Obsah

Zoznam obrázkov

1 Úvod

Učebný text je vytvorený podľa hlavného zdroja [6] s úpravami a doplneniami. Ďalej sú v texte použité informácie z literatúry.

Tento text vznikol ako praktická časť diplomovej práce Kristíny Vojtovej s názvom Interaktívny výučbový portál pre úvodný kurz počítačovej grafiky v roku 2017.

2 Nevyhnutná matematika

2.1 Základné pojmy rastrovej grafiky

Jednou z najdôležitejších úloh počítačovej grafiky je zobrazovanie modelov objektov zo spojitého priestoru E^3 do dvojrozmerného diskrétneho priestoru, ktorý predstavuje obrazovka počítača [6].

Prakticky všetky súčasné grafické displeje používajú metódu rastrového skenovania. Jej podstata spočíva v tom, že počítačový program pripravuje celý obraz v **pamäťovom bufri (frame buffer)** nazývanom aj **bitmapa**, pričom operuje s diskrétnymi bodmi. Aj samotný obraz pozostáva z tzv. obrazovkových bodov známych ako **pixely (picture elements)**, ktoré môžu byť zapnuté alebo vypnuté. Pixel je najmenší adresovateľný element obrazovky, jej najmenšia časť, ktorú môžeme nezávisle ovládať, čiže môžeme jej priradiť napríklad určitú farbu príp. odtieň – hodnotu šedej, ak zariadenie je monochromatické. Také jednoduché útvary, akými sú napr. krivky, špeciálne úsečky, potom zobrazujeme tak, že rozsvietime určitou intenzitou (jasom) reťazec pixlov medzi jej začiatočným a koncovým bodom. Dôležitú úlohu pri tvorbe obrazov dvoj a viac- rozmerných útvarov hrá vypĺňanie oblastí takýmito pixlami. Keďže pixely nie sú ideálne geometrické body, ale isté podmnožiny (najčastejšie štvorčeky) konečnej roviny, netvoria takto skonštruované objekty krivky, úsečky, mnohouholníky v matematickom zmysle slova, ale my ich budeme za krivky, úsečky resp. mnohouholníky považovať, pričom sa budeme snažiť, aby sa im čo najviac podobali. Ako sme povedali vyššie, každému pixlu je v bitmape priradené určité číslo špecifikujúce jeho farbu, odtieň šedej a pod. Z praktických dôvodov je užitočné predstavovať si obrazovku ako pravouholníkovú mriežku, ktorá má r riadkov očíslovaných zdola nahor nezápornými celými číslami $0, 1, ..., r-1$ a s stĺpcov očíslovaných zľava doprava číslami $0, 1, ..., s-1$. Potom každý pixel bude jednoznačne určený dvojicou tzv. obrazovkových súradníc $(x, y) \in \{0, 1, ..., r-1\} \times \{0, 1, ..., s-1\}$ (Obr. 1.) [6].

Na druhej strane bitmapa býva uložená v pamäti počítača ako jednorozmerné pole, ktorého indexy i $\in \{0, 1, ..., r \times s - 1\}$ sú také, že pixlu (x, y) priradia index $i = x + ys$. Napr. pre $r = 5$ a $s = 7$ je všetkých pixlov 35 a pixlu $(5, 3)$ prislúcha $i = 5 + 3.7 = 26$. Na Obr. 1 je ilustrované vzájomne jednoznačné priradenie medzi množinou pixlov a frame bufrom (bitmapou) [6].

	(0,y); i	(1,y); i	(2,y); i	(3,y); i	(4,y); i	(5,y); i	(6,y); i
4	(0,4); 28	(1,4); 29	(2,4); 30	(3,4); 31	(4,4); 32	(5,4); 33	(6,4); 34
3	(0,3); 21	(1,3); 22	(2,3); 23	(3,3); 24	(4,3); 25	(5,3); 26	(6,3); 27
2	(0,2); 14	(1,2); 15	(2,2); 16	(3,2); 17	(4,2); 18	(5,2); 19	(6,2); 20
1	(0,1); 7	(1,1); 8	(2,1); 9	(3,1); 10	(4,1); 11	(5,1); 12	(6,1); 13
0	(0,0); 0	(1,0); 1	(2,0); 2	(3,0); 3	(4,0); 4	(5,0); 5	(6,0); 6
	0	1	2	3	4	5	6

Obr. 1: Bitmapa [6]

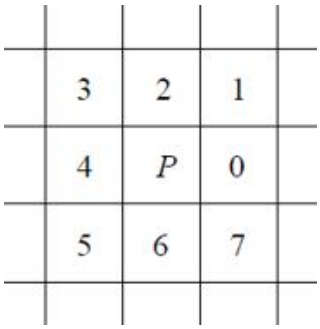
Obrazovka s rozlíšiteľnosťou napr. 512 na 512 pixlov sa skladá z $512^2 = 262144$ pixelov uložených do 512 riadkov a 512 stĺpcov očíslovaných číslami $0, 1, \ldots, 511$. Tie vytvárajú množinu usporiadaných dvojíc nezáporných celých čísel $0, 1, ..., 511 \times 0, 1, ..., 511$, ktorá sa nazýva **súradnicovým priestorom obrazovky** – alebo **DC priestorom (Device Coordinates Space)**. Ak by aplikačný program pracoval len v malých celočíselných používateľských súradniciach, mohli by sme tento priestor považovať za vyhovujúci pracovný priestor pre počítačovú grafiku. Existujúce grafické systémy však nemôžu aplikačné programy takto obmedzovať, a preto je potrebné, aby DC-priestor akceptoval aj reálne súradnice. Treba ho teda rozšíriť na priestor reálnych súradníc a to tak, aby pixely zodpovedali práve tým bodom rozšíreného DC, ktoré majú celočíselné súradnice. Toto možno dosiahnuť napr. tak, že ideálnemu bodu obrazovky s reálnymi súradnicami

(x, y); $x \geq 0$, $y \geq 0$ priradíme pixel so súradnicami (round x,round y), kde napríklad: round x = k $\Rightarrow$ ak k je to jediné nezáporné celé číslo, pre ktoré platí: $x \in \langle k - 0.5, k + 0.5)$.

Pripomenieme ešte, že dĺžky na obrazovke sa nemerajú v cm ani v palcoch, ale v pixloch, čo predstavuje dĺžku jednej strany pixla. Keďže každému ideálnemu bodu (x, y); $x, y \in \mathbb{R}$ (presnejšie z istej podmnožiny $\mathbb{R}$) vieme jednoznačne priradiť pixel a pixlu miesto vo frame bufri, prislúcha toto miesto aj bodu (x, y) [6].

Bitová mapa je teda časť pamäte počítača, ktorú si predstavme ako štvorcovú sieť. Jednotlivé jej prvky zodpovedajú obrazovým bodom na rastrovom zariadení. Ak je v niektorom prvku bitovej mapy zapísaná hodnota, tak je príslušný pixel rastra vysvietený farbou prislúchajúcou tejto hodnote. Pri vykresľovaní úsečky sa pre vykreslenie jednotlivých bodov volá funkcia, ktorá na určené miesto bitovej mapy zapíše hodnotu farby.

V niektorých algoritmoch vypĺňania budeme potrebovať pojem **susednosť**. V bitovej mape má každý vnútorný bod ôsmich susedov. Je zvykom označovať ich spôsobom, ako je uvedené na obrázku 2. Susedov s číslami 0, 2, 4, 6 nazývame **priami susedia, alebo 4-susedia**, susedov s číslami 1, 3, 5, 7 nazývame **nepriami susedia**. Všetkých spolu nazývame **8-susedia**.



Obr. 2: Susednosť v bitovej mape [6]

Oblasť je množina prvkov bitovej mapy, ktoré sú súvislo pospájané. Prvkom oblasti hovoríme aj **body**. Oblasti môžu byť zadané všetkými svojimi bodmi (používa sa aj pojem vnútorne definovaná oblasť) je zadaná farba týchto bodov a typ súvislosti. Pri oblasti danej hranicou je zadaná farba hranice.
Podľa typu súvislosti rozdeľujeme oblasti na:

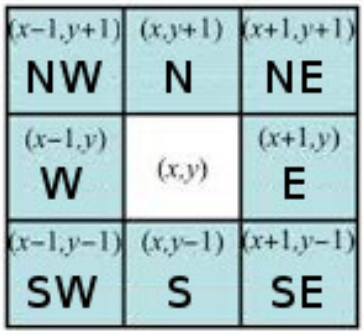
- **4-súvislé (4-connected):** Oblasť, ktorej každé dva prvky môžeme spojiť postupnosťou 4-susedov. Sused sa teda môže nachádza iba vo vodorovnom alebo zvislom smere. Smery označujeme pomocou skratiek N, S, E, W (z ang. North - Sever, South - Juh, East - Východ, West - Západ)(Obr. 3b).

- **8-súvislé (8-connected):** Oblasť, ktorej každé dva prvky môžeme spojiť postupnosťou 8-susedov, teda najbližší sused každého pixla môže ležať v ôsmich okolitých smeroch. Tieto smery označujeme pomocou skratiek N, S, E, W, NE, SE, SW, NW (z ang. Northeast - Severovýchod, Southeast - Juhovýchod, Southwest - Juhozápad, Northwest - Severozápad). 8-susednosť sa využíva aj pri algoritmoch vyplňovania oblastí (Obr. 3a).

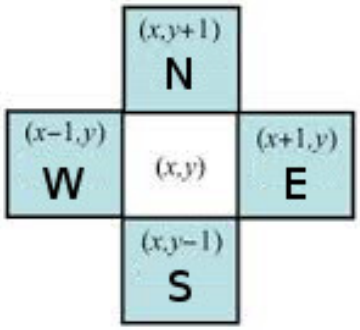
Je zrejmé, že každá 4-súvislá množina je 8-súvislá, ale naopak to neplatí.

2.2 Dvojrozmerné geometrické transformácie

Keďže téma transformácie je dobre spracovaná v použitej slovenskej literatúre, v tejto podkapitole čerpáme informácie z [1].



(a) 8 susednosť



(b) 4 susednosť

Obr. 3: Susednosť pixelov

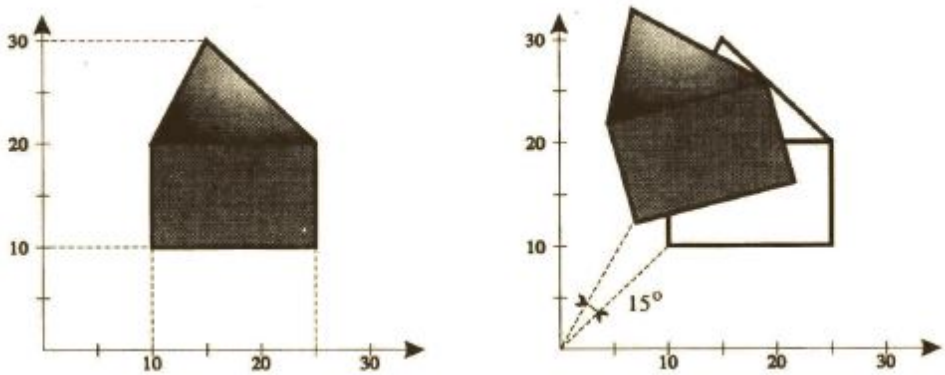
V úlohách často vytvárame alebo používame obraz určitého objektu, ktorý treba vhodne transformovať, teda vybrať určitú časť a prepočítať súradnice na dané výstupné zariadenie. V tejto podkapitole si ukážeme stručne vzájomnú súvislosť medzi maticami a lineárnymi transformáciami v rovine a odvodíme matice najčastejšie používaných rovinných transformácií: posunutia, otočenia a škálovania.

Medzi transformáciami a maticami je vzťah: každej lineárnej transformácií vieme priradiť určitý typ matice. Pri lineárnych transformáciách dvojrozmerných vektorových priestorov používame maticu typu 2x2. Každý bod v rovine budeme stotožňovať s vektorom typu 1x2. Pri vyjadrovaní bodov pomocou matíc 1x3 hovoríme o **homogénnych súradniciach**. Bodu so súradnicami (x, y) priradíme homogénnu súradnicu $(x, y, 1)$ a vektoru so súradnicami (x, y) priradíme $(x, y, 0)$.

2.2.1 Otočenie alebo rotácia

Transformácia objektu po kruhovej dráhe sa nazýva **otočenie**.

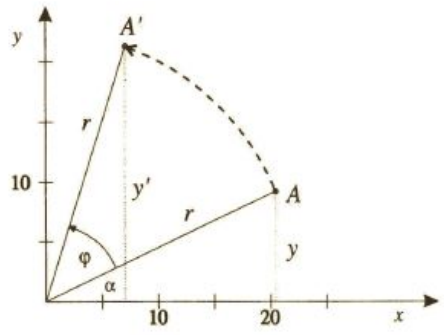
Je určené uhlom a stredom otočenia (pevným bodom). Príklad na obr. 4 ukazuje otočenie objektu so stredom v začiatku súradnicovej sústavy o uhol 15 stupňov.



Obr. 4: Otočenie pre daný objekt [1]

Odvodíme vzťah pre vyjadrenie tohto zobrazenia. Objekt môžeme otočiť o ľubovolný uhol φ vzhľadom k začiatku súradnicovej sústavy. Keďže sa všetky body otočia zhodne, stačí ukázať, čo sa deje s ľubovoľne vybratým bodom.

Pri otočení o uhol φ sa bod $A(x, y)$ zobrazí do bodu $A(x', y')$ (Obr. 5).



Obr. 5: Otočenie bodu A do bodu A'[1]

Použijeme polárne súradnice bodov:

$$x = r.cos(\alpha) \qquad x' = r.cos(\alpha + \varphi)$$
$$y = r.sin(\alpha) \qquad y' = r.sin(\alpha + \varphi)$$

Pomocou súčtu uhlov pre goniometrické funkcie $sin(\alpha + \varphi)$ a $cos(\alpha + \varphi)$ môžeme vyjadriť x' a y' ako:

$$x' = r.cos(\alpha).cos(\varphi) - r.sin(\alpha).sin(\varphi) = x.cos(\varphi) - y.sin(\varphi)$$
$$y' = r.cos(\alpha).sin(\varphi) + r.sin(\alpha).cos(\varphi) = x.sin(\varphi) + y.cos(\varphi)$$

Tým sme dostali transformačnú maticu R otočenia o uhol φ v homogénnych súradniciach:

$$R = \begin{pmatrix} cos(\varphi) & sin(\varphi) & 0 \\ -sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2.2.2 Škálovanie alebo zmena mierky

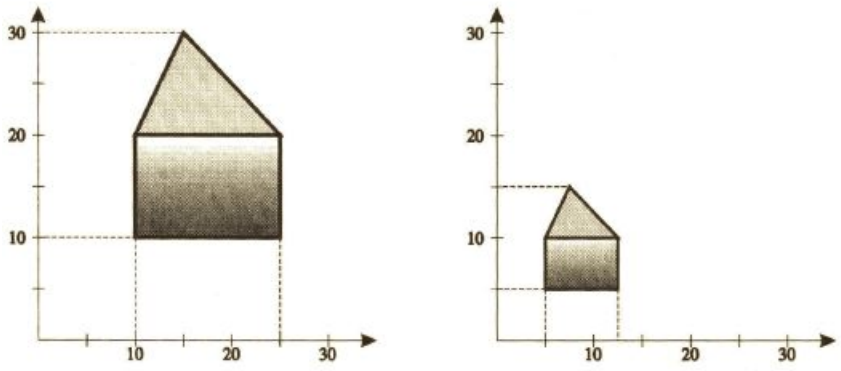
Pomocou tejto transformácie môžeme zmeniť veľkosť objektu. Tento typ zobrazenia všeobecne môžeme zapísať nasledovne:

$$x' = s_x.x$$
$$y' = s_y.y$$

kde s_x a s_y sa nazývajú škálovacie faktory a zodpovedajúca transformačná matica S má tvar:

$$S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Pre škálovanie vždy existuje pevný bod, v našom prípade nech je to začiatok súradnicovej sústavy. Na Obr 6 vidíme škálovanie objektu v tvare domčeka pre hodnoty $s_x = 0.5$ a $s_y = 0.5$.



Obr. 6: Škálovanie objektu[1]

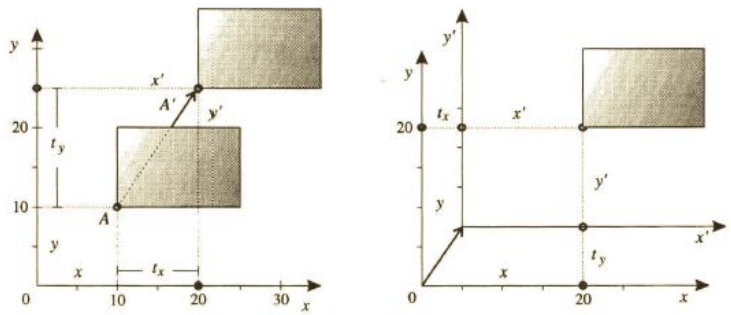
2.2.3 Posunutie objektu a sústavy súradníc

Posunutie je premiestnenie objektu z jednej pozície do druhej.

Na rozdiel od rotácie a škálovania nemá pevný bod. Posunutie v rovine je definované pomocou vektora (t_x, t_y). Zodpovedajúcu transformačnú maticu T môžeme zapísať nasledovne:

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Ale ako sa zmenia súradnice bodov pri posunutí sústavy súradníc? Na Obr 7 vidíme, že ten istý bod má vzhľadom k posunutej sústave súradníc súradnice posunuté o opačný vektor $(-t_x, -t_y)$.



Obr. 7: Posunutie bodu a sústavy súradníc [1]

Zodpovedajúca matica T' má potom tvar:

$$T' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix}$$

Matica T' je inverzná k matici posunutia T. Obdobne vieme odvodiť inverzné matice pre škálovanie a otočenie sústavy súradníc ako:

$$S' = \begin{pmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R' = \begin{pmatrix} cos(\varphi) & -sin(\varphi) & 0 \\ sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Všeobecne platí že pre transformáciu objektov a transformáciu súradnicovej sústavy platí vzťah navzájom inverzného zobrazenia, pokiaľ sú tieto zobrazenia jednoznačné.

2.2.4 Súmernosti podľa priamky

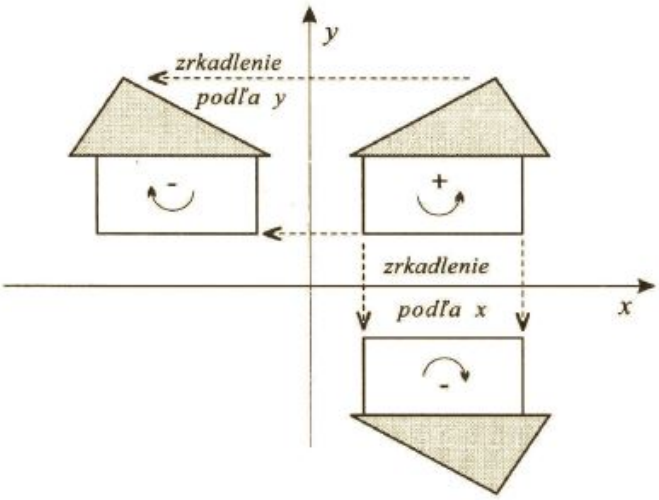
Niektoré aplikácie počítačovej grafiky vyžadujú aj iné transformácie v rovine. V prípade, že uvažujeme len transformácie zachovávajúce začiatok sústavy súradníc, potom môžeme napísať matice pre súmernosti(zrkadlenia) podľa osi x a y.

$$Z_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Z_y = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ide o špeciálny prípad škálovania, kde $s_x = -1$ alebo $s_y = -1$.

V obidvoch prípadoch transformácie menia orientáciu (Obr 8). Na toto treba dávať pozor. Ak totiž máme mnohouholník orientovaný proti smeru chodu ručičiek, potom takáto transformácia zmení orientáciu mnohouholníka na opačnú. Maticu súmernosti podľa ľubovolnej priamky získame skladaním s príslušnými otočeniami a základnými súmernosťami podľa osí.



Obr. 8: Súmernosti podľa osi x a y[1]

2.2.5 Zloženie dvojrozmerných transformácií

Ukážeme, ako môžeme využiť násobenie matíc pri skladaní zobrazení na príklade.

Chceme vyjadriť zmenu mierky so stredom v ľubovoľnom pevnom bode $A(x, y)$. Riešime to tak, že uskutočníme za sebou tri zobrazenia, a to:

1. Posunieme sústavu súradníc do bodu A.

2. Uskutočníme zmenu mierky v začiatku súradnicovej sústavy.

3. Posunieme späť bod A do pôvodného začiatku.

Každej tejto transformácií zodpovedá jedna transformačná matica. Výslednej zloženej transformácii zodpovedá nasledujúca matica:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{pmatrix} * \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ x(1-s_x) & y(1-s_y) & 1 \end{pmatrix}$$

Podobne otočenie zo stredom v ľubovoľnom bode $A(x, y)$ vykonáme pomocou týchto troch transformácií:

1. Posunieme sústavu súradníc do bodu A.

2. Uskutočníme otočenie okolo začiatku súradnicovej sústavy o uhol φ.

3. Posunieme späť bod A do pôvodného začiatku.

Výslednú zloženú transformáciu vyjadríme ako:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{pmatrix} * \begin{pmatrix} cos(\varphi) & sin(\varphi) & 0 \\ -sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{pmatrix} =$$

$$\begin{pmatrix} cos(\varphi) & sin(\varphi) & 0 \\ -sin(\varphi) & cos(\varphi) & 0 \\ x(1-cos(\varphi)) + ysin(\varphi) & y(1-cos(\varphi)) - xsin(\varphi) & 1 \end{pmatrix}$$

3 Rasterizácia

Podľa typu zobrazovacieho zariadenia sú výsledkami algoritmov buď postupnosti bodov – pixely alebo postupnosti úsečiek. V prvom prípade dostaneme rastrový obraz – raster a v tom druhom obraz vektorový [2].

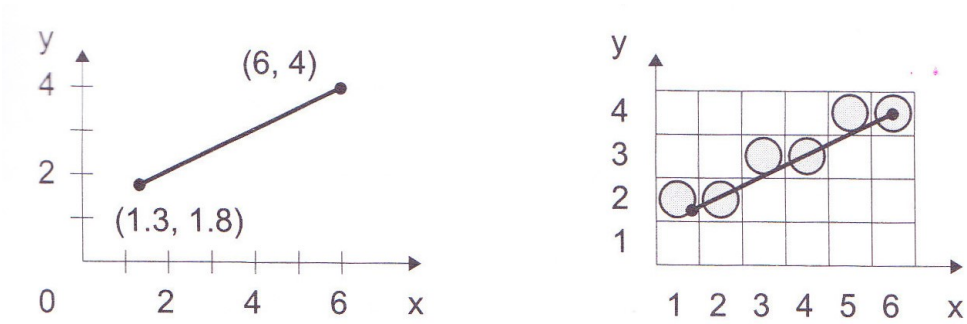
Rastrový obraz alebo raster si môžeme predstaviť ako celočíselnú sieť, ktorej každý uzol je stred kruhu o polomere $\frac{1}{2}$, predstavujúceho pixel. Cieľom je zobraziť (vysvietiť na obrazovke) množinu pixlov, ktorých geometrické stredy ležia na úsečke alebo blízko nej (pozdĺž danej úsečky). Tento proces sa nazýva **rasterizácia** a môžeme si ho predstaviť ako rozsvecovanie jednotlivých bodov rastra. Pod rasterizáciou vektora budeme rozumieť rasterizáciu orientovanej úsečky.

Za základné dvojrozmerné objekty považujeme úsečky, lomené čiary, kružnice, elipsy, mnohouholníky, krivky a textové reťazce [1]. Tieto objekty nazývame **základné grafické výstupné prvky (output primitives)** [2]. Počítačová grafika je orientovaná hlavne na tvorbu rastrového obrazu, a teda pri tvorbe rastrového obrazu je potrebné nájsť všetky pixely, ktoré reprezentujú daný grafický prvok [2].

