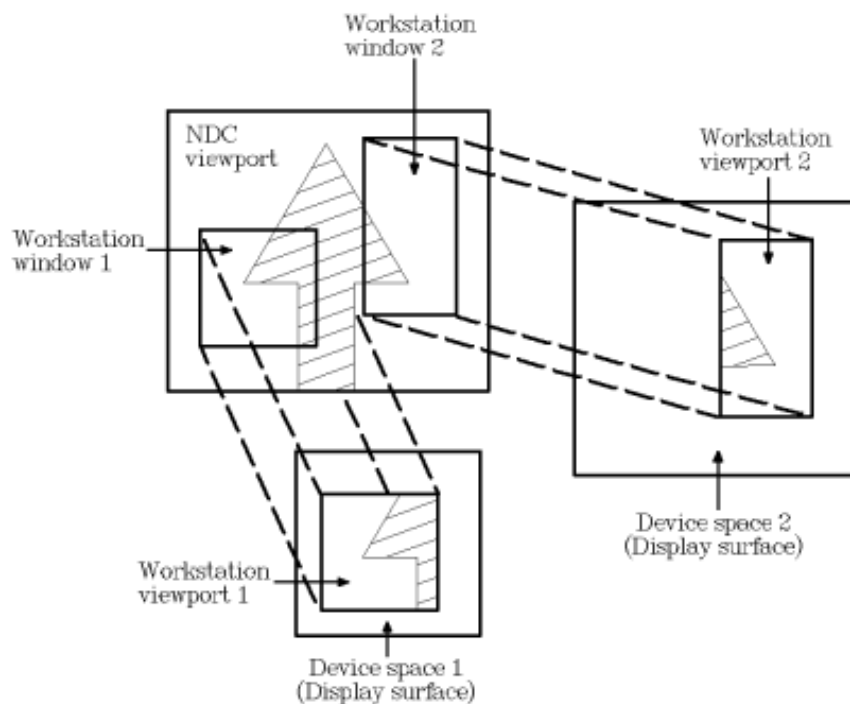
Workstation window to viewport transformation at IBM Infocenter. <http://pic.dhe.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.admk100%2Fadmki1a0036.htm>

16.3 Spracovanie výstupu v oknovom systéme

Oknový systém musí mať na spracovanie výstupu aspoň tieto základné funkcie:

- Create Window (name)** - vytvorí okno s daným menom
- Set Position (xmin, ymin)** - nastaví pozíciu aktuálneho okna
- Set Size (height, width)** - nastaví veľkosť aktuálneho okna
- Select Window (name)** - určí aktuálne okno
- Show Window** - zobraz aktuálne okno
- Hide Window** - skry aktuálne okno
- Set Title (name)** - nastav meno aktuálneho okna
- Get Position (xmin, ymin)** - zisti pozíciu aktuálneho okna
- Get Size (height, width)** - zisti veľkosť aktuálneho okna
- Bring To Top** - pošli aktuálne okno na vrch všetkých okien
- Send To Bottom** - pošli aktuálne okno na dno, za všetky okná
- Delete Window** - zruš aktuálne okno

Tým sme popísali minimálnu funkčnosť oknového systému pri spracovaní výstupu, pričom algoritmicke riešenia tejto funkčnej špecifikácie nás na tejto úrovni nezaujímajú, hoci niektoré úvahy môžeme naznačiť. Napr. uvedené funkcie predpokladajú obdĺžnikové okno s menom, rozmermi a pozíciou na obrazovke, súbor takýchto okien s



2.7 Zobrazenie okna na zobrazovacie pole

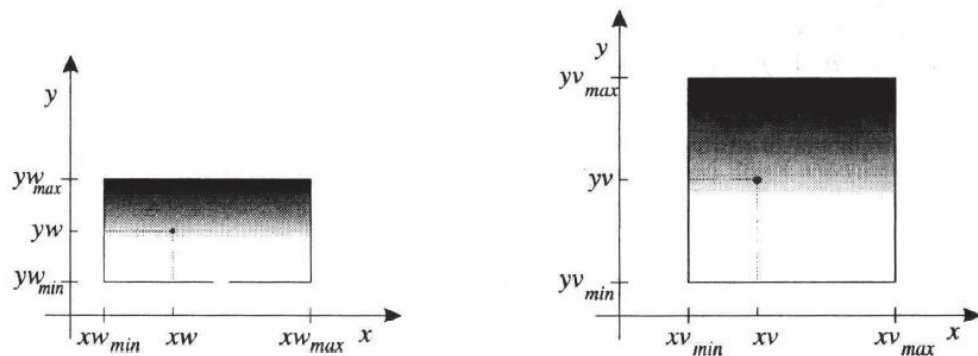
Užívateľ obvykle zadáva objekty vo svojich užívateľských súradniciach. Na zobrazenie **oblasti záujmu** si zvolí minimálne a maximálne súradnice v oboch smeroch, tzv. **okno**. Funkciu na definovanie okna viditeľnosti budeme označovať

