Rasterizácia (Scan conversion)

by Peter Gejguš and Marek Zimányi

Obsah

1. Uvod

V tejto prednáške sa sústredíme na pestrú paletu algoritmov implementovaných v grafických systémoch. V našom prístupe *zhora nadol* sa väčšinou sústredíme na vysoko úrovňovú počítačovú grafiku. V nasledujúcich lekciách budeme uvažovať, ako sú tieto veci implementované. Uvažujme otázku, ako mapovať dvojrozmerné grafické objekty na množinu pixelov, ktoré budú vyfarbené. Tento proces sa nazýva *scan conversion* alebo *rasterizácia*. Začneme diskusiou o rasterizačnom najjednoduchšom probléme – kreslením úsečky.

2. Rasterizácia

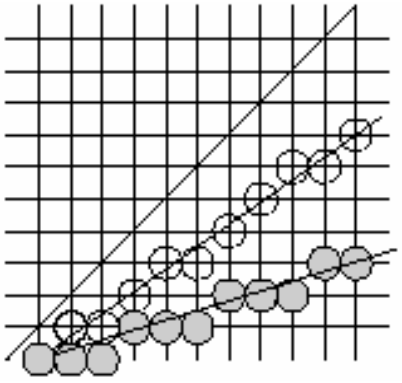
Predpokladajme, že naša obrazovka je reprezentovaná ako celočíselná mriežka, ktorej každý pixel predstavuje kružnicu o polomere 1/2 umiestnenú v každom bode mriežky. Chceli by sme zvýrazniť množinu bodov, ktoré ležia na alebo blízko k čiare, t.j. chceme nakresliť úsečku z bodu $q=(q_x,q_y)$ do bodu $r=(r_x,r_y)$, kde súradnice sú celočíselné mriežkové body (ktoré dostaneme operáciou zaokrúhľovania). Uvažujme, že sklon úsečky je medzi 0 a 1 a tiež nech platí $q_x<r_x$. Môže to vyzerať na prvý pohľad obmedzujúco. Ale nie je ťažké pretransformovať ľubovoľný problém kreslenia čiary, aby spĺňal práve tieto kritériá. Napríklad, ak absolútna hodnota sklonu úsečky je väčšia ako 1, tak sa vymení úloha x a y, čoho výsledkom je úsečka s inverzným sklonom. Ak sklon je záporný, tak algoritmus sa ľahko modifikuje (namiesto pripočítavania použijeme odpočítavanie). Výmenou koncových bodov môžeme kresliť stále zľava doprava.

2.1. Prírastkový algoritmus rasterizácie úsečky

- naive algorithm, Digital Differential Analyzer (DDA), 3D DDA
- pomocou funkcie round()

Dané:

- usečka AB ; nie je rovnobežná s osou y a $A = [x_A, y_A]$; $B = [x_B, y_B]$; a nech $x_A > x_B$.
- úsečka: $y = mx + b$
- smernica: $m = \frac{\Delta y}{\Delta x} = \frac{y_B - y_A}{x_B - x_A}$; ***nech smernica*** $m \in \langle 0, 1 \rangle$ ..... (2.1)



Obr. 1: Algoritmus DDA

- posun v smere x o $\Delta x = 1$
- Zrejme teda najjednoduchší algoritmus pre vykreslenie úsečky bude vyzerať takto (funkcia *round*() zaokrúhli reálne číslo na celé):

1. vykresli bod $[x_A ; y_A]$, $x := x_A$, $y := y_A$
2. $x := x + 1$, $y := y + m$, vykresli bod $[x; round(y)]$
3. ak $x < x_B$ tak opakuj krok 2

2.2. Bresenhamov algoritmus

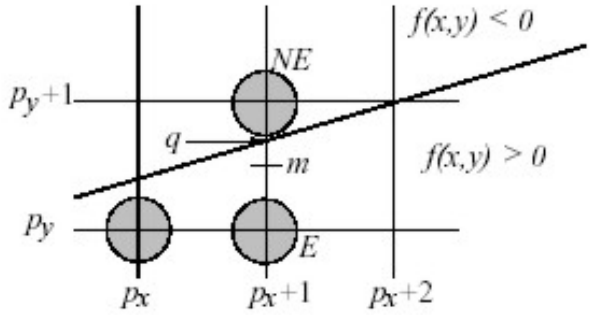
Je to jeden z najstarších známych algoritmov v oblasti počítačovej grafiky. Začneme predpokladom, že úsečka je v implicitnom tvare (využije sa to iba pre odvodenie a nie je explicitne vypočítaná v algoritme). Mame úsečku z $q = (q_x; q_y)$ do $r = (r_x; r_y)$,

$$f(x,y)=ax+by+c=0$$

Ak zvolíme $d_x = r_x - q_x$, $d_y = r_y - q_y$, ľahko sa ukáže (pomocou substitúcie) , že $a = d_y$, $b=-d_x$ a $c_x=-(q_x r_y - r_x q_y)$. Všimnime si, že všetky koeficienty sú celočíselné. Tiež si všimnime, že $f(x,y)>0$ pre body, ktoré ležia pod čiarou a $f(x,y)<0$ pre body nad čiarou. Pre dôvody, ktoré budú neskôr jasné, budeme používať ekvivalentnú reprezentáciu vynásobenú 2

