Orezávanie a prieniky

by Peter Gejguš and Marek Zimányi

Pri orezávaní v počítačovej grafike ide o to, rozhodnúť, či dane objekty ležia v zadanom okne a môžu sa cele vykresliť, alebo nie a pripadne určiť ich prienik s oknom. Pojem okna, ako obdĺžnika so stranami rovnobežnými so súradnicovými osami je používaný nielen v obrazovom ale aj modelovom priestore.

Nech je okno, do ktorého budeme orezávať objekty, dane súradnicami ľavého dolného vrcholu [x_{min}, y_{min}]a pravého horného vrcholu [x_{max}, y_{max}].

Orezávanie bodov

Orezávanie bodov do daného okna je triviálne: bod so súradnicami [x; y] leží v okne vtedy a len vtedy, keď sú súčasne splnené nerovnosti $x_{min} < x < x_{max}$ a $y_{min} < y < y_{max}$.

Orezávanie úsečiek

Pri orezávaní úsečiek môže nastať niekoľko jednoducho rozhodnuteľných situácií:

- oba koncové body úsečky ležia vnútri okna, úsečku je možné celú vykresliť
- oba koncové body ležia nad oknom, úsečka nemá s oknom prienik a vykresľovať sa nebude
- oba koncové body ležia pod oknom, úsečka nemá s oknom prienik a vykresľovať sa nebude
- oba koncové body ležia vľavo od okna, úsečka nemá s oknom prienik a vykresľovať sa nebude
- oba koncové body ležia vpravo od okna, úsečka nemá s oknom prienik a vykresľovať sa nebude

Ak nenastane ani jeden z týchto prípadov (ich overenie je možné jednoduchými nerovnosťami), je treba zistiť prienik úsečky s niektorou stranou okna (prípadne s priamkou na ktorej strana leží) a dve vzniknuté úsečky znovu preskúmať podľa podmienok uvedených vyššie.

Ako príklad určenia prieniku úsečky so stranou okna uveďme vzorec pre orezávanie zhora. Z podobnosti trojuholníkov vyplýva, že x-ová súradnica prieniku má hodnotu $x = x_1 + x_2$

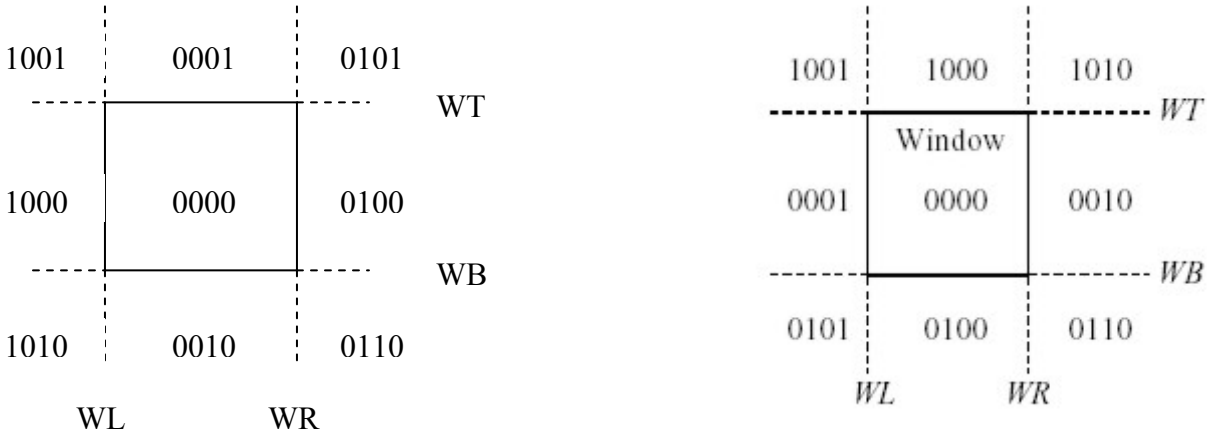
Cohen-Sutherlandov orezávací algoritmus

Uvažujme problém orezávania úsečiek, ktorých koncové body majú súradnice $P_0=(x_0,y_0)$ a $P_1=(x_1,y_1)$, vzhľadom na obdĺžnik, ktorého horná, dolná, ľavá a pravá strana sú označené WT, WB, WL a WR. Teraz si predstavíme Cohen-Sutherlandov orezávací algoritmus.

Základná myšlienka všetkých orezávacích algoritmov je, že veľa úsečiek potrebuje veľmi jednoduchú analýzu na určenie, či sú celé viditeľné alebo neviditeľné. Ak sú tieto obidva testy neúspešné, potrebujeme použiť oveľa zložitejšie algoritmy na hľadanie priesečníkov.

Pre otestovanie, či úsečka je celá viditeľná alebo neviditeľná, použijeme nasledujúcu (nie dokonalú, ale efektívnu) heuristiku. Nech koncové body úsečky budú orezané. Vypočítame 4 bitový kód pre každý z koncových bodov P_0 a P_1.Kód bodu (x , y) je definovaný nasledovne.

Bit 1: 1, ak je bod nad oknom, t.j. $y > WT$.
Bit 2: 1, ak je bod pod oknom, t.j. $y < WB$.
Bit 3: 1, ak je bod napravo od okna, t.j. x > *WR*.
Bit 4: 1, ak je bod naľavo od okna, t.j. x < *WL*.