$$\text{window}(xw_{\min}, xw_{\max}, yw_{\min}, yw_{\max}).$$

Avšak toto okno užívateľ nemusí chcieť zobraziť na celú obrazovku (zobrazovaciu časť príslušného výstupného zariadenia). Preto je prirodzené zaviesť pojem **zobrazovacie pole** resp. **záber**. Je to tá časť, na ktorú sa bude transformovať okno. Funkciu pre definovanie zobrazovacieho poľa (záberu) označíme

$$\text{viewport}(xv_{\min}, xv_{\max}, yv_{\min}, yv_{\max}).$$

Ako vyzerá transformačná matica, ktorá bude realizovať príslušné zobrazenie okna na zobrazovacie pole?



Obr. 2.7 Transformácia okna na zobrazovacie pole (záber)

Predpokladáme, že bod (xw, yw) sa zobrazí do bodu (xv, yv) , pozri obrázok 2.7. Prírodná požiadavka je, aby sa pri transformácii **zachovali pomery strán**. Preto podľa označenia z obrázku 2.7 požadujeme, aby platili tieto rovnosti

$$\frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}} = \frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}},$$

$$\frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}} = \frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}}.$$

Odtiaľ môžeme vyjadriť hodnoty xv a yv

$$xv = s_x \cdot (xw - xw_{\min}) + xv_{\min},$$

$$yv = s_y \cdot (yw - yw_{\min}) + yv_{\min},$$

kde

$$s_x = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}, \quad s_y = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}.$$

V tomto poslednom vyjadrení s_x a s_y sú koeficienty pre zmenu mierky z okna na záber a xv_{\min} , yv_{\min} sú relatívne hodnoty posunu. Nakoniec rovnosti môžeme upraviť na tvar, kde pre transformáciu jednej súradnice máme len jednu operáciu súčtu a násobenia:

$$xv = s_x \cdot xw + a,$$

$$yv = s_y \cdot yw + b,$$

kde

$$a = -s_x \cdot xw_{\min} + xv_{\min}, \quad b = -s_y \cdot yw_{\min} + yv_{\min}.$$

Hľadaná matica je teda

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ a & b & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ -s_x \cdot xw_{\min} + xv_{\min} & -s_y \cdot yw_{\min} + yv_{\min} & 1 \end{pmatrix}.$$

V grafických systémoch sa táto matica zostrojí automaticky po zadaní funkcií *window* a *viewport*. Kvôli úspore pamäti sa zvyčajne ukladajú len prvé dva stĺpce, pretože pre afinné transformácie je tretí stĺpec rovnaký.

Konštrukcia matice na konštrukciu zobrazovacieho kanála: okno/svetové súradnice WC, kompozícia scény/normalizované súradnice zariadenia NDC, záber/súradnice zariadenia DC. Obr. 1.8 ilustruje transformácie medzi súradnicovými systémami šípkami doprava na zobrazenie a šípkami doľava na interakciu. Bod (x, y) má homogénne súradnice $(x, y, 1)$. Tento zápis dovoľuje maticovo vyjadriť skladanie škálovania/rotácie a posunutia. Počítame teda 2D súradnice v 3D na rovine $z=1$. Vektor zapíšeme ako rozdiel bodov $(x, y, 0)$. Zadaním obdĺžnika určíme aj súradnicovú sústavu, t.j. kde leží bod $(0, 0)$. Vzhľadom k tomuto pevnému bodu majú matice škálovania a rotácie jednoduchý tvar, umožňujúci ich názorné invertovanie. Posunutie invertujeme násobkom -1 .