V nasledujúcej kapitole ukážeme, ako je možné zobraziť pomocou rasterizácie úsečku, kružnicu a elipsu využitím troch základných algoritmov.

3.1 Algoritmy na rasterizáciu úsečky

Úsečka je vo všeobecnosti vyjadrená neceločíselnými súradnicami koncových bodov. Algoritmy na vykresľovanie do rastra ale počítajú s celočíselnými súradnicami. Preto sa na vstupe algoritmu koncové body zaokrúhľujú do celočíselnej aritmetiky. Chyba, ku ktorej v dôsledku zaokrúhlenia dôjde, je považovaná za bezvýznamnú (Obr. 9).



Obr. 9: Spojité a rastrové zobrazenie úsečky [2]

Úsečka je segment priamky a uvedieme tri spôsoby opisu:

- **Smernicová rovnica priamky:**

$$y = mx + b, \tag{3.1}$$

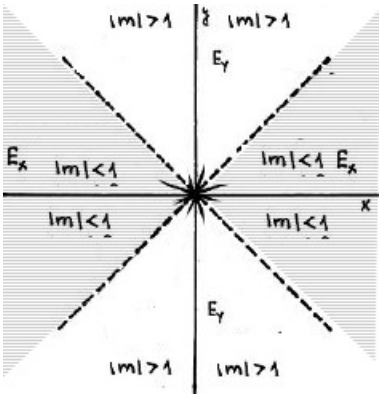
  kde m je smernica priamky a b je y-ová súradnica priesečníka priamky s osou y. Smernica priamky vyjadruje tangens uhla, ktorý zviera priamka s kladnou časťou osi x.
  Ak priamka prechádza bodom O (začiatok súradnicovej sústavy), tak $b = 0$, t.j.:

$$y = mx \tag{3.2}$$

- **Priamka určená dvoma bodmi:**
  Máme dva krajné body úsečky AB, $A = [x_A, y_A]$, $B = [x_B, y_B]$, $x_A \neq x_B$, $x_i, y_i \in \mathbb{N} \cup \{0\}, x_i \in \langle x_A, x_B \rangle, y_i \in \langle y_A, y_B \rangle$. Priamku určenú týmito dvoma bodmi dostaneme zo smernicového tvaru rovnice:
  $$y - y_A = \frac{y_B - y_A}{x_B - x_A}(x - x_A) \tag{3.3}$$
  Po úprave $y = mx + b$, kde $m = \frac{y_B - y_A}{x_B - x_A}$ a $b = y_B - mx_A$. Hodnoty $x_B - x_A$ a $y_B - y_A$ vyjadrujú prírastok v smere osi x a y a môžeme ich označiť ako $dx = x_B - x_A$ a $dy = y_B - y_A$. V literatúre sa dx označuje aj ako Δx a dy ako Δy.

- **Všeobecná rovnica priamky:**
  Rovnicu (3.1) vieme upraviť na tvar
  $$ax + by + c = 0, \tag{3.4}$$
  kde $a, b, c \in \mathbb{R}$. Toto vyjadrenie nazývame všeobecná rovnica priamky.

Ďalej nás budú zaujímať priamky $y = x$ a $y = -x$. Tie rozdelia rovinu na dve oblasti E_x a E_y, z ktorých každá je zjednotením dvoch protiľahlých vrcholových uhlov (Obr. 10).



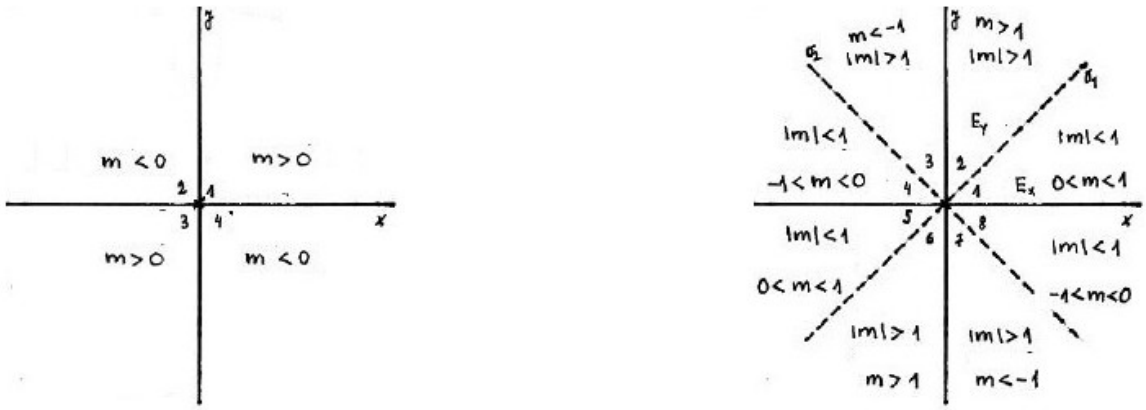
Obr. 10: Rozdelenie roviny na E_x a E_y [6]

- Oblasť E_x obsahuje os x-ovú. Je zrejmé, že do oblasti E_x patria tie a len tie priamky, ktorých smernice spĺňajú podmienku $|m| \leq 1$ a teda $| \ dy \ | \leq | \ dx \ |$. Sú to priamky s miernym stúpaním resp. klesaním vzhľadom k osi x. Ak úsečka patrí do oblasti E_x hovoríme, že má dominantný smer x.

- Oblasť E_y obsahuje os y. Do oblasti E_y patria tie a len tie priamky, ktorých smernice spĺňajú podmienku $|m| \geq 1$ a zároveň $| \ dy \ | \geq | \ dx \ |$. To sú priamky so strmým stúpaním resp. klesaním vzhľadom na os x, čiže s miernym stúpaním resp. klesaním vzhľadom na os y. Ak úsečka patrí do oblasti E_y má dominantný smer y.

Hraničné priamky $y = x$ a $y = -x$ môžeme zaradiť do ktorejkoľvek z týchto oblastí, dohodnime sa, že ich zaradíme napr. do oblasti E_x.

Súradnicové osi x a y rozdelia rovinu na štyri kvadranty (Obr. 11a). Ak k nim pridáme ešte priamky $y = x$ a $y = -x$, dostaneme rozdelenie roviny na osem oktantov (Obr. 11b). Úsečke AB vieme podľa hodnoty smernice m priradiť príslušnosť do kvadrantu (Obr. 11a) alebo oktantu (Obr. 11b). Medzi základné algoritmy rasterizácie úsečky patria tri algoritmy: DDA, Bresenhamov a Midpoint algoritmus.
Pri všetkých algoritmoch platí, že vstupom je začiatočný bod $A = [x_A, y_A]$ a koncový bod úsečky $B = [x_B, y_B]$ a výstupom množina bodov rastra, ktoré aproximujú danú úsečku AB.



(a) Rozdelenie roviny do 4 kvadrantov

(b) Rozdelenie roviny do 8 oktantov

Obr. 11: Rozdelenie roviny [6]

3.1.1 DDA algoritmus (Digital Differential Analyzer Algorithm)

História

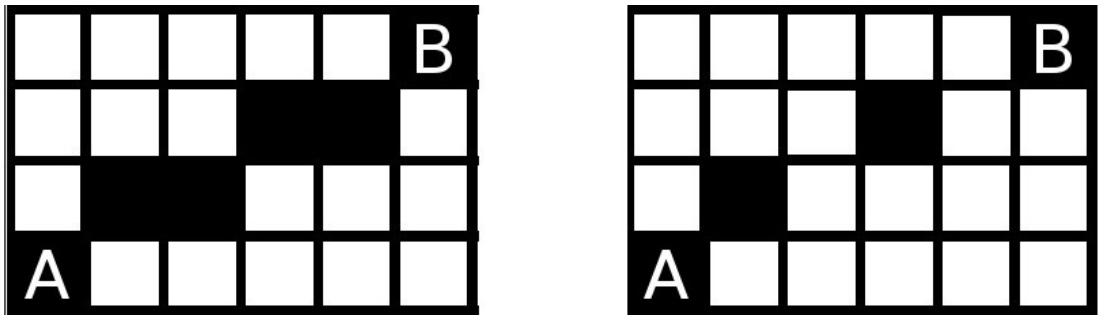
Je to jeden z prvých algoritmov používaných v počítačovej grafike.

Princíp algoritmu

DDA algoritmus je rýchlejšia metóda pre výpočet polôh pixlov ako použiť smernicovú rovnicu priamky (3.1).

V tomto iteratívnom krokovom algoritme musíme pracovať v reálnej aritmetike a robiť zaokrúhlenia. Nepoužívame pri ňom násobenie, ale výpočet súradníc pixlov pozdĺž priamky v reálnej aritmetike. Chyba zaokrúhlenia sa napriek tomu považuje za nepodstatnú.

DDA (Digital Differential Analyzer) je prírastkový algoritmus. Vychádza z rovnice $dy = mdx$, ktorú dostaneme ako formálny prepis smernicovej rovnice priamky. Úsečka je krokovaná jednotkovým krokom v smere jednej z osí x alebo y a druhá súradnica sa vyjadrí zo smernice priamky $m = \frac{dy}{dx}$. Os, v ktorej smere prebieha vzorkovanie, sa nazýva riadiaca/hlavná os a druhá je vedlajšia os. Aby sme sa pri tomto postupe vyhli výskytu medzier na zobrazených úsečkách, nemôže zobrazovaná priamka prudko stúpať alebo klesať od riadiacej osi (Obr. 12a a Obr. 12b). Ak tento prípad nastane, vymeníme úlohu súradnicových osí, t.j. jednotkové krokovanie budeme realizovať v smere druhej súradnicovej osi.



(a) Vzorkovanie v smere osi x

(b) Vzorkovanie v smere osi y

Obr. 12: Úsečka z bodu A do bodu B vykreslená pomocou dvoch vzorkovaní

Nachádzame sa v bode úsečky (x_i, y_i) a potrebujeme určiť nasledujúci bod (x_{i+1}, y_{i+1}), ktorý vieme vyjadriť ako

$$x_{i+1} = x_i + incr_x,$$
$$y_{i+1} = y_i + incr_y,$$

kde $incr_x$ a $incr_y$ je prírastok (z ang. increment) v smere x a y.

Uvažujme úsečku AB. Prípady, ktoré môžu nastať:

- Ak $|m| < 1 \Rightarrow| dy |<| dx |$ a $x_A < x_B$ (bod A sa nachádza vľavo od bodu B), tak úsečka má dominantný smer x. Vzorkovanie bude teda prebiehať v smere osi x o konštantnú hodnotu +1. Hodnotu nasledujúcej y-ovej súradnice určíme z rovnice priamky (3.1) ako $y_{i+1} = mx_{i+1} + b = m(x_i + 1) + b = y_i + m]$. Teda platí

$$x_{i+1} = x_i + 1,$$
$$y_{i+1} = y_i + m.$$

- Ak $|m| < 1 \Rightarrow| dy |<| dx |$ a $x_A > x_B$ (bod A sa nachádza vpravo od bodu B), môžeme body A a B navzájom vymeniť alebo ekvivalentne postupovať tak, že položíme hodnotu $incr_x = -1$. Potom $y_{i+1} = my_{i+1} + b = m(x_i - 1) + b = y_i - m$. Teda

$$x_{i+1} = x_i + 1,$$
$$y_{i+1} = y_i - m.$$

- Ak $|m| > 1 \Rightarrow| dy |>| dx |$ a $y_A < y_B$ (bod A sa nachádza nižšie ako bod B), tak úsečka má dominantný smer y. Vzorkovanie bude prebiehať v smere osi y o konštantnú hodnotu +1. Hodnotu nasledujúcej x-ovej súradnice určíme z rovnice priamky (3.1) ako $x_{i+1} = (y_{i+1} - b)\frac{1}{m} = (y_i + 1 - b)\frac{1}{m} = (mx_i + b + 1 - b)\frac{1}{m} = x_i + \frac{1}{m}$. Teda platí

$$x_{i+1} = x_i + \frac{1}{m},$$
$$y_{i+1} = y_i + 1.$$

- Ak $|m| > 1 \Rightarrow| dy |>| dx |$ a $x_A > x_B$ (bod A sa nachádza vyššie ako bod B), môžeme body A a B navzájom vymeniť alebo ekvivalentne postupovať tak, že položíme hodnotu $incr_y = -1$. Potom $x_{i+1} = (y_{i+1} - b)\frac{1}{m} = (y_i - 1 - b)\frac{1}{m} = (mx_i + b - 1 - b)\frac{1}{m} = x_i - \frac{1}{m}$. Teda

$$x_{i+1} = x_i - \frac{1}{m},$$
$$y_{i+1} = y_i - 1.$$

Je zrejmé, že počet vykreslených bodov v rastri pozdĺž úsečky sa rovná maximálnej hodnote rozdielu x/y-ovej súradnice krajných bodov, teda $n = max\{| dy |, | dx |\}$. Môžeme povedať, že pre $incr_x$ a $incr_y$ platí:

- $incr_x = \frac{dx}{n}$
- $incr_y = \frac{dy}{n}$

Postup

1. Vlož dva krajné body (x_A, y_A) a (x_B, y_B) a zober ľavý (s menšou x-ovou súradnicou) ako bod (x_0, y_0).
2. Nahraj (x_0, y_0) do frame buffera, teda zobraz bod (x_0, y_0) do rastra.
3. Vypočítaj konštanty $dx = x_B - x_A$, $dy = y_B - y_A$, $m = \frac{dy}{dx}$, $n = max\{\mid dy \mid, \mid dx \mid\}$ a hodnoty $incr_x = \frac{dx}{n}$, $incr_y = \frac{dy}{n}$.
4. V každej pozícii x_i počnúc $i = 1$ až po $i = n$ pozdĺž úsečky vykonaj:
   (a) $(x_{i+1}, y_{i+1}) = (x_i + incr_x, y_i + incr_y)$.
   (b) Zaokrúhli (x_{i+1}, y_{i+1}) na celé čísla.
   (c) Vykresli bod (x_{i+1}, y_{i+1}).

Príklad

Na ilustráciu algoritmu si ukážeme rasterizáciu úsečky s krajnými bodmi $(x_A, y_A) = (0, 6)$ a $(x_B, y_B) = (4, 0)$.

1. Označíme $(x_0, y_0) = (x_A, y_A) = (0, 6)$
2. Zobrazíme tento bod do rastra.
3. $dx = 4 - 0 = 4$,
   $dy = 0 - 6 = -6$
   $m = \frac{-3}{2}$
   $n = 6$
   $incr_x = \frac{2}{3}$
   $incr_y = -1$
4. $i = 1 :$ $(x_1, y_1) = (x_0 + incr_x, y_0 + incr_y) = (\frac{2}{3}, 5)$. Zaokrúhlenie $(x_1, y_1) = (1, 5)$. Vykresli bod (x_1, y_1).

   $i = 2 :$ $(x_2, y_2) = (x_1 + incr_x, y_1 + incr_y) = (\frac{4}{3}, 4)$. Zaokrúhlenie $(x_2, y_2) = (1, 4)$. Vykresli bod (x_2, y_2).

   $i = 3 :$ $(x_3, y_3) = (x_2 + incr_x, y_2 + incr_y) = (\frac{6}{3}, 3)$. Zaokrúhlenie $(x_3, y_3) = (2, 3)$. Vykresli bod (x_3, y_3).

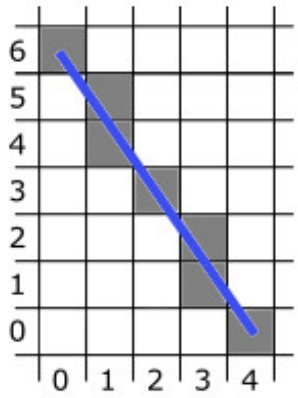
   $i = 4 :$ $(x_4, y_4) = (x_3 + incr_x, y_3 + incr_y) = (\frac{8}{3}, 2)$. Zaokrúhlenie $(x_4, y_4) = (3, 2)$. Vykresli bod (x_4, y_4).

   $i = 5 :$ $(x_5, y_5) = (x_4 + incr_x, y_4 + incr_y) = (\frac{10}{3}, 1)$. Zaokrúhlenie $(x_5, y_5) = (3, 1)$. Vykresli bod (x_5, y_5).

   $i = 6 :$ $(x_6, y_6) = (x_5 + incr_x, y_5 + incr_y) = (\frac{12}{3}, 0)$. Zaokrúhlenie $(x_6, y_6) = (4, 0)$. Vykresli bod (x_6, y_6).Hodnota $i = n$ a dostali sme sa do koncového bodu (x_B, y_B). Výsledok môžeme vidieť na Obr. 13.

Pseudokód

Implementácia DDA algoritmu je zhrnutá v nasledujúcom pseudokóde. Na vstupe sú krajné body úsečky. Parametre dx a dy predstavujú horizontálnu a vertikálnu vzdialenosť dvoch krajných bodov úsečky. Väčšia z týchto hodnôt určuje hodnotu parametru `steps`.



Obr. 13: Výsledok

Ak absolútna hodnota dx je väčšia ako absolútna hodnota dy a x_A je menšie ako x_B, hodnoty prírastku `xIncrement` a `yIncrement`sú 1 a m. V opačnom prípade, ak $x_A > x_B$, tak hodnoty prírastkov `xIncrement` a `yIncrement` sú -1 a $-m$. Funkcia `ROUND(x)` predstavuje zaokrúhlenie hodnoty x na celé čísla.

```
lineDDA (int xA, int yA, int xB, int yB)
{

  int dx = xB − xA, dy = yB − yA, steps,k;
  float xIncrement, yIncrement, x = xA, y = yA;

  if(abs(dx) > abs(dy)) steps = abs(dx);
  else steps = abs(dy);
  xIncrement = dx/(float) steps;
  yIncrement = dy/(float) steps;

  vykresliPixel(ROUND(x), ROUND(y));
  for(k = 0; k < steps; k++){
    x += xIncrement;
    y += yIncrement;
  vykresliPixel(ROUND(x), ROUND(y));
  }
}
```

Literatúra

Algoritmus DDA je spracovaný v anglickej literatúre [3] od strany 87. S jeho odvodením sa stretneme aj v [1] na strane 55 a na strane 81 v [2].

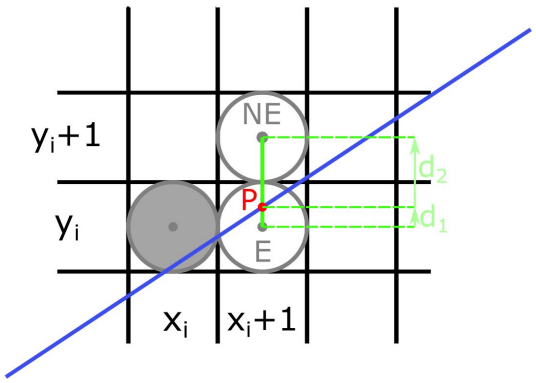
3.1.2 Bresenhamov algoritmus rasterizácie úsečky (Bresenham's Line Algorithm)

Princíp algoritmu

Tento algoritmus vykresľuje body rastra, ktoré ležia najbližšie ku geometrickému obrazu úsečky v zvislom/vodorovnom smere. Využíva sa výhradne celočíselná aritmetika.

Ilustrujeme postup pre časť úsečky AB (bod A leží naľavo od bodu B), ak riadiacou osou je x-ová súradnicová os a pre smernicu tejto úsečky AB platí $0 < m < 1$. Prírastok v smere osi x sa konštantne zväčšuje o hodnotu +1 (Obr. 14).

Predpokladajme, že sme v pozícii bodu úsečky $X = (x_i, y_i)$. Vzhľadom na jednotkové krokovanie v smere osi x, kandidátmi na ďalší bod sú pixely $E = (x_i+1, y_i)$ alebo $NE = (x_i+1, y_i+1)$. Je to ten z nich, ktorý je bližšie ku geometrickému priesečníku P danej úsečky so spojnicou týchto pixlov. Označíme $d_1 = y - y_i$ a $d_2 = (y_i + 1) - y$, kde y je y-ová súradnica bodu P na priamke a teda platí $y = m(x_i+1)+b$ (Obr. 14). Z toho $d_1 = m(x_i+1)+b-y_i$ a $d_2 = (y_i+1)-m(x_i+1)+b$. Zvolíme si premennú $\Delta d = d_1 - d_2 = 2m(x_i + 1) - 2y_i + 2b - 1$, podľa ktorej vieme určiť, ktorý



Obr. 14: Časť úsečky AB

z dvoch možných pixlov je bližšie k úsečke AB. Je zrejmé, že:

- Ak $\Delta d < 0 \Leftrightarrow d_1 < d_2$, tak bližšie k úsečke je pixel $E = (x_i + 1, y_i)$.
- Ak $\Delta d > 0 \Leftrightarrow d_1 > d_2$, tak bližšie je pixel $NE = (x_i + 1, y_i + 1)$.
- V prípade, ak $\Delta d = 0 \Leftrightarrow d_1 = d_2$, je jedno, ktorý z týchto pixlov sa vykreslí, obyčajne ten s väčšou y-ovou súradnicou.

Z uvedeného postupu vyplýva, že pre určenie, ktorý pixel sa vykreslí, nie je dôležité vypočítať hodnotu premennej Δd, ale iba určiť jej znamienko. Preto pri rozhodovaní môžeme túto hodnotu nahradiť ľubovoľným jej kladným násobkom, konkrétne parametrom $p_i = dx\Delta d$. Týmto krokom prenesieme výpočet do celočíselnej aritmetiky, pretože tým eliminujeme jediný neceločíselný člen, smernicu $m = \frac{dy}{dx}$.

Pre zjednodušenie výpočtov je vhodné si vyjadriť parameter p_i rekurentne:

$p_i = 2dyx_i - 2dxy_i + C$, kde $C = 2dy + dx(2b - 1)$ je konštanta nezávislá od i.

$p_{i+1} = 2dyx_{i+1} - 2dxy_{i+1} + C$

Z toho: $p_{i+1} - p_i = -2dx(y_{i+1} - y_i) + 2dy \Rightarrow p_{i+1} = p_i - 2dx(y_{i+1} - y_i) + 2dy$. Aby sme mohli tento predpis využiť, potrebujeme ešte hodnotu $p_0 = 2dy - dx$, kde sme pri výpočte využili, že y_0 je bod na priamke a platí $y_0 = mx_0 + b$. Teraz môžeme iteračným spôsobom počítať hodnoty každého nasledujúceho parametra p z jeho predchádzajúcej hodnoty. Teda:

- Ak $p_i < 0 \Leftrightarrow d_1 < d_2$, tak vykreslíme bod na pozícií $E = (x_i + 1, y_i)$, t.j. $y_{i+1} = y_i$ a teda $p_{i+1} = p_i + 2dy$
- Ak $p_i \geq 0 \Leftrightarrow d_1 > d_2$, tak vykreslíme bod na pozícií $NE = (x_i + 1, y_i + 1)$, t.j. $y_{i+1} = y_i + 1$ a teda $p_{i+1} = p_i + 2(dy - dx)$

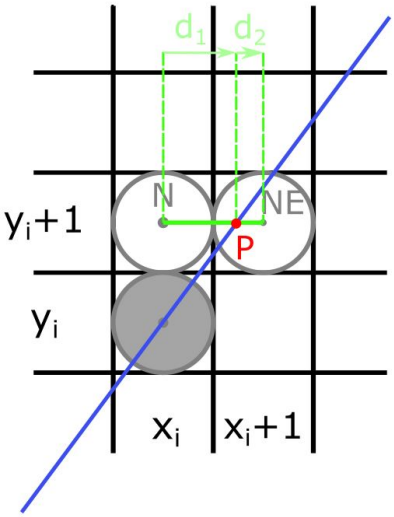
Všeobecne:

Vo vyššie opísanom postupe sme predpokladali, že smernica $0 < m < 1$ a teda $x_{i+1} = x_i + 1$. Ak pre smernicu m platí $-1 < m < 0$, tak sa jednotkový krok na x-ovej osi zmení na -1, teda $x_{i+1} = x_i - 1$.

Princíp algoritmu je rovnaký aj pre úsečku AB so smernicou $m > 1$.