Obr. 6.: Kódy oblastí v Cohen-Sutherlandovom algoritme

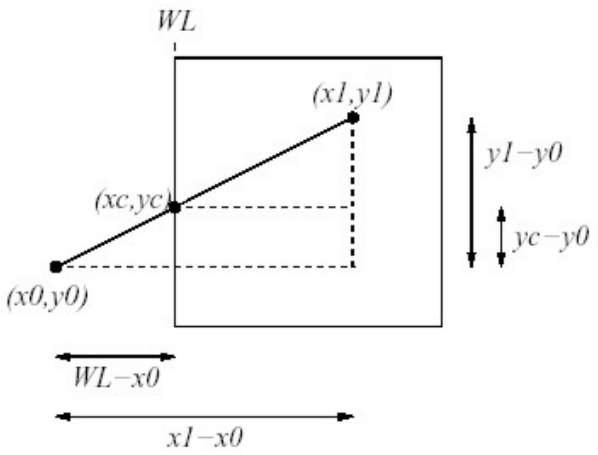
Toto rozdeľuje rovinu do 9 oblastí, čo závisí od týchto kódov, viď obrázok. Všimnime si, že úsečka je celá viditeľná vtedy a len vtedy, ak obidva kódy koncových bodov sú rovné nule. T.j. ak $C_0 \vee C_1 = 0$, potom je úsečka viditeľná a môžeme ju vykresliť. Ak obidva koncové body úsečky ležia nad, pod, napravo alebo naľavo, potom môžeme prehlásiť celú úsečku za neviditeľnú. Inými slovami, ak $C_0 \wedge C_1 \neq 0$, potom je celá úsečka neviditeľná.

Inak musíme úsečku orezať. Vieme, že jeden z kódov musí byť nenulový, predpokladajme že to je (x_0 ,x_1). (Inak vymeníme koncové body.) Teraz vieme, že niektoré bity kódu sú nenulové. Predpokladajme, že to je bit číslo 4, čo implikuje $x_0 < WL$. Môžeme usudzovať, že $x_1 > WL$, inak by sme prehlásili úsečku za neviditeľnú. Chceme určiť bod (x_c , y_c) v ktorom úsečka pretína *WL*. Je jasné, že potom

$$X_c = WL,$$

a použitím podobnosti trojuholníkov vidíme, že

$$(y_c - y_0)/(y_1 - y_0) = (WL - x_0)/(x_1 - x_0)$$



Obr. 7.: Orezávanie na ľavej strane okna

Z tohoto môžeme vyrátať y_c ako $y_c = (WL - x_0)/(x_1 - x_0)*(y_1 - y_0) + y_0$

Preto nahradíme (x_0 , y_0) s (x_c , y_c) , prepočítame hodnoty kódov a pokračujeme. Toto sa opakuje dokiaľ nie je celá úsečka jednoducho akceptovaná (všetky bity kódu sú 0) alebo dokiaľ nie je celá úsečka odmietnutá. To isté môžeme spraviť pre všetky ostatné prípady.

Algoritmus Cohen-Sutherland

```
Procedure Clipping;
begin
    1.  accept:=false;                                  {nastav – P1P2 sa nevykresľuje}
        outcod (x2, y2, cd2);                           {zistime kod bodu P2}
    2.  repeat
            outcod (x1, y1, cd1);                       {zistime kód bodu P1}
    3.      if and (cd1, cd2) !=0 then done:=true       {1.jednoduchy test–mimo okna}
            else
    4.          begin
                if (cd1=0 and cd2=0) then               {2.jednoduchy test–vnútri okna}
                                begin accept:=true;     {žiadaj vykreslenie v kroku 10.}
                                      done:=true end
                else
    5.              begin
                    if cd1=0 then swap(P1,P2);          {zameníme body, aby 1. bol von}
    6.              if cd1 & (0x0001) then          { /*cd1 in (1, 5, 9)*/orezávanie zhora}
                        begin
                        x1:=x1+(x2-x1)*(ymax-y1)/(y2-y1);
                        y1:=ymax;
                        end
    7.              else if cd1 & (0x0010) then         {/* cd1 in (2, 6, 10) */orež zdola}
                        begin
                        x1:=x1+(x2-x1)*(ymin-y1)/(y2-y1);
                        y1:=ymin;
                        end
    8.              else if cd1 & (0x0100) then         {/* cd1 in (4, 5, 6) */ orež sprava}
                        begin
                        y1:=y1+(y2-y1)*(xmax-x1)/(x2-x1);
                        x1:=xmax;
                        end
    9.              else if cd1 & (0x1000) then         {/* cd1 in (8, 9, 10)*/ orež zľava}
                        begin
                        y1:=y1+(y2-y1)*(xmin-x1)/(x2-x1);
                        x1:=xmin;
                        end
                    end
                end
            until done
    10. if accept then draw(P1,P2);                     {vykresli úsečku}
end.
```

Algoritmus postupného delenia

Pretože numericky najnáročnejšou operáciou predošlého postupu je počítanie prieniku, má praktický význam takáto modifikácia algoritmu.

Úsečku, ktorá neleží celá ani mimo okna ani celá v okne, rozdeliť na dve polovice (to je numericky rýchle) a ďalej uvažuj tieto dve úsečky. Keďže pracujeme v diskrétnej rovine je tento algoritmus konečný a pomerne efektívny.

Úlohy

1. Zistite koľko najviac prienikov je nutné počítať pri orezávaní úsečky.
2. Rozhodnite či je možné, že pri rozdelení úsečky v algoritme postupného delenia je niekedy nutné ďalej deliť obe vzniknuté úsečky.
3. A čo v ďalšom kroku? Mohol by nastať prípad, keď by nebolo možné vylúčiť ani jednu zo štyroch vniknutých úsečiek?