$$f(x,y)=2ax+2by+2c=0$$

Teraz nasleduje intuitívne vysvetlenie Bresenhamovho algoritmu. Pre každú celočíselnú hodnotu x chceme určiť, ktorá celočíselná hodnota y je najbližšie k čiare. Predpokladajme že sme práve vykreslili pixel (p_x,p_y) a pýtame sa, ktorý pixel ďalej vykreslíme. Pretože sklon je medzi 0 a 1, z toho vyplýva, že ďalší pixel, ktorý bude vykreslený je buď pixel na východ ($E =(p_x+1,p_y)$) alebo pixel na severovýchod ($NE =(p_x+1,p_y+1)$). Nech q označuje presnú y - hodnotu (reálne číslo) čiary v bode $x= p_x+1$. Nech $m=p_y+1/2$ označuje hodnotu medzi E a NE. Ak $q<m$, ako ďalší bod vyberieme E, inak vyberieme NE. Ak $q=m$, tak môžeme vybrať napr. E. Viď obr. 2.



Obr. 2.: Bresenhamov algoritmus

Pre určenie bodu, ktorý vyberieme, použijeme rozhodovaciu premennú *D*, ktorá bude predstavovať hodnotu *f* v strednom bode *m*. Preto

$$D = f(p_x + 1, p_y + (1/2))$$
$$= 2a(p_x + 1) + 2b(p_y + 1/2) + 2c$$
$$= 2ap_x + 2bp_y + (2a + b + 2c)$$

Ak $D>0$, tak *m* je pod čiarou a preto bod *NE* je bližšie k čiare. Na druhej strane, ak $D<0$, tak *m* je nad čiarou a preto bod *E* je bližšie k čiare. (Poznámka: Teraz je jasné, prečo sme vynásobili *f(x,y)* dvoma. *D* bude mať celočíselnú hodnotu).

Dobrá správa je, že *D* je celočíselná veličina. Zlá správa je, že potrebujeme najmenej 2 násobenia a 2 sčítania pre výpočet *D*. Jeden zo šikovných trikov za Bresenhamovým algoritmom je vypočítať *D* inkrementálne. Predpokladajme, že poznáme aktuálnu hodnotu *D* a chceme určiť jej ďalšiu hodnotu. Ďalšia hodnota *D* závisí od týchto dvoch okolností:

Pokračuj ďalej na E: Potom ďalší stredný bod bude mať súradnice $(p_x+2, p_y+(1/2))$ a preto nová hodnota *D* bude

$$D_{new} = f(p_x + 2, p_y + (1/2))$$
$$= 2a(p_x + 2) + 2b(p_y + 1/2) + 2c$$
$$= 2ap_x + 2bp_y + (4a + b + 2c)$$
$$= 2ap_x + 2bp_y + (2a + b + 2c) + 2a$$
$$= D + 2a = D + 2d_y$$

, čo je vlastne aktuálna hodnota *D* plus $2d_y$.

Pokračuj ďalej na NE: Potom ďalší stredný bod bude mať súradnice $(p_x+2, p_y+1+(1/2))$ a preto nová hodnota *D* bude

$$D_{new} = f(p_x + 2, p_y + 1 + (1/2)) = 2a(p_x + 2) + 2b(p_y + 3/2) + 2c = 2ap_x + 2bp_y + (4a + 3b + 2c) = 2ap_x + 2bp_y + (2a + b + 2c) + (2a + 2b) = D + 2(a + b) = D + 2(d_y - d_x)$$

, čo je vlastne aktuálna hodnota D plus $2(d_y - d_x)$.

Všimnime si, že v tomto prípade musíme vykonať iba jedno sčítanie (ak predpokladáme, že máme predpočítané hodnoty $2d_y$ a $2(d_y - d_x)$). Preto vnútorný cyklus algoritmu je veľmi efektívny. Jediná vec ktorá nám ešte ostáva je vyrátať počiatočnú hodnotu D. Keďže začíname v bode (q_x, q_y), počiatočný stredný bod je v $(q_x + 1, q_y + 1/2)$ a počiatočná hodnota D je

$$D_{init} = f(q_x + 1, q_y + (1/2)) = 2a(q_x + 1) + 2b(q_y + 1/2) + 2c = (2aq_x + 2bq_y + 2c) + (2a + b) = 0 + 2a + b = 2d_y - d_x$$

Teraz si zhrnieme celý algoritmus. Odvoláme sa na naše predpoklady, že $q_x < r_x$ a sklon je medzi 0 a 1. Všimnime si, že hodnoty $2d_y$ a $2(d_y - d_x)$, ktoré sa objavili v cykle, sú predpočítané a preto