Vtedy sa bod A nachádza nižšie ako bod B, t.j. $y_A < y_B$ (Obr. 15). Úsečka je priklonená k osi y. Preto zvolíme vzorkovanie v smere osi y o hodnotu +1. Ak sa nachádzame v pixli $X = (x_i, y_i)$, tak kandidáti na vykreslenie sú na pozícii $N = (x_i, y_i + 1)$ a $NE = (x_i + 1, y_i + 1)$. Označíme P geometrický priesečník danej úsečky so spojnicou týchto pixlov. Hodnoty d_1 a d_2 reprezentujú vzdialenosť medzi priesečníkom P a stredom pixlov N a NE (Obr. 15). Teda $d_1 = x - x_i$ a $d_2 = x_{i+1} - x$, kde x je x-ová súradnica bodu P a z rovnice priamky (3.1) pre ňu dostaneme $x = \frac{1}{m}(y_{i+1} - b)$.

Parameter p_i je možné určiť ako $p_i = dy(d_1 - d_2) = 2dxy_i - 2dyx_i + C$, kde $C = 2dx(1-b) - dy$ je konštanta nezávislá od i. Potom $p_{i+1} = p_i + 2dx - 2dy(x_{i+1} - x_i)$ a $p_0 = 2dx - dy$



Obr. 15: Časť úsečky AB

Teda pre vzorkovanie v smere osi y platí:

- Ak $p_i < 0 \Leftrightarrow d_1 < d_2$, tak vykreslíme bod na pozícií $N = (x_i, y_i + 1)$, t.j. $x_{i+1} = x_i$ a $p_{i+1} = p_i + 2dx$
- Ak $p_i \geq 0 \Leftrightarrow d_1 > d_2$, tak vykreslíme bod na pozícií $NE = (x_{i+1}, y_{i+1})$, t.j. $x_{i+1} = x_i + 1$ a $p_{i+1} = p_i + 2(dx - dy)$

Postup pre $0 < m < 1$

1. Vlož dva krajné body (x_A, y_A) a (x_B, y_B) a zober ľavý (s menšou x-ovou súradnicou) ako bod (x_0, y_0).
2. Nahraj (x_0, y_0) do frame buffera, teda zobraz bod (x_0, y_0) do rastra.
3. Vypočítaj konštanty dx, dy, $2dy$, $2(dy - dx)$ a urči hodnotu parametra p_0.
4. V každej z pozícii x_i počnúc $i = 0$ pozdĺž úsečky vykonaj test:
   (a) Ak $p_i < 0$: nasledujúci bod je $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$ a $p_{i+1} = p_i + 2dy$.
   (b) Inak je nasledujúci bod $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i + 1)$ a $p_{i+1} = p_i + 2dy - 2dx$.
   (c) Vykresli bod (x_{i+1}, y_{i+1}).
5. Opakuj krok 4. dx-krát.

Príklad

Na ilustráciu algoritmu si ukážeme rasterizáciu úsečky s krajnými bodmi $(x_A, y_A) = (0, 0)$ a $(x_B, y_B) = (7, 4)$.

1. Označíme $(x_0, y_0) = (x_A, y_A) = (0, 0)$
2. Zobrazíme tento bod do rastra.
3. $dx = 7 - 0 = 7$,
   $dy = 4 - 0 = 4$
   (smernica $m = \frac{4}{7} < 1$ a teda sa priamka leží v I. oktante a riadiaca os je x)
   $2dy = 2.4 = 8$
   $2(dy - dx) = 2.(4 - 7) = -6$
   $p_0 = 2.4 - 7 = 1$
4. $i = 0 :\ p_0 > 0 \Rightarrow (x_1, y_1) = (x_0 + 1, y_0 + 1) = (1, 1)$ a $p_1 = p_0 + 2(dy - dx) = -5$. Vykresli bod (x_1, y_1).

   $i = 1 :\ p_1 < 0 \Rightarrow (x_2, y_2) = (x_1 + 1, y_1) = (2, 1)$ a $p_2 = p_1 + 2dy = 3$. Vykresli bod (x_2, y_2).

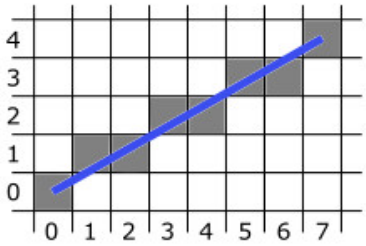
   $i = 2 :\ p_2 > 0 \Rightarrow (x_3, y_3) = (x_2 + 1, y_2 + 1) = (3, 2)$ a $p_3 = p_2 + 2(dy - dx) = -3$. Vykresli bod (x_3, y_3).

   $i = 3 :\ p_3 < 0 \Rightarrow (x_4, y_4) = (x_3 + 1, y_3) = (4, 2)$ a $p_4 = p_3 + 2dy = 5$. Vykresli bod (x_4, y_4).

   $i = 4 :\ p_4 > 0 \Rightarrow (x_5, y_5) = (x_4 + 1, y_4 + 1) = (5, 3)$ a $p_5 = p_4 + 2(dy - dx) = -1$. Vykresli bod (x_5, y_5).

   $i = 5 :\ p_5 < 0 \Rightarrow (x_6, y_6) = (x_5 + 1, y_5) = (6, 3)$ a $p_6 = p_5 + 2dy = 7$. Vykresli bod (x_6, y_6).

   $i = 6 :\ p_6 > 0 \Rightarrow (x_7, y_7) = (x_6 + 1, y_6 + 1) = (7, 4)$. Príkaz sme vykonali dx-krát a dostali sme sa do krajného bodu (x_B, y_B). Vykresli bod (x_7, y_7). Výsledok je zobrazený na Obr. 16.



Obr. 16: Výsledok rasterizácie úsečky AB

Pseudokód

Implementácia Bresenhamovho algoritmu pre smernicu úsečky $0 < m < 1$ je zhrnutá v nasledujúcom pseudokóde. Na vstupe sú dva krajné body úsečky $A = (x_A, y_A)$ a $B = (x_B, y_B)$.

```
lineBres (int xA, int yA, int xB, int yB)
{

  int dx = abs(xB − xA), dy = abs(yB − yA);
  int p=2 * dy - dx;
  int x, y, xKoncovy;

  /* Rozhodnutie, ktorý krajný bod sa použije ako začiatočný a koncový
  if (xA > xB) {
    x = xB;
    y = yB;
    xKoncovy = xA;
  }
  else {
    x = xA;
    y = yA;
    xKoncovy = xB;
  }
  vykresliPixel(x,y);

  while(x < xKoncovy) {
    x++;
    if(p < 0) p += 2 * dy;
    else {
      y++;
      p += 2 * (dy - dx);
    }
  vykresliPixel(x, y);
  }
}
```

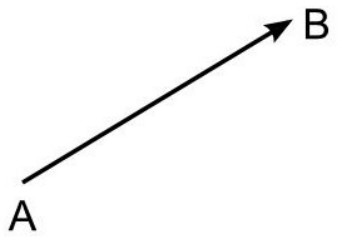
Literatúra

Bresenhamov algoritmus je podrobne spracovaný v literatúre [3] od strany 88. V slovenskej/českej literatúre ho nájdeme odvodený v [1] na strane 58 a taktiež v [2] od strany 82.

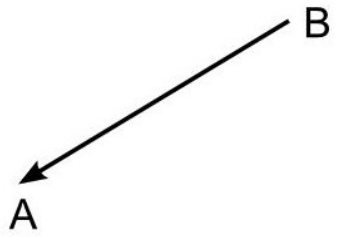
3.1.3 Bresenhamov stredový algoritmus rasterizácie úsečky (Bresenham's Midpoint Line Algorithm)

Princíp algoritmu

Orientovaná úsečka je nenulová úsečka, ktorej krajné body sú usporiadané. Hovoríme o začiatočnom a koncovom bode orientovanej úsečky (Obr. 17. a Obr. 18). Hovoríme tiež, že orientovaná úsečka zo svojho začiatočného bodu vychádza a do koncového bodu vchádza. Orientovaná úsečka orientuje prirodzeným spôsobom priamku, na ktorej leží a naopak. Orientáciu úsečky môžeme intuitívne vnímať ako jeden z dvoch smerov pohybu po nej [8].

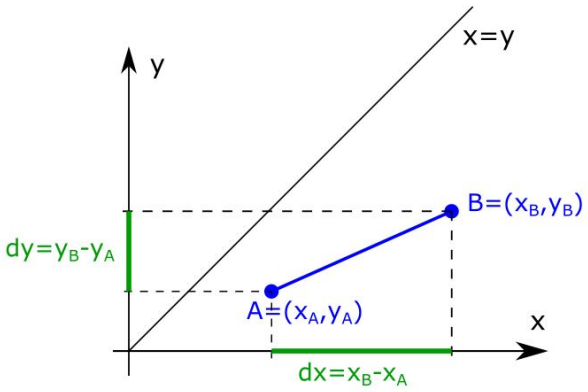


Obr. 17: Orientovaná úsečka AB



Obr. 18: Orientovaná úsečka BA

Uvažujeme orientovanú úsečku AB, kde $x_A < x_B$ (bod A je vľavo od bodu B) a smernica m úsečky AB je $0 \leq m \leq 1$ (Obr. 19.).



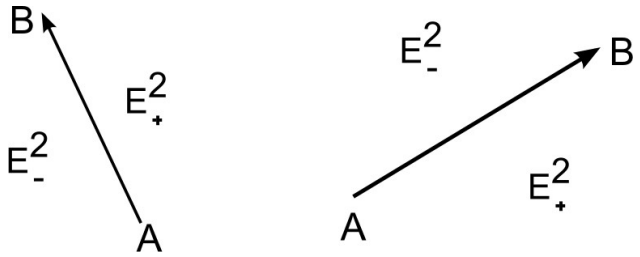
Obr. 19: Úsečka AB

Rovnicu priamky potom možno napísať v implicitnom tvare $f(x, y) = ax + by + c = 0$, kde $a = dy, b = -dx, c = dx.y_B - dy.x_B$. Všimnime si, že všetky koeficienty sú celočíselné. Budeme používať ekvivalentnú reprezentáciu priamky $f(x, y) = 2ax + 2by + 2c = 0$. Kvôli vhodne zvolenej reprezentácii sa presunieme do celočíselnej aritmetiky.

Táto priamka rozdeľuje rovinu E^2 na dve polroviny (Obr. 20.):

- $E^2_- = \{(x, y); f(x, y) < 0\}$. Je to otvorená polrovina, ktorú nazývame ľavá polrovina.
- $E^2_+ = \{(x, y); f(x, y) \geq 0\}$. Je uzavretá polrovina a nazývame ju pravá polrovina.

Z toho môžeme povedať, že:



Obr. 20: Polroviny vzhľadom na orientovanú úsečku AB

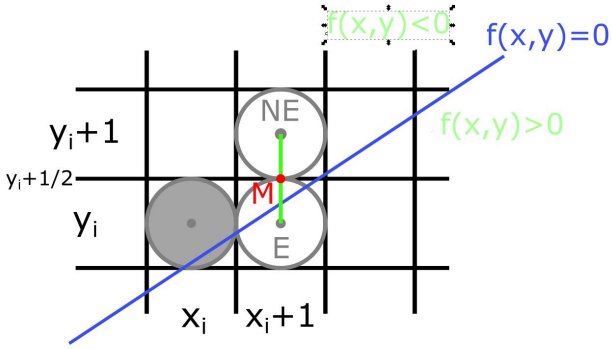
- bod $X = (x, y)$ leží vpravo od priamky $AB \Leftrightarrow f(x, y) > 0$
- bod $X = (x, y)$ leží vľavo od priamky $AB \Leftrightarrow f(x, y) < 0$
- bod $X = (x, y)$ leží na priamke $AB \Leftrightarrow f(x, y) = 0$.

Predpoklad, že pre smernicu vykresľovanej úsečky platí $0 \leq m \leq 1$, môžeme odstrániť. Ak $|m| > 1$, tak zámena $x \leftrightarrow y$ vedie k úsečke s prevrátenou hodnotou smernice. Ak $m < 0$, algoritmus vieme modifikovať zámenou prírastok $\leftrightarrow$ úbytok alebo stúpanie $\leftrightarrow$ klesanie. Výmenou vstupných bodov $A \leftrightarrow B$ je možné vždy zabezpečiť kreslenie zľava doprava.

Opäť uvažujme, že pre smernicu platí $0 < m < 1$. Pri rasterizácii sme dosiahli pixel (x_i, y_i). Potrebujeme rozhodnúť, ktorý z nasledujúcich pixlov $E = (x_i+1, y_i)$ alebo $NE = (x_i+1, y_i+1)$ vykreslíme.

Použijeme bod $M = stred(E, NE) = (x_i + 1, y_i + \frac{1}{2})$ a jeho polohu vzhľadom na priamku $f(x, y)$ (Obr. 21.). Označíme $D = f(M) = 2a(x_i + 1) + 2b(y_i + \frac{1}{2}) + 2c = 2ax_i + 2by_i + (2a + b + 2c) \in \mathbb{N}$. Zrejme platí:

- Ak $D > 0$, tak bod M leží v E^2_+ a z uvažovaných pixlov je bližšie NE, ktorý sa vysvieti.
- Ak $D < 0$, tak bod M leží v E^2_- a bližšie je bod E, ten sa rozsvieti.
- Ak nastáva $D = 0$ vysvietime hociktorý z bodov NE, E, zvyčajne NE.



Obr. 21: Výber NE resp. E

Je výhodné, že D je celočíselné, ale jeho výpočet si vyžaduje dve násobenia a dve sčítania, ak nemeniaca sa zložka v zátvorke je prepočítaná. Jedným z veľmi šikovných trikov je však, že hodnota D sa počíta prírastkovo.

Predpokladajme, že poznáme aktuálnu hodnotu D a chceme vypočítať jeho nasledujúcu hodnotu:

1. Ak sme ako nový vysvietený pixel vybrali $E = (x_i + 1, y_i)$, tak v nasledujúcom kroku k nemu prislúcha nový stred $M_{new} = ((x_i + 1) + 1, y_i + \frac{1}{2})$ a nové $D_{new} = f(M_{new}) = 2a(x_i + 2) + 2b(y_i + \frac{1}{2}) + 2c = 2a(x_i + 1) + 2by_i + (2a + b + 2c) = D + 2a = D + 2dy$

2. Ak sme však ako nový vysvietený pixel vybrali bod $NE = (x_i + 1, y_i + 1)$, tak k nemu prislúcha nový stred $M_{new} = ((x_i+1)+1, (y_i+1)+\frac{1}{2})$ a $D_{new} = f(M_{new}) = 2a(x_i+2)+2b(y_i+1+\frac{1}{2})+2c = 2a(x_i+1)+2b(y_i+1)+(2a+b+2c) = D+2a+2b = D+2(dy-dx)$.

Platí teda:

- Ak $D < 0$, tak $D_{new} = D + 2dy$ a vysvietime bod E
- Ak $D \geq 0$, tak $D_{new} = D + 2(dy - dx)$ a vysvietime bod NE

Z toho vyplýva, že parameter D je ekvivalentný parametru p z predchádzajúceho algoritmu a možno ho považovať za rozhodovací parameter Bresenhamovho algoritmu rasterizácie úsečky. Teda algoritmus možno dokončiť ako Bresenhamov line algoritmus.

V literatúre parameter D autori často uprednostňujú pred parametrom p z predchádzajúceho algoritmu, lebo princíp jeho konštrukcie je možno použiť aj v ďalších algoritmoch ako napr. v Bresenhamovom algoritme rasterizácie kružnice.

Postup pre $|m| < 1$

Kroky algoritmu vieme zhrnúť rovnakým spôsobom ako v predošlom algoritmu, keďže parameter D je ekvivalentný parametru p.

1. Vlož dva krajné body (x_A, y_A) a (x_B, y_B) a zober ľavý (s menšou x-ovou súradnicou) ako bod (x_0, y_0).
2. Nahraj (x_0, y_0) do frame buffera, teda zobraz bod (x_0, y_0) do rastra.
3. Vypočítaj konštanty dx, dy, $2dy$, $2(dy - dx)$ a urči počiatočnú hodnotu parametra $D = f(M) = 2ax_i + 2by_i + (2a + b + 2c)$, kde $M = (x_i + 1, y_i + \frac{1}{2})$.
4. V každej pozícií x_i začínajúc s $i = 0$ pozdĺž úsečky vykonaj test:
   (a) Ak $D < 0$: nasledujúci bod je $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$ a $D_{new} = D + 2dy$.
   (b) Inak je nasledujúci bod $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i + 1)$ a $D_{new} = D + 2dy - 2dx$.
   (c) Vykresli bod (x_{i+1}, y_{i+1}) a $D = D_{new}$.
5. Opakuj krok 4. dx-krát.

Príklad

Ako sme uviedli parameter D je ekvivalentný parametru p z predchádzajúceho algoritmu, postup výpočtu príkladu je rovnaký ako v predchádzajúcom príklade.

Pseudokód

Keďže parameter D je ekvivalentný parametru p z predchádzajúceho algoritmu, implementácia Bresenhamovho stredového algoritmu pre smernicu úsečky $0 < m < 1$ je zhrnutá v predchádzajúcej kapitole.

Literatúra

Bresenhamov stredový algoritmus sa nenachádza v [3] ani v [7]. Zo slovenskej/českej literatúry sa taktiež nenachádza v [1] ani [2].

3.2 Algoritmy na rasterizáciu kružnice

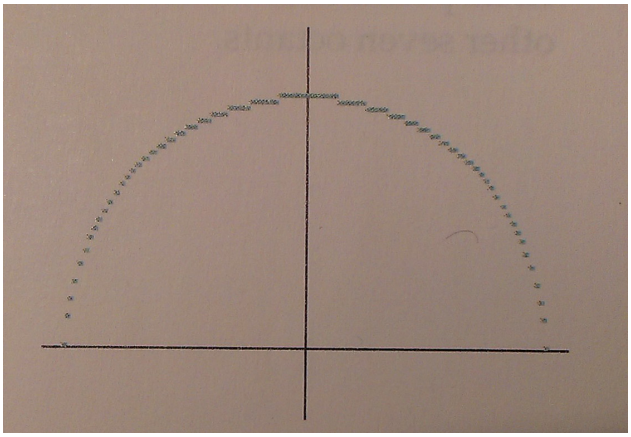
Kružnice sú často používané komponenty v obrázkoch a grafoch, a preto sú procedúry na ich generovanie zahrnuté v grafických knižniciach. Kružnica v E^2 je definovaná ako množina všetkých bodov v rovine, ktoré sú vo vzdialenosti r od stredu kružnice (x_C, y_C). Číslo r nazývame polomerom kružnice. Implicitnú rovnicu kružnice v karteziánskej súradnicovej sústave môžeme zapísať ako množinu bodov (x, y), ktoré vyhovujú rovnici

$$(x - x_C)^2 + (y - y_C)^2 - r^2 = 0, \tag{3.5}$$

kde (x_C, y_C) je stred kružnice a r je polomer danej kružnice. Túto rovnicu je možné upraviť na explicitné vyjadrenie

$$y = y_C \pm \sqrt{r^2 - (x_C - x)^2}, \tag{3.6}$$

v ktorom môžeme využiť jednotkový krok na osi x, a $x \in$
$\langle x_C - r, x_C + r \rangle$ Táto metóda však nie je najvýhodnejšia pre generovanie kružníc. Problémom je veľa výpočtov v každom kroku a pri vykresľovaní sa objavujú medzery medzi jednotlivými pixelmi (Obr. 22). Môžeme zameniť úlohy x a y a postupovať jednotkovým krokom po y-ovej



Obr. 22: Kružnica vykreslená v kladnej polrovine pomocou rovnice (2.6) [3]

súradnici, čím však zvýšime počet potrebných výpočtov a predĺžime čas behu algoritmu. Určitú eliminácia nerovnomerne zobrazených pixlov (na Obr. 22) môžeme zabezpečiť pomocou parametrizovaného vyjadrenia kružnice.

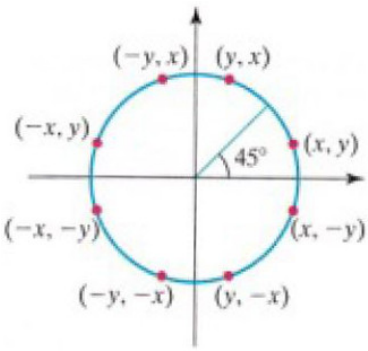
Parametrizovaná rovnica kružnice:

$$\begin{aligned} x(t) &= x_C + r.\cos(t), \\ y(t) &= y_C + r.\sin(t), \end{aligned} \tag{3.7}$$

kde parameter $t \in \langle 0, 2\pi)$, r je polomer kružnice a (x_C, y_C) je stred danej kružnice.

Ak generujeme kružnicu pomocou týchto rovníc pri rovnomernom uhlovom kroku, kružnica je zobrazená s rovnomerne vzdialenými bodmi pozdĺž kružnice. Väčšie medzery medzi bodmi kružnice môžu byť spojené úsečkami, aby aproximovali kružnicový tvar. Pre spojité zobrazenie kružnice môžeme použiť krokovanie o veľkosti $\frac{1}{r}$. Získané body následne budú vzdialené približne jeden pixel od seba.

Generovanie bodov kružnice môže byť zjednodušené využitím osových a stredových súmerností kružnice (Obr. 23). Segmenty kružnice sú v jednotlivých oktantoch 1-8 rovnaké. Preto stačí



Obr. 23: Symetrie bodu (x, y) na kružnici[3]

generovať iba segment kružnice v jednom oktante a ten pomocou súmerností zobraziť do ostatných.

Generovanie kružnice pomocou vyššie opísaných postupov vyžaduje značné množstvo výpočtov a teda dlhší čas behu algoritmu. V nasledujúcich podkapitolách opíšeme algoritmy na efektívne vykreslenie kružnice: minimalizácia výpočtov a s využitím len celočíselnej aritmetiky.

Pri všetkých algoritmoch máme na vstupe polomer r a stred (x_C, y_C) kružnice K. Na výstupe dostaneme množinu bodov rastra, ktoré aproximujú danú kružnicu K.

3.2.1 Bresenhamov kružnicový algoritmus (Bresenham's Circle Algorithm)

Princíp algoritmu

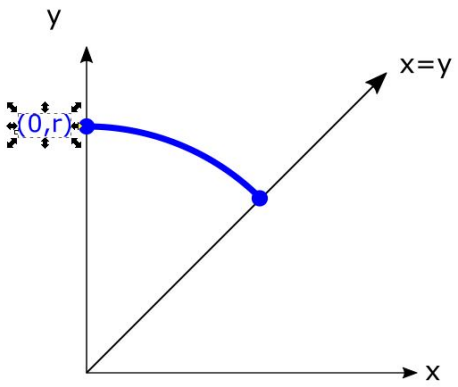
Tento algoritmus predstavuje zovšeobecnenie Bresenhamovho algoritmu rasterizácie úsečky na kružnicu. Rozklad zrýchlime tým, že budeme generovať len segment kružnice v jednom oktante. Ostatné časti kružnice dostaneme súmernosťou podľa súradnicových osí x, y a priamok $x = y$ a $y = -x$.

Výber pixlov je založený na rovnakom princípe ako pri úsečke, kde z dvoch možných kandidátov na vysvietenie vyberáme toho, ktorého stred je bližšie ku danej kružnici. Využívame parameter $p_i = d_1 - d_2$, ktorý sa vyčísľuje rekurentne, kde hodnoty d_1 a d_2 sú štvorcami rozdielov y-ových súradníc.