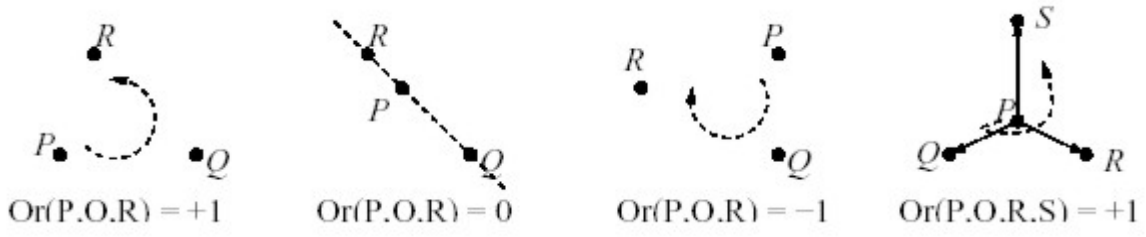
Orientácia:

Majme dve reálne čísla p a q. Sú tri možné spôsoby ich usporiadania: $p<q$, $p=q$, $p>q$. Môžeme definovať funkciu orientácie, ktorá nadobúda hodnoty +1, 0, -1 pre každý z týchto prípadov. Inak povedané, $Or_1(p, q) = sign(p - q)$, kde $sign(x)$ je –1, 0 alebo +1, v závislosti od x (t.j., či je záporné, nulové alebo kladné). Zaujímavá otázka je, či sa to dá zovšeobecniť aj pre vyššie dimenzie.

Odpoveď je áno, ale namiesto porovnávania dvoch bodov môžeme vo všeobecnosti definovať orientáciu d+1 bodov v d-rozmernom priestore. Orientáciu definujeme ako znamienko determinantu, ktorý sa skladá z homogénnych súradníc týchto bodov. Napríklad v 2-rozmernom a v 3-rozmernom priestore, orientácia troch bodov P,Q,R je definovaná nasledovne:

$$Or_2(P,Q,R) = sign\begin{vmatrix} 1 & 1 & 1 \\ p_x & q_x & r_x \\ p_y & q_y & r_y \end{vmatrix}, \qquad Or_3(P,Q,R,S) = sign\begin{vmatrix} 1 & 1 & 1 & 1 \\ p_x & q_x & r_x & s_x \\ p_y & q_y & r_y & s_y \\ p_z & q_z & r_z & s_z \end{vmatrix},$$

$$Or_2(P,Q,R) = p_x(q_y - r_y) + p_y(r_x - q_x) + q_x r_y - q_y r_x$$



Obr. 2.: Orientácia v 2 a 3-rozmernom priestore

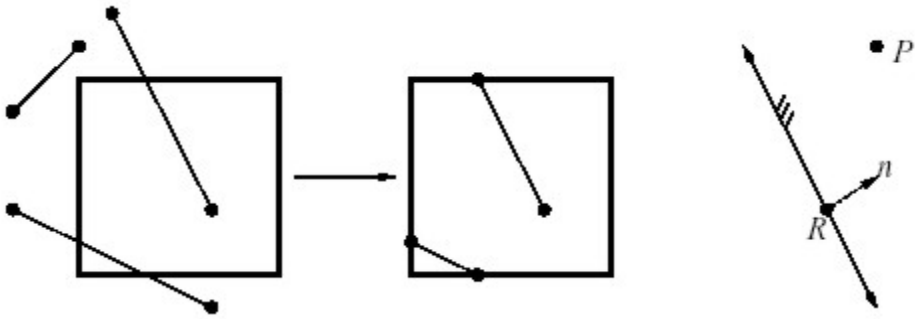
Čo znamená orientácie intuitívne? Orientácia troch bodov v rovine je +1, ak trojuholník *PQR* je orientovaný proti smeru hodinových ručičiek, -1, ak je orientovaný v smere hodinových ručičiek a 0, ak všetky tri body sú kolineárne. V 3-rozmernom priestore kladná orientácia znamená, že tieto body tvoria pravotočivú skrutkovicu, ak navštívime body v poradí *PQRS*.

Záporná orientácia znamená ľavotočivú skrutkovicu a nulová orientácia znamená, že body sú koplanárne. Všimnime si, že poradie argumentov je dôležité. Orientácia bodov (*P,Q,R*) je negáciou orientácie bodov (*P,R,Q*). Podobne ako u determinantov, výmena ľubovoľných dvoch elementov zmení znamienko orientácie. Môžeme sa opýtať prečo dávame homogénnu súradnicu na prvé miesto.

Matematik by odpovedal na túto otázku asi takto: Ak by sme ju dali na posledné miesto, tak kladne orientované body budú orientované pravotočivo v párnych dimenziách a ľavotočivo v nepárnych dimenziách. Ak je na prvom mieste, kladne orientované body majú stále pravotočivú orientáciu. Dávanie homogénnej súradnice na posledné miesto vzniklo v inžinierstve a adaptovalo sa aj do počítačovej grafiky. Ak je vám bližší inžiniersky spôsob, vypočítajte determinant (homogénna súradnica je posledná) a výsledok vynásobte –1, ak dimenzia je nepárna. Hodnota samotného determinantu je obsah rovnobežníka definovaného vektormi *Q-P* a *R-P* a preto je determinant veľmi vhodný pre výpočet obsahu. Neskôr si ukážeme inú metódu.