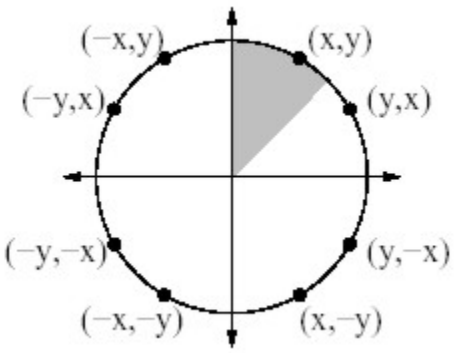
```
void bresenham(IntPoint q, IntPoint r)
{
     int dx, dy, D, x, y;
     dx = r.x – q.x;          // line width and height
     dy = r.y – q.y;
     D = 2*dy – dx;           // initial decision value
     y = q.y;                 // start at (q.x,q.y)
     for (x=q.x; x<=r.x; x++) {
          writePixel(x,y);
          if (D<=0) D += 2*dy;// below midpoint–go to E
          else {              //above midpoint–go to NE
               D += 2*(dy-dx); y++;
          }
     }
}
```

každý krok zahrňuje iba porovnanie a sčítania celočíselných hodnôt.

Bresenhamov algoritmus môže byť modifikovaný pre kreslenie iných druhov kriviek. Napríklad, existuje podobný Bresenhamov algoritmus na kreslenie kruhového oblúka. Zovšeobecnenie Bresenhamovho algoritmu sa nazýva algoritmus stredného bodu (*midpoint algorithm*) kvôli použitiu stredného bodu medzi dvoma bodmi ako základný diskriminátor.

2.3. Rasterizácia kružnice

Uvažujme ako zovšeobecniť Bresenhamov algoritmus rasterizácie úsečku na rasterizáciu kružnice. Urobíme niekoľko predpokladov, aby sme zjednodušili odvodenie algoritmu. Po prvé, predpokladajme, že stred kružnice leží v začiatku súradnicovej sústavy. Nech R (celé číslo) označuje polomer kružnice. Prvé pozorovanie o kružnici je také, že stačí uvažovať o kreslení kružnicového oblúka v kladnom kvadrante od $\pi/4$ do $\pi/2$, pretože všetky ostatné body môžu byť odvodené z 8-symetrie.



Obr.3.: 8-symetria kružnice

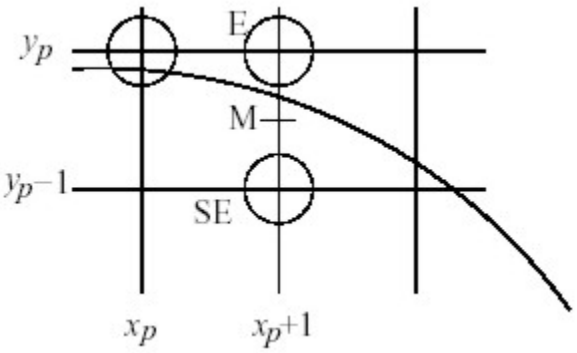
Podobne ako pri rasterizácii úsečky potrebujeme reprezentáciu v implicitnom tvare. Preto budeme používať

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

Všimnime si, že pre body vnútri kružnice (pod oblúkom) je tento výraz záporný, a pre body mimo kružnice (nad oblúkom) je tento výraz kladný.

Predpokladajme, že sme práve dokončili kreslenie bodu (x_p,y_p) a chceme vybrať ďalší bod na kreslenie (kreslíme v smere hodinových ručičiek). Pretože sklon kružnicového oblúka je medzi 0 a -1, naša voľba v každom kroku je medzi susedom na východ E a susedom na juhovýchod SE.

Ak kružnica prechádza na stredným bodom *M* medzi týmito bodmi, potom budeme pokračovať na *E*, inak na *SE*.



Obr.4. : Bresenhamov algoritmus pre kružnicu

Ďalej, budeme potrebovať rozhodovaciu premennú. Nech jej hodnota je *F(M)* definovaná ako

$$D = F(M) = F(x_p + 1 , y_p - 1/2) = (x_p+1)^2+(y_p -1/2)^2-R^2$$

Ak $D < 0$, potom *M* je pod oblúkom a preto bod *E* je najbližšie k čiare. Na druhej strane, ak $D <= 0$, potom *M* je nad oblúkom a preto bod *SE* najbližšie k čiare. Nová hodnota *D* bude závisieť od nášho výberu.

Pokračujeme na E: Nasledujúci stredný bod bude mať súradnice ($x_p + 2, y_p - 1/2$) a preto nová hodnota bude

$$\begin{aligned} D_{new} &= F(x_p + 2, y_p - 1/2) \\ &= (x_p + 2)^2 + (y_p - \tfrac{1}{2})^2 - R^2 \\ &= (x_p^2 + 4x_p + 4)+(y_p - \tfrac{1}{2})^2 - R^2 \\ &= (x_p^2 + 2x_p + 1) + (2x_p + 3) + (y_p - \tfrac{1}{2})^2 - R^2 \\ &= (x_p + 1)^2 + (2x_p + 3) + (y_p - \tfrac{1}{2})^2 - R^2 \\ &= D + (2x_p + 3) \end{aligned}$$

Nová hodnota *D* bude preto aktuálna hodnota plus $2x_p+3$.