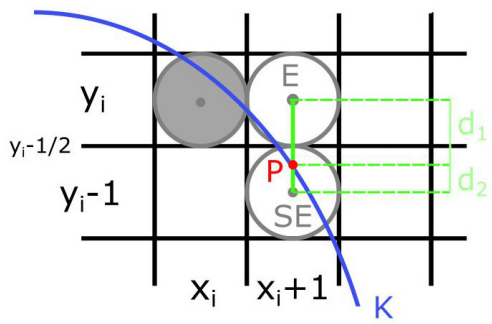
Pre ilustráciu algoritmu volíme kružnicu so stredom v začiatku súradnicovej sústavy a zobrazíme jej segment, ktorý sa začína v bode $(0, r)$ a končí, v bode (x_i, y_i), kde $x_i \geq y_i$ (Obr. 24). V tomto prípade je jednotkový krok v smere súradnicovej osi x. Nachádzame sa v bode (x_i, y_i) a potrebujeme sa rozhodnúť, ktorý z pixlov $E = (x_i + 1, y_i)$ alebo $SE = (x_i + 1, y_i - 1)$ si vyberieme (Obr. 25).

Označíme si $d_1 = y_i^2 - y^2$ a $d_2 = y^2 - (y_i - 1)^2$, kde y je hodnota určená rovnicou $y^2 = r^2 - (x_i + 1)^2$ z (3.5). Hodnota d_1 vyjadruje vzdialenosť medzi bodmi E a $(x_i + 1, y)$ a hodnota d_2 medzi $(x_i + 1, y)$ a SE. Určíme $p_i = d_1 - d_2 = 2(x_i + 1)^2 + y_i^2 + (y_i - 1)^2 - 2r^2$. Zrejme platí

- Ak $p_i < 0 \Leftrightarrow d_1 < d_2$, tak bod E je bližšie k bodu na kružnici $(x_i + 1, y_i)$ a preto vysvietime pixel E.
- Ak $p_i \geq 0 \Leftrightarrow d_1 \geq d_2$, tak je bližšie bod SE, ktorý následne vysvietime. Pri rovnosti môžeme vybrať ľubovoľný z bodov E, SE, no zvyčajne sa vyberá vyššie položený bod.



Obr. 24: Generovaný segment



Obr. 25: Bresenhamov algoritmus generovania kružnice

Teraz vyjadríme parameter p_i rekurentne pomocou p_{i+1}
$p_{i+1} = 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2$ a rozdielu
$p_{i+1} - p_i = 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i)$.
Možno zapísať rekurentný vzťah $p_{i+1} = p_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i)$. Teda:

- Ak $p_i < 0 \Leftrightarrow d_1 < d_2$, nasledujúci bod je $(x_{i+1}, y_{i+1}) = E = (x_i+1, y_i)$ a $p_{i+1} = p_i + 4x_i + 6$
- Ak $p_i \geq 0 \Leftrightarrow d_1 \geq d_2$, nasledujúci bod je $(x_{i+1}, y_{i+1}) = SE = (x_i+1, y_i-1)$, a preto $p_{i+1} = p_i + 4x_i + 6 + 2(y_i - y_i^2)$

Potom pre začiatočnú hodnotu p_0 v bode $(x_C, y_C) = (0, r)$ dostaneme $p_0 = 3 - 2r$. Hoci pri výpočte parametra sa používa násobenie, príslušný násobok je druhou mocninou a je možná implementácia cez logické operácie. Ostatné operácie sú celočíselné sčítanie a odčítanie.

Postup

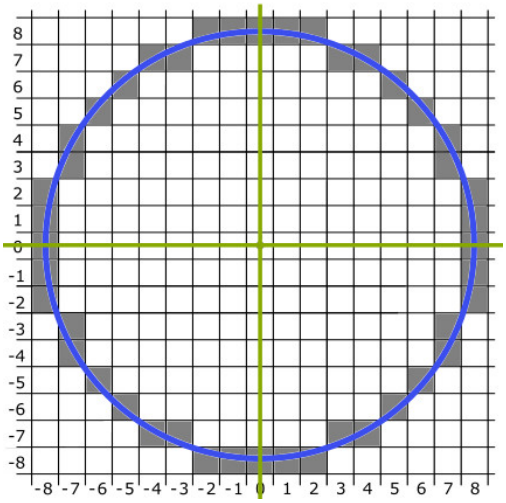
Kroky algoritmu vieme zhrnúť nasledovne:

1. Zadaj polomer r a stred kružnice (x_C, y_C) a urči prvý bod na kružnici so stredom v začiatku ako $(x_0, y_0) = (0, r)$.
2. Vypočítaj začiatočnú hodnotu rozhodovacieho parametra ako $p_0 = 3 - 2r$
3. V každej pozícii x_i, štartujúc v $i = 0$, vykonaj nasledujúci test:
   (a) Ak $p_i < 0$: nasledujúci bod pozdĺž kružnice so stredom $(0, 0)$ bude $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$ a $p_{i+1} = p_i + 4x_i + 6$.
   (b) Inak je nasledujúci bod $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)$ a $p_{i+1} = p_i + 4(x_i - y_i) + 10$.
4. Urči súmerne položené body v ostatných siedmich oktantoch.
5. Posuň každú vypočítanú pixlovú pozíciu (x, y) na kružnicu so stredom (x_C, y_C) a zobraz bod so súradnicami: $x = x + x_C$,$y = y + y_C$.
6. Opakuj kroky 3. až 5. pokiaľ $x_i \geq y_i$.

Príklad

Na ilustráciu algoritmu si ukážeme rasterizáciu kružnice s polomerom $r = 8$ a stredom kružnice $(x_C, y_C) = (1, 2)$.

1. $(x_0, y_0) = (0, r) = (0, 8)$
2. $p_0 = 3 - 2r = -13$
3. $i = 0: \; p_0 < 0 \Rightarrow (x_1, y_1) = (x_0 + 1, y_0) = (1, 8)$ a $p_1 = p_0 + 4x_0 + 6 = -7$
   $i = 1: \; p_1 < 0 \Rightarrow (x_2, y_2) = (x_1 + 1, y_1) = (2, 8)$ a $p_2 = p_1 + 4x_1 + 6 = 3$
   $i = 2: \; p_2 > 0 \Rightarrow (x_3, y_3) = (x_2 + 1, y_2 - 1) = (3, 7)$ a $p_3 = p_2 + 4(x_2 - y_2) + 10 = -11$
   $i = 3: \; p_3 < 0 \Rightarrow (x_4, y_4) = (x_3 + 1, y_3) = (4, 7)$ a $p_4 = p_3 + 4x_3 + 6 = 7$
   $i = 4: \; p_4 > 0 \Rightarrow (x_5, y_5) = (x_4 + 1, y_4 - 1) = (5, 6)$ a $p_5 = p_4 + 4(x_4 - y_4) + 10 = 5$
   $i = 5: \; p_5 > 0 \Rightarrow (x_6, y_6) = (x_5 + 1, y_5 - 1) = (6, 5)$ a $x_6 \geq y_6$ teda algoritmus končí.
4. Určíme súmerne položené body v ostatných oktantoch (Obr. 26).
5. Posunieme každú hodnotu pixla do stredu $(1, 2)$. Teda body: $(2, 10), (3, 10), (4, 9), (5, 9), (6, 8), ...$



Obr. 26: Výsledná kružnica so stredom v bode $(0, 0)$

Pseudokód

Implementácia Bresenhamovho kružnicového algoritmu je zhrnutá v nasledujúcom pseudokóde. Na vstupe je stred (x_C, y_C) a polomer kružnice r.

```
circleBres (int xC, int yC, int r)
{
  int x = 0;
  int y = r;
  int p = 3 - (2 * r);
  void vykresliBodKruznice(int, int, int, int);

  /* Vykreslí prvý bod kružnice
```

```
  vykresliBodKruznice(xC, yC, x, y);

  while(x < y) {
    x++;
    if(p < 0) p = p + (4 * x) + 6;
    else {
      y--;
      p = p + 4 * (x - y) + 10;
    }
  vykresliBodKruznice(xC, yC, x, y);
  }
}

void vykresliBodKruznice(int xC, int yC, int x, int y);
{
  vykresliPixel(xC + x, yC + y);
  vykresliPixel(xC - x, yC + y);
  vykresliPixel(xC + x, yC - y);
  vykresliPixel(xC - x, yC - y);
  vykresliPixel(xC + y, yC + x);
  vykresliPixel(xC - y, yC + x);
  vykresliPixel(xC + y, yC - x);
  vykresliPixel(xC - y, yC - x);
}
```

Literatúra

Bresenhamov kružnicový algoritmus sa nachádza podrobne spracovaný v literatúre [3]. Jeho myšlienka je načrtnutá aj v [1] na strane 62 a taktiež jeho vysvetlenie sa nachádza v [2] od strany 88.

3.2.2 Bresenhamov stredový kružnicový algoritmus (Midpoint Circle Algorithm)

Princíp algoritmu

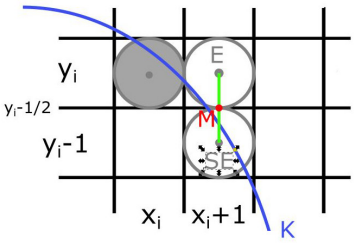
Algoritmus predstavuje úpravu Bresenhamovho midpoint line algoritmu pre kružnicu. Algoritmus vychádza z polomeru r kružnice K a jej stredu v začiatku súradnicovej sústavy, t.j. v bode $(x_C, y_C) = (0, 0)$.

Ak sa stred (x_C, y_C) nenachádza v začiatku, tak pozíciu bodu kružnice (x, y) vieme vypočítať pričítaním x_C k x-ovej súradnici a y_C k y-ovej. Vychádzame z rovnice kružnice: $f(x, y) = x^2 + y^2 - r^2 = 0$.

Platí:

- Ak $f(x, y) < 0$, tak bod (x, y) sa nachádza vnútri kružnice K. Takýto bod nazývame vnútorný bod
- Ak $f(x, y) = 0$, tak bod (x, y) sa nachádza na kružnici K
- Ak $f(x, y) > 0$, tak bod (x, y) sa nachádza mimo kružnice K. Takýto bod nazývame vonkajší bod.

Opäť sa nachádzame na segmente kružnice od bodu $(0, r)$ až po bod, pre ktorý je $x_i \geq y_i$ Nech sme v bode (x_i, y_i) a hľadáme nasledujúci bod. Rozhodujeme sa medzi dvoma kandidátmi $E = (x_i + 1, y_i)$ a $SE = (x_i + 1, y_i - 1)$ (Obr. 27).



Obr. 27: Midpoint algoritmus

Určíme $M = stred(E, SE) = (x_i + 1, y_i - \frac{1}{2})$. Budeme zisťovať, či tento bod je vnútorný alebo vonkajší. Definujeme rozhodovací parameter p_i: $p_i = f(M) = (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - r^2$ a odvodíme z neho rekurentný vzorec:
$p_{i+1} = f((x_{i+1} + 1, y_{i+1} - \frac{1}{2}) = (x_{i+1} + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - r^2$
Potom $p_{i+1} - p_i = 2x_{i+1} + (y_{i+1}^2 - y_{i+1}) - (y_i^2 - y_i) + 1$ rekurentný predpis má tvar:
$p_{i+1} = p_i + 2x_{i+1} + (y_{i+1}^2 - y_{i+1}) - (y_i^2 - y_i) + 1$

Zrejme platí:

- Ak $p_i < 0$, vyberáme ako nasledujúci pixel $(x_{i+1}, y_{i+1}) = E = (x_i + 1, y_i)$ a $p_{i+1} = p_i + 2x_i + 3$
- Ak $p_i \geq 0$, vyberáme bod $(x_{i+1}, y_{i+1}) = SE = (x_i + 1, y_i - 1)$ a teda $p_{i+1} = p_i + 2(x_i - y_i) + 5$

Potrebujeme ešte poznať hodnotu začiatočného parametra $p_0 = (x_0 + 1)^2 + (y_0 - \frac{1}{2})^2 - r^2$, ktorý pre bod $(x_0, y_0) = (0, r)$ sa rovná $p_0 = \frac{5}{4} - r$. Ak je polomer špecifikovaný ako celé číslo, môžeme hodnotu p_0 zaokrúhliť ako $p_0 = 1 - r$, lebo všetky prírastky sú celočíselné.

Na rozdiel od lineárnych algoritmov rasterizácie úsečky, sme v prípade kružnicových algoritmov nezískali rovnaké rozhodovacie parametre.

Postup algoritmu sme ilustrovali pre kružnicový oblúk, ktorý sa nachádza v II. oktante, avšak algoritmus je možné rovnakým spôsobom odvodiť aj pre ktorýkoľvek z ostatných oktantov. Podobne ako Bresenhamov line algoritmus, aj tento vypočítava pixely pozdĺž kružnice, používajúc pri tom len celočíselné sčítania a odčítania za predpokladu, že stred a polomer kružnice sú špecifikované v celočíselných obrazových súradniciach.

Postup

Kroky algoritmu vieme zhrnúť nasledovne:

1. Zadaj polomer r a stred kružnice (x_C, y_C) a urči prvý bod na kružnici so stredom v začiatku $(x_0, y_0) = (0, r)$.
2. Vypočítaj začiatočnú hodnotu rozhodovacieho parametra ako $p_0 = 1 - r$ (pretože predpokladáme, že r je celočíselné).
3. V každej pozícii x_i, štartujúc v $i = 0$, vykonaj nasledujúci test:
   (a) Ak $p_i < 0$: nasledujúci bod pozdĺž kružnice so stredom $(0, 0)$ bude $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$ a $p_{i+1} = p_i + 2x_i + 3$.
   (b) Inak je nasledujúci bod $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)$ a $p_{i+1} = p_i + 2(x_i - y_i) + 5$.
4. Urči súmerné body v ostatných siedmich oktantoch.
5. Posuň každú vypočítanú pixlovú pozíciu (x, y) na kružnicu so stredom (x_C, y_C) a zobraz bod so súradnicami: $x = x + x_C$,$y = y + y_C$.
6. Opakuj kroky 3. až 5. pokiaľ $x_i \geq y_i$.

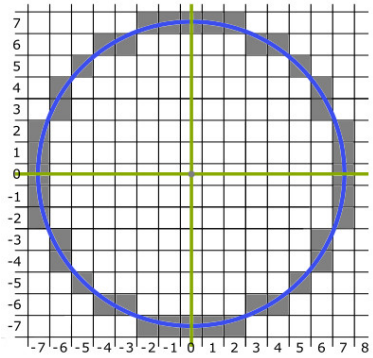
Príklad

Na ilustráciu algoritmu si ukážeme rasterizáciu kružnice so stredom v bode $(x_C, y_C) = (1, 2)$ a polomerom $r = 7$.

1. $(x_0, y_0) = (0, r) = (0, 7)$
2. $p_0 = 1 - r = -6$
3. $i = 0 :\ p_0 < 0 \Rightarrow (x_1, y_1) = (x_0 + 1, y_0) = (1, 7)$ a $p_1 = p_0 + 2x_0 + 3 = -3$
   $i = 1 :\ p_1 < 0 \Rightarrow (x_2, y_2) = (x_1 + 1, y_1) = (2, 7)$ a $p_2 = p_1 + 2x_1 + 3 = 2$
   $i = 2 :\ p_2 > 0 \Rightarrow (x_3, y_3) = (x_2 + 1, y_2 - 1) = (3, 6)$ a $p_3 = p_2 + 2(x_2 - y_2) + 5 = -3$
   $i = 3 :\ p_3 < 0 \Rightarrow (x_4, y_4) = (x_3 + 1, y_3) = (4, 6)$ a $p_4 = p_3 + 2x_3 + 3 = 6$
   $i = 4 :\ p_4 > 0 \Rightarrow (x_5, y_5) = (x_4 + 1, y_4 - 1) = (5, 5)$ a $x_5 \geq y_5$ teda algoritmus končí.
4. Určíme súmerné body v ostatných oktantoch (Obr. 28).
5. Posunieme každú hodnotu pixla do stredu $(1, 2)$. Teda body: $(2, 9), (3, 9), (4, 8), (5, 8), (6, 8), ...$

Pseudokód

Implementácia Bresenhamovho stredového kružnicového algoritmu je zhrnutá v nasledujúcom pseudokóde. Na vstupe je stred kružnice (x_C, y_C) a polomer kružnice r.



Obr. 28: Výsledná kružnica so stredom v bode $(0, 0)$

```
circleMidpoint (int xC, int yC, int r){
  int x = 0;
  int y = r;
  int p = 1 - r;
  void vykresliBodKruznice(int, int, int, int);

  /* Vykreslí prvý bod kružnice
  vykresliBodKruznice(xC, yC, x, y);

  while(x < y) {
    x++;
    if(p < 0) p += 2 * x + 3;
    else {
      y--;
      p += 2 * (x - y) + 5;
    }
  vykresliBodKruznice(xC, yC, x, y);
  }
}
void vykresliBodKruznice(int xC, int yC, int x, int y); {

  vykresliPixel(xC + x, yC + y);
  vykresliPixel(xC - x, yC + y);
  vykresliPixel(xC + x, yC - y);
  vykresliPixel(xC - x, yC - y);
  vykresliPixel(xC + y, yC + x);
  vykresliPixel(xC - y, yC + x);
  vykresliPixel(xC + y, yC - x);
  vykresliPixel(xC - y, yC - x);
}
```

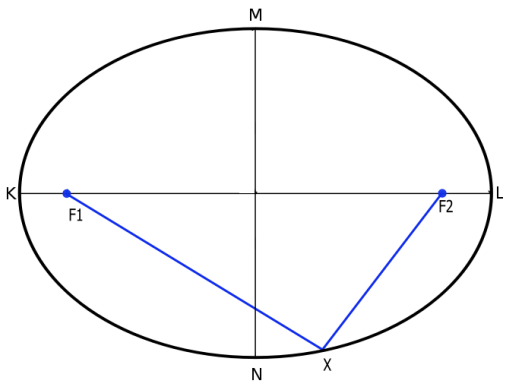
Literatúra

Bresenhamov stredový kružnicový algoritmus je detailne spracovaný v knihe [3] od strany 98. Zo slovenskej/českej literatúry nie je uvedený ani v [1] ani v [2].

3.3 Algoritmy na rasterizáciu elipsy

Elipsa je rovinná krivka, ktorá patrí do triedy kužeľosečiek.

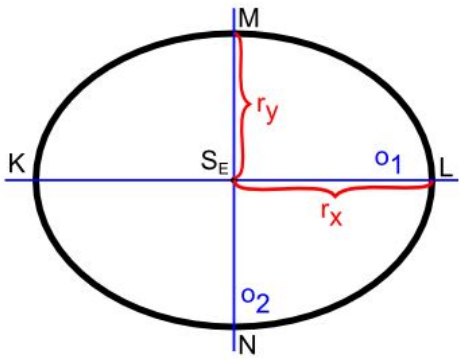
Elipsa E je definovaná ako množina všetkých bodov roviny, ktoré majú od dvoch rôznych pevných bodov F_1, F_2 rovnaký súčet vzdialeností, ktorý je väčší ako vzdialenosť týchto bodov (Obr. 29.).



Obr. 29: Elipsa

Body F_1 a F_2 nazývame ohniská elipsy. Priamka prechádzajúca týmito dvoma bodmi sa nazýva hlavná os elipsy a označíme ju o_1. Stred úsečky F_1F_2 nazývame stred elipsy a označíme ho ako $S_E = (x_C, y_C)$. Priamku kolmú na os o_1 prechádzajúcu bodom S_E nazývame vedľajšia os elipsy a označíme ju o_2. Body, v ktorých elipsa E pretne os o_1, nazývame hlavné vrcholy (body K a L na Obr. 30.) a body prieniku osi o_2 s elipsou E nazývame vedľajšie vrcholy elipsy (body M,N na Obr. 30.).

Dĺžku úsečky S_EL označíme ako r_x a nazývame ju hlavná poloos elipsy. Analogicky dĺžku úsečky S_EM označujeme r_y a nazývame vedľajšia poloos.

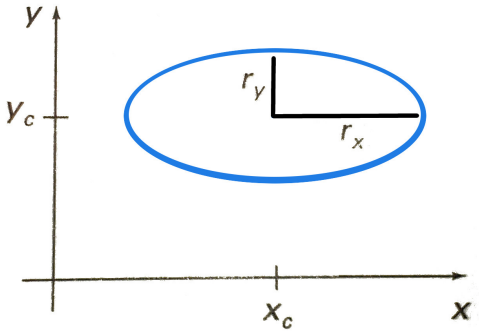


Obr. 30: Časti elipsy

Ďalej sa budeme zaoberať iba špeciálnou polohou elipsy, v ktorej je hlavná a vedľajšia os rovnobežná so súradnicovými osami. Túto polohu nazývame základná pozícia elipsy (Obr.31.).

V tomto prípade môžeme rovnicu elipsy zapísať ako

$$\left(\frac{x - x_C}{r_x}\right)^2 + \left(\frac{y - y_C}{r_y}\right)^2 = 1 \tag{3.8}$$



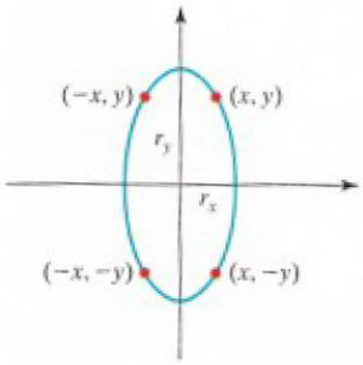
Obr. 31: Elipsa so stredom v bode (x_C, y_C) v základnej pozícii [3]

Pomocou parametrických súradníc vieme elipsu v základnej pozícií vyjadriť pomocou rovníc

$$\begin{aligned} x(t) &= x_C + r_x \cos(t), \\ y(t) &= y_C + r_y \sin(t), \end{aligned} \tag{3.9}$$

kde parameter $t \in \langle 0, 2\pi)$.

Na zrýchlenie výpočtov pixlov pozdĺž elipsy môžeme využiť súmernosti elipsy. Elipsa v základnej pozícii na rozdiel od kružnice je súmerná iba podľa súradnicových osí x a y. Preto môžeme vyčísliť pozície pixelov v jednom kvadrante a ostatné dostaneme zo súmerností (Obr. 32.).



Obr. 32: Súmerností elipsy [3]

V nasledujúcej podkapitole si ukážeme jeden algoritmus na zobrazenie elipsy do rastra. Na vstupe algoritmu máme polomer stred (x_C, y_C) elispy E a hodnoty r_x a r_y. Na výstupe dostaneme množinu bodov rastra, ktoré aproximujú danú elipsu E.

3.3.1 Bresenhamov stredový elipsový algoritmus (Midpoint Ellipse Algorithm)

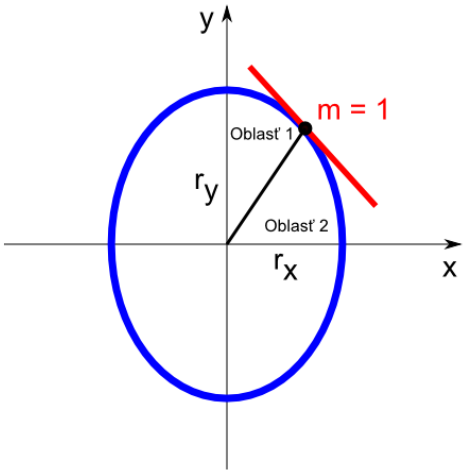
Princíp algoritmu

Bresenhamov stredový elipsový algoritmus je založený na rovnakom princípe ako Bresenhamov kružnicový algoritmus. Dané sú parametre r_x, r_y a stred elipsy (x_C, y_C). Vyčísľujeme body elipsy v základnej pozícii a to so stredom v bode $(0, 0)$. Následne sa všetky body elipsy posunú tak, že bod (x_C, y_C) je jej stredom.

Ak chceme vykresliť elipsu v inej ako základnej polohe (ak hlavná a vedľajšia os elipsy nie sú rovnobežné so súradnicovými osami), môžeme elipsu otočiť okolo tredu otáčania (viď Kapitola 1. rotácia).

V nasledujúcej časti ukážeme rasterizáciu elipsy so stredom v bode $(0, 0)$ a základnej pozícii. Algoritmus budeme aplikovať v I. kvadrante v dvoch oblastiach (Obr. 33.). Tieto oblasti vzniknú rozdelením I. kvadrantu na dva v bode elipsy, v ktorom dotyčnica k elipse má smernicu $m = -1$.

Hranicou medzi oblasťami je priamka spájajúca stred elipsy s dotykovým bodom.