Priesečník úsečiek:

Orientácia je dôležitá operácia, ktorú by sme si mali zapamätať. Napríklad, ak chceme vedieť, či sa úsečky *PQ* a *RS* pretnú v rovine. Aj keď to na prvý pohľad vyzerá ako jednoduchý problém, obsahuje špeciálne prípady a úskalia. Napríklad, ak uvažujeme možnosť, že úsečky sú rovnobežné, kolineárne, alebo sa dotýkajú v bode (T-junction). Budeme uvažovať iba o regulárnych priesečníkoch, t.j. takých, že vnútra obidvoch úsečiek sa pretínajú v práve jednom bode.



Obr. 3.: Rôzne typy priesečníkov

Všimnime si, že ak ľubovoľná trojica bodov je kolineárna, potom neexistuje reguárny priesečník. Ak uvažujeme, že žiadne tri body nie sú kolineárne, potom si všimnime, že segmenty sa pretínajú vtedy a len vtedy, ak body *P* a *Q* ležia na rôznych stranách priamky $\overline{RS}$ a body *R* a *S* ležia na opačných stranách priamky $\overline{PQ}$. Môžeme to redukovať na test orientácie. Ak *R* a *S* ležia na opačných stranách priamky $\overline{PQ}$, potom $Or_2(P,Q,R)$ a $Or_2(P,Q,S)$ majú opačné znamienko, čo implikuje, že ich súčin je záporný. Jednoduchá implementácia je ukázaná nižšie.

```
bool ProperIntersect(Point P, Point Q, Point R, Point S)
{
     return (Or2(P,Q,R)*Or2(P,Q,S)<0)&&
            (Or2(R,S,P)*Or2(R,S,Q)<0);
}
```

Alg.1.: test na priesečník dvoch úsečiek

Všimnime, si že týmto máme ošetrenú aj kolinearitu bodov. Ak nejaká trojica bodov je kolineárna, potom jeden z testov orientácie vráti 0. Mimochodom, toto nie najefektívnejšie testovanie priesečníka, pretože, ak sa orientácia vyratáva osobitne, opakuje sa tam množstvo výpočtov. Čitateľ si môže premyslieť ako to prekódovať do efektívnejšej podoby.

Orezávanie úsečiek:

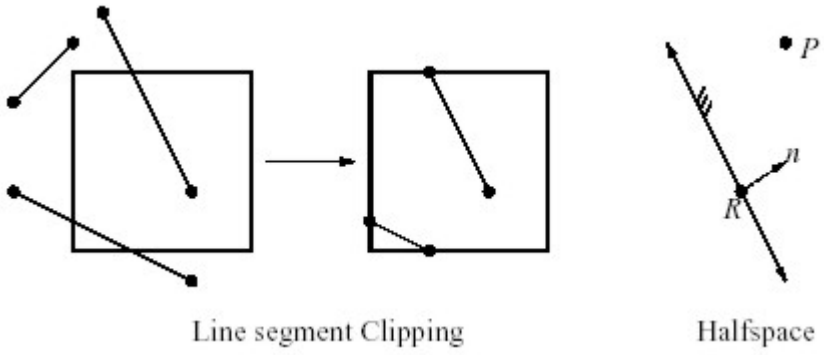
Pre demonštráciu ideí, ktoré tu boli prezentované, ukážeme si na súradniciach nezávislý algoritmus orezávania úsečiek vzhľadom na konvexný polygón v rovine. Orezávanie je proces upravovania grafický primitívov (napr. úsečiek, kružníc, vyplnených polygónov) vzhľadom na hranice nejakého okna (viď obrázok nižšie). Často sa to aplikuje v 2-rozmernom priestore na obdĺžnikové okná. Ako uvidíme neskôr, táto procedúra sa aplikuje aj na neobdĺžnikové okná v 3-rozmernom priestore, ako časť všeobecnejšieho procesu nazývaného *perspektívne orezávanie*.

Pretože tento algoritmus je dosť často používaný, je potrebné ho čo najefektívnejšie implementovať. Pre rovinné orezávanie sa to nazýva algoritmus *Liang-Barsky*. Výhoda na súradniciach nezávislého algoritmu je jeho všeobecnosť a ľahké odvodenie. Aplikuje sa na ľubovoľný typ orezávania úsečiek a pre všetky dimenzie. Budeme používať zovšeobecnenie tejto procedúry na priesečník lúča s mnohostenom v *ray shooting*.

Definujme *polrovinu* v 2-rozmernom priestore ako časť roviny ležiacu na jednej strane od priamky. Vo všeobecnosti, pre dimenziu *d*, definujeme *polpriestor* ako časť *d*-rozmerného priestoru, ležiaci na jednej strane od (*d*-1) - rozmernej *hyperroviny*. V ľubovoľnej dimenzii, polpriestor *H* môže byť reprezentovaný dvojicou $\langle R, \vec{n} \rangle$, kde *R* je bod ležiaci v rovine a $\vec{n}$ je normála smerujúca do polpriestoru, viď obrázok nižšie. Bod *P* leží v polpriestore vtedy a len vtedy, ak uhol medzi vektormi *P*-*R* a $\vec{n}$ je nanajvýš 90 stupňov, t.j.:

$$((P\text{-}R) . \vec{n}) \geq 0.$$

Ak skalárny súčin je nula, potom bod *P* leží v rovine ohraničujúcej polpriestor.