Pokračujeme na SE: Nasledujúci stredný bod bude mať súradnice ($x_p + 2, y_p - 1 - 1/2$) a preto nová hodnota *D* bude

$$
\begin{aligned}
D_{new} &= F(x_p + 2, y_p - 3/2) \\
&= (x_p + 2)^2 + (y_p - 3/2)^2 - R^2 \\
&= (x_p^2 + 4x_p + 4) + (y_p^2 - 3y_p + 9/4) - R^2 \\
&= (x_p^2 + 2x_p + 1) + (x_p + 3) + (y_p^2 - 3y_p + 9/4) - R^2 \\
&= (x_p + 1)^2 + (2x_p + 3) + (y_p^2 - y_p + 1/4) + (-2y_p + 8/4) - R^2 \\
&= (x_p + 1)^2 + (y_p - \tfrac{1}{2})^2 + (2x_p + 3) + (2y_p + 8/4) - R^2 \\
&= D + (2x_p - 2y_p + 5)
\end{aligned}
$$

Nová hodnota D bude preto aktuálna hodnota plus $2(x_p - y_p) + 5$.

Ešte musíme vypočítať iniciálnu hodnotu D. Keďže začíname v $x = 0, y = R - 1/2$, preto začiatočná hodnota D bude

$$
\begin{aligned}
D_{init} &= F(1, R - 1/2) \\
&= 1 + (R - 1/2)^2 - R^2 \\
&= 1 + R^2 - R + \tfrac{1}{4} - R^2 \\
&= 5/4 - R
\end{aligned}
$$

Niečo tu ešte nie je v poriadku, pretože sme sa chceli vyhnúť reálnej aritmetike. Aj napriek tomu tu existuje veľmi šikovná pomôcka, ktorú môžeme použiť. My sa iba zaujímame, o to či D je kladné alebo záporné. Kedykoľvek zmeníme hodnotu D, robíme to pomocou celočíselného prírastku. Preto D je v tvare $D' + 1/4$, kde D' je celé číslo. Tento výraz je kladný práve vtedy keď D' je kladné. Preto môžeme zanedbať hodnotu 1/4. Preto inicializujeme $D_{init} = 1 - R$ (odčítali sme 1/4) a algoritmus funguje ako predtým.

1. Úlohy

1. Ako vyzerá 3D DDA?
2. Ako vyzerá 3D Bresenhamov alg.?

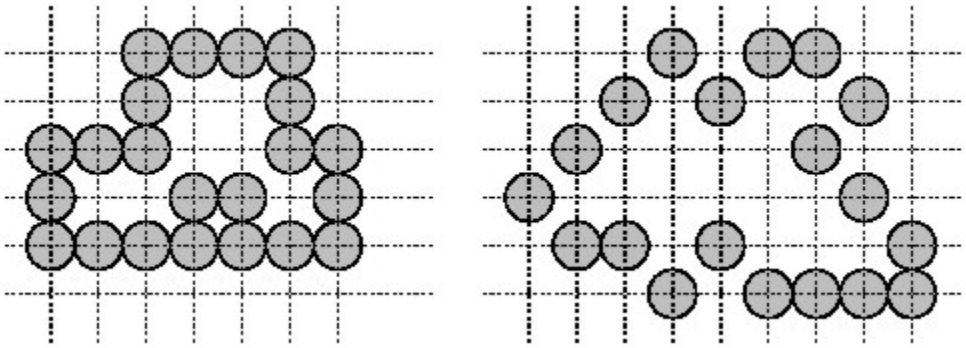
3. Vypĺňanie oblastí

V mnohých prípadoch nechceme kresliť iba samotnú krivku, ale namiesto toho chceme vyplniť oblasť . Sú dve základné metódy pre definovanie oblasti, ktorá je potrebné vyplniť. Prvá z nich je vypĺňanie polygónu, pre ktorý máme zadané vrcholy. Túto metódu prediskutujeme neskôr. Druhá metóda je založená na tom, že hranica oblasti je definovaná pomocou množiny bodov a cieľom je vyplniť všetko vo vnútri oblasti. Najprv prediskutujeme tento druhý typ metódy, pretože nám prinesie niekoľko zaujímavých záverov.

Intuitívna predstava, ktorú máme je, že uvažujeme o množine bodov, ktoré predstavujú hranicu nejakej oblasti, podobne ako uzavretá krivka v rovine. Takáto množina bodov by mala byť spojitá a podobne ako krivka by mala rozdeliť nekonečnú mriežku bodov na dve časti, *vnútro* (*interior*) a *vonkajšok* (*exterior*). Definujme *4-susedov* nejakého bodu ako body, ktoré sa bezprostredne nachádzajú na sever, juh, východ alebo západ od tohoto bodu. Definujme *8-susedov* ako zjednotenie 4-susedov a 4 najbližších diagonálnych susedov. Preto existujú dva prirodzené spôsoby ako definovať spojitosť, podľa toho ktorý typ susedov uvažujeme.