Obr. 33: I. kvadrant elipsy rozdelený na dve oblasti [3]

Začíname v bode elipsy $(0, r_y)$ a postupujeme v smere chodu hodinových ručičiek. Najprv postupujeme jednotkovým krokom v smere osi x v prvej oblasti, až pokým sa nedostaneme na hranicu medzi oblasťami. Vtedy prejdeme na jednotkový krok v smere osi y, až kým neprejdeme celý I. kvadrant.Tento postup je potrebný, pretože ak by sme postupovali jednotkovým krokom iba v smere jednej zo súradnicových osí, pri rasterizácii by vznikali medzery medzi zobrazenými pixlami a výsledná elipsa by bola nespojitá.

Pre smernicu m dotyčnice v bode elipsy $X = (x, y)$ na hranici oblasti platí $m = -1 \Rightarrow \frac{dy}{dx} = \frac{2r_y^2 x}{2r_x^2 y} \Rightarrow 2r_y^2 x = 2r_x^2 y$. Teda pre smernicu m dotyčnice v bode elipsy $X = (x, y)$ v prvej oblasti I. kvadrantu platí $m > -1$ a v druhom $m < -1$. Preto môžeme povedať, že bod elipsy $X = (x, y)$ sa nachádza v prvej oblasti, ak platí $2r_y^2 x \geq 2r_x^2 y$, t.j. vzorkujeme jednotkovým krokom v smere osi x, inak v smere osi y.

Samozrejme, modifikáciou algoritmu je možné postupovať zo začiatočného bodu $(r_x, 0)$ proti smeru hodinových ručičiek.

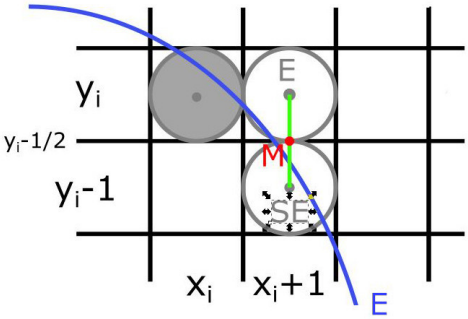
Zapíšeme rovnicu elipsy vyjadrenú v rovnici (3.8) so stredom $(x_C, y_C) = (0, 0)$ ako $f(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$.

Je zrejmé, že platí:

- Ak $f(x, y) < 0$ tak bod $X = (x, y)$ je vnútorný bod elipsy
- Ak $f(x, y) = 0$ tak bod $X = (x, y)$ je bodom elipsy
- Ak $f(x, y) > 0$ tak bod $X = (x, y)$ je vonkajším bodom elipsy

Napr. nachádzame sa v bode elipsy (x_i, y_i) a rozhodujeme sa, ktorý z nasledujúcich bodov pozdĺž elipsy vykreslíme. Môžu nastať dve situácie:

1. Nachádzame sa v prvej oblasti. Teda platí $2r_y^2x_i \leq 2r_x^2y_i$ a kandidáti na nasledujúci bod sú body na mieste $E = (x_i + 1, y_i)$ a $SE = (x_i + 1, y_i - 1)$ (Obr. 34.). Ako v oboch



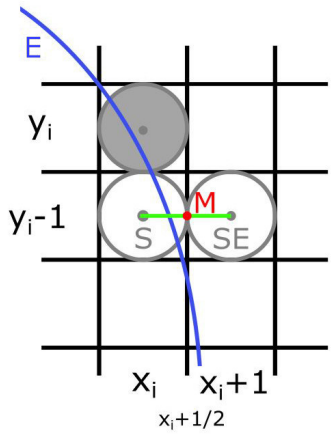
Obr. 34: Dvaja kandidáti na vykreslenie

Midpoint algoritmoch, použijeme bod $M = stred(E, SE) = (x_i + 1, y_i - \frac{1}{2})$. Vyjadríme si rozhodovací parameter pre prvú oblasť ako $p1_i = f(M) = f(x_i + 1, y_i - \frac{1}{2}) = r_y^2(x_i + 1)^2 + r_x^2(y_i - \frac{1}{2})^2 - r_x^2r_y^2$ a odvodíme rekurentný vzorec.
$p1_{i+1} = f(x_{i+1} + 1, y_{i+1} - \frac{1}{2}) = r_y^2\left(x_{i+1} + 1\right)^2 + r_x^2\left(y_{i+1} - \frac{1}{2}\right)^2 - r_x^2r_y^2$. Z toho
$p1_{i+1} = p1_i + 2r_y^2(x_i + 1) + r_y^2 + r_x^2\left((y_{i+1} - \frac{1}{2})^2 - (y_i - \frac{1}{2})^2\right)$.
Zrejme platí:

- Ak $p1_i < 0$, tak vyberáme za nasledujúci pixel $(x_{i+1}, y_{i+1}) = E = (x_i + 1, y_i)$ a $p1_{i+1} = p1_i + 2r_y^2x_{i+1} + r_y^2$.
- Ak $p1_i \geq 0$, tak vyberáme za nasledujúci pixel $(x_{i+1}, y_{i+1}) = SE = (x_i + 1, y_i - 1)$ a $p1_{i+1} = p1_i + 2r_y^2x_{i+1} + r_y^2 - 2r_x^2y_{i+1}$.

Potrebujeme ešte vyčísliť hodnotu začiatočného parametra v začiatočnej pozícii pre $(x_0, y_0) = (0, r_y)$, teda $p1_0 = f(1, r_y - \frac{1}{2}) = r_y^2 - r_x^2r_y + \frac{1}{4}r_x^2$

2. Ak $2r_y^2x_i \geq 2r_x^2y_i$ nachádzame sa v druhej oblasti. Teda kandidáti na nasledujúci bod sú body na mieste $S = (x_i, y_i - 1)$ a $SE = (x_i + 1, y_i - 1)$ (Obr. 35.).V tomto prípade použijeme bod $M = stred(S, SE) = (x_i + \frac{1}{2}, y_i - 1)$. Vyjadríme si rozhodovací parameter pre druhú



Obr. 35: Dvaja kandidáti na vykreslenie

oblasť ako $p2_i = f(M) = f(x_i + \frac{1}{2}, y_i - 1) = r_y^2(x_i + \frac{1}{2}) + r_x^2(y_i - 1) - r_x^2 r_y^2$ a odvodíme rekurentný vzorec.
$p2_{i+1} = f(x_{i+1} + \frac{1}{2}, y_{i+1} - 1) = r_y^2(x_{i+1} + \frac{1}{2})^2 + r_x^2\,(y_i - 1)^2 - r_x^2 r_y^2$. Z toho
$p2_{i+1} = p2_i + 2r_x^2(y_i - 1) + r_x^2 + r_y^2\left((x_{i+1} + \frac{1}{2})^2 - (x_i + \frac{1}{2})^2\right)$.
Zrejme platí:

- Ak $p2_i < 0$, tak vyberáme za nasledujúci pixel $(x_{i+1}, y_{i+1}) = S = (x_i, y_i - 1)$ a $p2_{i+1} = p2_i - 2r_x^2 y_{i+1} + r_x^2$.
- Ak $p2_i \geq 0$, tak vyberáme za nasledujúci pixel $(x_{i+1}, y_{i+1}) = SE = (x_i + 1, y_i - 1)$ a $p2_{i+1} = p2_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2$.

Potrebujeme ešte vyčísliť hodnotu začiatočného parametra v poslednom bode z prvej oblasti, ktorý je vlastne bod (x_0, y_0) v oblasti dva. Teda $p2_0 = f(x_0 + \frac{1}{2}, y_0 - 1) = r_y^2(x_0 + \frac{1}{2})^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$.

Pre zjednodušenie môžeme pixely v druhej oblasti I. kvadrantu vyčísľovať taktiež z bodu $(r_x, 0)$ proti smeru chodu hodinových ručičiek, až kým sa nedostaneme do posledného vyčísleného bodu prvej oblasti.

Tak ako v kružnicovom algoritme, pri výpočte $p1_{i+1}$ a $p2_{i+1}$ používame iba operácie sčitovania a odčitovania. Hodnoty $2r_y^2 x_{i+1}$ a $2r_x^2 y_{i+1}$ je možno počítať s pomocou konštantného prírastku. Pre hodnotu začiatočného parametra v bode $(x_0, y_0) = (0, r_y)$ platí $2r_y^2 x_0 = 0$ a $2r_x^2 y_0 = 2r_x^2 r_y$. Ako sa budú hodnoty x_i a y_i zvyšovať, budú sa tieto parametre meniť v I kvadrante takto:

- Ak $x_{i+1} > xi$ platí $2r_y^2 x_{i+1} = 2r_y^2 x_i + 2r_y^2$, inak $2r_y^2 x_{i+1} = 2r_y^2 x_i$.
- Ak $y_{i+1} < yi$ platí $2r_x^2 y_{i+1} = 2r_x^2 x_i - 2r_x^2$, inak $2r_x^2 y_{i+1} = 2r_x^2 y_i$.

Postup

Kroky algoritmu vieme zhrnúť nasledovne:

1. Zadaj r_x a r_y a stred elipsy (x_C, y_C) a urči prvý bod na elipse so stredom v začiatku $(x_0, y_0) = (0, r_y)$.
2. Vypočítaj začiatočnú hodnotu rozhodovacieho parametra v prvej oblasti ako $p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$, urči $2r_y^2 x_0 a 2r_x^2 y_0$.
3. V každej pozícii x_i v prvej oblasti, štartujúc v $i = 0$, vykonaj nasledujúci test:
   (a) Ak $p1_i < 0$: nasledujúci bod pozdĺž elipsy so stredom $(0, 0)$ bude $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$ a $p1_{i+1} = p1_i + 2r_y^2 x_{i+1} + r_y^2$.
   (b) Inak je nasledujúci bod $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)$ a $p1_{i+1} = p1_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_y^2$.

   a urči $2r_y^2 x_{i+1} = 2r_y^2 x_i + 2r_y^2$ a $2r_x^2 y_{i+1} = 2r_x^2 y_i - 2r_x^2$
4. Opakuj krok 3. až pokým $2r_y^2 x \leq 2r_x^2 y$.
5. Urči začiatočnú hodnotu rozhodovacieho parametra v druhej oblasti využitím posledného bodu z prvej oblasti ako $p2_0 = r_y^2(x_0 + \frac{1}{2})^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$
6. V každej pozícii x_i v druhej oblasti, štartujúc v $i = 0$, vykonaj nasledujúci test:
   (a) Ak $p2_i > 0$: nasledujúci bod pozdĺž elipsy so stredom $(0, 0)$ bude $(x_{i+1}, y_{i+1}) = (x_i, y_i - 1)$ a $p2_{i+1} = p2_i - 2r_x^2 y_{i+1} + r_x^2$.
   (b) Inak je nasledujúci bod $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)$ a $p2_{i+1} = p2_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2$.

   a urči $2r_y^2 x_{i+1} = 2r_y^2 x_i + 2r_y^2$ a $2r_x^2 y_{i+1} = 2r_x^2 y_i - 2r_x^2$.
7. Urči súmerné body v ostatných troch kvadrantoch.
8. Posuň každú vypočítanú pixlovú pozíciu (x, y) na elipsu so stredom (x_C, y_C) a zobraz bod so súradnicami: $x = x + x_C$,$y = y + y_C$.

Príklad

Na ilustráciu algoritmu si ukážeme rasterizáciu elipsy s parametrami $r_x = 8$ a $r_y = 6$ a stredom elipsy v bode $(0, 7)$

1. $(x_0, y_0) = (0, r_y) = (0, 6)$.
2. $p1_0 = -332$, $2r_y^2 x_0 = 0$ a $2r_x^2 y_0 = 2.8^2.6 = 768$. Výraz $2r_y^2 x$ sa bude zvyšovať o hodnotu $2r_y^2 = 72$ a výraz $2r_x^2 y$ o hodnotu $-2r_x^2 = -128$.
3. $i = 0 :\ p1_0 < 0 \Rightarrow (x_1, y_1) = (x_0 + 1, y_0) = (1, 6)$, $p1_1 = p1_0 + 2r_y^2 x_1 + r_y^2 = -224$, $2r_y^2 x_1 = 72$ a $2r_x^2 y_1 = 768$.

   $i = 1 :\ p1_1 < 0 \Rightarrow (x_2, y_2) = (x_1 + 1, y_1) = (2, 6)$, $p1_2 = p1_1 + 2r_y^2 x_2 + r_y^2 = -44$, $2r_y^2 x_2 = 72 + 72 = 144$ a $2r_x^2 y_2 = 768$.

   $i = 2 :\ p1_2 < 0 \Rightarrow (x_3, y_3) = (x_2 + 1, y_2) = (3, 6)$, $p1_3 = p1_2 + 2r_y^2 x_3 + r_y^2 = 208$, $2r_y^2 x_3 = 144 + 72 = 216$ a $2r_x^2 y_3 = 768$.

$i = 3: \ p1_3 > 0 \Rightarrow (x_4, y_4) = (x_3 + 1, y_3 - 1) = (4, 5), p1_4 = p1_3 + 2r_y^2 x_4 - 2r_x^2 y_4 + r_y^2 = -108$, $2r_y^2 x_4 = 216 + 72 = 288$ a $2r_x^2 y_4 = 768 - 128 = 640$.

$i = 4: \ p1_4 < 0 \Rightarrow (x_5, y_5) = (x_4 + 1, y_4) = (5, 5)$, $p1_5 = p1_4 + 2r_y^2 x_5 + r_y^2 = 288$, $2r_y^2 x_5 = 288 + 72 = 360$ a $2r_x^2 y_5 = 640$.

$i = 5: \ p1_5 > 0 \Rightarrow (x_6, y_6) = (x_5 + 1, y_5 - 1) = (6, 4), p1_6 = p1_5 + 2r_y^2 x_6 - 2r_x^2 y_6 + r_y^2 = 244$, $2r_y^2 x_6 = 360 + 72 = 432$ a $2r_x^2 y_6 = 640 - 128 = 512$.

$i = 6: \ p1_6 > 0 \Rightarrow (x_7, y_7) = (x_6 + 1, y_6 - 1) = (7, 3)$, $2r_y^2 x_7 = 432 + 72 = 504$ a $2r_x^2 y_7 = 512 - 128 = 384$. Platí $2r_y^2 x_7 \geq 2r_x^2 y_7$, preto prechádzame do druhej oblasti.

4. $(x_0, y_0) = (7, 3)$ a teda $p2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2 = -23$.

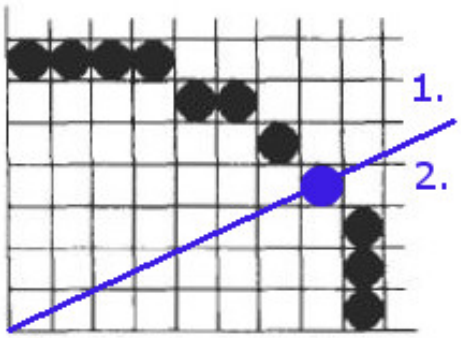
5. $i = 0: \ p2_0 < 0 \Rightarrow (x_1, y_1) = (x_0 + 1, y_0 - 1) = (8, 2), p2_1 = p2_0 + 2r_y^2 x_1 - 2r_x^2 y_1 + r_x^2 = 233$, $2r_y^2 x_1 = 504 + 72 = 576$ a $2r_x^2 y_1 = 384 - 128 = 256$.

   $i = 1: \ p2_1 > 0 \Rightarrow (x_2, y_2) = (x_1, y_1 - 1) = (8, 1)$, $p2_2 = p2_1 + 2r_x^2 y_2 + r_x^2 = 169$, $2r_y^2 x_2 = 576$ a $2r_x^2 y_2 = 256 - 128 = 128$.

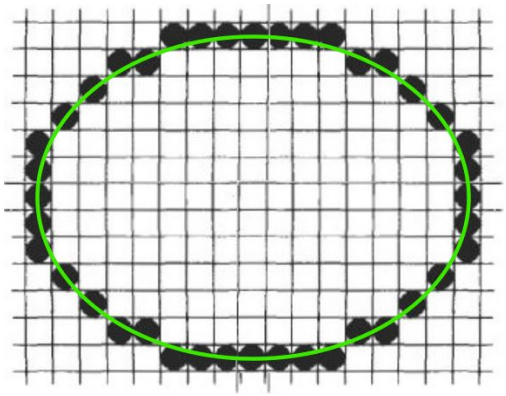
   $i = 2: \ p2_2 > 0 \Rightarrow (x_3, y_3) = (x_2, y_2 - 1) = (8, 0)$, algoritmus končí, pretože sme sa dostali na hranicu I. oktantu. Výsledné body možno vidieť na Obr. 36.

6. Určíme súmerné body v ostatných oktantoch. Výslednú elipsu možno vidieť na Obr. 37.

7. Posunieme každú hodnotu pixla do stredu $(0, 7)$. Teda dostaneme body: $(0, 13)$,$(1, 13)$,$(2, 13)$, $(3, 13)$, $(4, 12)$, ...



Obr. 36: Body elipsy v 1. kvadrante



Obr. 37: Výsledná elipsa

Pseudokód

Implementácia Bresenhamovho stredového elipsového algoritmu je zhrnutá v nasledujúcom pseudokóde. Na vstupe sú parametre r_x, r_y a stred elipsy(x_C, y_C).

```
ellipseMidpoint (int xC, int yC, int rx, int ry)
{
  int p;
  int x = 0;
  int y = ry;
  int px = 0;
  int py = 2 * rx * rx * y;
```

```
  void vykresliBodElipsy(int, int, int, int);

  /* Vykreslí prvý bod elipsy
  vykresliBodElipsy(xC, yC, x, y);

  /* Región 1
  p=ROUND(ry * ry -(rx * rx * ry) + (0.25 * rx * rx));
  while(px <py) {
    x++;
    px += 2 * ry * ry;
    if(p < 0) p += ry * ry + px;
    else {
      y--;
      py -= 2 * rx * rx;
      p += ry * ry + px - py;
    }
  vykresliBodElipsy(xC, yC, x, y);
  }

  /* Región 2
  p=ROUND(ry * ry * (x+0.5) * (x+0.5) + rx * rx * (y - 1) * (y - 1)
  - rx * rx * ry * ry);
  while(y > 0) {
    y--;
    py += 2 * rx * rx;
    if(p > 0) p += rx * rx + py;
    else {
      x++;
      px -= 2 * ry * ry;
      p += rx * rx + px;
    }
  vykresliBodElipsy(xC, yC, x, y);
  }
}

void vykresliBodElipsy(int xC, int yC, int x, int y);
{
  vykresliPixel(xC + x, yC + y);
  vykresliPixel(xC - x, yC + y);
  vykresliPixel(xC + x, yC - y);
  vykresliPixel(xC - x, yC - y);
}
```

Literatúra

Bresenhamov stredový elipsový algoritmus je podrobne spracovaný v [3] od strany 102. Je uvedený aj v [2] od strany 88, avšak je uvedený iba výsledný vzorec na výpočet bodov pozdĺž elipsy, bez odvodenia.

4 Vypĺňanie

4.1 Vypĺňanie oblastí

V počítačovej grafike v mnohých prípadoch nie je potrebné vykresliť jedinú priamku alebo krivku, ale vyplniť určitú oblasť.

V niektorých algoritmoch sa pri hľadaní a následnom vypĺňaní pixlov predpokladá, že poznáme aspoň jeden vnútorný bod v súvislej oblasti - **semienko**. Takýto typ vypĺňania sa preto nazýva **semienkové vypĺňanie**

Pri vypĺňaní prechádzajú algoritmy všetkými pixlami súvislej oblasti a farbia ich farbou, ktorou sa má táto oblasť zafarbiť (tzv. nová farba).

Body 4- alebo 8-susedných oblastí klasifikujeme takto:

- Testovaný bod je vnútorný, ak má inú farbu ako hraničný. Vypĺňanie založené na tomto princípe sa nazýva **hraničné vypĺňanie (po hranicu)**.
- Testovaný bod je vnútorný, pokiaľ má farbu, ktorou sú zafarbené všetky body vnútra. Toto je charakteristické pre tzv. **záplavové / lavínové vypĺňanie**, pri ktorom sa súvislá oblasť prefarbuje novou farbou (všetky jej pixle tou istou).
- Testovaný bod je vnútorný, ak má farbu, ktorá sa výrazne líši od farby hranice. Je to špeciálny prípad hraničného vypĺňania, v prípade, že hranica má len približne určený farebný odtieň a jas, čo býva obyčajne dôsledkom vyhladzovania obrazu a špeciálne hranice. V tomto prípade hovoríme o **mäkkom vypĺňaní**.

Vo všeobecnosti existujú dve možnosti pre definovanie oblasti, ktorá sa má vyplniť: oblasť zadaná svojimi vnútornými bodmi alebo oblasť zadaná hranicou.

Intuitívny prístup, ktorý nás napadne, je chápať zadanú množinu hraničných pixlov ako pre hranicu určitej oblasti, podobne ako je to s bodmi uzavretej krivky v rovine. Takáto množina pixlov by mala byť súvislá a podobne ako uzavretá rovinná krivka by mala rozdeľovať nekonečnú mriežku na dve časti: vnútro a vonkajšok.

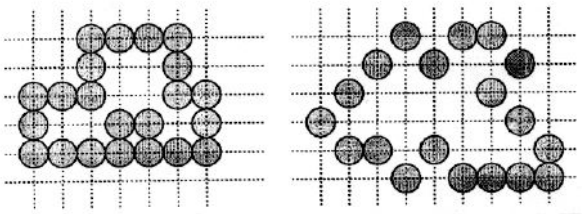
Podľa známej Jordanovej vety platí, že každá uzavretá krivka v rovine rozdeľuje rovinu na dve súvislé oblasti: vnútro a vonkajšok. Zatiaľ však nemáme definované, čo sa rozumie uzavretou krivkou v našom prípade, dokonca nám bez tejto definície môžu nastať určité problémy (Obr. 38).

Ak hraničná krivka je 8-súvislá, tak vo všeobecnosti neplatí, že rozdeľuje nekonečnú sieť do dvoch 8-súvislých oblastí, pretože (ako vidieť na Obr.38) vnútro a vonkajšok sa dajú navzájom spojiť 8-súvislou cestou. Existuje však zaujímavý spôsob, ako upraviť tento problém. Špeciálne, ak požadujeme, aby hraničná krivka bola 8-súvislá, tak je potrebné, aby ňou definovaná oblasť bola 4-súvislá. Podobne, ak chceme, aby hranica bola 4-súvislá, je vhodné požadovať, aby oblasť, ktorú definuje, bola 8-súvislá.

V tejto kapitole si uvedieme tri algoritmy pre vypĺňanie a pre oblasť danú hranicou uvedieme jeden vylepšený algoritmus.

4.1.1 Vnútorný bod mnohouholníka

Jeden zo základných úloh počítačovej grafiky je určenie vnútorného bodu mnohouholníka.



Obr. 38: 4-súvislá (vľavo) a 8-súvislá (vpravo) množina pixlov [6]

Na objasnenie pojmov ako sú vnútro alebo vonkajšok mnohouholníka si uvedieme základné definície [8].

Okolie bodu je každá otvorená množina obsahujúca daný bod.

Bod A priestoru $X \subset E^n$ je vnútorný bod množiny M⊂X, ak existuje jeho okolie ležiace v M. Množina všetkých vnútorných bodov množiny M sa nazýva vnútro množiny M a označuje sa int$M$(z angl. interior = vnútro).

Bod A je hraničný bod množiny M, ak každé jeho okolie pretína množinu M aj jej doplnok (t.j. obsahuje bod patriaci množine M aj bod jej nepatriaci). Množina všetkých hraničných bodov množiny M sa nazýva hranica množiny M a označuje sa ∂M alebo bd$M$.

Bod A je vonkajší bod množiny M, ak existuje jeho okolie nepretínajúce množinu M. Množina všetkých vonkajších bodov množiny M sa nazýva vonkajšok množiny M a označuje sa ext$M$(z angl. exterior = vonkajšok).