Obr. 4.: Orezávanie a polpriestory

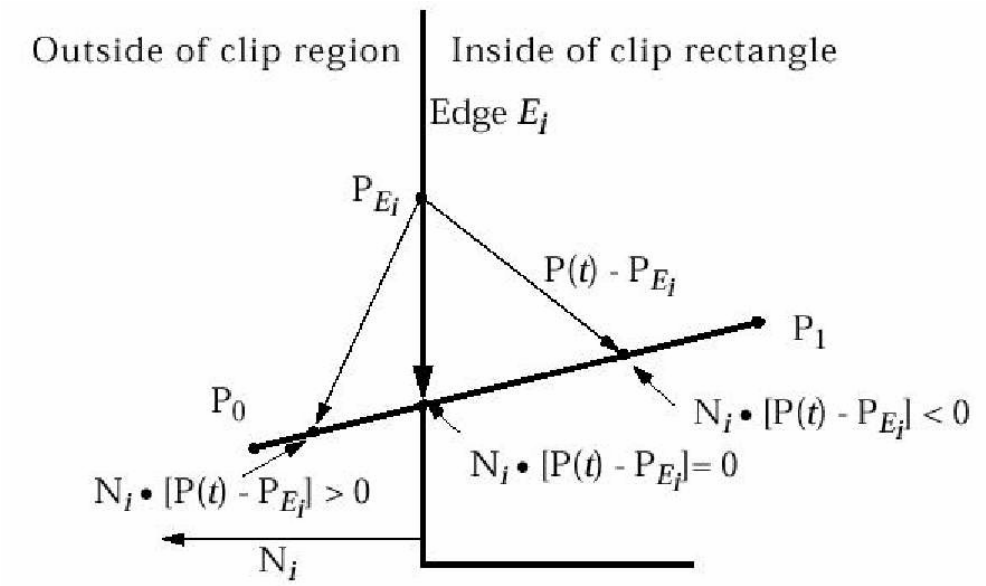
Konvexný polygón v rovine je priesečníkom konečnej množiny polrovín (Táto definícia nie je presná, pretože pripúšťa aj neohraničené konvexné polygóny, ale pre naše účely to postačuje). Vo všeobecnom rozmere, konvexný mnohosten je definovaný ako prienik konečnej množiny polpriestorov. Budeme uvažovať algoritmus v rovinnom prípade, ale neskôr uvidíme, že nám nič nebráni prejsť do vyšších dimenzií. Vstup algoritmu je množina polrovín H_1, ..., H_m, kde $H_i = \langle R_i, \vec{n}_i \rangle$ a množina úsečiek L_1, ..., L_n, kde každá úsečka je reprezentovaná dvojicou bodov $\overline{P_{i,0}P_{i,1}}$. Algoritmus orezáva všetky úsečky a výstupom sú orezané úsečky. Stačí uvažovať prípad jedinej úsečky $\overline{P_0P_1}$.

Parametrické orezávanie úsečiek (Cyrus-Beck/Liang-Barsky Parametric Line Clipping)

Úsečku budeme reprezentovať *parametricky*, použitím konvexnej kombinácie. Ľubovoľný bod úsečky $\overline{P_0P_1}$ môže byť vyjadrený :

$$P(\alpha) = (1-\alpha)P_0 + \alpha P_1 \text{ , kde } 0 \le \alpha \le 1.$$

Algoritmus vypočíta hodnoty parametrov α_0 a α_1, a výsledná úsečka je $\overline{P(\alpha_0)P(\alpha_1)}$. Budeme požadovať, aby $\alpha_0 < \alpha_1$. Na začiatku nastavíme $\alpha_0 = 0$ a $\alpha_1 = 1$. Preto začiatočná orezaná úsečka je rovná pôvodnej úsečke.



Obr. 5.: L-B alg.

Náš prístup bude orezávať úsečku vzhľadom na polrovinu polygónu. Uvažujme ako orezať jednu parametrizovanú úsečku vzhľadom na polrovinu $\langle R, \vec{n} \rangle$. Počas algoritmu sa hodnota α_0 zvyšuje a hodnota α_1 znižuje. Ak $\alpha_0 > \alpha_1$, orezaná úsečka je prázdna a my sa môžeme vrátiť.

Chceme vedieť hodnotu α v ktorej priamka určená orezávanou úsečkou pretína hraničnú priamku polroviny. Aby sme to vypočítali, vložíme definíciu $P(\alpha)$ do vyššie uvedenej podmienky patričnosti do polroviny,

$$((P(\alpha) - R) \,.\, \vec{n}) \geq 0 ,$$

a vyrátame α. Pomocou jednoduchej afinnej algebry dostaneme:

$$\begin{aligned} (((1-\alpha)P_0 + \alpha P_1) - R) \,.\, \vec{n}) &\geq 0 \\ ((\alpha(P_1 - P_0) - (R - P_0)) \,.\, \vec{n}) &\geq 0 \\ \alpha ((P_1 - P_0) \,.\, \vec{n}) - (R - P_0) \,.\, \vec{n} &\geq 0 \\ \alpha ((P_1 - P_0) \,.\, \vec{n}) &\geq (R - P_0) \,.\, \vec{n} \\ \alpha d_1 &\geq d_r \end{aligned}$$

kde $d_1 = ((P_1 - P_0) \,.\, \vec{n})$ a $d_r = ((R - P_0) \,.\, \vec{n})$.

Z tohoto dostávame niekoľko prípadov závislých od d_1.

$d_1 > 0$: Potom $\alpha \geq d_r / d_1$. Položíme $\alpha_0 = \max(\alpha_0 , d_r / d_1)$.

Ak je výsledok $\alpha_0 > \alpha_1$, potom nastavíme príznak označujúci, že orezaná úsečka je prázdna.