4-susednosť: Množina je 4-susedná, ak pre ľubovoľné dva body z množiny existuje cesta z prvého bodu do druhého, ktorá celá leží v množine pohybujúc sa z daného bodu do jedného z jeho 4-susedov.

8-susednosť: Množina je 8-susedná, ak pre ľubovoľné dva body z množiny existuje cesta z prvého bodu do druhého, ktorá celá leží v množine pohybujúc sa z daného bodu do jedného z jeho 8-susedov.



Obr. 5.: 4-susedná (vľavo) a 8-susedná (vpravo) množina bodov

Všimnime si, že 4-susedná množina je aj 8-susedná, ale opačne to neplatí. Jordanova teoréma hovorí, že uzavretá krivka v rovine rozdeľuje túto rovinu na 2 spojité oblasti – vnútro a vonkajšok. Nedefinovali sme, čo v tomto kontexte rozumieme pod uzavretou krivkou, ale aj okrem toho sú tu nejaké problémy. Všimnime si, že ak ohraničujúca krivka je 8-spojitá, potom vo všeobecnosti nie je pravda, že táto krivka rozdeľuje nekonečnú mriežku na 8-susedé oblasti, pretože ako vidno na obrázku 5, obe oblasti (vnútro aj vonkajšok) môžu byť spojené pomocou 8-susednej cesty. Ale je tu zaujímavý spôsob ako vyriešiť tento problém. Ak predpokladáme, že hraničná krivka je 8-susedná, potom oblasť, ktorú definuje, je 4-susedná. Podobne ak uvažujeme, že hranica je 4-susedná, potom je zrejmé že ňou definovaný región je 8-susedný.

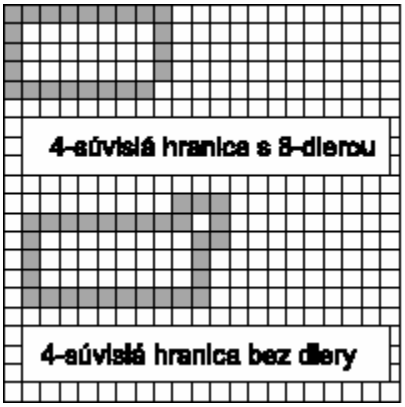
3.1.Rekurzívne vypĺňanie (Flood fillig)

Bez ohľadu na to, ako je definovaná spojitosť, my sa zaoberáme algoritmickou otázkou ako vyplniť oblasť. Predpokladajme, že máme daný začiatočný bod $p=(p_x,p_y)$. Chceme navštíviť všetky body v tej istej spojitej oblasti (použijeme napr. 4-susednosť) a priradiť im tú istú farbu. Budeme predpokladať, že všetky body na začiatku majú spoločnú farbu pozadia a priradíme im novú farbu oblasti. Princíp spočíva v pohybovaní sa dovtedy, kým neuvidíme 4-suseda s farbou pozadia a priradíme mu farbu oblasti. Aby sme sa vyhli backtrackingu, môžeme udržiavať zásobník nedokončených bodov. Jeden spôsob ako to implementovať tento zásobník je použitie rekurzie. Táto metóda sa nazýva *flood filling*. Výsledná procedúra sa ľahko zapíše, ale nie je to najefektívnejší spôsob ako vyriešiť tento problém.

```
void floodFill(intPoint p) {
     if ( getPixel(p.x,p.y)==backgroundColor ) {
          setPixel(p.x,p.y,regionColor);
          floodFill(p.x-1,p.y);    // apply to 4-neghbours
          floodFill(p.x+1,p.y);
          floodFill(p.x,p.y-1);
          floodFill(p.x-1,p.y+1);
     }
}
```

3.2.Alg. vypĺňania do hraničných bodov (Bound fill)

Na vstupe nech je 4-suvisla hranica farby BOUND_COLOR. (V tomto algoritme potrebujeme hranicu ktorá je 4-suvisla v silnejšom zmysle ako sme definovali v úvode. Hranica nesmie mat 8-dieru. Vid. obrázok 2) Na výstupe ma byt oblasť ohraničená touto hranicou zafarbená farbou NEW_COLOR. Parametre x, y sú znova súradnice vnútorného bodu oblasti.