Platí, že bod je vnútorným bodom mnohouholníka, ak polpriamka rovnobežná s osou x vychádzajúca z tohto bodu pretne mnohouholník v nepárnom počte hrán [1]. Predpokladáme pri tom, že polpriamka neprechádza žiadnym vrcholom mnohouholníka. Ak by takáto situácia nastala, môžeme mnohouholník posúvať.

4.1.2 Záplavový algoritmus (Flood fill Algorithm)

Princíp algoritmu

Na vstupe nech je 4-súvislá oblasť daná všetkými svojimi bodmi s farbou $stara - farba$.

Na výstupe má byť tá istá oblasť zafarbená farbou $nova - farba$. Procedúra ešte pozná súradnice jedného vnútorného bodu (x, y)oblasti. Je zrejmé, že ide o tzv. semienkové vypĺňanie. V algoritme sa predpokladá, že hodnoty priradené jednotlivým pixlom sa dajú priamo prečítať z obrazovej pamäte.

Tento algoritmus pracuje na princípe rekurzie. Rekurzia predstavuje využitie časti vlastnej vnútornej štruktúry, najmä definovanie funkcie pomocou seba samej resp. samotná táto funkcia.

Pseudokód

Implementácia rekurzívneho Flood-fill algoritmu pre 4-súvislú oblasť je zhrnutá v nasledovnom zápise. Na vstupe nech je 4-súvislá oblasť daná všetkými svojimi bodmi s farbou $stara - farba$.

Na výstupe má byť tá istá oblasť zafarbená farbou $nova - farba$. Procedúra, ktorá oblasť vyfarbí má okrem farieb ešte súradnice jedného vnútorného bodu oblasti (x, y).

Funkcia $zapis - pixel(x, y, color)$ má tri parametre: súradnice vykresleného bodu rastra a jeho

príslušnú farbu. Funkcia $nacitaj - pixel(x, y)$ vráti farbu zadaného pixlu (x, y).

```
Flood-fill-4(int x, int y, color stara-farba, color nova-farba)
{
  if (nacitaj-pixel(x,y) = stara-farba) {
    zapis-pixel(x,y,nova-farba);
    Flood-fill-4(x,y-1,stara-farba, nova-farba);
    Flood-fill-4(x,y+1,stara-farba, nova-farba);
    Flood-fill-4(x-1,y,stara-farba, nova-farba);
    Flood-fill-4(x+1,y,stara-farba, nova-farba);
  }
}
```

Tento algoritmus sa obmedzuje síce na 4-súvislú oblasti, ale je jednoducho modifikovateľný pre 8-súvislú oblasť.

Literatúra

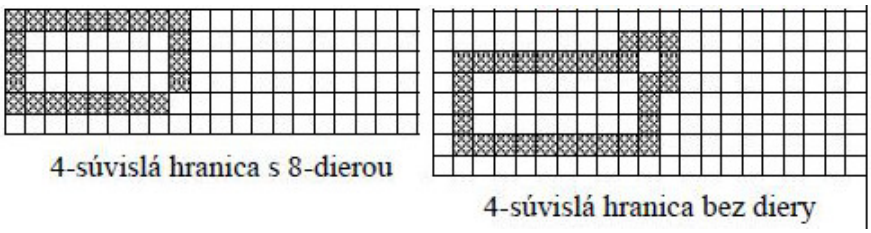
Záplavový algoritmus vieme nájsť podrobne spracovaný v literatúre [3] na strane 130. Jeho princíp je aj v [1] na 65. strane.

4.1.3 Algoritmus vypĺňania do hraničných bodov (Bound fill Algorithm)

Princíp algoritmu

Na vstupe je 4-súvislá hranica farby $hranicna - farba$. Na výstupe má byť oblasť ohraničená touto hranicou a zafarbená farbou $nova - farba$. V tomto algoritme potrebujeme hranicu, ktorá je 4-súvislá v silnejšom zmysle ako sme definovali v úvode. Hranica nesmie mať 8-dieru (Obr. 39.)

Ako vidno, myšlienka tohto algoritmu sa podobá na $Flood - fill - 4$. Nestačí však testovať, či bod (x, y) patrí oblasti. Potrebné sú dva testy, či je vo vnútri oblasti a či mu už skôr nebola pridelená nová farba. Aj modifikácia tohto algoritmu na 8-súvislé oblasti je pomerne jednoduchá.



Obr. 39: 4-súvislá hranica [6]

Pseudokód

Implementácia rekurzívneho Bound-fill-4 algoritmu je zhrnutá v nasledovnom zápise. Na vstupe nech je 4-súvislá hranica farby $hranicna - farba$. Na výstupe má byť oblasť ohraničená touto hranicou a zafarbená farbou $nova - farba$. Algoritmus opäť obsahuje funkciu $zapis - pixel(x, y, color)$ a funkciu $nacitaj - pixel(x, y)$, ktorá vráti farbu zadaného pixlu (x, y). Opäť potrebujeme poznať ešte súradnice jedného vnútorného bodu oblasti (x, y).

```
Bound-fill-4(int x, int y, color hranicna-farba, color nova-farba)
{
```

```
  if (nacitaj-pixel(x,y) ≠ hranicna-farba && nacitaj-pixel(x,y)
  ≠ nova-farba) {
    zapis-pixel(x,y,nova-farba);
    Bound-fill-4(x,y-1,hranicna-farba, nova-farba);
    Bound-fill-4(x,y+1,hranicna-farba, nova-farba);
    Bound-fill-4(x-1,y,hranicna-farba, nova-farba);
    Bound-fill-4(x+1,y,hranicna-farba, nova-farba);
  }
}
```

Túto procedúru vieme rozšíriť aj na osem susedných bodov:

```
Bound-fill-8(int x, int y, color hranicna-farba, color nova_color)
{
  if (nacitaj-pixel(x,y) ≠ hranicna-farba && nacitaj-pixel(x,y)
  ≠ nova-farba) {
    zapis-pixel(x,y,nova-farba);
    Bound-fill-8(x,y-1,hranicna-farba, nova-farba);
    Bound-fill-8(x,y+1,hranicna-farba, nova-farba);
    Bound-fill-8(x-1,y,hranicna-farba, nova-farba);
    Bound-fill-8(x+1,y,hranicna-farba, nova-farba);
    Bound-fill-8(x-1,y-1,hranicna-farba, nova-farba);
    Bound-fill-8(x+1,y-1,hranicna-farba, nova-farba);
    Bound-fill-8(x-1,y+1,hranicna-farba, nova-farba);
    Bound-fill-8(x+1,y+1,hranicna-farba, nova-farba);
  }
}
```

Literatúra

Algoritmus Bound fill vieme nájsť podrobne spracovaný v literatúre [3] na strane 130 ako aj v literatúre [1] na strane 66 a [2].

4.1.4 Riadkové semienkové vypĺňanie (Scan line seed fill Algorithm)

Princíp algoritmu

Rekurzívne algoritmy Flood fill a Bound fill sú jednoduché, no nie príliš vhodné pre implementáciu. Okrem veľkých pamäťových nárokov (väčšinou sa používa zásobník pre rekurziu), navyše pracujú dosť neefektívne, lebo v každom kroku vypĺňajú len štyri susedné body, ktoré sú naviac rozmiestnené v rôznych smeroch.

Preto sa pre implementáciu semienkového vypĺňania používajú rekurzívne algoritmy, ktoré pracujú po riadkoch a vypĺňajú postupne všetky body napravo i naľavo až po nájdenie hraničných bodov. Body (u nekonvexných oblastí) sa ukladajú do zásobníka. Tieto algoritmy sú známe ako **riadkové semienkové vypĺňanie (scan line seed fill)**.

4.2 Rozklad mnohouholníka do rastra

Pod rozkladom mnohouholníka budeme rozumieť vykreslenie mnohouholníka zadaného vrcholmi na rastrovom zariadení. Ako prvý spôsob riešenia problému, by nás mohol napadnúť nasledovný postup: rasterizáciou strán mnohouholníka vytvoríme hranične zadanú oblasť a následne použijeme algoritmus na vypĺňanie oblasti. V tomto texte si uvedieme efektívnejšiu metódu - Scan line algoritmus.

4.2.1 Riadkový skenovací algoritmus (Scan line Algorithm)

Princíp algoritmu

Scan Line algoritmus rozkladu mnohouholníka patrí do skupiny algoritmov, ktoré používajú princíp skenovacej priamky: ak je možné vyriešiť problém, úlohu z pohľadu jednej vodorovnej alebo zvislej priamky, urobí sa tak. Úlohu takejto priamky hrá často riadok alebo stĺpec pixlov rastra. Okrem zúženia pohľadu je druhou vlastnosťou skenovacích algoritmov využitie informácií z jedného kroku, v nasledujúcom kroku.

V algoritme bude skenovacia priamka vodorovná s celočíselnými súradnicami. Jej rovnica teda bude $y = y_i$, kde y_i nadobúda celočíselné hodnoty od najmenšej po najväčšiu y-ovú súradnicu vrcholov mnohouholníka.

Scan Line algoritmus na rozklad mnohouholníka funguje tak, že v každom kroku zistí prieniky skenovacej priamky so stranami mnohouholníka, tieto prieniky usporiada do dvojíc a úseky medzi dvojicami vyfarbí.

Ak urobíme prienik skenovacej priamky so stranami mnohouholníka, chceme získať párny počet bodov. Preto musíme zo zoznamu strán mnohouholníka vylúčiť vodorovné strany (tie sa vykreslia priamo) a ešte musíme skrátiť zdola strany v neextremálnych bodoch (vo vrcholoch, z ktorých idú strany do opačných polrovín vzhľadom na skenovaciu priamku).

Konkrétna realizácia tohto skrátenia bude zrejmá z inicializácie dátovej štruktúry, ktorú algoritmus používa.

Dátové štruktúry pre Scan Line algoritmus

Základným kameňom dátových štruktúr bude záznam o strane mnohouholníka. Tento záznam bude obsahovať tri čísla:

1. maximálnu y- ovú súradnicu strany; je to väčšia z y-ových súradníc vrcholov strany, toto číslo použijeme pri rozhodovaní, či je nutné zisťovať prienik strany so skenovacou priamkou

2. x-ovú súradnicu bodu s minimálnou y-ovou súradnicou ; toto číslo je vlastne x-ová súradnica bodu, v ktorom skenovacia priamka prvýkrát pretne stranu a v priebehu algoritmu je táto hodnota modifikovaná

3. prevrátenú hodnotu smernice priamky, na ktorej strana leží: $\frac{1}{m} = \frac{x_A - x_B}{y_A - y_B}$, kde A,B sú vrcholy strany.

V predchádzajúcej časti sme hovorili o skrátení strany zdola v neextremálnom vrchole. Toto skrátenie zrealizujeme tak, že druhú hodnotu záznamu pre stranu budeme inicializovať nie na x-ovú súradnicu vrcholu strany, ktorý má menšiu y- ovú súradnicu, ale ak je tento vrchol neextremálny, na jeho x-ovú súradnicu zväčšenú o $\frac{1}{m}$. Pri tomto skrátení vychádzame z toho, že ak na

priamke so smernicou m leží bod so súradnicami $[x, y]$, tak na nej leží aj bod so súradnicami$[x + \frac{1}{m}, y + 1](m(x + \frac{1}{m}) + q = mx + q + 1 = y + 1)$.

Algoritmus používa dve dátové štruktúry: tabuľku strán (TS) a tabuľku aktívnych strán (TAS).

TS sa vytvára pri inicializácii a má riadky označené hodnotami, ktoré skenovacia priamka nadobudne v jednotlivých krokoch (celočíselné hodnoty od najmenšej po najväčšiu y- ovú súradnicu vrcholov mnohouholníka). V jednotlivých riadkoch je zoznam záznamov strán, ktoré majú túto hodnotu ako svoju minimálnu y- ovú súradnicu. Ak je vrchol strany s menšou y- ovou súradnicou neextremálny, strana bude zapísaná až v ďalšom riadku TS (pretože je zdola skrátená). Všimnime si, že až teraz, keď je strana zaradená do TS, máme o nej úplnú informáciu.

TAS je zoznam strán, s ktorými má aktuálna skenovacia priamka prienik a vytvára sa počas behu algoritmu vyberaním záznamov o stranách z TS. V tejto tabuľke sa druhá položka záznamu o každej strane mení tak, aby vždy mala hodnotu x-ovej súradnice prieniku strany a skenovacej priamky.

Postup

Na vstupe algoritmu nech je usporiadaný zoznam vrcholov mnohouholníka s celočíselnými súradnicami. Na výstupe algoritmu musí byť tento mnohouholník rozložený do rastra t.j. v rastri majú byť vyfarbené príslušné obrazové body.

1. Zisti, ktoré strany mnohouholníka sú vodorovné, ktoré vrcholy neextremálne.
2. Strany, ktoré nie sú vodorovné zapíš do TS, TAS inicializuj ako prázdnu, teda $y = y_{min}$
3. Kým TS alebo TAS sú neprázdne opakuj:
   (a) Vyber TS strany v riadku y a daj ich do TAS.
   (b) Usporiadaj strany v TAS podľa x-ovej súradnice (druhá položka v jednotlivých záznamoch).
   (c) Vyber za sebou idúce úseky a vykresli ich.
   (d) Zruš tie strany v TAS, pre ktoré $y_{max} = y$
   (e) Pre strany v TAS zmeň x na $x + \frac{1}{m}$
   (f) $y = y + 1$

Príklad

Pre mnohouholník $A_0A_1A_2A_3A_4A_5A_6$, kde $A_0 = (0, 4)$, $A_1 = (3, 0)$, $A_2 = (5, 3)$, $A_3 = (4, 5)$, $A_4 = (3, 2)$, $A_5 = (2, 7)$, $A_6 = (1, 7)$, opíšeme TS, jednotlivé kroky TAS ako aj výstup algoritmu a jednotlivé vykresľované úseky. Mnohouholník môžeme vidieť na Obr. 40.

1. Na obrázku môžeme vidieť, že mnohouholník má jednu vodorovnú hranu f, tú do TS nezaradíme. Tiež obsahuje dva neextremálne vrcholy A_0 a A_2.
2. $y_{min} = 0$ a $y_{max} = 7$ a TS vyzerá nasledovne:

   $0: \ a = 4 \mid 3 \mid -\frac{3}{4}, b = 3 \mid 3 \mid \frac{2}{3}$

   $1:$

   $2: \ d = 5 \mid 3 \mid \frac{1}{3}, e = 7 \mid 3 \mid -\frac{1}{5}$

   $3:$

$4: \ c = 5 \mid 5 - \frac{1}{2} \mid -\frac{1}{2}$

$5: \ g = 7 \mid 0 + \frac{1}{3} \mid \frac{1}{3}$

$6:$

$7:$

3. $i = 0:$ (a) TAS: $a = 4 \mid 3 \mid -\frac{3}{4}$, $b = 3 \mid 3 \mid \frac{2}{3}$
   (b) TAS sa po usporiadaní nezmení.
   (c) Vykresli úsek $[3, 0]$ až $[3, 0]$, t.j. pixel $[3, 0]$.
   (d) Z TAS sa žiadna hrana nezruší.
   (e) TAS: $a = 4 \mid \frac{9}{4} \mid -\frac{3}{4}$, $b = 3 \mid \frac{11}{3} \mid \frac{2}{3}$
   (f) $y = 1$

   $i = 1:$ (a) TAS sa nezmení, t.j. TAS: $a = 4 \mid 3 \mid -\frac{3}{4}$, $b = 3 \mid 3 \mid \frac{2}{3}$
   (b) TAS sa po usporiadaní nezmení.
   (c) Vykresli úsek $[\frac{9}{4}, 1]$ až $[\frac{11}{3}, 1]$, t.j. pixle $[2, 1]$,$[3, 1]$,$[4, 1]$.
   (d) Z TAS sa žiadna hrana nezruší.
   (e) TAS: $a = 4 \mid \frac{6}{4} \mid -\frac{3}{4}$, $b = 3 \mid \frac{11}{3} \mid \frac{2}{3}$
   (f) $y = 2$

   $i = 2:$ (a) TAS: $a = 4 \mid \frac{6}{4} \mid -\frac{3}{4}$, $b = 3 \mid \frac{11}{3} \mid \frac{2}{3}$, $d = 5 \mid 3 \mid \frac{1}{3}$, $e = 7 \mid 3 \mid -\frac{1}{3}$
   (b) TAS: $a = 4 \mid \frac{6}{4} \mid -\frac{3}{4}$, $d = 5 \mid 3 \mid \frac{1}{3}$, $e = 7 \mid 3 \mid -\frac{1}{3}$, $b = 3 \mid \frac{11}{3} \mid \frac{2}{3}$.
   (c) Vykreslia sa úseky $[\frac{6}{4}, 2]$ až $[3, 2]$ a $[3, 2]$ až $[\frac{13}{3}, 2]$, t.j. pixle $[2, 2]$,$[3, 2]$,$[4, 2]$.
   (d) Z TAS sa žiadna hrana nezruší.
   (e) TAS: $a = 4 \mid \frac{3}{4} \mid -\frac{3}{4}$, $d = 5 \mid \frac{10}{3} \mid \frac{1}{3}$, $e = 7 \mid \frac{14}{5} \mid -\frac{1}{5}$, $b = 3 \mid 5 \mid \frac{2}{3}$.
   (f) $y = 3$

   $i = 3:$ (a) TAS sa nemení.
   (b) TAS: $a = 4 \mid \frac{3}{4} \mid -\frac{3}{4}$, $e = 7 \mid \frac{14}{5} \mid -\frac{1}{5}$, $d = 5 \mid \frac{10}{3} \mid \frac{1}{3}$, $b = 3 \mid 5 \mid \frac{2}{3}$.
   (c) Vykreslia sa úseky $[\frac{3}{4}, 3]$ až $[\frac{14}{3}, 3]$ a $[\frac{10}{3}, 3]$ až $[5, 3]$, t.j. pixle $[1, 3]$,$[2, 3]$, $[3, 3]$, $[4, 3]$, $[5, 3]$.
   (d) TAS: $a = 4 \mid \frac{3}{4} \mid -\frac{3}{4}$, $e = 7 \mid \frac{14}{5} \mid -\frac{1}{5}$, $d = 5 \mid \frac{10}{3} \mid \frac{1}{3}$.
   (e) TAS: $a = 4 \mid 0 \mid -\frac{3}{4}$, $e = 7 \mid \frac{13}{5} \mid -\frac{1}{5}$, $d = 5 \mid \frac{11}{3} \mid \frac{1}{3}$.
   (f) $y = 4$

   $i = 4:$ (a) TAS: $a = 4 \mid 0 \mid -\frac{3}{4}$, $e = 7 \mid \frac{13}{5} \mid -\frac{1}{5}$, $d = 5 \mid \frac{11}{3} \mid \frac{1}{3}$, $c = 5 \mid \frac{9}{2} \mid -\frac{1}{2}$.
   (b) TAS sa po usporiadaní nezmení.
   (c) Vykreslia sa úseky $[0, 4]$ až $[\frac{13}{5}, 4]$ a $[\frac{11}{3}, 4]$ až $[\frac{9}{2}, 4]$, t.j. pixle $[0, 4]$,$[1, 4]$, $[2, 4]$, $[3, 4]$, $[4, 4]$, $[5, 4]$.
   (d) TAS: $e = 7 \mid \frac{13}{5} \mid -\frac{1}{5}$, $d = 5 \mid \frac{11}{3} \mid \frac{1}{3}$, $c = 5 \mid \frac{9}{2} \mid -\frac{1}{2}$.
   (e) TAS: $e = 7 \mid \frac{12}{5} \mid -\frac{1}{5}$, $d = 5 \mid 4 \mid \frac{1}{3}$, $c = 5 \mid 4 \mid -\frac{1}{2}$.
   (f) $y = 5$

   $i = 5:$ (a) TAS: $e = 7 \mid \frac{12}{5} \mid -\frac{1}{5}$, $d = 5 \mid 4 \mid \frac{1}{3}$, $c = 5 \mid 4 \mid -\frac{1}{2}$, $g = 7 \mid \frac{1}{3} \mid \frac{1}{3}$.
   (b) TAS: $g = 7 \mid \frac{1}{3} \mid \frac{1}{3}$, $e = 7 \mid \frac{12}{5} \mid -\frac{1}{5}$, $d = 5 \mid 4 \mid \frac{1}{3}$, $c = 5 \mid 4 \mid -\frac{1}{2}$.
   (c) Vykreslia sa úseky $[\frac{1}{3}, 5]$ až $[\frac{12}{5}, 5]$ a $[4, 5]$ až $[4, 5]$, t.j. pixle $[0, 5]$,$[1, 5]$, $[2, 5]$, $[4, 5]$.
   (d) TAS: $g = 7 \mid \frac{1}{3} \mid \frac{1}{3}$, $e = 7 \mid \frac{12}{5} \mid -\frac{1}{5}$.

(e) TAS: $g = 7 \mid \frac{2}{3} \mid \frac{1}{3}$, $e = 7 \mid \frac{11}{5} \mid -\frac{1}{5}$.

(f) $y = 6$

$i = 6$: (a) TAS sa nemení.

(b) TAS sa po usporiadaní nezmení.

(c) Vykreslí sa úsek $[\frac{2}{3}, 6]$ až $[\frac{11}{5}, 6]$, t.j. pixle $[1, 6]$, $[2, 6]$.

(d) Z TAS sa žiadna hrana nezruší.

(e) TAS: $g = 7 \mid 1 \mid \frac{1}{3}$, $e = 7 \mid 2 \mid -\frac{1}{5}$.

(f) $y = 7$

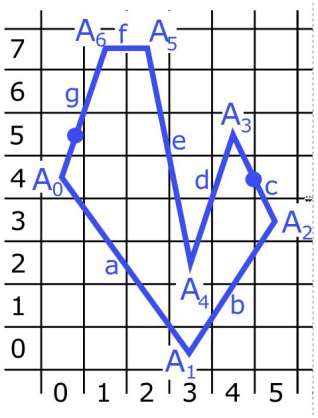
$i = 7$: (a) TAS sa nemení.

(b) TAS sa po usporiadaní nezmení.

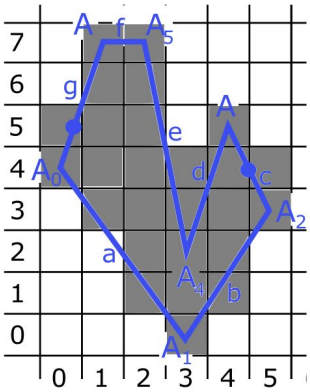
(c) Vykreslí sa úsek $[1, 7]$ až $[2, 7]$, t.j. pixle $[1, 7]$, $[2, 7]$.

(d) TAS = $\emptyset$.

(e) $y = y_{max}$, algoritmus končí. Výsledok je zobrazený na Obr. 41.



Obr. 40: Príklad mnohouholníka



Obr. 41: Výsledný vyplnený mnohouholník

Pseudokód

Pseudokód Scan-line algoritmu môžeme nájsť na strane 122 v literatúre [3]. Pre jeho veľký rozsah a zložitosť ho tu neuvádzame.

Literatúra

Algoritmus Scan-line vieme nájsť podrobne spracovaný v [3] na strane 117 ako aj v slovenskej literatúre [1] na strane 70 a [2]. Je v krátkosti spomenutý aj v kapitole o rasterizovaní polygónov v [7].

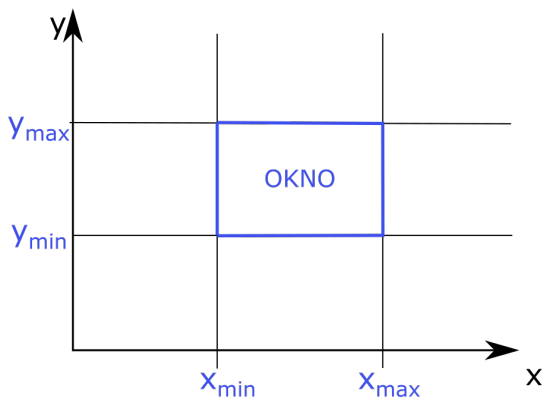
5 Orezávanie

5.1 Algoritmy na orezávanie bodu

Predpokladáme, že máme bod $A = (x_A, y_A)$ a okno, vzhľadom na ktoré orezávame je dané svojimi min-max súradnicami $x_{min}, x_{max}, y_{min}, y_{max}$.