$d_1 < 0$: Potom $\alpha \leq d_r / d_1$. Položíme $\alpha_1 = \min(\alpha_1 , d_r / d_1)$.

Ak je výsledok $\alpha_0 < \alpha_1$, potom nastavíme príznak označujúci, že orezaná úsečka je prázdna.

$d_1 = 0$: Potom α je nedefinované. Geometricky to znamená, že ohraničujúca priamka je rovnobežná s úsečkou. V tomto prípade stačí otestovať ľubovoľný bod úsečky. Preto, ak je ((P_0 - R) . $\vec{n}$) < 0, potom vieme, že celá úsečka leží mimo polroviny a nastavíme príznak označujúci, že orezaná úsečka je prázdna. Inak nerobíme nič.

Algoritmus Cyrus-Becka

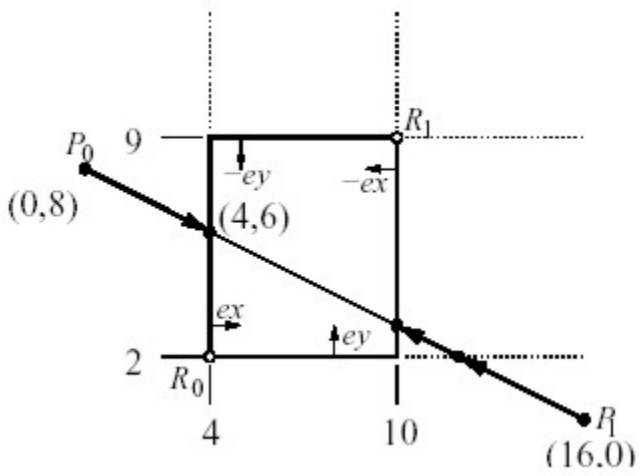
```
Procedure Cyrus-Beck;
begin
   1. if (P1 = P2) then clip_point;                {usecka je bod, orez bod}
      else
      begin
   2.    te:=0;                                    {priprava parametrov}
         tl:=1;
   3.    for e[i] (i=1 to 4)                       {pre kazdu hranu okna}
            begin
            nd:=N[i].D;
   4.       if nd !=0 then t=N.(P1-P[i])/nd;       {vypocet parametrov}
            if nd>0 then te:=max(te, t);           {uprav parameter potenc. vstupu}
            if nd<0 then tl:=min(tl, t)            {uprav parameter pot. vystupu}
            end
   5.    if te>tl then return nil;                 {usecka mimo okna}
         else
      return (P[te], P[tl]);                       {vystup – orezana usecka}
      end
end.
```

Príklad:

Uvažujme príklad tohoto algoritmu. Na obrázku nižšie máme konvexné okno, ohraničené 4 stranami, $4 \le x \le 10$ a $2 \le y \le 9$. Aby sme odvodili reprezentáciu strán pomocou polrovín, vytvoríme dva body so súradnicami $R_0 = (4,2,1)^T$ a $R_0 = (10,9,1)^T$. Nech $\vec{e}_x = (1,0,0)^T$ a $\vec{e}_y = (0,1,0)^T$ sú súradnice jednotkových vektorov. Preto máme 4 polroviny

$$H_1 = \langle R_0, \vec{e}_x \rangle$$
$$H_2 = \langle R_0, \vec{e}_y \rangle$$
$$H_3 = \langle R_1, -\vec{e}_x \rangle$$
$$H_4 = \langle R_1, -\vec{e}_y \rangle$$

Orežme úsečku určenú bodmi P_0 = (0, 8, 1) a P_1 = (16, 0, 1).



Obr. 6.: Orezávací algoritmus

Na začiatku je $\alpha_0 = 0$ a $\alpha_1 = 1$. Najprv uvažujme ľavú stenu, $\langle R_0, \vec{e}_x \rangle$. Dosadením do našich rovníc dostaneme:

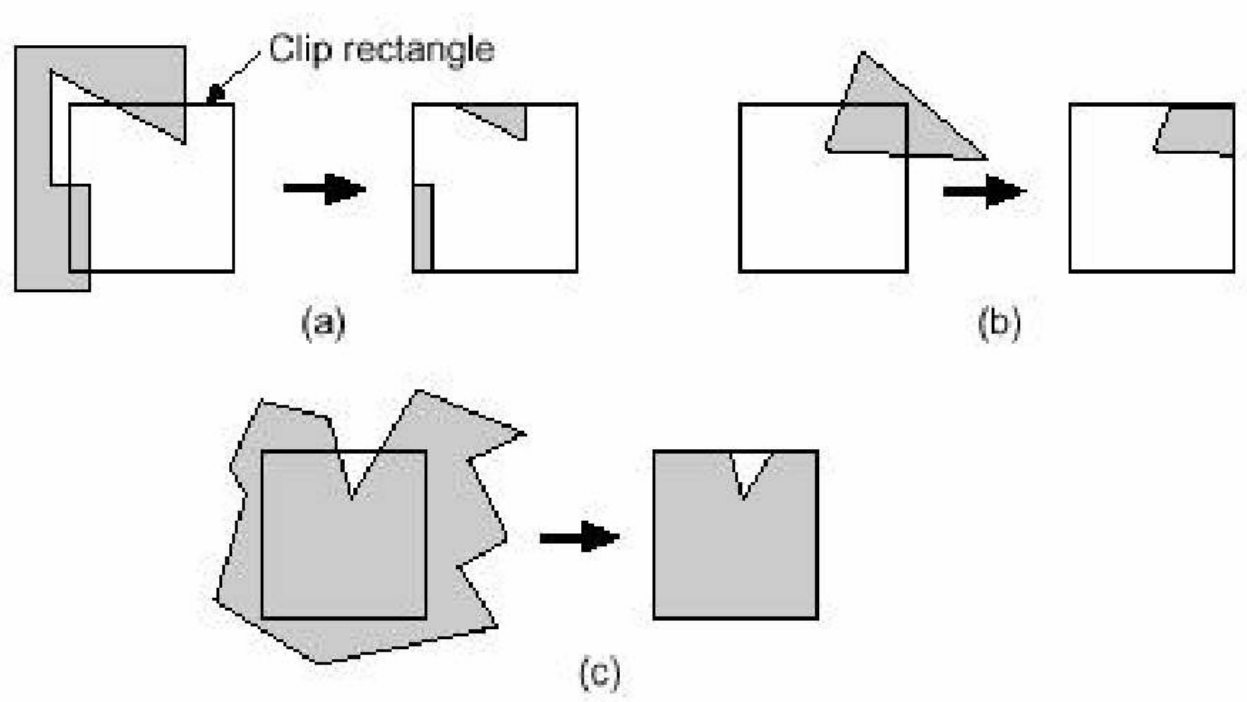
$$d_1 = ((P_1 - P_0).\vec{e}_x)$$
$$= (((16,0,1)-(0,8,1)).(1,0,0)) = ((16,8,0).(1,0,0)) = 16,$$
$$d_r = ((R_0 - P_0).\vec{e}_x)$$
$$= (((4,2,1)-(0,8,1)).(1,0,0)) = ((4,-6,0).(1,0,0)) = 4.$$

Keďže $d_1 > 0$, položíme $\alpha_0 = \max(\alpha_0 , d_r / d_1) = \max(0,4/16) = 1/4$.

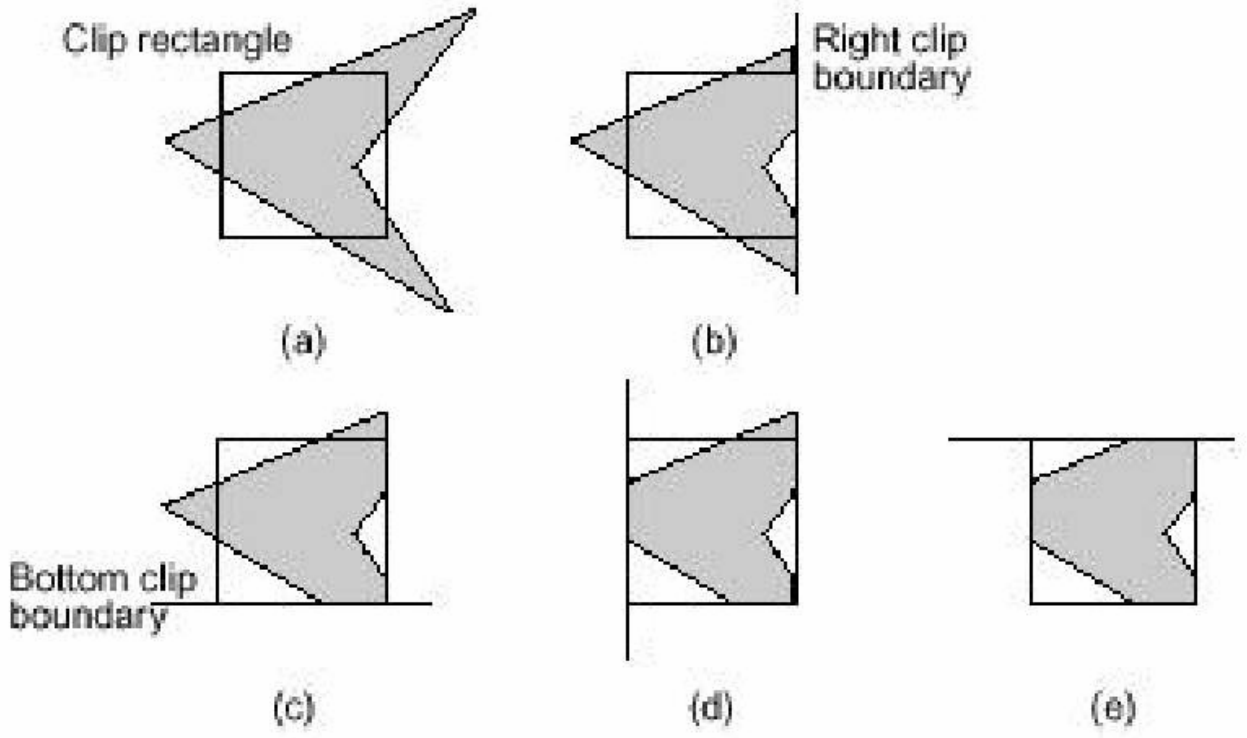
Všimnime si, že výsledný bod je $P(\alpha_0) = (1-\alpha_0)P_0 + \alpha_0 P_1 = (3/4)(0,8,1)+(1/4)(16,0,1) =$ $= (4,6,1)$.

Toto je bod priesečníka ľavej steny s úsečkou. Algoritmus pokračuje orezávaním vzhľadom na ostatné polroviny. Zvyšok príkladu ponechávame ako domácu úlohu pre čitateľa, ale ako pomôcku pre polrovinu H_2 dostaneme $\alpha_1 \leq 3/4$, pre H_3 dostaneme $\alpha_1 \leq 5/8$ a pre H_4 $\alpha_0 \geq -1/16$. Výsledné hodnoty sú $\alpha_0 = 1/4$ a $\alpha_1 = 5/8$.