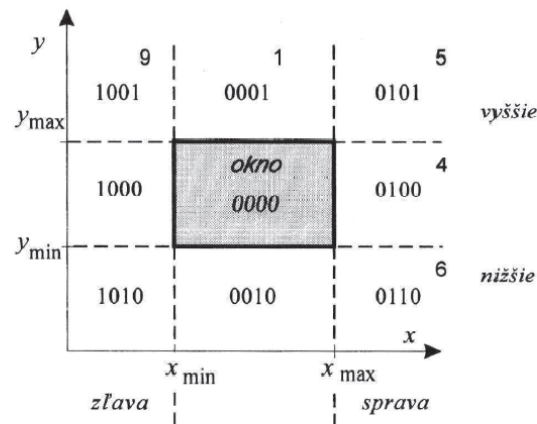
Do okien a záberov treba orezať body, úsečky či výplňové oblasti, aby sme nepočítali ani nezobrazovali zbytočné. Tak ako sme pridali ku 2D ďalší rozmer, pridáme k oknu v rovine ďalšie jej časti, každý bod v Euklidovej rovine patrí do jednej z deviatich. Zatiaľ sme využili algoritmickú stratégiu **Duality**, orezávanie realizuje Odsekni a hľadaj (**Prune & Search**). Rasterizácia uplatní **Iteration, Divide & Conquer** (FloodFill) a **Sweeping Technique**. Viac o algoritmických stratégiách: Chalmovianský, P. et al. 2001. **Zložitosť geometrických algoritmov.**

3.3.1 Algoritmus orezovania Cohena-Sutherlanda

Tento algoritmus rýchlo vylúči vyššie spomenuté prípady. Zvlášť rýchly je v prípade okna, ak obsahuje veľa úsečiek vo vnútri a taktiež pri takom okne, ak väčšina úsečiek je mimo okna. V týchto prípadoch sa úsečka podľa polohy buď celá zobrazí alebo nezobrazí.

Na začiatku algoritmu nastavíme 4-bitové hodnoty (kód) pre obidva koncové body úsečky podľa polohy vzhľadom na okno. Pre každú hraničnú priamku okna nastavíme jeden bit podľa toho, či leží bod v danej polovine (pozri obr. 3.1). Pravidlo upresníme podľa polohy bodu vzhľadom na okno:

1. bit - bod leží vyššie od okna ($x_{max} < x$);
2. bit - bod leží nižšie od okna ($x_{min} < x$);
3. bit - bod leží sprava od okna ($y_{max} < y$);
4. bit - bod leží zľava od okna ($y_{min} < y$).



Obr. 3.1 Kódovanie jednotlivých častí roviny

Algoritmus Cohena-Sutherlanda

Procedure Clipping;

begin

1. **accept**:= false; { nastav - P1P2 sa nevykresľuje }
 $outcod(x2, y2, cd2)$; { zistíme kód bodu P2 }
 2. **repeat**
 $outcod(x1, y1, cd1)$; { zistíme kód bodu P1 }
 3. **if and** ($cd1, cd2$) $\neq 0$ **then done**:= true { 1. jednoduchý test - mimo okna }
else
 4. **begin**
if ($cd1=0$ and $cd2=0$) **then** { 2. jednoduchý test - vnútri okna }
{ žiadaj vykreslenie v kroku 10. }
begin **accept**:= true;
done:= true **end**
 5. **else**
begin
if $cd1=0$ **then** $swap(P1, P2)$; { zameníme body, aby 1. bol von }
 6. **if** $cd1 \in (1, 5, 9)$ **then** { orezovanie zhora }
begin
 $x1 := x1 + (x2-x1)*(ymax-y1)/(y2-y1)$;
 $y1 := ymax$;
end
 7. **else if** $cd1 \in (2, 6, 10)$ **then** { orež zdola }
begin
 $x1 := x1 + (x2-x1)*(ymin-y1)/(y2-y1)$;
 $y1 := ymin$;
end
 8. **else if** $cd1 \in (4, 5, 6)$ **then** { orež sprava }
begin
 $y1 := y1 + (y2-y1)*(xmax-x1)/(x2-x1)$;
 $x1 := xmax$;
end
 9. **else if** $cd1 \in (8, 9, 10)$ **then** { orež zľava }
begin
 $y1 := y1 + (y2-y1)*(xmin-x1)/(x2-x1)$;
 $x1 := xmin$;
end
 10. **end** { od kroku 5 }
end { od kroku 4 }
 - until done**
 10. **if** **accept** **then** $draw(P1, P2)$; { vykresli úsečku }
- end.**