Ak bod spĺňa všetky nerovnosti nižšie, bod leží v okne a vykreslíme ho. Inak tento bod zahadzujeme.

$$\begin{aligned} x_{min} \leq x_A \leq x_{max} \\ y_{min} \leq y_A \leq y_{max} \end{aligned} \tag{5.1}$$



Obr. 42: Orezávanie danej úsečky

5.2 Algoritmy na orezávanie úsečky do axiálneho okna

Orezávanie bodu je pomerne jednoduché. Na zložitejšie objekty však potrebujeme viac operácií, a preto sa snažíme ich počet minimalizovať. Tak je to aj s úsečkou vo všeobecnej polohe. Najprv otestujeme jej krajné body, a ak patria do okna, tak aj úsečka leží v okne.

V prípade, že úsečka neleží v okne, otestujeme jej prienik s oknom. Ak úsečka pretína okno, tak ju orezávame, a na to potrebujeme zistiť body prieniku (t.j. kde úsečka pretne hranice okna).

Podstatnou vlastnosťou pri orezávaní je rýchlosť algoritmu na nájdenie bodov prieniku (pri vykresľovaní scény môžeme mať rádovo tisíce objektov, ktoré treba orezať, za čo najkratší čas). Preto vzniklo niekoľko algoritmov na orezávanie. Najprv ale musíme urobiť testy, či naozaj treba hľadať body prieniku. Úsečky, ktoré ležia v okne, orezávať netreba, tie môžeme rovno vykresliť. Ak oba krajné body úsečky ležia nad oknom, t.j. majú súradnicu $y > y_{max}$ okna, tak úsečku nevykreslíme. Podobne ak oba krajné body úsečky ležia pod oknom, naľavo či napravo od okna, tak úsečku nevykreslíme.

V tejto kapitole si predstavíme jeden algoritmus na orezávanie úsečky.

5.2.1 Algoritmus Cohen-Sutherland

Princíp algoritmu

Algoritmus Cohen-Sutherland slúži na orezávanie úsečky do axiálneho okna, ktoré má strany rovnobežné so súradnicovými osami. Algoritmus prebieha počas behu algoritmu a minimalizuje počet orezávaní.

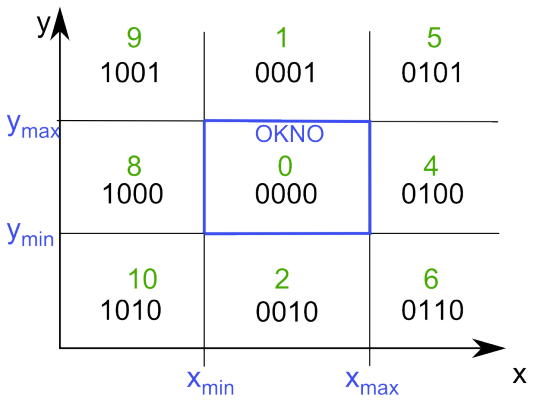
Prvá časť tohto algoritmu slúži na vylúčenie úsečiek, u ktorých netreba počítať prienik s oknom. Takáto úsečka sa buď nachádza celá v okne alebo mimo okna.

Najprv rozdelíme rovinu na deväť častí hraničnými priamkami okna. Súradnice (x_{min}, x_{max}, y_{min}, y_{max}) definujú okno, vzhľadom na ktoré budeme orezávať úsečky. Každej časti priradíme 4-bitový kód, ktorý dostane aj každý bod úsečky, ktorý do tejto oblasti padne (Obr. 43).

Jednotlivé bity pre bod (x, y) sa nastavia nasledovne:

- 1. bit – bod leží naľavo od okna ($x_{min} > x$)
- 2. bit – bod leží napravo od okna ($x_{max} < x$)
- 3. bit – bod leží nižšie od okna ($y_{min} > y$)
- 4. bit – bod leží vyššie od okna ($y_{max} < y$)

Preto napríklad bod s kódom 0101 leží napravo a vyššie od okna.

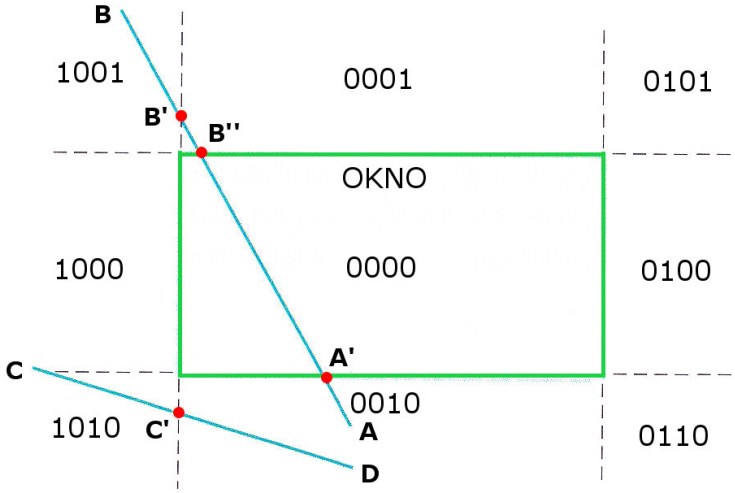


Obr. 43: Kódy oblastí určené oknom

Bod, ktorý sa nachádza vo vnútri okna, má kód 0000. Úsečka je v okne celá a môžeme ju vykresliť, ak oba jej krajné body majú kód 0000. Tento prípad nastane, ak kod(A)||kod(B) = 0000, kde A a B sú krajné body orezávanej úsečky a symbol || predstavuje binárne sčítanie. Takýto test budeme nazývať **Trivial accept**.

Ďalej môžeme vylúčiť úsečku bez vykreslenia, ak oba jej krajné body sú vľavo, vpravo, hore alebo dole od okna. Ľahko to identifikujeme práve podľa kódov krajných bodov úsečky tak, že urobíme ich logický súčin ((1 and 1)=1, inak 0). Teda kod(A)&& kod(B) $\neq$ 0000. Takýto test budeme označovať pojmom **Trivial reject**.

Ak je logický súčin rôzny od 0000, tak úsečka nepretína okno, a teda ju môžeme vylúčiť z vykresľovania. V opačnom prípade úsečku nemôžeme vylúčiť z vykresľovania a musíme ju otestovať na prienik vzhľadom na hranicu okna, v najhoršom prípade 4-krát.



Obr. 44: Orezávanie danej úsečky

Myšlienku orezávania úsečky AB vzhľadom na axiálne okno môžeme vidieť na Obr. 44.

Vyberieme bod mimo okna, napr. bod A. Bitový kód bodu A je 0010, preto má zmysel orezávať úsečku AB priamkou prechádzajúcou dolnou hranou okna ($y = y_{min}$). Bod prieniku úsečky a okna označíme ako $A^{'}$. Ten predstavuje nový krajný bod orezávanej úsečky, teda úsečku $A^{'}B$. Bod $A^{'}$ sa teraz nachádza na spodnej hrane okna, teda jeho bitový kód predstavuje 0000.

Prechádzame na bod B, druhý krajný bod orezávanej úsečky. Z jeho bitového kódu vidíme, že sa nachádza vľavo od okna. Orezávame podľa priamky $x = x_{min}$ a nachádzame bod $B^{'}$. Zahadzujeme pôvodný bod B a nahradíme ho bodom $B^{'}$. Dostávame úsečku $A^{'}B^{'}$. Z bitového kódu bodu $B^{'}$ vidíme, že sa nachádza nad oknom a preto ho orezávame priamkou $y = y_{max}$. Nachádzame bod $B^{''}$, ktorého bitový kód je 0000. Úspešne sme orezali úsečku AB.

V prípade úsečky CD po prvom orezávacom kroku zistíme, že úsečka sa celá nachádza pod/vedľa axiálneho okna, preto ju celú zahadzujeme.

Matematicky je možno vyjadriť prienik orezávanej úsečky s priamkami prechádzajúcimi hranami axiálneho okna pomocou smernicovej rovnice priamky (3.1).

- Orezávanie **zľava** priamkou $x = x_{min}$ (Obr 45b):

$$x'_B = x_B + \Delta x_B = x_B + (x_{min} - x_B) = x_{min},$$
$$y'_B = y_B + \Delta y_B = y_B + \Delta x_B \mid m \mid = y_B + \mid m \mid (x_{min} - x_B)$$

- Orezávanie **sprava** priamkou $x = x_{max}$ (Obr 45a):

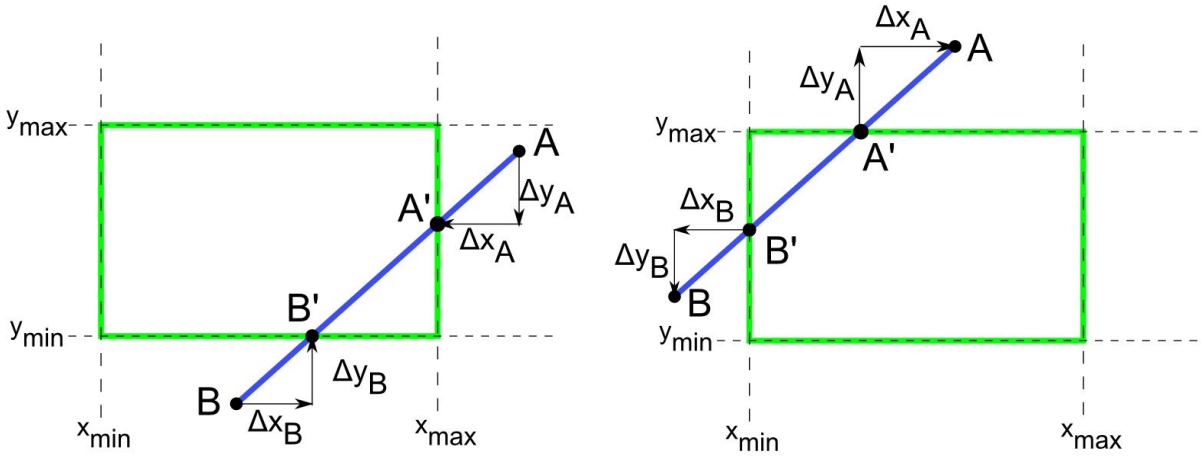
$$x'_A = x_A + \Delta x_A = x_A + (x_{max} - x_A) = x_{max},$$
$$y'_A = y_A + \Delta y_A = y_A + \Delta x_A \mid m \mid = y_A + \mid m \mid (x_{max} - x_A)$$

- Orezávanie **zdola** priamkou $y = y_{min}$ (Obr 45a):

$$x'_B = x_B + \Delta x_B = x_B + \frac{1}{\mid m \mid}\Delta y_B = x_B + \frac{1}{\mid m \mid}(y_{min} - y_B),$$
$$y'_B = y_B + \Delta y_B = y_B + (y_{min} - y_B) = y_{min}$$

- Orezávanie **zhora** priamkou $y = y_{max}$ (Obr 45b):

$$x'_A = x_A + \Delta x_A = x_A + \frac{1}{\mid m \mid}\Delta y_A = x_A + \frac{1}{\mid m \mid}(y_{max} - y_A),$$
$$y'_A = y_A + \Delta y_A = y_A + (y_{max} - y_A) = y_{max}$$



(a) Orezávanie sprava a zdola

(b) Orezávanie zľava a zhora

Obr. 45: Orezávanie

Algoritmus Cohen-Sutherland v danej verzii nevie orezať úsečku rovnobežnú zo súradnicovou osou y podľa axiálneho okna, pretože pre smernicu m takejto úsečky platí $m = \frac{\Delta y}{\Delta x}$, kde $\Delta x = 0$.

Postup

Postup orezávania úsečky s krajnými bodmi $A = (x_A, y_A)$ a $B = (x_B, y_B)$ do okna určeného bodmi $M = (x_M, y_M)$ a $N = (x_N, y_N)$.

1. Zisti bitový kód bodu A kod(A) a bitový kod bodu B kod(B).
2. Otestuj Trivial accept bodov A a B. Ak vyšlo true vykresli celu úsečku, ak false pokračuj.
3. Otestuj Trivial reject bodov A a B. Ak vyšlo true zahoď celu úsečku, ak false pokračuj.
4. Vypočítaj smernicu m úsečky AB ako $m = \frac{y_B - y_A}{x_B - x_A}$ a konštanty $x_{min} = min(x_A, x_B)$, $x_{max} = max(x_A, x_B)$, $y_{min} = min(y_A, y_B)$ a $y_{max} = max(y_A, y_B)$.
5. Vyber krajný bod, ktorý sa nenachádza vnútri axiálneho okna, nech je to bod A.
6. Kým $kod1$ bodu A sa nerovná 0000 postupuj:
   (a) Orezanie zľava do bodu $A' = (x'_A, y'_A)$: $x'_A = x_{min}$ a $y'_A = y_A + \mid m \mid (x_{min} - x_A)$. Zahoď úsečku AA', teda $A = A'$
   (b) Orezanie sprava do bodu $A' = (x'_A, y'_A)$: $x'_A = x_{max}$ a $y'_A = y_A + \mid m \mid (x_{max} - x_A)$. Zahoď úsečku AA', teda $A = A'$
   (c) Orezanie zdola do bodu $A' = (x'_A, y'_A)$: $x'_A = x_A + \dfrac{1}{\mid m \mid}(y_{min} - y_A)$ a $y_A = y_{min}$. Zahoď úsečku AA', teda $A = A'$
   (d) Orezanie zhora do bodu $A' = (x'_A, y'_A)$: $x'_A = x_A + \dfrac{1}{\mid m \mid}(y_{max} - y_A)$ a $y_A = y_{max}$. Zahoď úsečku AA', teda $A = A'$
7. Vymeň bod A a bod B a opakuj bod 6.

Príklad

Na ilustráciu algoritmu Cohen-Sutherland si ukážeme orezanie úsečky s krajnými bodmi $A = (x_A, y_A) = (0, 0)$ a $B = (x_B, y_B) = (8, 7)$ do axiálneho okna určeného bodmi $M = (x_M, y_M) = (2, 1)$ a $N = (x_N, y_N) = (6, 5)$.

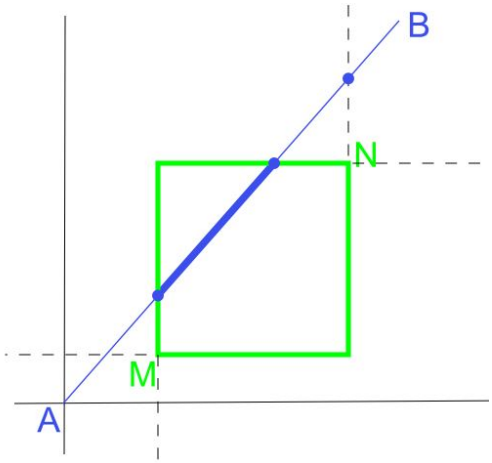
1. Bitový kód bodu A kod(A)=1010 a bitový kod bodu B kod(B)=0101.
2. Trivial accept(A, B) = false.
3. Trivial reject(A, B) = false.
4. $m = \frac{7}{8}$, $x_{min} = min(2, 6) = 2$, $x_{max} = max(2, 6) = 6$, $y_{min} = min(1, 5) = 1$ a $y_{max} = max(1, 5) = 5$.
5. Vyber krajný bod, nech je to bod A.
6. Kým kod(A) bodu A sa nerovná 0000 postupuj:

   zľava: $x'_A = x_{min} = 1$ a $y'_A = 0 + \frac{7}{8}(2 - 0)$. Teda $A = (x'_A, y'_A) = (2, \frac{7}{4})$ a kod(A)=0010.
7. Vymeň bod A a bod B, teda $A = (8, 7)$ a $B = (2, \frac{7}{4})$. V tomto prípade kod(A)=0101.

   sprava: $x'_A = x_{max} = 6$ a $y'_A = 7 + \dfrac{7}{8}(6 - 8)$. Teda $A = (x'_A, y'_A) = (6, \frac{21}{4})$ a kod(A)=0001.

   zhora: $x'_A = 6 + \dfrac{8}{7}(5 - \frac{21}{4}) = \frac{40}{7}$ a $y_A = 5$. Teda $A = (x'_A, y'_A) = (\frac{40}{7}, 5)$ a kod(A)=0000.

8. Algoritmus končí. Krajné body orezanej úsečky sú $A = (\frac{40}{7}, 5)$, $B = (2, \frac{7}{4})$. Výslednú orezanú úsečku môžete vidieť na Obr 46.



Obr. 46: Výsledná orezaná úsečka algoritmom Cohen-Sutherland

Pseudokód

Implementácia algoritmu Cohen-Sutherland je zhrnutá v nasledujúcom pseudokóde.

Na vstupe sú dva krajné body úsečky $A = (x_A, y_A)$ a $B = (x_B, y_B)$. Súradnice (x_{min}, x_{max}, y_{min}, y_{max}) definujú okno, podľa ktorého orezávame úsečku. Procedúra $ZistiKod(x, y, kod)$, priradí premennej kod 4-bitový kód bodu (x, y).

Procedúra TrivialAccept(kod1, kod2) porovná bitové kódy dvoch rôznych bodov kod1 a kod2. Vráti hodnotu true, ak sa oba body nachádzajú vnútri okna a false ak nie.

Procedúra TrivialReject(kod1, kod2) porovná bitové kódy dvoch rôznych bodov kod1 a kod2. Vráti hodnotu true, ak sa oba body nachádzajú vľavo/vpravo/zľava alebo sprava od okna a false ak nie.

```
Cohen-Sutherland (int x_A, int y_A, int x_B, int y_B)
{
  ZistiKod(x_1, y_1, kod1)
  ZistiKod(x_2, y_2, kod2)
  while(kod≠0000) {
    if (TrivialAccept(kod1, kod2) == true) koniec;
    elseif (TrivialReject(kod1, kod2) == true) koniec;
    else {
      if(kod1 ≠ 0000) KodVonku=kod1;
      else KodVonku=kod2;
      if (KodVonku & HORE) { /bod sa nachádza nad oknom
        x = x_A + (x_B − x_A) * (y_max − y_A)/(y_B − y_A);
        y = y_max;
      } else if (KodVonku & DOLU) { /bod sa nachádza pod oknom
        x = x_A + (x_B − x_A) * (y_min − y_A)/(y_B − y_A);
        y = y_min;
      } else if (KodVonku & VPRAVO) { /bod sa nachádza vpravo
      od okna
```

```
      y = y_A + (y_B - y_A) * (x_max - x_A)/(x_B - x_A);
      x = x_max;
    } else if (KodVonku & VLAVO) { /bod sa nachádza vl'avo
    od okna
      y = y_A + (y_B - y_A) * (x_min - x_A)/(x_B - x_A);
      x = x_min;
    }
    // Presunieme sa na druhý koncový bod úsečky, ktorý
    sa nachádza mimo okna
    if (KodVonku=kod1){
      x_A = x;
      y_A = y;

    } else {
      x_B = x;
      y_B = y;

    }
    KodVonku=kod2
  }
 }
}
```

Literatúra

Algoritmus Cohen-Sutherland vieme nájsť podrobne spracovaný v anglickej literatúre [3] od strany 226. S jeho vysvetlením sa môžeme stretnúť aj v [1] o strany 26 a v českej literatúre [2] sú tomuto algoritmu venované dve strany 106-107.

5.3 Algoritmy na orezávanie úsečky mnohouholníkom

5.3.1 Algoritmus Liang-Barsky

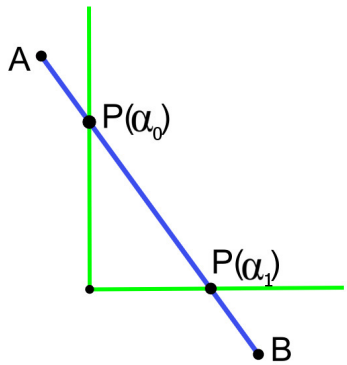
Princíp algoritmu

Algoritmus Liang-Barsky slúži na orezávanie úsečky do konvexného mnohouholníka. Bod na úsečke AB je reprezentovaný pomocou parametrického vyjadrenia úsečky,

$$P(\alpha) = (1 - \alpha)A + \alpha B, \tag{5.2}$$

kde $0 \leq \alpha \leq 1$.

Algoritmus počíta dve hodnoty parametra α_0, α_1, ktorým prislúcha výsledná orezaná úsečka $P(\alpha_0)P(\alpha_1)$, kde $\alpha_0 < \alpha_1$ (Obr 47).

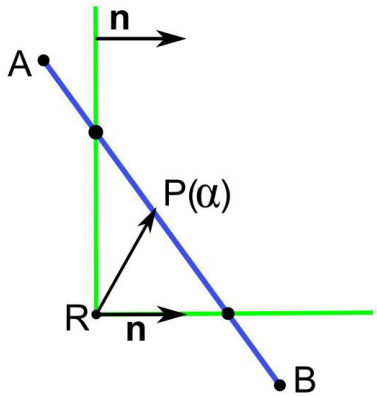


Obr. 47: Priesečníky úsečky a okna

Na začiatku algoritmu sa inicializujú hodnoty ako $\alpha_0 = 0$ a $\alpha_1 = 1$, teda $P(\alpha_0)P(\alpha_1) = AB$.

K orezávaniu úsečky pristupujeme postupne vzhľadom na polroviny mnohouholníka. Skúmame, ako oreže úsečku polrovina určená $< R, \mathbf{n} >$, kde *n* je normálový vektor hranice smerujúci dovnútra polroviny (Obr 48).

Počas priebehu algoritmu bude hodnota α_0 rásť (neklesať) a hodnota α_1 klesať (nerásť). Ak $\alpha_0 > \alpha_1$, tak orezávaná úsečka je prázdna a algoritmus končí.



Obr. 48: Polrovina určená bodom R a vektorom **n** [6]

Potrebujeme zistiť hodnotu α (ak existuje), pre ktorú priamka, na ktorej leží úsečka, pretína hranicu polroviny. Presnejšie potrebujeme nájsť podmienku, aby $P(\alpha)$ patrí polrovine. Aby sme takéto α vypočítali, dosadíme $P(\alpha)$ do podmienky patriť polrovine:

$$\begin{aligned}
(P(\alpha) - R).\mathbf{n} &\geq 0, \\
((1-\alpha)A + \alpha B) - R).\mathbf{n} &\geq 0, \\
(A + \alpha(B - A)) - R).\mathbf{n} &\geq 0, \\
\alpha(B - A).\mathbf{n} + (A - R).\mathbf{n} &\geq 0, \\
\alpha(B - A).\mathbf{n} &\geq (R - A).\mathbf{n}, \\
\alpha d_l &\geq d_r,
\end{aligned} \tag{5.3}$$

kde $d_l = (B - A).\mathbf{n}$ a $d_r = (R - A).\mathbf{n}$ (Obr 48).

Potom existuje niekoľko prípadov v závislosti od hodnoty d_l:

1. $d_l > 0$: Potom $\alpha \geq \frac{d_r}{d_l}$. Aby sme to zabezpečili, stačí položiť: $\alpha_0 = max(\alpha_0, \frac{d_r}{d_l})$.

   Ak je $\alpha_0 > \alpha_1$ ako výsledok dostaneme, že orezávaná úsečka je prázdna a algoritmus končí.

2. $d_l < 0$: Potom $\alpha \leq \frac{d_r}{d_l}$. Aby sme to zabezpečili, stačí položiť: $\alpha_1 = min(\alpha_1, \frac{d_r}{d_l})$.

   Ak ako výsledok dostaneme $\alpha_0 > \alpha_1$, tak opäť orezávaná úsečka je prázdna a algoritmus končí.

3. $d_l = 0$: Potom α nie je definované. Geometricky to znamená, že hraničná priamka a orezávaná úsečka sú rovnobežné. V tomto prípade stačí vziať ľubovoľný bod na úsečke napr. A a ak bude $(A - R).\mathbf{n} < 0$, tak bude jasné, že celá úsečka je mimo polroviny, a preto prienik s mnohouholníkom je prázdny. Inak neurobíme nič, čiže pokračujeme v orezávaní ďalšími polrovinami.