Obr. 4-súvislá hranica

```
void Bound_fill_8(int x,int y,int bound_color,int new_color)
{
        if (read_pixel(x,y)<>bound_color && read_pixel(x,y)<>new_color)
        {
                write_pixel(x,y,new_color);
                Bound_fill_8(x,y-1,bound_color,new_color);
                Bound_fill_8(x,y+ 1,bound_color,new_color);
                Bound_fill_8(x-1,y,bound_color,new_color);
                Bound_fill_8(x+ 1,y,bound_color,new_color);
                Bound_fill_8(x-1,y-1,bound_color,new_color);
                Bound_fill_8(x+ 1,y-1,bound_color,new_color);
                Bound_fill_8(x-1,y+ 1,bound_color,new_color);
                Bound_fill_8(x+ 1,y+ 1,bound_color,new_color);
        }
}
```

3.3.Rasterizácia polygónu

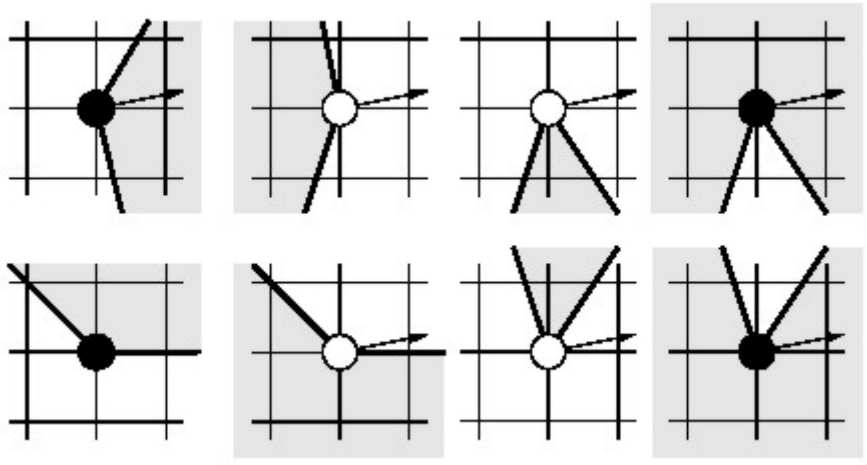
Vypĺňanie polygónu: Jedna z fundamentálnych nízkoúrovňových operácií, ktorá musí byť efektívne implementovaná v grafických systémoch, je rasterizácia polygónov. Budeme uvažovať prípad, keď polygón je vyplnený jedinou farbou, ale vypĺňanie polygónu typicky zahrňuje

a tieňovanie ako aj mapovanie textúr. Aj napriek tomu, základná operácia určovania bodov patriacich do vypĺňanej oblasti je fundamentálna aj pre tieto ostatné úlohy.

The braking rules: Úloha vypĺňania polygónov je oveľa prefíkanejšia ako na prvý pohľad vyzerá. Napríklad je veľmi dôležité, aby oblasti, ktoré zdieľajú spoločnú hranu, boli vykreslené akoby tam žiadna hranica nebola. Prekrytie (zapísanie toho istého bodu dvakrát) tiež predstavuje nebezpečenstvo, pretože existujú zápisové režimy (napr. použitie akumulačného buffra) , kde dvojnásobné zapísanie toho istého pixela má za následok úplne odlišnú farbu.

V tomto texte navrhneme nasledujúce kritériá pre presné určenie, ktoré body vykresliť ako súčasť polygónu. Tieto pravidlá budeme nazývať *ILB* (interior-left-bottom). Pixel sa vykreslí, ak leží buď vnútri polygónu, alebo na ľavej hrane, resp. spodnej hrane. Bohužiaľ, je veľmi ťažké opísať podmienky za ktorých tieto pravidlá platia. Namiesto toho budeme používať trochu modifikované pravidlá. Budeme ich nazývať *LrsU* (Look Right and slightly Up).

Myšlienka je nasledovná. Predpokladajme, že máme bod o ktorom chceme určiť, či je časťou vyplneného polygónu alebo nie. Uvažujme bod infinitezimálne napravo od tohoto bodu., ale nie priamo napravo, pretože tam môže byť horizontálna hrana. Namiesto toho sa budeme pozerať napravo a mierne hore. Ak tento bod bude vnútri polygónu, tak ho vyplníme, inak ho nevyplníme. Príklad aplikovania tohoto pravidla je zobrazený na obrázku.



Obr. 6.: LrsU pravidlá

Môžeme sa opýtať ako vypočítať infinitezimálnu veličinu. Je potrebné poznamenať, že nikdy nepočítame infinitezimálne hodnoty, Namiesto toho jednoducho algoritmus robí diskrétne rozhodnutia, ako keby boli aplikované infinitezimálne perturbácie.

Scan line algoritmus: Štandardný algoritmus na vypĺňanie polygónov sa nazýva *scan-line* algoritmus. Je to príklad všeobecnej paradigmy geometrických algoritmov, ktorá sa vo výpočtovej geometrii nazýva *zametací algoritmus*. Tieto algoritmy skenujú riadok za riadkom zdola polygónu nahor, pričom riadky sa prechádzajú zľava doprava. Ak sa v skenovacom riadku prvý krát dostaneme k hrane polygónu, začneme kresliť pixely. Keď sa dostaneme k nasledujúcej hrane, kreslenie prestane. Tento proces sa jednoducho opakuje, dokiaľ nie je preskenovaný celý polygón. Aj napriek tomu, táto jednoduchá idea v sebe ukrýva zákerné špeciálne prípady, ktoré je treba pozorne vyšetriť. Napríklad, čo sa stane, ak skenovací riadok prechádza cez vrchol polygónu? V ďalšom texte načrtneme spôsoby ako sa vysporiadať s rôznymi špeciálnymi prípadmi pomocou ignorovania alebo rešpektovania určitých hrán a vrcholov.