5.2. Rastrový rozklad úsečky	55
5.3. Rastrový rozklad kružnice	60
5.4. Vyhľadzovanie (antialiasing) a hrúbka čiary	62
5.5. Vypĺňanie oblasti	64
5.6. Rastrový rozklad mnohoúhelníka	70

5.2 Rastrový rozklad úsečky

Uvedieme dva algoritmy generovania úsečiek do rastrovej formy. Pri rozklade úsečky budeme predpokladať, že oba koncové body majú celočíselné súradnice (x_1, y_1) , (x_2, y_2) .

5.2.1 Jednoduchý prírastkový algoritmus

Analytické vyjadrenie priamky, ktorá nie je rovnobežná s osou y , vyjadrujeme v tvare:

$$y = m \cdot x + b,$$

kde m je smernica priamky a b posun na osi y . Koncové body úsečky určujú priamku s parametrami m a b :

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{a} \quad b = \frac{x_2 \cdot y_1 - x_1 \cdot y_2}{x_2 - x_1}.$$

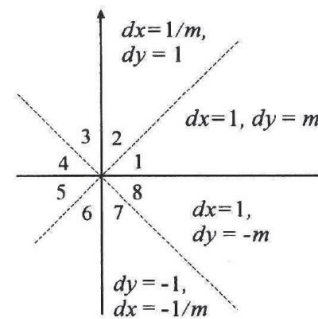
Prírastkový algoritmus úsečky DDA (Digital Differential Analyzer)

```

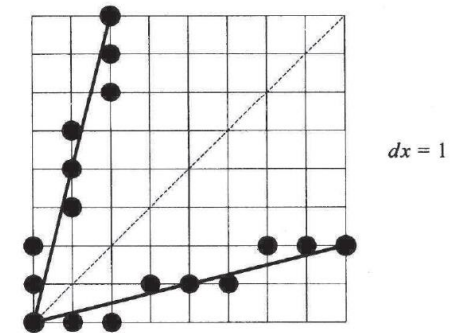
procedure DDA_line ( x1, y1,           { začiatkový bod P }
                    x2, y2,           { koncový bod Q }
                    colour : integer ); { farba vykreslenej úsečky }

var dx, dy, x, y, m: real;
begin
  if x1 > x2 then swap;           { zameň body, aby prvý bod bol ľavý }
  if x1 < x2 then
    begin
      dx := x2 - x1;
      dy := y2 - y1;
      m := dy/dx;
      y := y1;
      for x := x1 to x2 do
        begin
          write_pixel (x, round(y), colour);   { zobraz bod (x, y) }
          y := y + m                          { inkrementuj súradnicu y }
        end
      end
    else if y1 = y2 then write_pixel (x1, y1, colour) { zapíš len jeden bod }
    else error;
end.

```



Inkrementálne $dy = 1$



Obr. 5.1 Prírastky pre niektoré oktanty a dve úsečky v 1. a 2. oktante

Prírastkový algoritmus úsečky pre všetky oktanty

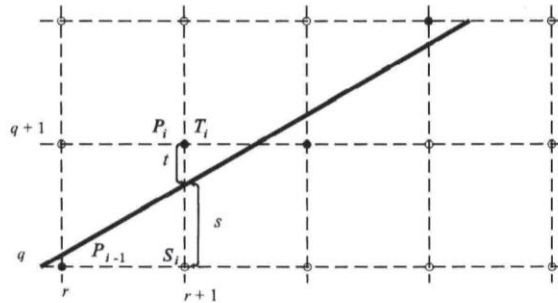
```

Procedure line ( x1, y1,           { začiatkový bod }
                 x2, y2,           { koncový bod }
                 colour : integer ); { farba vykreslenej úsečky }

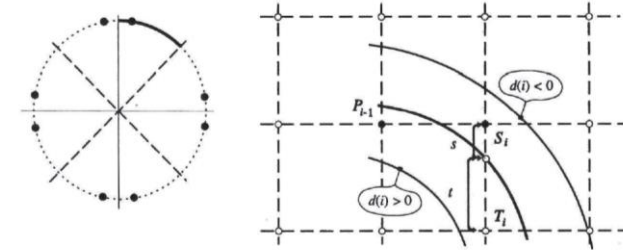
var dx, dy, x, y, m: real;
begin
  dx := x2 - x1;
  dy := y2 - y1;
  if abs(dx) > abs(dy) then
    begin
      { oktanty 1, 4, 5, 8 }
      { zameň body, aby prvý bod bol ľavo }
      if x1 > x2 then swap;
      if y1 < y2 then Yinc := 1;
      else Yinc := -1;
      begin
        { generovanie úsečky pre 1. oktant }
        DDA_Line1(x1, y1, x2, y2, colour);
      end;
    end;
  else begin
    { abs(dx) < abs(dy), t.j. oktanty 2, 3, 6, 7 }

```

$$d(P(i)) = (x_i^2 + y_i^2) - r^2.$$



Obr. 5.2 Rozklad úsečky do rastra



Obr. 5.3 Symetria kružnice a rozklad kružnice do rastra podľa hodnôt s a t

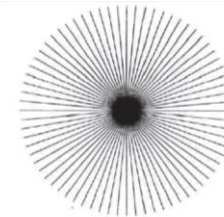
Bresenhamov algoritmus úsečky

```

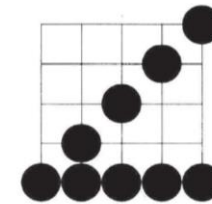
procedure Bres_line ( x1, y1,          { začiatkový bod }
                      x2, y2,          { koncový bod }
                      colour : integer ); { farba vykreslenej úsečky }

var dx, dy, d, incr1, incr2, x, y, m : integer;
begin
  dx := abs(x2 - x1); dy := abs(y2 - y1);
  d := 2*dy - dx;          { začiatková hodnota d }
  incr1 := 2*dy;          { prírastok pre d < 0 horizontálne }
  incr2 := 2*(dy-dx);     { prírastok pre d ≥ 0 diagonálne }
  if x1 > x2 then begin x := x2 ; y := y2 ; xend := x1 end
  else begin x := x1 ; y := y1 ; xend := x2 end;
  write_pixel (x, y, colour); { vykresli 1. bod }
  while x < xend do
    begin
      x := x + 1 ;
      if d < 0 then d := d + incr1 { vyber bod S(i), horizontálne }
      else begin
        y := y + 1 ;
        d := d + incr2; { vyber bod T(i), diagonálne }
      end
      write_pixel (x, y, colour); { vykresli bod }
    end
  end.

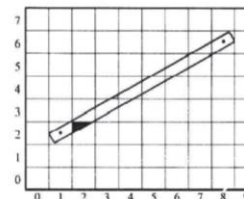
```



Obr. 5.4 Aliasing úsečiek

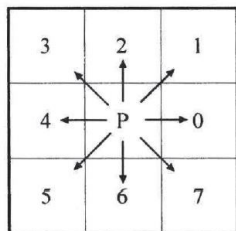
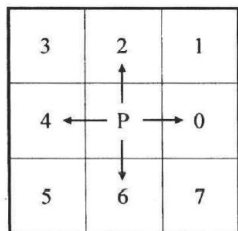


Na obrázku 5.5 je zobrazená úsečka nenulovej hrúbky, položená do rastra. Rastrová sieť je posunutá tak, aby mriežové body ležali vo vnútri každého obdĺžnika siete a nie v priesečníkoch mriežky. Každý obrazový bod je zobrazený obdĺžnikom.

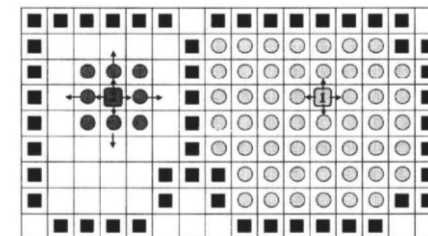


Algoritmus DDA sa dá zovšeobecniť, kým Bresenham optimalizuje rýchlosť výpočtu celočíselnou aritmetikou.

Video Bresenham: https://www.youtube.com/watch?v=ax_iADXP7co



Obr. 5.7 Susednosť (4 a 8) v štvorcovom rastru pre bod P



Obr. 5.8 Vypĺňanie oblastí farbou zadaného vnútorného bodu

Algoritmus vlnového vypĺňania *Flood fill*

```

procedure Flood_fill_4 ( x, y,           { začiatkový vnútorný bod vypĺňania oblasti }
                        old_colour,      { stará farba oblasti }
                        new_colour: integer; { nová farba oblasti }

begin
  if read_pixel (x,y) = old_colour then
    begin
      write_pixel (x, y, new_colour);
      Flood_fill_4 (x, y-1, old_colour, new_colour);
      Flood_fill_4 (x, y+1, old_colour, new_colour);
      Flood_fill_4 (x-1, y, old_colour, new_colour);
      Flood_fill_4 (x+1, y, old_colour, new_colour);
    end
  end.

```

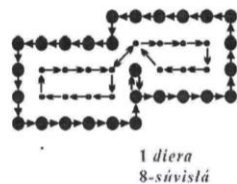
Algoritmus vypĺňania do hraničných bodov *Bound fill 4*

```

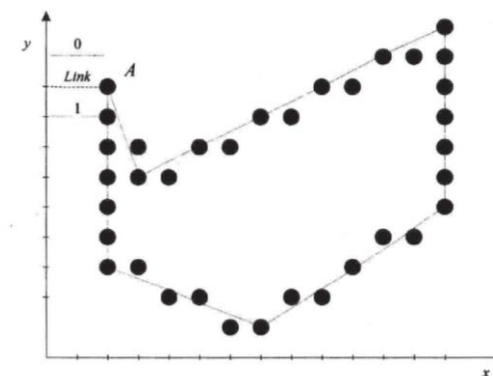
procedure Bound_fill_4 ( x, y,           { začiatkový bod vypĺňania oblasti }
                        bound_colour,    { farba hranice oblasti }
                        new_colour: integer; { nová farba oblasti }

begin
  if read_pixel (x, y) ≠ bound_colour and read_pixel (x, y) ≠ new_colour
  then begin
    write_pixel (x, y, new_colour);
    Bound_fill_4 (x, y-1, old_colour, new_colour);
    Bound_fill_4 (x, y+1, old_colour, new_colour);
    Bound_fill_4 (x-1, y, old_colour, new_colour);
    Bound_fill_4 (x+1, y, old_colour, new_colour);
  end
end.

```



Obr. 5.9 Vypĺňanie podľa 4-súvislosti a 8-súvislosti



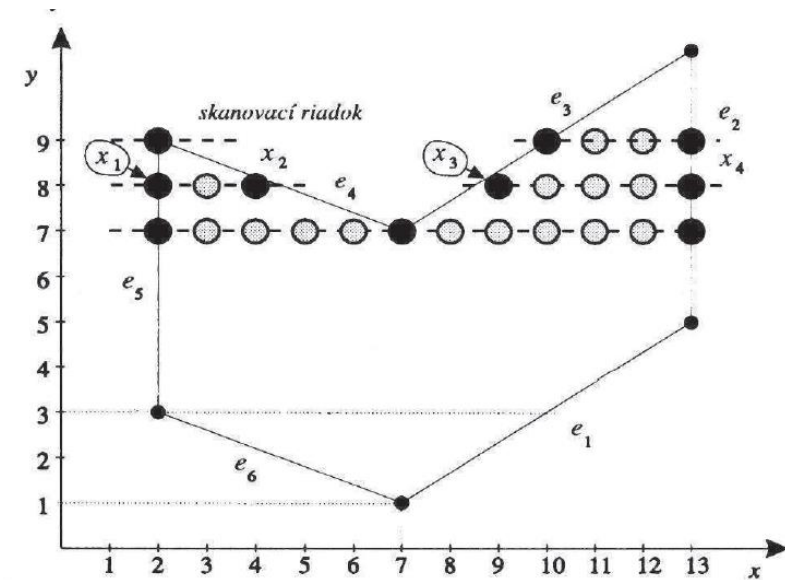
Obr. 5.11 Nekorektná hranica pre algoritmus vypĺňania podľa parity

Rekurzívny algoritmus FloodFill patrí do algoritmickej stratégie Rozdeľuj a panuj (Divide & Conquer).

Algoritmus vypíňania podľa parity *Parity fill*

```
procedure Parity_fill;
var x, y, Xmax, Ymax, par, error_flag,
    nad, pod, bound_colour, ab: integer;
begin
  for y:= 1 to Ymax do                { pre všetky riadky }
  begin
    par:= 0; error_flag:= 0;           { inicializácia premenných *}
    nad:= 0; pod:= 0; x:= 1;
    while x <= Xmax do
    begin
      if (h[x, y] ≠ bound_colour) then
      begin
        if (Odd(par)) then Fill_Pixel(x, y); { vyplň *}
        x:= x + 1;                          { posuň *}
      end
    else
    begin
      Link_4(x, y, nad, pod);             { zisti lokálne správanie hranice *}

      if ((nad=1) and (pod=1)) then par:= par + 1;
      ab:= nad + pod;
      if ((ab ≠ 0) or (ab ≠ 2)) then error_flag:= 1;
    end
  end;
  if (error_flag ≠ 0) then Title('Error in boundary'); { vypíš chybovú správu }
end.
```



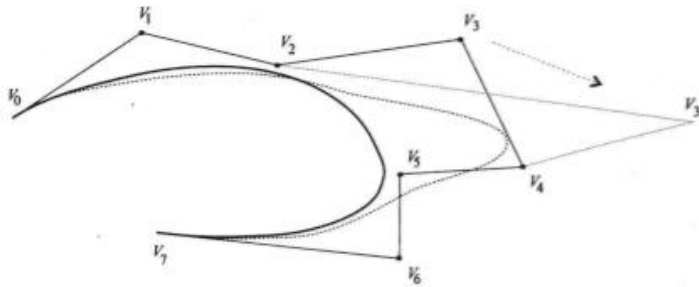
Obr. 5.13 Skanovací riadok pre rozklad mnohoúhelníka do rastra

Algoritmus Scan_Line

1. Skrátime zdola hrany naväzujúce vo vrchole monotónneho spojenia.
2. Vylúčime vodorovné hrany.
3. V cykle od minimálnej po maximálnu súradnicu y mnohoúhelníka:
 - 3.1 Nájde priesečníky skanovacej priamky so všetkými hranami.
 - 3.2 Usporiadame priesečníky podľa x -ovej súradnice.
 - 3.3 Vykreslíme všetky body, ktoré sú medzi dvojicami za sebou.
4. Vykreslíme hranicu mnohoúhelníka.

Algoritmy Parity i ScanLine patria do algoritmickej stratégie Zametacia technika (Sweeping), statický dvojrozmerný problém sa rieši postupnosťou jednorozmerných dynamických problémov.

Video ScanLine: <https://www.youtube.com/watch?v=loJ4B58VD8U>



Obr. 4.3 Zmena riadiaceho polygónu Bézierovej krivky

Sformulujeme niektoré vlastnosti Bézierových kriviek :

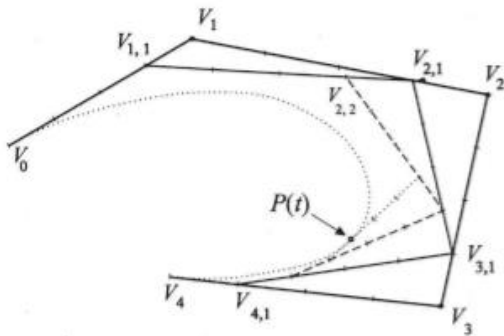
1. Začiatočným bodom Bézierovej krivky je bod V_0 a krivka sa v tomto bode dotýka priamky V_0V_1 . Podobne koncovým bodom krivky je bod V_n a krivka sa v tomto bode dotýka priamky $V_{n-1}V_n$.

2. Bézierova krivka je krivkou stupňa n (pre daných $n+1$ bodov). Polynómi stupňa n , pre veľké n , sú numericky nestabilné a takisto aj Bézierová krivka stupňa n . Z toho dôvodu sa začali používať splajnové krivky nižšieho stupňa.

3. Na obr. 4.4 je ukázaná podstata tzv. algoritmu Casteljau. Určenie bodu krivky $P(t)$ pre parameter t vykonáme postupným iteratívnym delením úsečiek riadiaceho polygónu pomerom závislým od parametru t . Na obrázku 4.4 sú znázornené tieto úsečky, kde pomer delenia je pre parameter $t = 0.75$ a $n = 4$. V každom i -tom priblížení definujeme body

$$V_{j,i} = (1-t) \cdot V_{j-1,i-1} + t \cdot V_{j,i-1},$$

pre $i = 1, 2, 3, 4$ a $j = i, \dots, 4$ a hľadaný bod je $P(t) = V_{4,4}$.



Obr. 4.4 Algoritmus de Casteljau