Vo všeobecnosti je algoritmus Liang-Barsky efektívnejší ako algoritmus Cohen-Sutherland, pretože netreba počítať prienik úsečky a okna. Každá aktualizácia parametru u vyžaduje iba operáciu delenia a prienik úsečky a okna sa počíta iba jedenkrát. Taktiež Cohen-Sutherland môže hľadať prienik okna s úsečkou aj keď tá sa nachádza mimo okna. Oba tieto algoritmy je možné rozšíriť do trojdimenzionálneho priestoru.

Postup

Na ilustráciu algoritmu Liang-Barsky si ukážeme orezanie úsečky s krajnými bodmi $A = (x_A, y_A)$ a $B = (x_B, y_B)$. Je dané okno, kde $x \in < x_{min}, x_{max} >$ a $y \in < y_{min}, y_{max} >$. Určenie polrovín si vyžaduje voľbu minimálne dvoch bodov, $R_0 = (x_{min}, y_{min})$ a $R_1 = (x_{max}, y_{max})$ a dvoch vektorov $\boldsymbol{e_x} = (1, 0)$ a $\boldsymbol{e_y} = (0, 1)$.

1. Vytvoríme nasledujúce štyri polroviny:

$$H_1 = \langle R_0, \boldsymbol{e_x} = (1,0)\rangle; \qquad H_3 = \langle R_1, -\boldsymbol{e_x} = (-1,0)\rangle$$
$$H_2 = \langle R_0, \boldsymbol{e_y} = (0,1)\rangle \qquad H_4 = \langle R_1, -\boldsymbol{e_y} = (0,-1)\rangle$$

2. Inicializácia $\alpha_0 = 0$ a $\alpha_1 = 1$.

3. Pre každú polrovinu $H = \langle R, \boldsymbol{n}\rangle$;

   (a) Vypočítaj $d_l = (B - A).\mathbf{n}$ a $d_r = (R - A).\mathbf{n}$

   (b) Ak $d_l > 0$, tak $\alpha \geq \frac{d_r}{d_l}$ a $\alpha_0 = max(\alpha_0, \frac{d_r}{d_l})$.

   (c) Ak $d_l < 0$, tak $\alpha \leq \frac{d_r}{d_l}$ a $\alpha_1 = min(\alpha_1, \frac{d_r}{d_l})$.

   (d) Ak $d_l = 0$, tak α nie je definované. Ak $(A-R).\mathbf{n} < 0$, a úsečka je mimo polroviny, a preto prienik s mnohouholníkom je prázdny. Inak pokračujeme v orezávaní ďalšími polrovinami.

   (e) Ak je $\alpha_0 > \alpha_1$ ako výsledok dostaneme, že orezávaná úsečka je prázdna a algoritmus končí

4. Krajné body orezanej úsečky sú:

   - $A = P(\alpha_0) = (1 - \alpha_0)A + \alpha_0 B$
   - $B = P(\alpha_1) = (1 - \alpha_1)A + \alpha_1 B$

Príklad

Na ilustráciu algoritmu Liang-Barsky si ukážeme orezanie úsečky s krajnými bodmi $A = (x_A, y_A) = (0, 8)$ a $B = (x_B, y_B) = (16, 0)$.

Je dané okno, kde $x \in < 4, 10 >$ a $y \in < 2, 9 >$. Určenie polrovín si vyžaduje voľbu minimálne dvoch bodov, $R_0 = (4, 2)$ a $R_1 = (10, 9)$ a dvoch vektorov $\boldsymbol{e_x} = (1, 0)$ a $\boldsymbol{e_y} = (0, 1)$.

1. Vytvoríme nasledujúce štyri polroviny:

$$H_1 = \langle R_0 = (4,2,), \boldsymbol{e_x} = (1,0)\rangle; \qquad H_3 = \langle R_1 = (10,9), -\boldsymbol{e_x} = (-1,0)\rangle$$
$$H_2 = \langle R_0 = (4,2,), \boldsymbol{e_y} = (0,1)\rangle \qquad H_4 = \langle R_1 = (10,9), -\boldsymbol{e_y} = (0,-1)\rangle$$

2. Inicializácia $\alpha_0 = 0$ a $\alpha_1 = 1$.

3. $H_1 = \langle R_0, \boldsymbol{e_x}\rangle$:
   $d_l = (B - A).\boldsymbol{e_x} = (16, -8).(1, 0) = 16 > 0$
   $d_r = (R_0 - A)\boldsymbol{e_x} = (4, -6).(1, 0) = 4 \Rightarrow \alpha \geq \frac{1}{4}$
   $d_l > 0 : \alpha_0 = max(0, \frac{1}{4}) = \frac{1}{4}$
   Čiastkový výsledok: $(\alpha_0 = \frac{1}{4}, \alpha_1 = 1)$

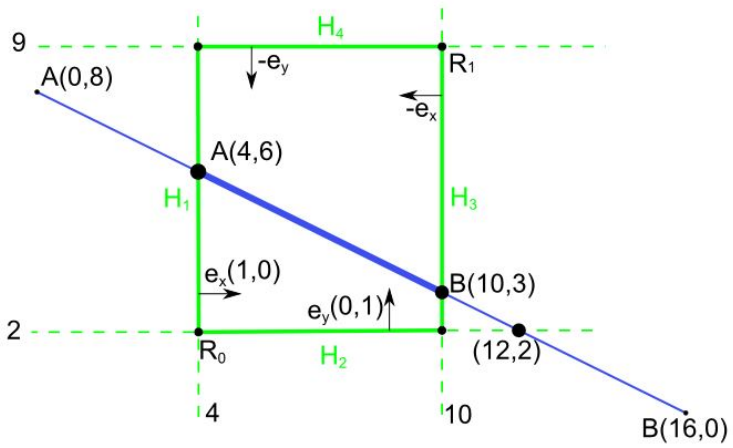
4. $H_2 = \langle R_0, \boldsymbol{e_y}\rangle$:
   $d_l = (B - A).\boldsymbol{e_y} = (16, -8).(0, 1) = -8 < 0$
   $d_r = (R_0 - A)\boldsymbol{e_y} = (4, -6).(0, 1) = -6 \Rightarrow \alpha \leq \frac{3}{4}$
   $d_l < 0 : \alpha_1 = min(1, \frac{3}{4}) = \frac{3}{4}$
   Čiastkový výsledok: $(\alpha_0 = \frac{1}{4}, \alpha_1 = \frac{3}{4})$

5. $H_3 = \langle R_1, -\boldsymbol{e_x} \rangle$:
   $d_l = (B - A).-\boldsymbol{e_x} = (16, -8).(-1, 0) = -16 < 0$
   $d_r = (R_1 - A).-\boldsymbol{e_x} = (10, 1).(-1, 0) = -10 \Rightarrow \alpha \leq \frac{5}{8}$
   $d_l < 0 : \alpha_1 = min(\frac{3}{4}, \frac{5}{8}) = \frac{5}{8}$
   Čiastkový výsledok: $(\alpha_0 = \frac{1}{4}, \alpha_1 = \frac{5}{8})$

6. $H_4 = \langle R_1, -\boldsymbol{e_y} \rangle$:
   $d_l = (B - A).-\boldsymbol{e_y} = (16, -8).(0, -1) = 8 > 0$
   $d_r = (R_1 - A).-\boldsymbol{e_y} = (10, 1).(0, -1) = -1 \Rightarrow \alpha \geq -\frac{1}{8}$
   $d_l > 0 : \alpha_0 = max(\frac{1}{4}, -\frac{1}{8}) = \frac{1}{4}$

7. Výsledok je $(\alpha_0 = \frac{1}{4}, \alpha_1 = \frac{5}{8})$. Teda krajné body orezanej úsečky sú:
   - Pre $\alpha_0 = \frac{1}{4} : \frac{3}{4}(0, 8) + \frac{1}{4}(16, 0) = (4, 6) = A$
   - Pre $\alpha_1 = \frac{5}{8} : \frac{3}{8}(0, 8) + \frac{5}{8}(16, 0) = (10, 3) = B$

8. Výslednú orezanú úsečku môžete vidieť na Obr 49.



Obr. 49: Výsledná orezaná úsečka algoritmom Cohen-Sutherland

Literatúra

Algoritmus Liang-Barsky vieme nájsť podrobne spracovaný v anglickej literatúre [3] od strany 230. Zo slovenskej a českej literatúry sa nenachádza v [1] ani v [2].

5.3.2 Algoritmus Cyrus-Beck

Princíp algoritmu

Staršia verzia Liang-Barského algoritmu je známa ako algoritmus Cyrus-Beck. Je to rýchlejšia metóda ako Algoritmus Cohen-Sutherland.

Je to orezávanie do konkrétneho okna, no nie nutne axiálneho.

Je založený na hľadaní tzv. vstupného bodu úsečky AB do okna, ktorý je určený maximálnym vstupným parametrom t_e a výstupného bodu úsečky AB z okna, ktorý je určený zasa minimálnym výstupným parametrom t_o v intervale $< 0, 1 >$. Predpokladá sa, že úsečka AB je určená parametrickým vyjadrením:

$$P(t) = A + dt, \tag{5.4}$$

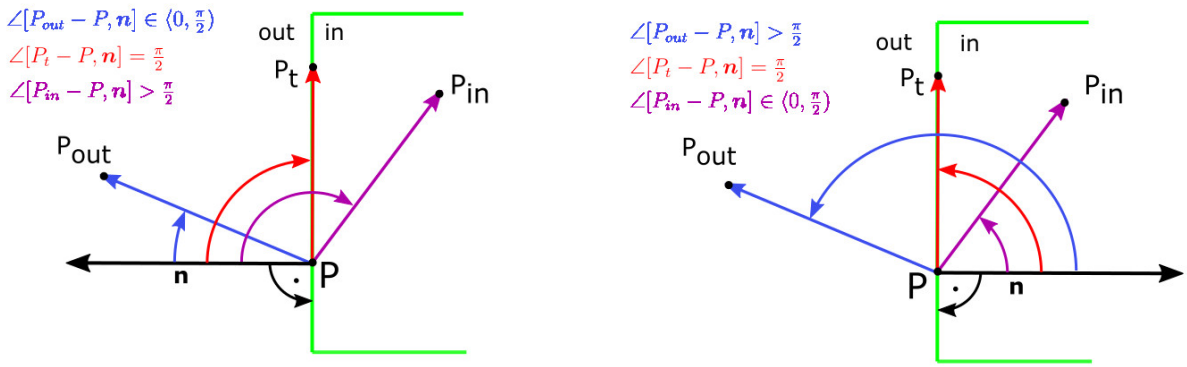
kde vektor $s = B - A$ a $t \in < 0, 1 >$ a chceme určiť jej prienik so stranou E (edge) konvexného mnohouholníka určenou bodom P a vonkajším normálovým vektorom **n**.

Vo všeobecnosti pre uhol vektora a vonkajší normálový vektor strany mnohouholníka platí:

- $\angle[P_{out} - P, \mathbf{n}] \in \langle 0, \frac{\pi}{2})$
- $\angle[P_{in} - P, \mathbf{n}] > \frac{\pi}{2}$
- $\angle[P_t - P, \mathbf{n}] = \frac{\pi}{2}$

a pre uhol vektora a vnútorný normálový vektor strany mnohouholníka platí:

- $\angle[P_{out} - P, \mathbf{n}] > \frac{\pi}{2}$
- $\angle[P_{in} - P, \mathbf{n}] \in \langle 0, \frac{\pi}{2})$
- $\angle[P_t - P, \mathbf{n}] = \frac{\pi}{2}$



(a) Vonkajší normálový vektor

(b) Vnútorný normálový vektor

Obr. 50: Orezávanie s použitím normálového vektora

Platí:

1. $P(t)$ je vonkajším bodom, ak $\angle[P(t) - P, \boldsymbol{n}] \in \langle 0, \frac{\pi}{2}) \Leftrightarrow [P(t) - P].\boldsymbol{n} > 0$
2. $P(t)$ je vnútorným bodom, ak $\angle[P(t) - P, \boldsymbol{n}] > \frac{\pi}{2} \Leftrightarrow [P(t) - P].\boldsymbol{n} < 0$

3. $P(t)$ leží na strane E, ak $\angle[P(t)-P,\boldsymbol{n}]=\frac{\pi}{2} \Leftrightarrow [P(t)-P].\boldsymbol{n}=0 \Leftrightarrow (A-P+t.\boldsymbol{d}).\boldsymbol{n}=0 \Leftrightarrow (A-P).\boldsymbol{n}+t.(\boldsymbol{d.n})=0 \Leftrightarrow \left[(\boldsymbol{d.n})\neq 0 \wedge t=-\frac{(A-P).\boldsymbol{n}}{(\boldsymbol{d.n})}\right] \vee \left[(\boldsymbol{d.n})=0 \wedge (A-P).\boldsymbol{n}=0\right] \vee \left[(\boldsymbol{d.n})=0 \wedge (A-P).\boldsymbol{n}\neq 0\right]$. Teda $t=-\frac{(A-P).\boldsymbol{n}}{(\boldsymbol{d.n})}$, kde $(\boldsymbol{d.n})\neq 0$.

Pokiaľ priamka AB obsahuje stranu mnohouholníka t.j. prienikom priamky a mnohouholníka je táto strana, čiže strana je orezaním priamky AB mnohouholníkom – algoritmus končí.

Ak priamka AB a strana E nemajú spoločný ani jeden bod, čiže priamka je so stranou E rovnobežná, pričom môže, ale nemusí pretínať ďalšie strany – prejdeme k ďalšej strane.

Môžu nastať dve situácie:

1. Ak $\boldsymbol{d.n} < 0$, nazveme ho potenciálne vstupným parametrom a označíme t_e. Bod $P(t_e)$ budeme nazývať potenciálne vstupným bodom úsečky AB do okna.

2. Ak $\boldsymbol{d.n} > 0$, nazveme ho potenciálne výstupným parametrom a označíme t_o. Bod $P(t_o)$ budeme nazývať potenciálne výstupným bodom úsečky AB z okna.

Nevýhodou tohto algoritmu je, že nevieme ošetriť prípad, keď sa úsečka nachádza celá mimo mnohouholníka.

Postup

Na ilustráciu algoritmu Cyrus-Beck si ukážeme orezanie úsečky s krajnými bodmi $A=(x_A,y_A)$ a $B=(x_B,y_B)$ do mnohouholníka určeného vrcholmi A_i.

1. Zvolíme si jednu z dvoch orientácií mnohouholníka, nech je kladná, t.j. proti smeru hodinových ručičiek. Uvažujeme vnútorné normály strán mnohouholníka.

2. Inicializácia $t_e=0, t_o=1$, urči smerový vektor orezávanej úsečky $\boldsymbol{s}=B-A$.

3. Pre každý vektor hrany $\boldsymbol{e_i}$ a jemu prislúchajúci normálový vektor $\boldsymbol{n_i}$ vykonáme:

   (a) Urči vektor $\boldsymbol{e_i}$ a normálový vektor danej hrany $\boldsymbol{n_i}$ a jeden bod ležiaci na hrane, nech je to bod A_i

   (b) Urči skalárny súčin $u=\langle \boldsymbol{s},\boldsymbol{n_i}\rangle$

   (c) Urči $t=\frac{\langle P_i-A,\boldsymbol{n_i}\rangle}{u}$

   (d) Ak $u<0$ platí $t_o=min(t_o,t)$. Ak $u>0$ platí $t_e=max(t_e,t)$.

4. Výsledná orezaná úsečka má krajné body:

   - Pre t_e: $P(t_e)=A+t_e(B-A)$
   - Pre t_o: $P(t_o)=A+t_o(B-A)$

Príklad

Na ilustráciu algoritmu Cyrus-Beck si ukážeme orezanie úsečky s krajnými bodmi $A=(x_A,y_A)=(-2,2)$ a $B=(x_B,y_B)=(8,3)$ do mnohouholníka určeného vrcholmi A_i, kde $A_1=(2,1)$, $A_2=(5,2)$, $A_3=(4,5)$ a $A_4=(2,3)$.

1. Zvolíme si jednu z dvoch orientácií mnohouholníka, nech je kladná, t.j. proti smeru hodinových ručičiek. Uvažujeme vnútorné normály strán mnohouholníka.

2. $t_e=0, t_o=1$ a $\boldsymbol{s}=(8,3)-(-2,2)=(10,1)$.

3. $i = 1:$ (a) $\boldsymbol{e_1} = A_2 - A_1 = (5,2) - (2,1) = (3,1)$, $\boldsymbol{n_1} = (-1,3)$, $P_1 = (3,1)$.
   (b) $u = \langle \boldsymbol{s}, \boldsymbol{n_1} \rangle = \langle (10,1), (-1,3) \rangle = -7$
   (c) $t = \frac{\langle P_1 - A, \boldsymbol{n_1} \rangle}{u} = \frac{\langle (5,-1),(-1,3) \rangle}{-7} = \frac{8}{7}$
   (d) $u < 0$, teda $t_o = min(1, \frac{8}{7}) = 1$.

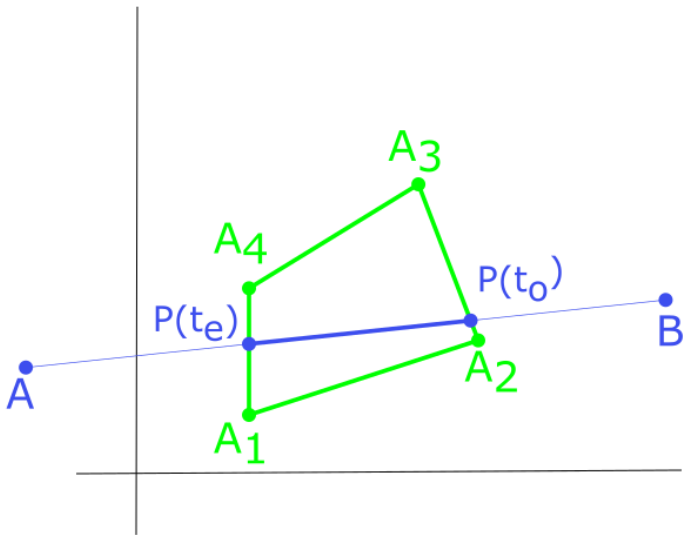
   $i = 2:$ (a) $\boldsymbol{e_2} = A_3 - A_2 = (4,5) - (5,2) = (-1,3)$, $\boldsymbol{n_2} = (-3,-1)$, $P_2 = (5,2)$.
   (b) $u = \langle \boldsymbol{s}, \boldsymbol{n_2} \rangle = \langle (10,1), (-3,-1) \rangle = -31$
   (c) $t = \frac{\langle P_2 - A, \boldsymbol{n_2} \rangle}{u} = \frac{\langle (7,0),(-3,-1) \rangle}{-31} = \frac{21}{31}$
   (d) $u < 0$, teda $t_o = min(1, \frac{21}{31}) = \frac{21}{31}$.

   $i = 3:$ (a) $\boldsymbol{e_3} = A_4 - A_3 = (2,3) - (4,5) = (-2,2)$, $\boldsymbol{n_3} = (2,-2)$, $P_3 = (4,5)$.
   (b) $u = \langle \boldsymbol{s}, \boldsymbol{n_3} \rangle = \langle (10,1), (2,-2) \rangle = 18$
   (c) $t = \frac{\langle P_3 - A, \boldsymbol{n_3} \rangle}{u} = \frac{\langle (6,3),(-3,-1) \rangle}{18} = \frac{1}{3}$
   (d) $u < 0$, teda $t_o = min(\frac{21}{31}, \frac{1}{3}) = \frac{21}{31}$.

   $i = 4:$ (a) $\boldsymbol{e_4} = A_1 - A_4 = (2,1) - (2,3) = (0,-2)$, $\boldsymbol{n_4} = (2,0)$, $P_4 = (2,3)$.
   (b) $u = \langle \boldsymbol{s}, \boldsymbol{n_4} \rangle = \langle (10,1), (2,0) \rangle = 20$
   (c) $t = \frac{\langle P_4 - A, \boldsymbol{n_4} \rangle}{u} = \frac{\langle (4,1),(2,0) \rangle}{20} = \frac{2}{5}$
   (d) $u > 0$, teda $t_e = max(0, \frac{2}{5}) = \frac{2}{5}$.

4. Krajné body orezanej úsečky:
   - Pre t_e: $P(t_e) = (-2,2) + \frac{21}{31}(10,1) = (\frac{148}{31}, \frac{83}{31})$
   - Pre t_o: $P(t_o) = (-2,2) + \frac{2}{5}(10,1) = (2, \frac{12}{5})$

5. Výslednú orezanú úsečku môžete vidieť na Obr. 51.



Obr. 51: Výsledná orezaná úsečka algoritmom Cyrus-Beck

Pseudokód

Implementácia algoritmu Cyrus-Beck je zhrnutá v nasledujúcom pseudokóde. Na vstupe sú dva krajné body úsečky A a B.

```
Cyrus-Beck (int x_A, int y_A, int x_B, int y_B)
{

  floor t_e, t_o, t;
  if (A = B) then orez-bod; {úsečka = bod, orež-bod}
  else {
    t_e = 0;
    t_o = 1;
    d = B − A;
    {pre každú hranu okna a jej vonkajší normálový vektor}
    for(i = 1; i ≤ 4; i++){
      if (n_i.d_i ≠ 0) t = (−n_i.(A−P_i)) / (n_i.d_i);
      if (n_i.d_i < 0 ∧ t ≤ 1) t_e = max(t_e, t);
      if (n_i.d_i > 0 ∧ t ≥ 0) t_o = max(t_o, t);
    }
    if (t_e > t_o) return; {úsečka mimo okna}
    else Vykresli-usecku(P(t_e), P(t_o));
}
```

Literatúra

Môžeme sa s ním stretnúť v [1] od strany 28 pod názvom parametrické orezávanie a v českej literatúre [2] môžeme nájsť vysvetlenie tohto algorimtmu od strany 107. V anglickej literatúre [3] sa tento algoritmus neuvádza.

6 Literatúra

[1] **RUŽICKÝ, E., FERKO, A.**, 1995. *Počítačová grafika a spracovanie obrazu*, Bratislava: Sapientia, 1995. ISBN 80-967180-2-9. Dostupné na internete:< http://www.sccg.sk/ ferko/PGASO2012-bookmarks.pdf>.
[2] **ŽÁRA, J. et al.** 2004. *Moderní počítačová grafika*, druhé vydání 2004. Brno: Computer Press, 2004. ISBN 80-251-0454-0
[3] **HEARN, D., BAKER, M. P.**, 1997. *Computer graphics*, C version, second edition, USA: Prentice Hall, 1997, ISBN 0-13-578634-7
[4] **HEARN, D., BAKER, M. P.**, 2004. *Computer graphics with OpenGl*, third edition, USA: Prentice Hall, 2004, ISBN 0-13-120238-3
[5] **HUGHES, J., F. et al.** 2013. *Computer Graphics Principles and Fundamentals*. third edition, USA: Addison-Wesley. 2014, ISBN 0-132-39952-8
[6] **ZAŤKO, V.**, 2014. *Poznámky z prednášok Počítačová grafika (1): 2-MPG-101*. [online]. 04/2014. [cit. 2.1.2015]. Dostupné na internete:< http://flurry.dg.fmph.uniba.sk/webog/sk/zatko-vyucba/389-pocitacova-grafika-1.html>.
[7] **WATT, A.** 2000. *3D Computer Graphics*. third edition, USA: Addison-Wesley. 2000, ISBN 0-201-39855-9
[8] **BOŽEK, M.**, 2014. *Učebné texty ku predmetu Geometria (1) – Mnohouholníky.*
[9] **FOLEY, J. D., VAN DAM, A.**, 1982. *The Fundamentals of Interactive Computer Graphics*. first edition, USA: Addison-Wesley. 1982, ISBN 0-201-14468-9
[10] **RUŽICKÝ, E.**, 1991. *Úvod do počítačovej grafiky*, Bratislava: Polygrafické stredisko UK, 1991. ISBN 80-223-0375-5