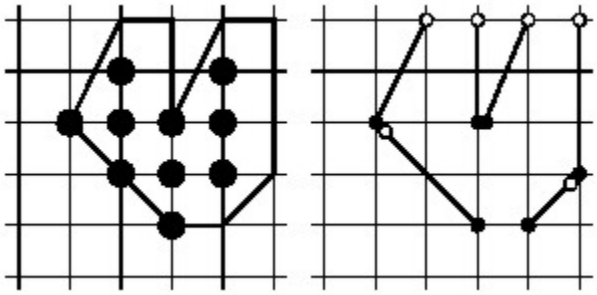
Horizontálne hrany: Všetky sú ignorované. Uvidíme, že toto vyrieši náš problém.

Bodom patriaci hrane: Ak bod patri priamo vnútru nehorizontálnej hrany, potom invertujeme kresliaci mód v tomto bode. T.j., ak sme nekreslili, tak začneme kresliť v tomto bode a ak sme kreslili, tak prestaneme kresliť v tomto bode.

Bod patriaci vrcholu: O každej nehorizontálnej hrane budeme uvažovať že je *otvorená* na vrchu a dole je *uzavretá*. T.j., najvrchnejší pixel nepatrí hrane, ale najspodnejší pixel patrí. Tu môžu vzniknúť rôzne prípady:

- **Jedna hrana hore, druhá hrana dole:** V tomto prípade sa kresliaci mód invertuje, lebo vrchol patrí práve jednej hrane.
- **Obidve hrany dole:** V tomto prípade bod nepatrí žiadnej hrane, preto sa kresliaci mód nemení.
- **Obidve hrany hore:** Pretože v tomto prípade bod patrí obidvom hranám, kresliaci mód sa invertuje dvakrát, a výsledok sa nemení.
- **Jedna horizontálna hrana:** Zadeklarovali sme, že takéto hrany ignorujeme, mód sa zmení, iba ak iný vrchol bude spodný.

Tvrdíme, že toto sú práve tie pravidlá, ktoré presne odpovedajú LRsU pravidlám pre kreslenie. Všimnime si, že sme vôbec nepoužili infinitezimálne hodnoty. Príklad je zobrazený na obrázku.



Obr. 6.: Scan-line algoritmus

Implementácia: Je tu niekoľko pozoruhodných aspektov pre efektívnu implementáciu scan-line algoritmu.

Na ktoré hrany natrafíme? Potrebujeme dátovú štruktúru, ktorá nám bude uchovávať riadok, ktorý v ďalšom bude skenovací riadok. Toto sa realizuje pomocou štruktúry *aktívna tabuľka hrán* (*active edge table AET*).

Kedy sa hrany pridajú do AET? Potrebujeme vedieť na začiatku každého nového skenovacieho riadku, ktoré hrany prestávajú byť aktívne a ktoré hrany začínajú byť aktívne. Toto vykonáme pomocou štruktúry nazvanej *tabuľka hrán* (edge table).

V ktorom bode skanovací riadok pretína hranu? Pre každú hranu v AET sledujeme jej x-ovú súradnicu v aktuálnom skenovacom riadku. Vo všeobecnosti je táto hodnota reálne číslo. Môžeme aktualizovať toto číslo použitím striktne celočíselnej aritmetiky?

Potrebný údaj, ktorý musíme vedieť o každej hrane, je jej aktuálna x-ová súradnica v skenovacom riadku. Ak inkrementujeme y-ovú súradnicu o 1, tak zodpovedajúca x-ová súradnica sa sa mení o $1/m$, kde m je sklon úsečky. Aby bolo jednoduché aktualizovať x-ovú súradnice každej hrany, budeme si uchovávať aktuálnu x-ovú súradnicu a hodnotu $1/m$ pre každú aktívnu hranu.

Tabuľka hrán: (Vo všeobecných zametacích algoritmoch potrebujeme *pioritnú frontu*, ktorá nám hovorí o udalostiach, ktoré nastanú) Pretože hlavné udalosti v scan-line algoritme je vstupovanie nových hrán, tabuľka hrán obsahuje zoznam hrán usporiadaný vzhľadom na ich spodné *y*-súradnice (horizontálne hrany sú jednoducho ignorované a v tabuľke nie sú uložené). Každá položka tabuľky hrán (aj aktívnej tabuľky hrán) má nasledujúcu štruktúru:

```
struct E_Node {
     int y_max;     // y-coordinate of edge's top vertex
     double x_curr; // x-coordinate of edge's botom vertex
     double recip_slope // reciprocal of edge slope (dx/dy)
     E_Node *next;
};
```

$$\text{recip\_slope} = \frac{1}{m} = \frac{x_A - x_B}{y_A - y_B}$$

Položka `x_curr` na začiatku obsahuje hodnotu *x*-ovú súradnicu spodného vrchola hrany. Neskôr uvidíme, že jej hodnota sa zmení, ak hrana sa stane aktívnou. Položky v tabuľke sú usporiadané podľa y_{min}, *y*-ovej súradnice spodného vrchola hrany. Máme pole `ET`, ktoré obsahuje jednu položku pre každý skenovací riadok. `ET[i]` ukazuje na spojovací zoznam vytvorený z `E_Node` pre ktoré y_{min}=`i`. Na začiatku sú všetky nehorizontálne hrany vložené do tabuľky hrán. Ak skenovací riadok navštívi riadok *i*, obsah `ET[i]`vložený do aktívnej tabuľky hrán.