Prienik mnohouholníka s polrovinou


Clip rectangle
(a)
(b)
(c)

Obr. 7


Clip rectangle
Right clip boundary
Bottom clip boundary
(a)
(b)
(c)
(d)
(e)

Obr. 8

Problém zistenia prieniku konvexného a všeobecného mnohouholníka je možné redukovať na zisťovanie prieniku polroviny a mnohouholníka. Konvexný mnohouholník je totiž prienikom polrovín.

Nech je daná polrovina určená orientovanou priamkou PQ a regulárny, kladne orientovaný mnohouholník $A_0A_1 \ldots A_n$. Označme $s_i = Or_2(A_i, P, Q)$.

Prienik mnohouholníka s polrovinou získame tak, že vytvoríme zoznam tých vrcholov mnohouholníka, ktoré ležia v polrovine. Tento zoznam potom rozdelíme na potrebný počet mnohouholníkov.

Skúmajme vždy hranu A_iA_{i+1} a podľa znamienok s_i; s_{i+1} rozhodnime, či do zoznamu vrcholov pridáme koncový bod hrany, bod prieniku hrany s hraničnou priamkou roviny (označujeme ho C), oba, prípadne ani jeden. Na jednoznačné rozhodnutie, je v niektorých prípadoch nutné zvážiť aj znamienko s_{i+2}. Klasifikácia hrán je zhrnutá v nasledovnej tabuľke.

s_i	s_{i+1}	poloha hrany	do zoznamu sa pridáva
+	+	vnútri	A_{i+1}
+	0	vnútri	A_{i+1}
+	−	vychádza	C
0	+	vchádza, A_i na hranici	A_{i+1}
0	0	celá na hranici	A_{i+1} ak $s_{i+2} > 0$, inak $\emptyset$
0	−	mimo, A_i na hranici	$\emptyset$
−	+	vchádza	C, A_{i+1}
−	0	A_{i+1} na hranici	A_{i+1} ak $s_{i+2} > 0$, inak $\emptyset$
−	−	mimo	$\emptyset$

Pri takto vytvorenom zozname leží na hraničnej priamke polroviny párny počet bodov z tohoto zoznamu. Tieto body je nutné usporiadať vzhľadom na hraničnú priamku polroviny a vytvoriť z nich dvojice. Ostatné body zoznamu sa pospájajú v prirodzenom poradí, tak ako boli do zoznamu pridávané.

Takto sa vytvorený zoznam rozpadne na príslu²ný počet mnohouholníkov. Poznamenajme ešte, že prienik dvoch nekonvexných mnohouholníkov je možné získať tiež pomocou prieniku mnohouholníka a polroviny, jeden z nekonvexných mnohouholníkov je však najprv nutné rozdeliť na mnohouholníky konvexné.

Algoritmus prieniku mnohouholnika s polrovinou

```
Procedure Clipping;
begin
1.  s:=d(A(1), P, Q);
    k:=0;                                   {index pre body C}
2.  for i:=2 to n do
    begin
3.      r:=s;                               {zamenime d pre A(i-1)}
        s:=d(A(i), P, Q);                   {pripravime d pre A(i)}
```

```
4.      if (r>0 and s>0) then next(A(i-1)):=i;          {pripad a) – potvrd smernik}
5.      else if (r>0 and s<=0) then begin
                        intersection(C);                {pocitaj prienik A(i-1)A(i) s PQ}
                        write(C, A, t);                 {zapis bod C na (n+k) miesto A a t()}
                        next(A(i-1)):=n+k;              {pripad b) – smernik C}
                              end;
6.      else if (r<=0 and s>0) then begin
                        intersection(C, t);             {pocitaj prienik A(i-1)A(i) s PQ}
                        write(C, A, t);                 {zapis bod C na (n+k) miesto A a t()}
                        next(A(n+k)):=i;                {pripad d) – smernik z bodu C}
                              end;
7.  end

            {2. cast : uzatvorime smernikovu strukturu}

8.  {Do matice A sa zapisalo k novych bodov C a do pola t() parametre priamky PQ }
    {Usporiadame hodnoty v poli t() a zaroven podla nich ulozime indexy do pola m() }
9.  {Podla vybranych dvojic uzavrieme smernikovu strukturu v poli A() }
    for I i:=1 to k {step 2} do
    begin
10.     next(A(n+m(i))):=n+m(i+1);
    end

            {3.cast vytvorime nove mnohouholniky}

    j:=0;                           {priprav index do pola B pre nove mnohouholniky}
    repeat
11. i:=1;                                   {zacni prezerat od zaciatku pole A()}
        repeat
12.     i:=i+1;                             {vyhladaj nenulovy smernik}
        until ( next(A(i)) != 0 or i=n)     {opakuj pokial este mame nenulovy smernik}
13. i1:=i+1;                                {priprav 1. index pre novy mnohouholnik}
14. if i1!=n then                           {vyhladaj v cykle}
            repeat                          {cely novy mnohouholnik}
15.         B(j):=A(i); temp:=i;            {odloz vrcholy do pola B}
            i:=next(A(i));next(A(temp)):=0;
            until (i=i1);
    until (i1=n)
end.
```

Úlohy

4. Ako vyzerá orezávanie v 3D?
    a. Urobte 3D Cohen-Sutherland algoritmus.
    b. Urobte Liang-Barsky algoritmus pre 3D.

Prieniky

Pozri dokument ***prieniky.pdf***