Aktívna tabuľka hrán: Aktívna tabuľka hrán (active edge table AET) je spojovací zoznam obsahujúci hrany, ktoré majú prienik so skenovacím riadkom. Sú uložené vo vzostupnom poradí vzhľadom na *x*-ové súradnice priesečníka hrany a aktuálneho skenovacieho riadku. (Všimnime si, ak predpokladáme, že hrany polygónu nie sú samopretínajúce sa, nepotrebujeme triediť tento zoznam, iba vtedy, ak sa pridáva nová hrana).

Scan-line algorithm: Here is the scan-line algorithm.

1. zisti, ktoré hrany mnohouholníka sú vodorovné, ktoré vrcholy neextremálne
2. hrany, ktoré nie sú vodorovné zapíš do TH, TAH inicializuj ako prázdnu, y := ymin
3. Kým TH, alebo TAH sú neprázdne opakuj

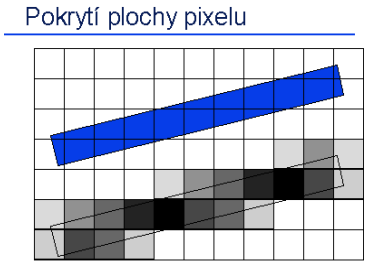
a. vyber z TH hrany v riadku y a daj ich do TAH
b. usporiadaj hrany v TAH podľa *x*-ovej súradnice (druhá položka v jednotlivých záznamoch)
c. vyber za sebou idúce úseky a vykresli ich
d. zruš tie hrany v TAH, pre ktoré $y_{max} = y$
e. pre hrany v TAH zmeň x na $x + 1/m$
f. $y := y + 1$

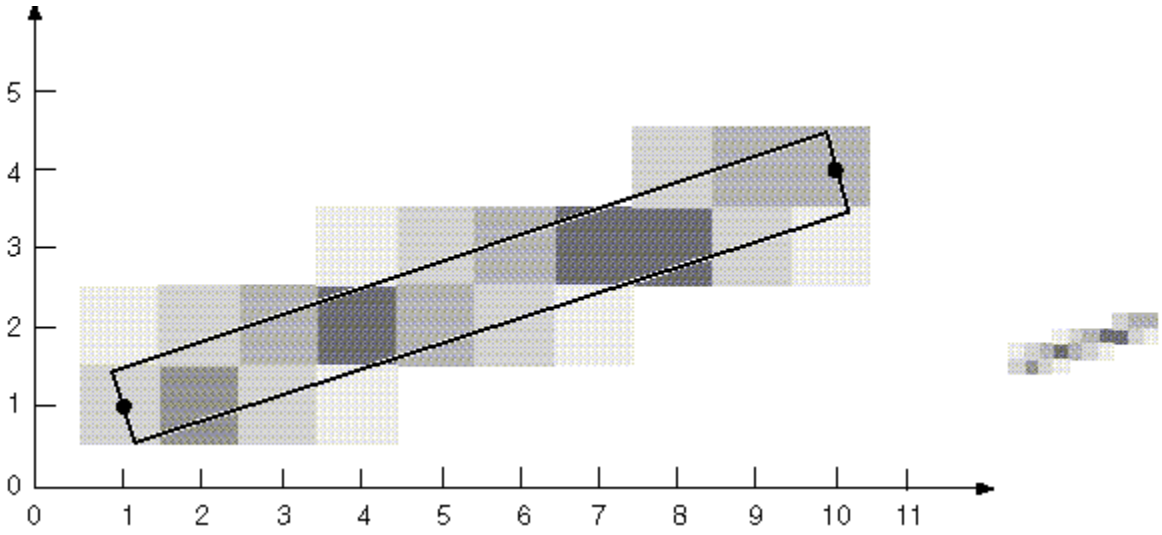
4. Anti-aliasing

Alias – neželaný optický jav, kaz obrazu.

- priečne pruhy pri trse úsečiek (dialóg. krok dlhší ako horizontálny)
-

Pokrytie plochy pixelu





Source: Foley, VanDam, Feiner, Hughes - Computer Graphics, Second Edition, Addison Wesley

Pokrytí plochy pixelu

- ke kreslení použiji **více odstínů** dané barvy
  - – zvětším prostorové rozlišení na úkor barevného
- pixely i kreslené objekty jsou **plošné útvary**
- každý pixel rozsvítím intenzitou úměrnou **ploše jeho pokryté části**

Prealiasing
Postaliasing
Supersampling
Literatura: