Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

BAKALÁŘSKÁ PRÁCE

Plzeň, 2006

Vít Ondračka

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Algoritmy ořezávání

Plzeň, 2006 Vít Ondračka

Abstract

Clipping algorithms

Clipping of graphic data is essential part of computer graphics. This work focuses on presenting and comparing algorithms dealing with clipping of graphical elements – lines, line segments and polygons. Fundamentals of basic methods are presented and examined. Efficient line segment clipping algorithm is modified for triangle clipping, implemented, tested and compared to standard Sutherland-Hodgman algorithm.

Obsah

Prohlášení

Prohlašuji, že jsem bakalářskou/diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. 4. 2006

Vít Ondračka

Úvod

Ořezávání grafických primitiv je stěžejní operací počítačové grafiky. Významně urychluje výpočet obrazu, protože z dalšího zpracování vyřazuje části scény, které nejsou viditelné. Existuje celá řada prací a algoritmů zabývajících se tímto tématem. Jsou známy postupy pro ořezávání úseček i polygonů jak v ploše, tak v euklidovském či projektivním prostoru, od nejjednodušších pravidelných oblastí až po nepravidelné mnohoúhelníky či mnohostěny.

Ořezáním grafického elementu rozumíme kontrolu, zda daný bod úsečky, přímky, či polygonu leží uvnitř určené oblasti. Pokud oblast rozděluje ořezávaný prvek na dvě nebo více částí, musí být vymezeny všechny části náležející vnitřku oblasti – určeny příslušné části úseček, přímek či polygonů.

Cílem této práce je podání přehledu o této oblasti počítačové grafiky a srovnání složitosti jednotlivých algoritmů. Získané poznatky jsou spolu s nedávnými poznatky z ořezávání úseček v prostoru homogenních souřadnic aplikovány na ořezávání nejjednodušších polygonů – trojúhelníků – včetně srovnání se známými postupy.

Práce tedy obsahuje popis základních principů metod ořezávání a jejich syntézu pro speciální případ ořezávání trojúhelníka normalizovaným osově orientovaným oknem. Jeden z efektivních algoritmů k ořezávání úseček je rozvinut pro ořezávání polygonů, konkrétně speciální případ trojúhelníků. Tento algoritmus je implementován a porovnán s implementací standardní verze Sutherland-Hodgmanova algoritmu.

Přehled metod

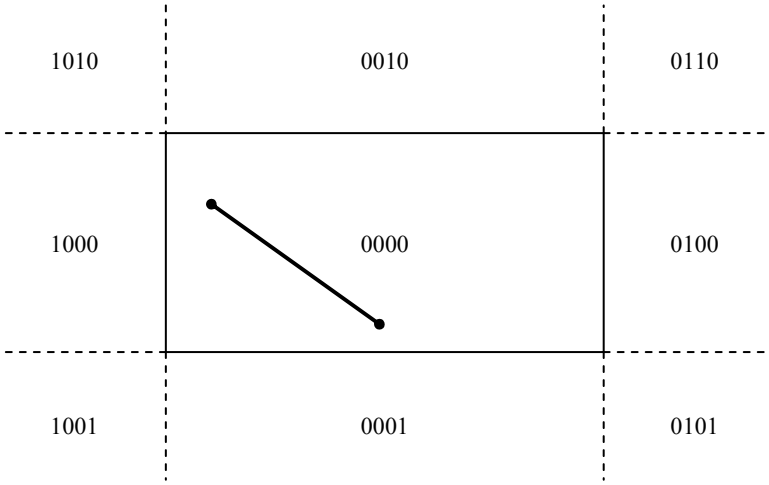
Výchozím bodem pro nás budou znalosti o principech činnosti metod ořezávání. Zaměřme se tedy na popis těchto algoritmů. U všech algoritmů můžeme vyjádřit jejich výpočetní složitost v závislosti na počtu průchodů, nutných ke zpracování dat. K tomu slouží několik postupů, přičemž nejrozšířenějším je tzv. *O*-notace. Ta určuje počet průchodů *g* algoritmem funkcí *f*, omezující shora nejpříznivější případ. Můžeme napsat:

$g = O(f) \Leftrightarrow \exists N_0, c : \forall N > N_0 : g(N) \leq c * f(N)$

kde N_0 je přirozené číslo, *c* kladné reálné číslo.

Cohen-Sutherland

Metoda, publikovaná v [Spro68], slouží primárně k ořezávání úseček v E2 obdélníkem. Jedná se o jednoduchou metodu, která je však výchozím bodem řady pokročilejších algoritmů. Základem metody je rozdělení plochy přímkami hranic ořezávané oblasti na devět segmentů, přičemž každému je přiřazen čtyřbitový kód.



Obr. 1 – ořezávání metodou Cohen-Sutherland

Koncové body úsečky v ploše označme *P*, *Q*, odpovídající kódy *K(P), K(Q)*. Libovolná úsečka spadá podle kódů do jedné ze tří skupin:

1. Platí-li *K(P) | K(Q) = 0*, leží úsečka celá uvnitř oblasti a je tedy přijata.
2. Platí-li *K(P) & K(Q) ≠ 0*, leží úsečka celá vně oblasti a můžeme ji zamítnout. Odstraní úsečky jak ležící celé uvnitř jednoho segmentu, tak některé protínající více segmentů.
3. Platí-li *K(P) & K(Q)* = 0, prochází úsečka více segmenty a je nutné ji oříznout. Přitom může (ale nemusí) zčásti ležet v ořezávané oblasti. Nejméně jeden z kódů *K(P)*, *K(Q)* je nenulový, zvolíme tedy (libovolný) bod s nenulovým kódem a podle polohy jedničky v kódu vybereme hranici, kterou úsečku zkrátíme (nalezením průsečíku). Poté zopakujeme algoritmus na tuto zkrácenou úsečku.

Zkrácení úsečky představuje jednoduchou operaci, jelikož hranice oblasti bývají rovnoběžné s osami souřadného systému, tedy jedna ze souřadnic je konstantní. Odpovídající souřadnice průsečíku tedy bude rovna této hodnotě, druhou lze snadno dopočítat. Nevýhodou algoritmu je jeho opakování na úsečky, které procházejí více segmenty, ale leží zcela vně oblasti. Algoritmická složitost metody je *O(N)*.

Metodu Cohen-Sutherland lze intuitivně rozšířit do E3, stačí prosté přidání dalších osmnácti segmentů (devět před a devět za ořezávaný prostor) a odpovídajícím způsobem rozšířit bitové kódy.

Cyrus-Beck

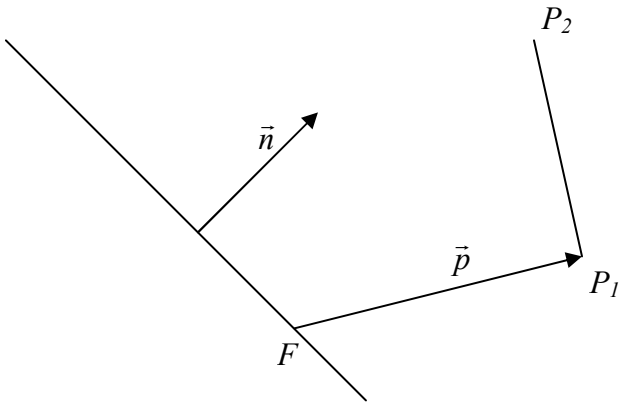
Algoritmus pro E2 publikovali M. Cyrus a J. Beck roku 1978 [Cyru78]. Zabývá se ořezáváním úseček či přímek konvexním polygonem.

Libovolný bod *P* ořezávané úsečky s koncovými body P_1, P_2 můžeme vyjádřit pomocí parametru t rovnicí:

$$P(t) = (1-t)P_1 + tP_2$$
$$P(t) = P_1 + t(P_2 - P_1)$$

$t \in \langle 0;1 \rangle$ (pokud $t \in R$, pak se jedná o rovnici přímky)

Cílem je určit body *P* úsečky (přímky), které leží uvnitř ořezávané oblasti, k čemuž využijeme jednoduchý test polohy bodu vzhledem k přímce představující hranici. Ta je určena bodem *F* a normálovým vektorem $\vec{n}$. Vektor polohy bodu *P* vůči vztažnému bodu *F* označíme $\vec{p}$.



Obr. 2 – poloha bodu vůči hranici

Výsledek skalárního součinu $\vec{n} \cdot \vec{p}$ určuje tři stavy:

1. $\vec{n} \cdot \vec{p} < 0$: bod *P(t)* leží vně oblasti
2. $\vec{n} \cdot \vec{p} = 0$: bod *P(t)* leží na hranici oblasti
3. $\vec{n} \cdot \vec{p} > 0$: bod *P(t)* leží uvnitř oblasti

Hledáme průsečík přímky (úsečky) s hranicí oblasti, tedy vlastně hodnotu parametru *t*.

Platí tedy:

$$\vec{n} \cdot \vec{p} = 0$$
$$\vec{n} \cdot \left[P(t) - F\right] = 0$$
$$\vec{n} \cdot \left[P_1 + t(P_2 - P_1) - F\right] = 0$$
$$-\vec{n} \cdot (P_1 - F) = t(\vec{n} \cdot (P_2 - P_1))$$
$$t = \frac{-\vec{n} \cdot (P_1 - F)}{\vec{n} \cdot (P_2 - P_1)}$$

přičemž $\vec{w} = (P_1 - F)$ je tzv. váhový vektor a $\vec{d} = (P_2 - P_1)$ je směrový vektor přímky.

Je zjevné, že výraz nemá smysl pro $\vec{n} \cdot \vec{d} = 0$, což odpovídá přímce rovnoběžné s hranicí. V opačném případě průsečík přímky s hranicí o příslušném normálovém vektoru existuje a vypočteme jej s využitím výše uvedeného vzorce pro parametr *t*. Pokud je hodnota t průsečíku mimo interval úsečky (na počátku <0, 1>), můžeme jej zamítnout. V opačném případě klasifikujeme průsečík jako vstupní či výstupní bod (určíme, zda s rostoucí hodnotou parametru t vstupujeme do oblasti nebo z ní vystupujeme), podle toho zamítneme jeden z koncových bodů a upravíme interval. Pokud jsme vyšetřili průsečíky se všemi hranami, máme ořezanou úsečku. Je tedy vidět, že metoda poskytuje poměrně široký prostor k optimalizaci, čehož řada dalších metod využívá. Složitost metody je *O(N)*.

Liang-Barsky

[Bars84]

Metoda je v podstatě založena na metodě Cyrus-Beck. Rovněž využívá parametrického zápisu ořezávané přímky (úsečky), avšak ořezávání je možné pouze osově orientovanými hranicemi okna, nikoliv hranicemi obecnými.

Body náležející přímce (koncové body úsečky) označíme V_0, V_1 jejich souřadnice (x_0, y_0) a (x_1, y_1). Pak pro souřadnice libovolného bodu $P(t)$ platí:

$$x(t) = x_0 + \Delta x \cdot t$$
$$y(t) = y_0 + \Delta y \cdot t$$

kde $\Delta x = x_1 - x_0$ a $\Delta y = y_1 - y_0$.

Bod P(t) leží uvnitř ořezávané oblasti, pokud platí:

$$x_{levá} \le x(t) \le x_{pravá}$$
$$y_{dolní} \le y(t) \le y_{horní}$$

Dosazením pak získáme:

$$x_{levá} \le x_0 + \Delta x \cdot t \le x_{pravá}$$
$$y_{dolní} \le y_0 + \Delta y \cdot t \le y_{horní}$$

Tyto čtyři nerovnosti upravíme do společného tvaru $p_i t \le q_i$:

$$-\Delta x t \le x_0 - x_{levá}$$
$$\Delta x \cdot t \le x_{pravá} - x_0$$
$$-\Delta y \cdot t \le y_0 - y_{dolní}$$
$$\Delta y \cdot t \le y_{horní} - y_0$$

Každá úsečka, která je rovnoběžná s jednou z hraničních přímek, má $p_k = 0$ pro k odpovídající této hranici. Pokud pro tuto hodnotu k je ještě $q_k < 0$, pak celá úsečka leží zcela mimo okénko a může být z dalších výpočtů vyloučena.

Je-li $p_k < 0$, pak do nekonečna prodloužená úsečka pokračuje z vnějšku dovnitř hranice. Pokud je $p_k > 0$, úsečka prochází zevnitř hranice ven. Pro nenulové hodnoty p_k můžeme vypočítat velikost parametru u, která odpovídá průsečíku do nekonečna protažené úsečky s prodlouženou hraniční čárou jako:

$$u = \frac{q_k}{p_k}$$

Pro každou úsečku lze spočítat hodnoty parametrů u_1 a u_2, které tak určují tu část úsečky, která leží uvnitř okna. Hodnota u_1 je stanovena podle těch hranic, u kterých ořezávaná úsečka prochází z vnějšku dovnitř ($p < 0$). Pro všechny tyto hranice vypočítáme:

$$r_k = \frac{q_k}{p_k}$$

a jako hodnotu u_1 zvolíme maximum z množiny, která obsahuje 0 a hodnoty r_k.

Podobně získáme velikost u_2 z hranic, u kterých úsečka prochází zevnitř ven ($p > 0$). Hodnoty r_k jsou počítány pro každou takovou hraniční úsečku a hodnota u_2 je potom minimem z množiny, která obsahuje *1* a vypočítané hodnoty r_k.

Je-li $u_1 > u_2$, úsečka leží zcela mimo okénko a může být vyloučena. V opačném případě stanovíme koncové body ořezané úsečky ze dvou hodnot parametru u.

Metodu je možné využít i k ořezávání polygonů [Lian83]. Oblast v E2 rozděluje na 9 segmentů s ohodnocením shodným s počtem sousedních segmentů (tj. 2–4). Vstupem i výstupem je seznam vrcholů. Metoda postupně nezávisle zpracovává jednotlivé hrany v parametrickém vyjádření tak, jak jdou po sobě vrcholy ve vstupním seznamu. Každá hrana, která není rovnoběžná s hranicemi, prodloužená do přímky, protíná všechny čtyři přímky hranic okna. Dva z těchto průsečíků jsou vstupními, druhé dva výstupními body ve vztahu k oknu, přičemž u vstupního bodu vzrůstá ohodnocení segmentu, do kterého přímka směřuje (růst parametru t), u výstupního naopak. Pokud seřazeny podle hodnoty t jsou nejprve oba vstupní body a až poté oba body výstupní, protíná přímka okno, pokud se body vstupní a výstupní střídají, míjí jej ([Lian 83], obr. 6). Podle vzájemné polohy těchto průsečíků, počátečních a koncových bodů úsečky a poloze vzhledem k oknu lze doplnit i body výsledného polygonu ležící v rozích okna. Výhodou této metody je její rychlost [Mail92]. Složitost metody je opět *O(N).*

Sutherland-Hodgman

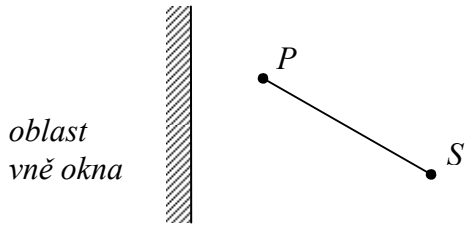
[Suth74]

Základní metoda užívaná k ořezávání polygonů konvexní oblastí (specielně obdélník v E2, resp. kvádr v E3). Matematický aparát je obdobný jako u metody Cyrus-Beck. Jedná se vlastně o ořezávání postupně dodávaných úseček a případné doplňování částmi hranic ořezávané oblasti do uzavřeného polygonálního tahu.

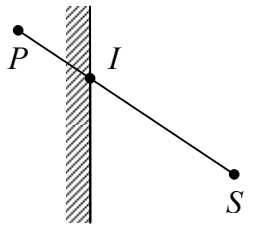
Vstupem metody je seznam bodů (vertexů) $P_1..Pn$ tvořících vstupní polygon. Předpokládá se, že body tvoří polygonální tah, který je případně doplněn na uzavřený (dodáním hrany P_nP_1). Výstupem je pak opět seznam bodů (jeho délka je zpravidla odlišná od délky vstupního seznamu, může být i prázdný).

Pro každou hranici (předpokládáme normálu směřující dovnitř oblasti) je testována úsečka tvořená vždy posledním ořezaným bodem (*S*) a bodem v seznamu následujícím (*P*), přičemž mohou nastat čtyři případy:

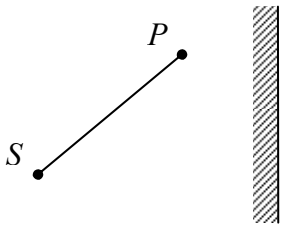
1. Úsečka *SP* leží v polorovině dané hranicí, výstupem je bod *P*.
2. *S* leží uvnitř poloroviny, *P* vně. Pak je výstupem průsečík *I* úsečky *SP* a hranice.
3. *SP* leží vně poloroviny, žádný výstup.
4. *S* leží vně, *P* uvnitř. Výstupem je nejdřív průsečík *I* úsečky *SU* a hranice poté bod *P*.



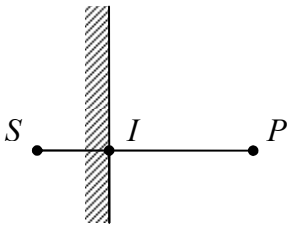
Obr. 3a – případ 1



Obr. 3b – případ 2



Obr. 3c – případ 3



Obr. 3d – případ 4

Algoritmus s výhodou využívá možnosti ořezání polygonu každou hranicí zvlášť, je reentrantní. Výsledky ořezání jednou hranicí předává k dalšímu ořezání, což umožňuje zřetězení zpracování (pipelining).

Pokud zpracováváme nekonvexní polygon, může se ořezávaný polygon rozdělit na více polygonů spojených obousměrnými hranami. Dle [Suth74] je možné využít modifikovanou metodu i k ořezávání úseček, není to však výhodné. Složitost metody pro polygony je *O(M*N)*, kde *M* je počet vrcholů ořezávaného polygonu.

Weiler-Atherton

Obecný algoritmus pro ořezávání nekonvexního polygonu (s dírami) nekonvexním oknem (polygonem). Původní algoritmus je prezentován ve [Weil77], jeho detailní popis uvádí např. [Žára04]. Vstupem je seznam vrcholů ořezávaného polygonu, dále vrcholů jeho případných děr uspořádaných v opačném smyslu, a konečně také vrcholů ořezávajícího polygonu. Výstupem je seznam vrcholů ořezaného polygonu.

Algoritmus nejprve nalezne průsečíky obou polygonů a uloží je do zvláštního seznamu. Takto nalezené průsečíky také zařadí na odpovídající místa v seznamech vrcholů jak ořezávaného, tak ořezávajícího polygonu. Poté vybere vždy jeden průsečík ze seznamu a podle orientace hran polygonů zvolí, zda do výstupního seznamu řadit body seznamu ořezávaného (jsme uvnitř okna) či ořezávajícího (vně okna) polygonu. Na výstup pak tyto body předává, dokud nedojde do dalšího průsečíku. Ten pak opět vybere ze seznamu a na výstup začne předávat body z druhého, než doposud procházeného seznamu. Algoritmus končí, pokud došel do výchozího bodu. Složitost algoritmu je *O(M*N)*.

Skala, Bui

[Bui98]

Metoda je určena k ořezávání úseček konvexním polygonem v E2. Oproti předchozím metodám je zvláště zajímavá, neboť namísto obvyklé lineární výpočetní složitosti *O(N)* přináší složitost logaritmickou *O(log N)*. Ořezávací polygon je chápán jako otevřený polygonální tah a průsečík úsečky s tímto tahem je hledán metodou půlení intervalu. Označíme-li koncové body tahu x_i, x_j (přičemž i, j odpovídají indexu bodu, $i < j$), pak mohou nastat dva případy:

1. x_i a x_j se nacházejí v opačných polorovinách oddělených ořezávanou přímkou, pak vzhledem ke konvexnosti ořezávajícího polygonu je mezi nimi jediný průsečík, který lze nalézt metodou půlení intervalu
2. x_i a x_j se nacházejí v jedné polorovině; pak je tah (jeho část) rozpůlen „vložením“ bodu x_k ($i < k < j$) a mohou nastat další případy:
   a. x_k leží ve opačné polorovině, než x_i, x_j; pak lze opět nalézt průsečíky binárním vyhledáváním
   b. x_k leží ve stejné polorovině; pak musíme zkoumat vzdálenost bodů od ořezávané přímky (označme čitatele vzdálenosti bodu a jako $F(a) = Ax + By + C$):
      i. x_i je nejbližší – pokud je $F(x_{i+1}) \geq F(x_i)$, znamená to, že se polygonální tah vzdaluje a nemůže protnout přímku – je zamítnuta; v opačném případě může protínat, pak vypustíme část polygonálního tahu mezi body x_k a x_j a zkoumáme dále jen tuto část (počínaje krokem 2)
      ii. x_j je nejbližší – obdobný případ
      iii. x_k je nejbližší – pak zkoumáme vzdálenost $F(x_{k+1})$ a $F(x_k)$; je li $F(x_{k+1}) > F(x_k)$, odstraníme část tahu mezi body x_k a x_j, v opačném případě část mezi body x_i a x_k a opět pokračujeme dělením tahu (krok 2)

Výhodou je právě využití dělení intervalu, které přináší algoritmu namísto obvyklé lineární složitosti složitost logaritmickou – *O(log N)*.

Další algoritmy

Stejně jako některé z již předestřených, jsou i tyto metody nezřídka založeny na základních principech, prezentovaných v algoritmech Cyrus-Beck, Cohen-Sutherland a Sutherland-Hodgman. Přinášejí však často významná vylepšení vedoucí zpravidla k vyšší rychlosti zpracování.

Maillot [Mail92]

Slouží k ořezávání polygonu libovolně orientovaným konvexním oknem v E2. Vychází z metody Cohen-Sutherland, snaží se využívat data vytvořená testem pro okamžité přijetí či zamítnutí úsečky. Každá hrana polygonu se zpracovává v jednom průchodu algoritmu, přičemž nejprve je jednoduchým způsobem oříznuta (C-S), následuje tzv. test rohu (turning-point test). Tento test zjišťuje, zda úsečka končí v jednom ze čtyř rohových segmentů, pokud ano, bude třeba přidat rohový bod okna jako bod výstupního polygonu. Pro složitější případy hran protínajících více segmentů mimo viditelnou oblast vytváří algoritmus pomocnou tabulku kódů, s jejíž pomocí pak určuje případné další rohové body. Přináší vyšší rychlost, avšak generuje některé další zbytečné degenerované hrany (obousměrné na hranici okna).

Vatti [Vatt92]

Definuje jiný přístup ke zcela obecnému případu ořezávání (nekonvexní polygon včetně sebeprotínajících nekonvexním oknem). Výstupem může být jak polygon, tak přímo čtyřúhelníky. Algoritmus nejprve nalezne protínající se hrany a vložením dalšího bodu tento problém odstraní. Poté rozdělí hrany polygonu na levou a pravou hranici a pro každou hranu určí její minimum a maximum (souřadnic). Poté je polygon pomyslně rozdělen na pruhy ležící mezi vodorovnými ($y = konst$) přímkami procházejícími jednotlivými vrcholy polygonu. V každém tomto pruhu jsou označeny hrany, které se v něm nacházejí, nalezeny průsečíky hran s jeho dolní hranicí, a seznam hran je podle hodnot souřadnice x seřazen. Následuje řešení výstupu odlišné pro lokální minimum (levý bod), lokální maximum (pravý bod) a body mezilehlé, přičemž pokud je řešen bod levé hranice polygonu, přidává se vrchol výstupního polygonu na „levou“ stranu výstupného řetězu a naopak. Seřazení hran zajišťuje korektní vytváření výstupního polygonu. Algoritmus přináší urychlení výpočtu i pro konvexní polygony oproti Sutherland-Hodgmanově algoritmu.

Greiner-Hormann [Grei98]

Další metoda pro ořezávání obecného polygonu v E2. Algoritmus hledá části hranice ořezávaného polygonu ležící uvnitř[1] ořezávajícího polygonu a poté je spojuje. Nalezení částí ležících uvnitř je obdobné jako u algoritmu Weiler-Atherton. Spojení je pak záležitostí nalezení hrany (řetězce hran) spojující nalezené průsečíky (koncové body viditelných hran ořezaného polygonu). Metoda přináší značné urychlení výpočtu oproti [Vatt92].

[1] Vnitřek zcela obecného polygonu je složitější definovat. Algoritmus k tomu užívá čísel: při postupném procházení vrcholů polygonu se určují uzavřené smyčky. S každou smyčkou po směru hodinových ručiček se hodnocení snižuje o 1, proti směru se o 1 zvyšuje. Pouze segmenty s lichým ohodnocením jsou vnitřkem polygonu.

Zhang-Sabharwal [Zhan02]

Jedná se o variantu algoritmu Liang-Barsky, ořezávající úsečky obdélníkovým oknem s osově orientovanými hranicemi. Urychlení výpočtu přináší především optimalizace včasné detekce triviálně zamítnutelných hran.

Další metody představili např. Andrejev (1989) pro obecné polygony, Blinn (1991) pro úsečky, Rappaport (1991) pro úsečky i polygony či Toussaint (1985) pro konvexní polygony.

Shrnutí

Většina popsaných metod je založena na jednoduchých algebraických operacích a jejich princip je poměrně snadno pochopitelný. Vzájemné srovnání je samozřejmě možné pouze u metod, zabývajících se stejným problémem. Vzhledem k tomu, že je časová náročnost popsaných metod vesměs lineární, nejsou mezi nimi zásadní rozdíly. Existují i algoritmy s nižší časovou náročností, přičemž jeden z nich je zde popsán, nicméně zpravidla je u nich nezbytné provést další výpočty před samotným ořezáváním.

Ořezávání v homogenních souřadnicích

Další možnosti pro ořezávání přináší zavedení tzv. prostoru homogenních souřadnic. Jak bude patrné z následujícího popisu, přináší v řadě případů značné usnadnění výpočtu.

Klasický přístup

Základním postupem pro ořezávání v homogenních souřadnicích je přístup odvozený z [Blinn78]. Polohu bodu v prostoru běžně určujeme v kartézských souřadnicích, v nichž má vektor polohy bodu tři složky. Nevýhodou kartézských souřadnic je ale nemožnost vyjádření bodů ležících v nekonečnu a především nemožnost zavedení některých transformací, především posunu a perspektivní projekce. Proto zavádíme tzv. homogenní souřadnice jako prostor, v němž má vektor bodu čtyři složky: (x, y, z, w).

Tento vektor se v homogenních souřadnicích zobrazuje na bod $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$;

pokud $w = 0$, bod leží v nekonečnu.

Do prostoru kartézských souřadnic se tedy pro $w \neq 0$ zobrazí na bod $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$.

Zjevně tedy platí, že vektory
$\vec{v}_1 = (x, y, z, w)$
$\vec{v}_2 = k \cdot \vec{v}_1 = k \cdot (x, y, z, w) = (kx, ky, kz, kw);\ k \in \{R - 0\}$
představují stejný bod.

Pro zjednodušení uvažujme ořezávání do obdélníkového okna v intervalu:
$-1 < x < 1$
$-1 < y < 1$

Tomu odpovídají rovnice hranic – ořezávacích rovin (v pořadí levá, pravá, dolní, horní):
$x = -1$ $\quad x = 1$ $\quad y = -1$ $\quad y = 1$

V homogenních souřadnicích odpovídají homogenním plochám:
$w + x = 0$ $\quad w - x = 0$ $\quad w + y = 0$ $\quad w - y = 0$

Řešíme-li pouze viditelnost bodů před pozorovatelem, což bývá zajištěno omezením hodnot souřadnice *z*, můžeme přepsat na nerovnice, které musí splňovat bod o souřadnicích (x, y), aby byl viditelný:
$w + x > 0$ $\quad w - x > 0$ $\quad w + y > 0$ $\quad w - y > 0$

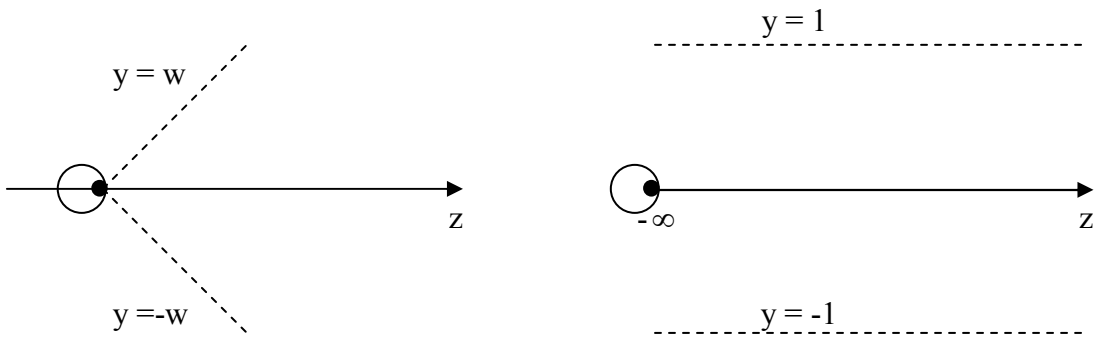
Závěr je tedy takový, že údaj o poloze bodu vzhledem k ořezávacím rovinám tedy v prostoru homogenních souřadnic zjistíme pouhým porovnáním složek vektoru jeho polohy.

Zobecníme-li rovnice pro případ obecné polohy hranic, získáme nerovnice:
$a < x < b$ $\quad a < 0 < b$
$c < y < d$ $\quad c < 0 < d$

obdobně odvodíme další kroky – ve výsledku přibude pouze operace násobení:

$x = a$:	$x = b$:	$y = c$:	$y = d$:
$aw + x > 0$	$bw - x > 0$	$cw + y > 0$	$dw - y > 0$

Díky vlastnostem homogenních souřadnic je zavedeno perspektivní zobrazení, které umožní vzájemně převést pozorovaný prostor tvaru jehlanu na oblast s pozorovatelem v nekonečnu (nekonečně dlouhý kvádr):



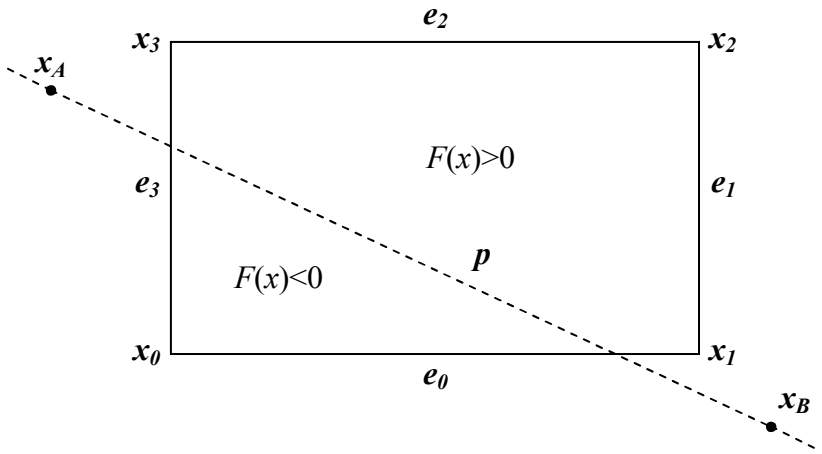
Obr. 4 – transformace oblasti

Takovou oblast tedy můžeme snadno ořezat výše uvedeným způsobem porovnáním s homogenní souřadnicí, spolu s omezením hodnot souřadnice *z*, tedy ořezáním přední a zadní stěnou prostoru. Tento postup je tedy velice výhodný.

Metoda s vektorovým součinem

Naproti výše popsanému postupu je ve [Skal05] publikována metoda, využívající vlastností prostoru homogenních souřadnic hlouběji – jedná se metodu ořezávání přímek a úseček konvexním polygonem, ovšem pouze v E2.

Předpokládejme tedy, že ořezáváme přímku *p* polygonem *P* o *N* vrcholech, přičemž je přímka reprezentována rovnicí $F(x) = ax + by + c$.



Obr. 5 – situace ořezávání přímky obdélníkovým oknem

Pro každý vrchol polygonu $\boldsymbol{x_i}$ stanovíme kód polohy c_i vůči přímce $\boldsymbol{p}$:

$F(x_i) \geq 0 \Rightarrow c_i = 1$

$F(x_i) < 0 \Rightarrow c_i = 0$

Získáme tak vektor polohy vrcholů ořezávacího polygonu $\boldsymbol{c}$:

$\vec{c} = [c_N, c_{N-1}, ..., c_1, c_0]^T$ kde N je počet vrcholů polygonu.

Pokud se kódy dvou po sobě následujících vrcholů $\boldsymbol{x_i}$, $\boldsymbol{x_{i+1}}$ liší (tedy platí $\boldsymbol{c_i \neq c_{i+1}}$), znamená to, že ořezávaná přímka protíná hranu $\boldsymbol{e_i}$ tvořenou těmito vrcholy. Pro daný ořezávací polygon lze tedy sestavit tabulku, která pro každou kombinaci $\boldsymbol{c}$ kódů c_i obsahuje indexy hran $\boldsymbol{e}$ protínaných ořezávanou přímkou, pro niž byly tyto kódy vypočteny.

Tabulka 1 je konkrétním případem obdélníkového okna na obr. 5 a obdobou tabulky ze [Skal05]. ***TAB1*** a ***TAB2*** jsou vektory obsahující indexy protínaných hran (postupně ve směru od bodu $\boldsymbol{x_A}$ k bodu $\boldsymbol{x_B}$), hodnota ***MASK*** je využívána při ořezávání úseček ležících částečně uvnitř okna a její význam je popsán níže.

Tabulka 1 – Hrany obdélníkového okna protínané přímkou

$\boldsymbol{c}$	c_3	c_2	c_1	c_0	***TAB1***	***TAB2***	***MASK***
0	0	0	0	0	-	-	-
1	0	0	0	1	0	3	0001
2	0	0	1	0	1	0	0100
3	0	0	1	1	1	3	0100
4	0	1	0	0	2	1	0010
5	0	1	0	1	nelze	nelze	nelze
6	0	1	1	0	2	0	0010
7	0	1	1	1	2	3	0010
8	1	0	0	0	3	3	1000
9	1	0	0	1	0	2	0001
10	1	0	1	0	nelze	nelze	nelze
11	1	0	1	1	1	2	0100
12	1	1	0	0	3	1	1000
13	1	1	0	1	0	1	0001
14	1	1	1	0	3	0	1000
15	1	1	1	1	-	-	-

Když tedy známe hrany protínané přímkou p, vypočítáme pro každou z nich průsečík I pomocí vektorového součinu. Rovnici ořezávané přímky označme $a_1 x + b_1 y + c_1 w = 0$, rovnice hranice jako $a_2 x + b_2 y + c_2 w = 0$. Pro průsečík I platí:

$$I = [a_1, b_1, c_1] \times [a_2, b_2, c_2] = \det \begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix} \quad (1)$$

přičemž $\vec{i} = [1,0,0]^T$, $\vec{j} = [0,1,0]^T$ a $\vec{k} = [0,0,1]^T$

Důkaz je rovněž uveden ve [Skal05].

Pokud dále uvažujeme ořezávání standardním osově orientovaným čtvercovým oknem, mají normálové vektory hranic vždy jednu složku nulovou. Operace vektorového součinu se tedy dále zjednoduší. Rozeberme si případy průsečíků úsečky *p* (s koncovými body *SP*) s jednotlivými hranicemi (význam vektorů ***i***, ***j*** a ***k*** je shodný s výše uvedeným):

$$\vec{p} = [a,b,c]^T = \det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_S & y_S & w_S \\ x_P & y_P & w_P \end{pmatrix} = [y_S w_P - y_P w_S, x_P w_S - x_S w_P, x_S y_P - x_P y_S]^T \qquad (2)$$

Vyjdeme tedy z rovnice (1), přičemž za *a*, *b*, *c* dosazujeme z (2).

Pro levou hranici platí: $\vec{n} = [1,0,1]^T$

$$I = n^T \times p^T = \det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 0 & 1 \\ a & b & c \end{pmatrix} = \vec{i}(-b) + \vec{j}(a-c) + \vec{k}(b) = [-b, a-c, b]^T \qquad (3a)$$

Pro pravou hranici: $\vec{n} = [-1,0,1]^T$

$$I = n^T \times n_p{}^T = \det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ -1 & 0 & 1 \\ a & b & c \end{pmatrix} = \vec{i}(-b) + \vec{j}(a+c) + \vec{k}(-b) = [-b, a+c, -b]^T \qquad (3b)$$

Obdobně pro horní hranici: $\vec{n} = [0,-1,1]^T$

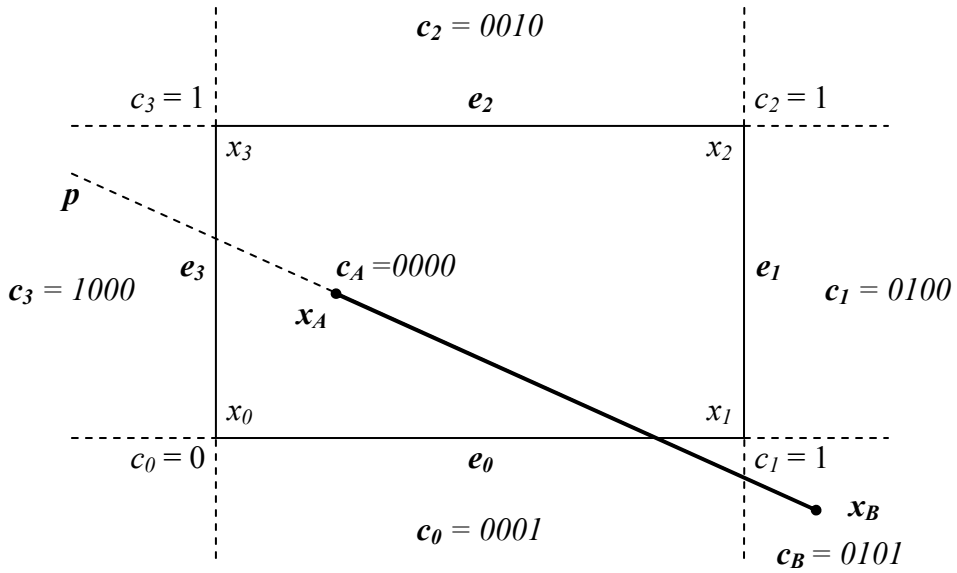
$$I = n^T \times n_p{}^T = \det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ 0 & -1 & 1 \\ a & b & c \end{pmatrix} = \vec{i}(-c-b) + \vec{j}(a) + \vec{k}(a) = [-c-b, a, a]^T \qquad (3c)$$

A konečně pro dolní hranici $\vec{n} = [0,1,1]^T$:

$$I = n^T \times n_p{}^T = \det\begin{pmatrix} \vec{i} & \vec{j} & \vec{k} \\ 0 & 1 & 1 \\ a & b & c \end{pmatrix} = \vec{i}(c-b) + \vec{j}(a) + \vec{k}(-a) = [c-b, a, -a]^T \qquad (3d)$$

Pokud chceme tímto postupem ořezávat úsečky místo přímek, přibývá potřeba omezit ořezávání jejími koncovými body. K tomu je využito principu Cohen-Sutherlandovy metody (viz str. 3) – pro koncové body úsečky určíme kódy polohy vůči hranicím okna. Pomocí nich rozlišíme, zda leží celá úsečka vně nebo uvnitř okna, či zda leží uvnitř jen jeden bod a musíme nalézt průsečík. V posledním uvedeném případě ještě musíme rozlišit, který z bodů leží vně okna. Na kód polohy tohoto bodu ($\boldsymbol{c_A}$ nebo $\boldsymbol{c_B}$) pak operací bitového součinu aplikujeme masku (položka ***MASK*** tabulky 1), jejíž pomocí rozlišíme, kterou ze dvou protínaných hran (příslušná hodnota ***TAB1*** nebo ***TAB2***) ořezávat. Hodnota masky odpovídá kódu polohy

vnějšího bodu vůči hranici s indexem ***TAB1***. Je-li výsledek bitového součinu roven 0, protíná úsečka hranici s indexem ***TAB2***, v opačném případě s indexem ***TAB1***.

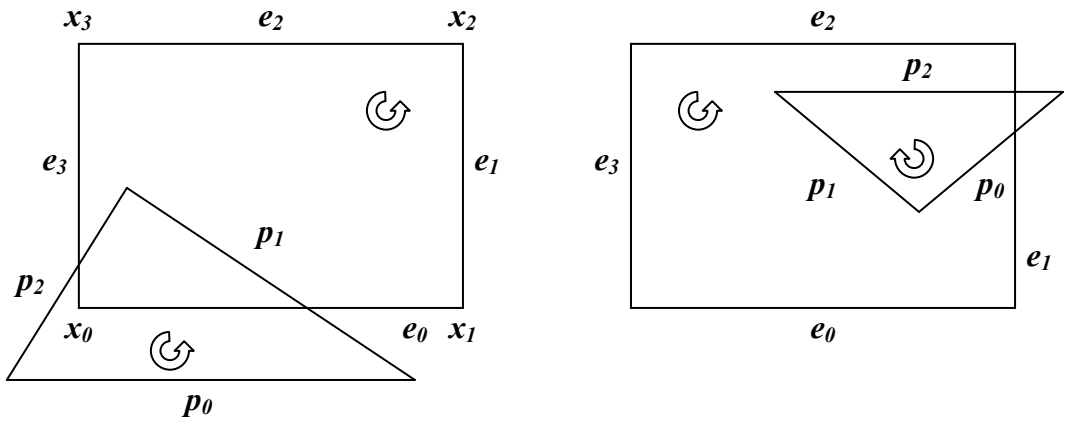


Obr. 6 – kódy vrcholů okna a bodů úsečky při ořezávání
kódy $\boldsymbol{c_i}$ *je maskou vůči hranici* $\boldsymbol{e_i}$

V situaci ilustrované obr. 6 tedy vidíme, že přímka $\boldsymbol{p}$ protíná nejprve hranu $\boldsymbol{e_3}$, poté hranu $\boldsymbol{e_0}$. Odpovídající kód vrcholů okna je 1110. Vně okna leží bod $\boldsymbol{x_B}$ s kódem $\boldsymbol{c_B} = [0101]^T$, kód bodu vnějšího vůči hraně $\boldsymbol{e_3}$ je $\boldsymbol{c_3} = [1000]^T$. Operací bitového součinu zjišťujeme:
$c_B \,\&\, c_3 = 0$, určovat tedy budeme průsečík s hranou $\boldsymbol{e_0}$.

Aplikace metod ořezávání úseček na ořezávání polygonů

Jakýkoliv postup ořezávání úseček, tudíž i výše popsanou metodu, můžeme snadno rozšířit na ořezávání polygonů. Ořezávaný polygon *P* chápeme jako množinu jeho hran $\boldsymbol{p_i}$.
Předpokladem je, že polygon *P* je orientován shodně s ořezávacím oknem (s vrcholy $\boldsymbol{x_i}$) tvořeným hranami $\boldsymbol{e_i}$:



Obr. 7 – shodná (vlevo) a rozdílná (vpravo) orientace polygonů

Všechny tyto hrany standardním způsobem konkrétní metody ořízneme, čímž získáme vrcholy ležící uvnitř okna a průsečíky s jeho hranicemi. V řadě případů tedy pro korektní výsledek musíme přidat jeden či více vrcholů ořezávacího okna – např. v případě, který je na obr. 7 vlevo. Vektor již získaných bodů označme ***m***, výstupní vektor ***n***, přičemž platí:
$\vec{m} = [m_0, m_1, ..., m_i]^T$

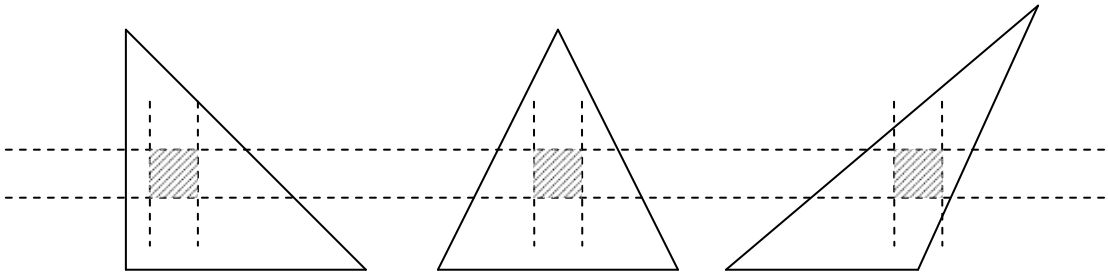
Následující postup pracuje s indexem vstupního vektoru *i*, výstupního vektoru *j* a body *S* a *P*:

Algoritmus 1 – doplnění průsečíků úseček polygonu na vrcholy úplného ořezaného polygonu

1. $S = m_0$, $P = m_1$, $j = 0$, $i = 1$
2. pokud *S* nebo *P* není průsečíkem, nebo oba body leží na stejné hraně okna, přidej *P* do *n* a zvětši *i* o 1; v opačném případě do *n* přidej vrchol okna x_k, kde *k* je indexem hrany okna, na níž leží vrchol *S*
3. pokud $n_j == m_0$, skonči
4. $S = n_j$, $P = m_i$, $j = j + 1$, pokračuj bodem 2

Vidíme, že u průsečíků musíme znát informaci, na které hraně ořezávaného polygonu leží. Přidání tohoto údaje tedy musíme přidat do algoritmu ořezávání úseček. V případě, že algoritmus využívá kódování koncových bodů úseček, není nutné provádět ohodnocení pro každou úsečku dvakrát. Polohu bodů stačí vyhodnotit pro každý vrchol polygonu a poté ji využít pro obě úsečky, jejichž koncovým bodem daný vrchol je.

Další komplikací je situace, kdy celé okno leží uvnitř trojúhelníka (viz obr. 8, 11f). Tento zvláštní případ nemůže Algoritmus 1 vyřešit, protože výstupem ořezávání jednotlivých úseček je prázdný seznam. Zde již musíme zkoumat buď konkrétní polohu jednotlivých vrcholů, nebo vlastnosti hran trojúhelníka. Možné polohy trojúhelníka vzhledem k oknu, pro něž nastává tento problém, ilustrujeme na obr. 8. Všechny další případy jsou pouze iterací, tj. pootočením a/nebo zrcadlením těchto stavů.



Obr. 8 – polohy trojúhelníka s oknem uvnitř
(zleva případ A, B a C)

Pokud máme vrcholy trojúhelníka ohodnoceny Cohen-Sutherlandovým kódem podle jejich polohy vzhledem k hranicím okna (viz obr. 1), můžeme snadno určit i ohodnocení hran. Pro hranu $\boldsymbol{e_i}$ s koncovými vrcholy s kódy $\boldsymbol{c_A}$, $\boldsymbol{c_B}$ platí:

$c_i = c_A \text{ xor } c_B$

Operaci „xor“ (exkluzivní bitový součet) s operandy *A*, *B* můžeme vyjádřit:

$$A \text{ xor } B = (A \mid \mathrm{B}) - (\mathrm{A} \,\&\, \mathrm{B})$$

Počet bitů s hodnotou „1“ v získaném kódu hrany c_i označíme jako ***bitovou délku hrany***.

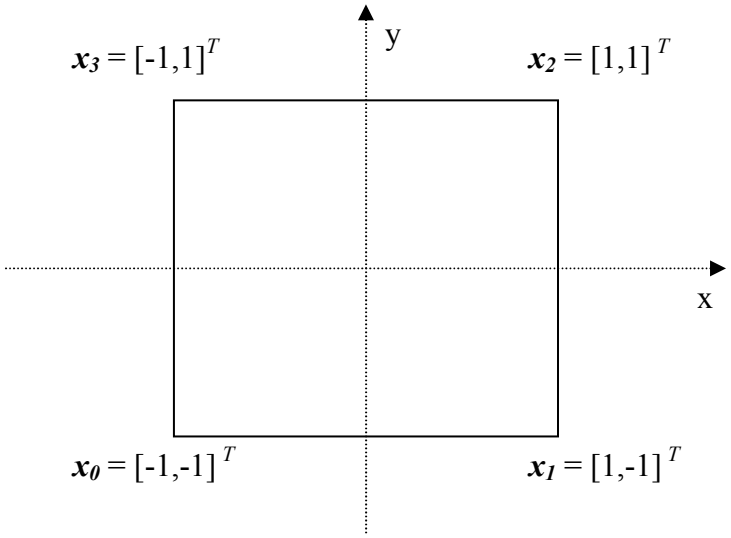
Předpokládejme tedy, že neexistují žádné průsečíky hran trojúhelníka s hranicemi okna. Aplikujeme-li poznatky o bitové délce hran na případy na obr. 8, leží okno uvnitř trojúhelníka jen a pouze v těchto případech bitových délek (iterace polohy nejsou zahrnuty):

1. 4 – 2 – 2 (případ *A*)
2. 3 – 3 – 2 (případ *B*)
3. 4 – 3 – 1 (případ *C*)

Všechny jiné kombinace bitových délek hran při absenci průsečíků s hranicemi okna znamenají, že trojúhelník leží zcela mimo okno.

Srovnání metod pro ořez trojúhelníka

Zaměřme se tedy na rovnání metody ořezávání vektorovým součinem v homogenních souřadnicích s běžnou Sutherland-Hodgmanovou metodou. Omezíme se pouze na speciální případ ořezávání trojúhelníka namísto obecného polygonu, a to standardním osově orientovaným čtvercovým oknem:



Obr. 9 – standardní osově orientované ořezávací okno

Jelikož ořezávaná oblast je konvexní a vstupním polygonem je též nutně konvexní trojúhelník, je konvexní i výstupní polygon – nevznikají degenerované hrany. Vyjma specifických případů se dostaneme do situace, kdy výstupní polygon má více než tři vrcholy, je tedy nezbytné užít patřičných datových struktur.

Metoda Sutherland-Hodgman

Algoritmus ořezává trojúhelník (obecně polygon) postupně po jednotlivých úsečkách. Poté, co zjistíme polohu koncových bodů úsečky, může nastat případ, kdy musíme nalézt průsečík s hranicí.

Uvažujme parametrickou rovnici úsečky s koncovými body P_i, P_{i+1}:

$$P_i = [x_i, y_i]^T,\ P_{i+1} = [x_{i+1}, y_{i+1}]^T$$
$$P(t) = (1-t)P_i + tP_{i+1} \qquad t \in \langle 0;1 \rangle \tag{4}$$

a její křížení např. s *levou* hranicí $x = -1$. Dosazením do rovnice hranice (4) získáme:

$$x = -1$$
$$(1-t)x_i + tx_{i+1} = -1$$

a odtud hodnotu parametru t průsečíku:

$$x_i - tx_i + tx_{i+1} = -1$$
$$t(x_{i+1} - x_i) = -x_i - 1$$
$$t = \frac{x_i + 1}{x_i - x_{i+1}} \tag{5a}$$

obdobně pro *pravou* hranici:

$$x = 1$$
$$(1-t)x_i + tx_{i+1} = 1$$
$$t(x_{i+1} - x_i) = 1 - x_i$$
$$t = \frac{x_i - 1}{x_i - x_{i+1}} \tag{5b}$$

pro *dolní* a *horní* hranici tedy lze snadno odvodit:

$$t = \frac{y_i + 1}{y_i - y_{i+1}} \qquad t = \frac{y_i - 1}{y_i - y_{i+1}} \tag{5c, 5d}$$

Se získanou hodnotou parametru t pak prostou lineární interpolací vypočteme hodnoty x a y průsečíku:

$$x_0 = (1-t)X_i + tX_{i+1}$$
$$y_0 = (1-t)Y_i + tY_{i+1} \tag{6}$$

Pro algoritmus Sutherland-Hodgman uvažujme modifikovaný pseudokód z [Bui99], str. 59.:

Algoritmus 2 – ořezání polygonu metodou Sutherland-Hodgman

```
type
    vertex = array[1..3] of real;
    polygon = array[1..7] of vertex;

procedure clip (in_v : polygon;        { vstupní vrcholy }
    var out_v : polygon;               { výstupní vrcholy }
    in_length : integer;               { počet vstupních vrcholů }
    var out_length : integer;          { počet výstupních vrcholů }
    int edge_id);                      { číslo hrany }
var I, P, S : vertex;
    j : integer
begin
    out_length = 0;
    S := in_v[in_length];
    for j := 1 to in_length do         { vnitřní smyčka }
     begin
        P := in_v[j];
        if INSIDE(P, edge_id) then
            if INSIDE(S, edge_id) then OUTPUT(out_v, P, out_length);
            else begin
                I := INTERSECT(S, P, edge_id);
                OUTPUT(out_v, I, out_length);
                OUTPUT(out_v, P, out_length);
            end
        else if INSIDE(S, edge_id) then begin
                I := INTERSECT(S, P, edge_id);
                OUTPUT(out_v, I, out_length);
            end
        S := P;
    end                                { konec vnitřní smyčky }
end {clip};
```

Příkaz *INSIDE* je kontrolou polohy bodu vůči dané hranici. Příkaz *INTERSECT* pak je výpočtem průsečíku podle vzorce ve tvaru rovnic (5a-d):

$$t = \frac{a_i \pm 1}{a_i - a_{i+1}}$$

a poté přiřazení dle rovnic (6) do složek vrcholu:

$$x_0 = (1-t)X_i + tX_{i+1}$$
$$y_0 = (1-t)Y_i + tY_{i+1}$$
$$w_0 = 1$$

Výpočet jediného průsečíku tedy zahrnuje tyto operace:

Tabulka 2.1 – počet operací s čísly s plovoucí desetinnou čárkou při výpočtu průsečíku metodou Sutherland-Hodgman

operace	=	<	±	*	/
počet	4	0	6	4	1

Počet počítaných průsečíků se pro jedinou hranici může výrazně lišit. Závisí samozřejmě jak na poloze hranice, tak také na pořadí ořezávání jednotlivými hranicemi. Při velkém množství náhodných vzorků však bude rozdíl v pořadí ořezávání zanedbatelný.

Příkaz *OUTPUT* je přiřazením bodu a inkrementací (celočíselného) čítače. Dále je třeba vést v patrnosti, že přiřazení bodu zahrnuje přiřazení jeho všech tří souřadnic.

V závislosti na polohách bodů *S* a *P* můžeme rozlišit čtyři případy průchodu algoritmem:

1. *S* i *P* leží uvnitř okna
2. *S* leží uvnitř, *P* vně
3. *S* leží vně, *P* uvnitř
4. *S* i *P* leží vně

Pro každý z těchto případů vytvoříme jeden řádek tabulky počtu operací pro jeden průchod vnitřní smyčky `for` algoritmu.

Tabulka 2.2 – počet operací při průchodu vnitřní smyčkou algoritmu Sutherland-Hodgman

operace	*počet operací dle případu*			
	1	*2*	*3*	*4*
<	2	2	2	2
=	9	13	16	6
±	0	6	6	0
*	0	4	4	0
/	0	1	1	0

Hodnoty v této tabulce využijeme později při porovnávání výpočetní náročnosti algoritmů.

Metoda s vektorovým součinem

Jak již bylo uvedeno, provede algoritmus nejprve ohodnocení vrcholů ořezávaného trojúhelníka Cohen-Sutherlandovým kódem, a to prostým porovnáním patřičných složek souřadnic se souřadnicemi hranic okna. Pokud je vrchol $x = [\mathrm{x}, \mathrm{y}]^{\mathrm{T}}$ uvnitř okna, platí:

$$\begin{aligned} x_{\min} \le x \le x_{\max} \\ y_{\min} \le y \le y_{\max} \end{aligned} \tag{7}$$

Do nerovnic (7) dosadíme hranice okna dle obr. 9:

$$\begin{aligned} -1 \le x \le 1 \\ -1 \le y \le 1 \end{aligned} \tag{8}$$

Pokud budou vrcholy trojúhelníka zadány v homogenních souřadnicích –
$x = [\mathrm{x}, \mathrm{y}, \mathrm{w}]^{\mathrm{T}},\ w > 0$, bude to pro výpočet znamenat drobnou úpravu nerovnic (8):

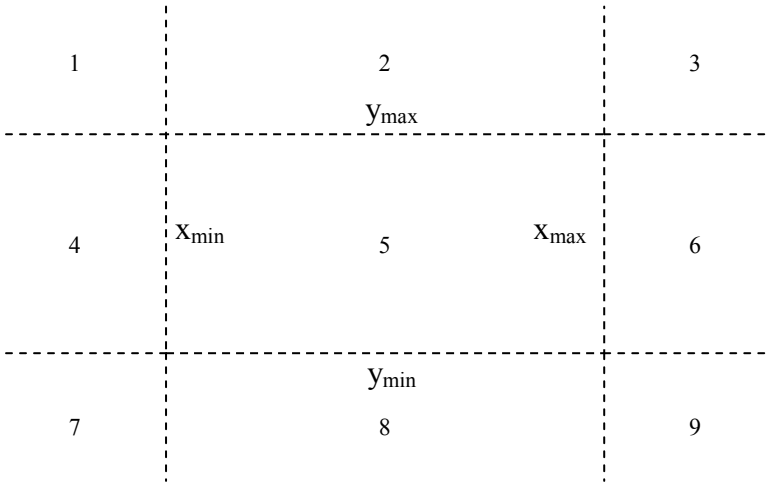
$$-1 \le \frac{x}{w} \le 1 \qquad -1 \le \frac{y}{w} \le 1$$
$$-w \le x \le w \qquad -w \le y \le w \tag{9}$$

Pro tuto akci nám tedy stačí velmi jednoduchý pseudokód, implementující nerovnice (9):

Algoritmus 3 – vytvoření C-S kódu pro koncový bod úsečky

```
function code (x);
begin
    c := [0000];
    if x < -w then c := [1000];
        else if x > w then c: = [0100];
    if y < -w then c := c | [0001];
        else if y > w then c: = c | [0010];
    code := c;
end;
```

Uvažujme nyní možnou polohu vrcholů v segmentech vzhledem k hranicím okna (obr. 10).



Obr. 10 – segmenty polohy vrcholů trojúhelníka

V segmentech 2, 3, 5 a 6 bude potřeba k určení kódu čtyř porovnání čísel s plovoucí desetinnou čárkou (případ a_1), v segmentech 1, 4, 8 a 9 tří (a_2) a v segmentu 7 pouze dvou (a_3).

Algoritmus pokračuje ořezáváním jednotlivých hran trojúhelníka. Pro ořezávání uvažujme následující pseudokód, převzatý ze [Skal05 – algoritmus 4]. Pro přehlednost v něm nejsou zahrnuty operace ukládání indexu incidující hrany okna u průsečíku, což neovlivní výsledek, protože se jedná o celočíselnou operaci.

Algoritmus 4 – ořezání úsečky metodou vektorového součinu s využitím homogenních souřadnic

```
procedure clip;
begin
    cA := A.code; cB := B.code; { hodnoty kódů jsou pro jednotlivé
                                  vrcholy již spočítány }
    if (cA | cB) = 0 then begin
        output(xA; xB); exit;
    end;
    if (cA & cB) ≠ 0 then exit; { & je bitový součin }
    p := xA x xB;               { vektorový součin }
    for k := 0 to 3 do
        if pT xk ≥ 0 then ck := 1 else ck := 0;
    if c = [0000]T or c = [1111]T then exit;
    i := TAB1[c]; j := TAB2[c];
    if cA ≠ 0 and cB ≠ 0 then begin
        xA := p x ei; xB := p x ej;
    end else
        if cA = 0 then begin
            if cB & MASK[c] ≠ 0 then xB := p x ei;
            else xB := p x ej;
        end else if cB = 0 then begin
            if cA & MASK[c] ≠ 0 then xA := p x ei;
            else xA := p x ej;
        end;
    end;
    output(xA; xB);
end
```

Operace vektorového součinu jsme již vyjádřili ve formě rovnic (2) a (3a-d). Nyní se zaměříme na operaci skalárního součinu:

$\vec{p}^T \vec{x}_k \geq 0$

Vlastnosti okna (viz obr. 9) opět umožní zjednodušení operace:

$\vec{p}^T \vec{x}_0 = [a,b,c] \cdot [-1,-1,1]^T = -a - b + c$

$\vec{p}^T \vec{x}_1 = [a,b,c] \cdot [1,-1,1]^T = a - b + c$

$\vec{p}^T \vec{x}_2 = [a,b,c] \cdot [1,1,1]^T = a + b + c$

$\vec{p}^T \vec{x}_3 = [a,b,c] \cdot [-1,1,1]^T = -a + b + c$

Pak můžeme stanovit průměrný počet operací s čísly s plovoucí desetinnou čárkou při ořezávání úseček jednoho trojúhelníka takto:

1. Úsečka leží zcela vně okna: algoritmus ořezávání skončí buďto ihned (případ *1a*), nebo provede jednu operaci vektorového součinu a tři zjednodušené operace skalárního součinu – vše nad vektory s desetinnými čísly (případ *1b*).
2. Úsečka leží zcela uvnitř okna: algoritmus ořezávání pouze přiřadí výstupní body a skončí (případ *2*)

3. Úsečka má jeden průsečík: provede se jedna operace obecného vektorového, jedna operace zjednodušeného vektorového (výpočet průsečíku) a tři operace skalárního součinu. (případ *3*)
4. Úsečka má dva průsečíky: provede se jedna operace obecného vektorového, dvě operace zjednodušeného vektorového (průsečíky) a tři operace skalárního součinu. (případ *4*).

Na každý vygenerovaný průsečík také připadají tři operace přiřazení. Zjednodušený vektorový součin je navíc za podmínek ilustrovaných na obr. 9 ve tvaru podle rovnic (3a-d).

Pokud tedy provedeme úhrn operací pro celý dosavadní postup ořezání, získáme následující údaje:

Tabulka 3 – odhad průměrného počtu operací s čísly s plovoucí desetinnou čárkou pro algoritmus s vektorovým součinem

operace	*počet operací dle případu*				
	1a	*1b*	*2*	*3*	*4*
<	1	1	1	1	1
=	0	6	6	12	12
±	0	9	0	11	13
*	0	6	0	6	6

Posledním krokem je vykonání Algoritmu 1 (viz str. 16). Ten z desetinných operací zahrnuje pouze operace přiřazení, a to tři pro každý vrchol výsledného polygonu. Počet vrcholů je samozřejmě závislý na poloze trojúhelníka.

Je také nutné podotknout, že postup zahrnuje ještě řadu dalších celočíselných operací, zpravidla přiřazení, porovnání či inkrementace. Jejich počet je však jen velmi obtížně vyčíslitelný, navíc jsou oproti operacím v plovoucí desetinné čárce výrazně rychlejší. Přesto je však odhad založený na těchto údajích třeba brát pouze jako spodní teoretickou hranici časové náročnosti.

Srovnání výpočetní náročnosti algoritmů

Podrobný rozbor porovnávaných algoritmů odhalil, že je značně komplikované určit průměrnou hodnotu počtu operací, a tím v hodnotě blízké střední srovnat jejich časovou náročnost. Je to způsobeno silným vlivem postavení vrcholů ořezávaných trojúhelníků na rychlost zpracování. Konfigurací vrcholů je navíc velké množství. Přesné vyčíslení prostřednictvím rozboru jednotlivých případů a jejich pravděpodobností je tudíž takřka nemožné.

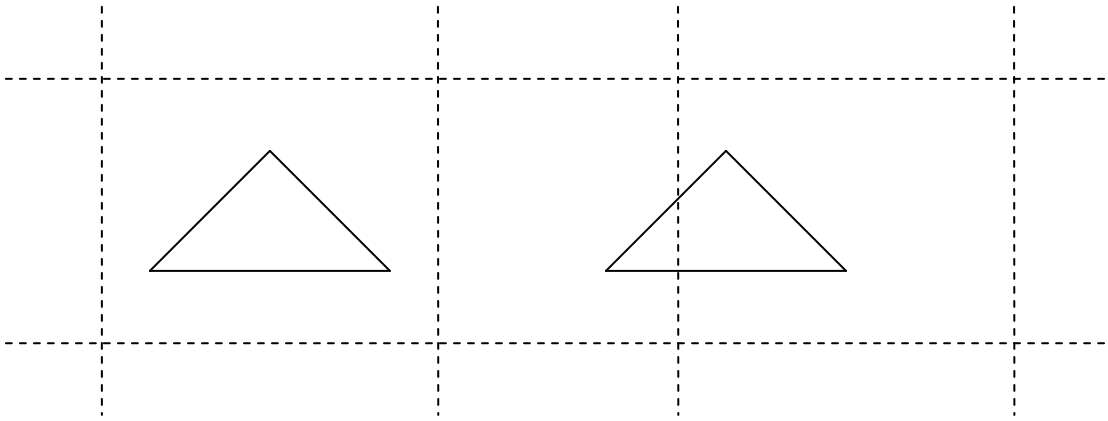
Zaměříme se tedy na několik modelových případů, které odhalí výhody a nevýhody jednotlivých algoritmů. Získáme tak hrubý odhad rozpětí, v němž se pohybuje vzájemný poměr rychlosti zpracování.

Zjevná slabina standardního Sutherland-Hodgmanova algoritmu spočívá v nutnosti počítat i průsečíky, které neleží přímo na hranicích okna, ale dále na přímkách, které tyto hranice určují. S rostoucím počtem takových průsečíků tedy u něj lze očekávat postupný pokles výkonnosti.

Pokud vezmeme v úvahu speciální případ trojúhelníka ležícího zcela mimo okno, je dalším faktorem rychlosti Sutherland-Hodgmanova algoritmu počet hranic, vůči kterým musí být vrcholy trojúhelníka testovány, než jsou zamítnuty. Dokonce i v této situaci mohou být zbytečně počítány průsečíky, které zpracování dále zpomalí.

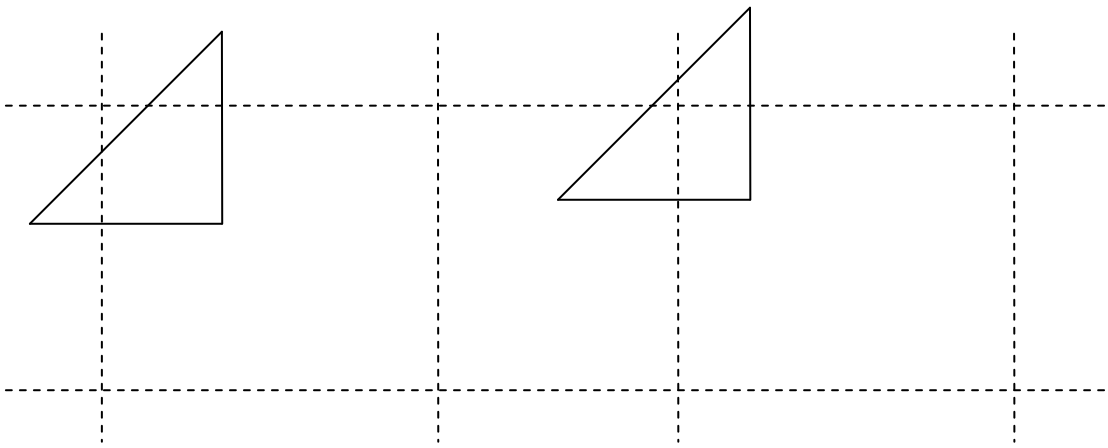
V našem srovnání budeme ořezávat postupně levou pravou, dolní a horní hranicí.

Rozebírané případy jsou ilustrovány na následujících obrázcích.



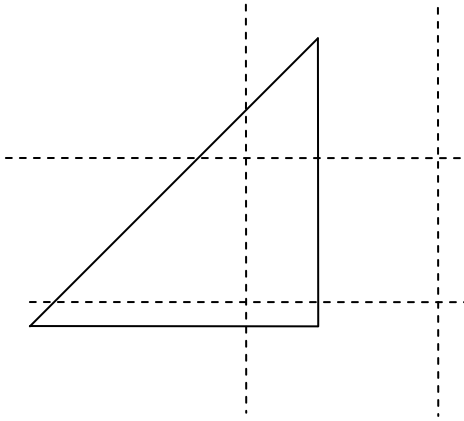
Obr. 11a – případ A polohy trojúhelníka

Obr. 11b – případ B

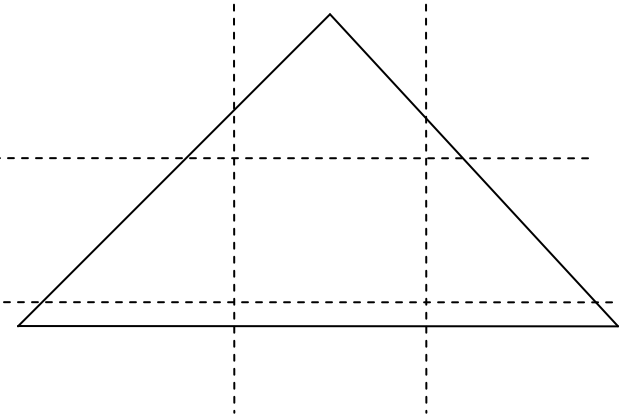


Obr. 11c – případ C

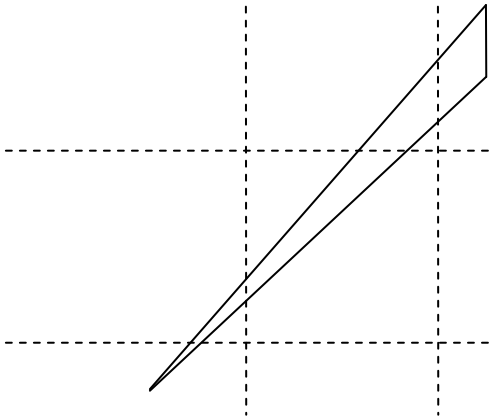
Obr. 11d – případ D



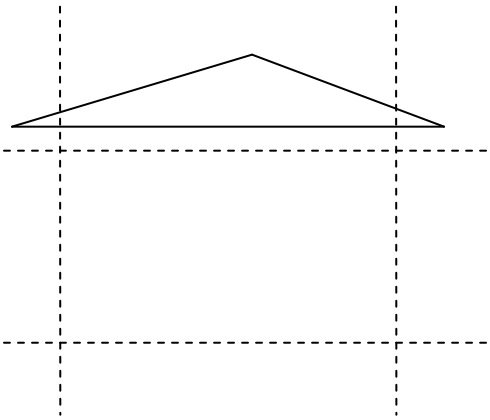
Obr. 11e – případ E



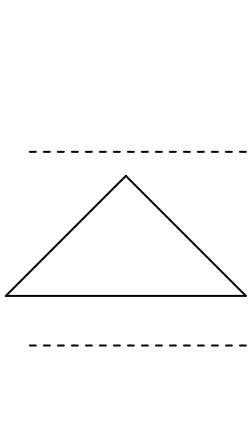
Obr. 11f – případ F



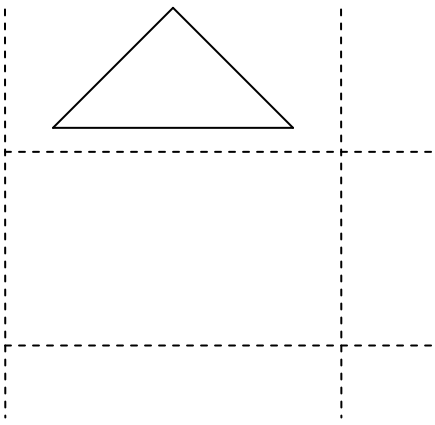
Obr. 11g – případ G



Obr. 11h – případ H



Obr. 11i – případ I



Obr. 11j – případ J

Teoretické srovnání

Provedený rozbor vlastností algoritmů implikuje nejvyšší relativní rychlost algoritmu Sutherland-Hodgman v případě *I*, nejnižší pak v případech *H* či *F*. Pro tyto případy tedy provedeme teoretické srovnání výpočetní náročnosti.

Uvažujme nyní časovou náročnost podle [Skal05], tab. 3, přičemž přidáme relativní časovou náročnost operace s čísly s plovoucí čárkou (vztaženo k „nejlevnější“operaci, tj. přiřazení):

Tabulka 4 – počet cyklů procesoru 750MHz a relativní časová náročnost operací

operace	=	<	±	*	/
počet cyklů	16	149	31	31	78
relativní náročnost	1	9,3125	1,9375	1,9375	4,875

Pro zmíněné případy tedy pomocí hodnot z této tabulky vyjádříme součet relativní časové náročnosti všech operací, přičemž náročnost algoritmu Sutherland-Hodgman označíme $t_{r\text{-}SH}$, náročnost algoritmu s vektorovým součinem $t_{r\text{-}C}$. Dále stanovíme poměr těchto náročností jako rychlost S-H algoritmu vůči rychlosti algoritmu s vektorovým součinem. Tato hodnota je rovná i podílu absolutních časových náročností t_{SH} a t_C, resp. jejich aritmetickému průměru:

$$v_i = \frac{t_{(r-SH)i}}{t_{(r-C)i}} = \frac{t_{(SH)i}}{t_{(C)i}} = \frac{\frac{1}{m}\sum_m \left(t_{(SH)i}\right)_m}{\frac{1}{n}\sum_n \left(t_{(C)i}\right)_n} \qquad (10)$$

Případ I

Tabulka 5.1.1 – odhad počtu operací případu *I*

operace		=	<	±	*	/
počet operací	*SH*	21	6	0	0	0
	C	0	9	0	0	0

Tabulka 5.1.2 – odhad relativní časové náročnosti pro případ *I*

operace	=	<	±	*	/	∑
$t_{(r\text{-}SH)I}$	21	55,875	0	0	0	76,875
$t_{(r\text{-}C)I}$	0	83,8125	0	0	0	83,8125

Poměr náročnosti dle vzorce (10) je tedy:

$$v'_I = \frac{t_{(r-SH)I}}{t_{(r-C)I}} \doteq 0{,}92$$

Vidíme, že standardní Sutherland-Hodgmanův algoritmus je zde teoreticky rychlejší. Slabinou algoritmu s vektorovým součinem je zde vytváření C-S kódu vrcholů ořezávaného trojúhelníka, protože pro každý vrchol provádí tři porovnání. Naproti tomu algoritmus Sutherland-Hodgman zamítne body ihned po prvním porovnání, tj. s levou hranicí. Je zřejmé, že se z hlediska tohoto algoritmu jedná o optimální případ.

Případ H

Tabulka 5.2.1 – odhad počtu operací případu *H*

operace		=	<	±	*	/
počet operací	*SH*	172	34	24	16	4
	C	0	12	0	0	0

Tabulka 5.2.2 – odhad relativní časové náročnosti pro případ *H*

operace	=	<	±	*	/	∑
$t_{(r\text{-}SH)H}$	172	316,625	46,5	31	19,5	585,625
$t_{(r\text{-}C)H}$	0	111,75	0	0	0	111,75

Poměr náročnosti dle vzorce (10):

$$v'_H = \frac{t_{(r-SH)H}}{t_{(r-C)H}} \doteq 5{,}24$$

Případ F

Tabulka 5.3.1 – odhad počtu operací případu *H*

operace		=	<	±	*	/
počet operací	*SH*	200	34	48	32	8
	C	24	9	18	12	0

Tabulka 5.3.2 – odhad relativní časové náročnosti pro případ *H*

operace	=	<	±	*	/	∑
$t_{r\text{-}SH}$	200	316,625	93	62	39	710,625
$t_{r\text{-}C}$	24	83,8125	34,875	23,25	0	165,9375

Poměr náročnosti dle vzorce (10):

$$v'_F = \frac{t_{(r-SH)F}}{t_{(r-C)F}} \doteq 4{,}28$$

Jak uvidíme při praktickém srovnání, je v případech *H* i *F* tento teoretický odhad nereálný. Zřejmě se jedná o vliv do výpočtu nezahrnutých celočíselných operací a dalších zdržení programu, např. při volání funkcí. Jelikož vypočtený poměr náročnosti je v obou případech velmi vysoký, má zde značný vliv i poměrně malé ovlivnění relativní náročnosti těmito dalšími činnostmi.

Pokud připustíme nárůst celkové relativní náročnosti obou algoritmů v případě *H* o 10 bodů, poměr poklesne ze stávající hodnoty $v_H = 5{,}24$ na hodnotu

$v'_{H2} = \frac{t_{(r-SH)H} + 10}{t_{(r-C)H} + 10} \doteq 4{,}89$, což je pokles o více než 6%.

Pokud budeme náročnost obou algoritmů dále zvyšovat, bude klesat i poměr:

$v'_{H3} = \frac{t_{(r-SH)H} + 30}{t_{(r-C)H} + 30} \doteq 4{,}34$, tedy pokles celkem už o 17%.

Vliv zpoždění t_i v případě i můžeme z teoretické hodnoty v_i' ***průměru*** $t_{(SH)i}$ a $t_{(C)i}$ naměřených hodnot vypočítat. Platí:

$$v_i' = \frac{t_{(SH)i} - t_i}{t_{(C)i} - t_i}$$

úpravou tedy získáme:

$$\begin{aligned}
&\left(t_{(C)i} - t_i\right)v_i' = t_{(SH)i} - t_i \\
&t_{(C)i}v_i' - t_i v_i' = t_{(SH)i} - t_i \\
&t_i\left(1 - v_i\right) = t_{(SH)i} - t_{(C)i}v_i' \\
&t_i = \frac{t_{(SH)i} - t_{(C)i}v_i'}{1 - v_i'} \qquad (11)
\end{aligned}$$

Praktické srovnání

Do realizovaných algoritmů je začleněn výpis času počátku a konce samotného výpočtu. Je tedy možné sledovat i praktické výsledky. Vzhledem k rychlosti samotného ořezávání bohužel nebylo možné implementovat zcela přesnou indikaci času jím vyžadovaného. Časy uvedené aplikací tedy navíc zahrnují např. volání či řadu celočíselných operací.

Srovnání bylo prováděno na jedenácti datových množinách. Prvních deset z nich představuje opakovaně zpracovávané trojúhelníky reprezentující případy na obr. 11a-j. Každé z těchto množin odpovídá vstupní soubor *inputX.txt*, kde *X* je nahrazeno velkým písmenem příslušného případu (tj. *A-J*). Přestože by bylo možné modifikovat program ke zpracování jediného trojúhelníka daným počtem iterací, je v souboru uloženo množství trojúhelníků shodných. Jedenáctou datovou množinou jsou náhodné trojúhelníky, jejichž všechny vrcholy leží v intervalu:
$x, y \in \langle -2,2 \rangle$ a jsou uloženy v souboru *input.txt*.
Ve všech případech bylo zpracováváno $\mathbf{10^5}$ trojúhelníků**.**

Pro všechny soubory dat bylo provedeno deset měření, aby byl eliminován vliv různých nežádoucích faktorů, jako aktivity procesů na pozadí, vyrovnávacích pamětí apod. Výsledky s výraznou odchylkou byly z dalšího zpracování vypuštěny (takové vzorky dat jsou v tabulce označeny kurzívou. Ze získaných hodnot je vypočten průměr (zaokrouhlený na celé číslo), který je dosazován do vzorce (10). Měření probíhalo na počítači vybaveném procesorem Intel Pentium 4 2.8GHz, 1MB cache a 2GB operační paměti. Jedná se o stroj umístěný v laboratoři KIV UL407.

Následují tabulky zobrazující naměřené časy pro jednotlivé případy *A-J* (tab. 6.1-6.10), poslední tabulka (6.11) obsahuje časy získané při zpracování náhodných trojúhelníků. Hodnoty $t_{r\text{-}SH}$ představují čas dosažený Sutherland-Hodgmanovým algoritmem, hodnoty $t_{r\text{-}C}$ čas spotřebovaný algoritmem s vektorovým součinem. Hodnoty dif_i udávají odchylku měření pro danou metodu od její průměrné hodnoty. V přidaném sloupci „∅“ je průměrný čas dosažený danou metodou, tučně je označena průměrná hodnota, spočítaná z dat očištěných o dílčí hodnoty s odchylkou dif_X větší než ±5%. Ke každé tabulce je připojena hodnota relativní časové náročnosti vypočtená dle vzorce (10) a zaokrouhlená na dvě desetinná místa.

Tabulka 6.1 – časy naměřené pro případ *A* (*obr. 11a*) – soubor *inputA.txt*

měření	1	2	3	4	5	6	7	8	9	10	∅
$t_{(SH)A}$ [ms]	379	379	379	379	*404*	392	379	379	379	379	383
dif_{SH} [%]	-1	-1	-1	-1	*5,5*	2,3	-1	-1	-1	-1	**380**
$t_{(C)A}$ [ms]	256	256	256	257	257	256	257	257	*305*	269	263
dif_C [%]	-2,7	-2,7	-2,7	-2,3	-2,3	-2,7	-2,3	-2,3	*16*	2,3	**258**

Z výpočtu vyřadíme hodnotu *5* veličiny $t_{SH(A)}$ a hodnotu 9 veličiny $t_{(C)A}$ kvůli výrazné odchylce:

$$v_A = \frac{t_{(SH)A}}{t_{(C)A}} = \frac{380}{258} \doteq 1{,}47$$

Tabulka 6.2 – časy naměřené pro případ *B* (*obr. 11b*) – soubor *inputB.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)B}$ [ms]	428	428	429	*465*	*489*	416	428	415	428	428	435
dif_{SH} [%]	-1,6	-1,6	-1,4	*6,9*	*12,4*	-4,4	-1,6	-4,6	-1,6	-1,6	**425**
$t_{(C)B}$ [ms]	*367*	318	318	318	318	306	318	*367*	*367*	318	332
dif_C [%]	*10,5*	-4,2	-4,2	-4,2	-4,2	-7,8	-4,2	*10,5*	*10,5*	-4,2	**316**

I zde jsou patrné výrazné odchylky – u měření $t_{(SH)B}$ vyřadíme hodnoty *4* a *5*, u měření $t_{(C)B}$ hodnoty *1*, *8* a *9*. Vypustíme-li je z výpočtu, vyjde poměr rychlostí:

$$v_B = \frac{t_{(SH)B}}{t_{(C)B}} = \frac{425}{316} \doteq 1{,}34$$

Tabulka 6.3 – časy naměřené pro případ *C* (*obr. 11c*) – soubor *inputC.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)C}$ [ms]	525	538	538	526	526	538	563	539	538	538	537
dif_{SH} [%]	-2,2	0,2	0,2	-2	-2	0,2	4,8	0,4	0,2	0,2	**537**
$t_{(C)C}$ [ms]	367	367	379	367	391	403	*428*	*427*	367	367	385
dif_C [%]	-4,7	-4,7	-1,6	-4,7	1,6	4,7	*11,2*	*10,9*	-4,7	-4,7	**376**

Po vypuštění hodnot 7 a 8 veličiny $t_{(C)C}$ získáme:

$$v_C = \frac{t_{(SH)C}}{t_{(C)C}} = \frac{537}{376} \doteq 1{,}43$$

Tabulka 6.4 – časy naměřené pro případ *D* (*obr. 11d*) – soubor *inputD.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)D}$ [ms]	538	526	526	538	538	538	538	538	526	538	534
dif_{SH} [%]	0,7	-1,5	-1,5	0,7	0,7	0,7	0,7	0,7	-1,5	0,7	**534**
$t_{(C)D}$ [ms]	318	330	318	318	*391*	318	318	*375*	319	318	332
dif_C [%]	-4,2	-0,6	-4,2	-4,2	*17,8*	-4,2	-4,2	*13*	-3,9	-4,2	**320**

Vypustíme hodnoty *5* a *8* veličiny $t_{(C)D}$:

$$v_D = \frac{t_{(SH)D}}{t_{(C)D}} = \frac{534}{320} \doteq 1{,}67$$

Tabulka 6.5 – časy naměřené pro případ *E* (*obr. 11e*) – soubor *inputE.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)E}$ [ms]	691	692	691	691	677	*749*	720	692	692	692	699
dif_{SH} [%]	-1,1	-1	-1,1	-1,1	-3,1	*7,2*	3	-1	-1	-1	**693**
$t_{(C)E}$ [ms]	360	361	360	360	360	375	361	361	375	375	365
dif_C [%]	-1,4	-1,1	-1,4	-1,4	-1,4	2,7	-1,1	-1,1	2,7	2,7	**365**

Bez hodnoty $t_{(SH)E}$ č. *6* vyjde:

$$v_E = \frac{t_{(SH)E}}{t_{(C)E}} = \frac{693}{365} \doteq 1{,}89$$

Tabulka 6.6 – časy naměřené pro případ *F* (*obr. 11f*) – soubor *inputF.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)F}$ [ms]	792	778	821	778	792	*850*	792	777	*849*	778	801
dif_{SH} [%]	-1,1	-2,9	2,5	-2,9	-1,1	*6,1*	-1,1	-3	*6*	-2,9	**789**
$t_{(C)F}$ [ms]	317	317	317	317	*360*	317	317	317	331	317	323
dif_C [%]	-1,9	-1,9	-1,9	-1,9	*11,5*	-1,9	-1,9	-1,9	2,5	-1,9	**319**

Opět tedy vynecháme silně odlišné hodnoty, č. *6* a *9* veličiny $t_{SH(A)}$ a č. *5* veličiny $t_{(C)A}$:

$$v_F = \frac{t_{(SH)F}}{t_{(C)F}} = \frac{789}{319} \doteq 2{,}47$$

Porovnáme-li tento výsledek s odhadovaným poměrem náročností v'_F, vidíme, že se jedná o hodnotu o cca 40% nižší. Vyjádříme tedy hodnotu zpomalení t_H dle vzorce (11):

$$t_F = \frac{t_{(SH)F} - t_{(C)F} v'_F}{1 - v'_F} = \frac{789 - 319 \cdot 4{,}28}{1 - 4{,}28} \doteq 175{,}71$$

Tato hodnota po přepočtu dle tab. 4 odpovídá přibližně 2800 cyklům.

Tabulka 6.7 – časy naměřené pro případ *G* (*obr. 11g*) – soubor *inputG.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)G}$ [ms]	677	691	692	*749*	735	677	677	691	735	677	700
dif_{SH} [%]	-3,3	-1,3	-1,1	*7*	5	-3,3	-3,3	-1,3	5	-3,3	**695**
$t_{(C)G}$ [ms]	432	432	432	417	432	432	432	432	432	432	431
dif_C [%]	0,2	0,2	0,2	-3,2	0,2	0,2	0,2	0,2	0,2	0,2	**431**

Vynecháme pouze hodnotu *4* $t_{(SH)G}$ a získáme tak poměr:

$$v_G = \frac{t_{(SH)G}}{t_{(C)G}} = \frac{695}{431} \doteq 1{,}61$$

Tabulka 6.8 – časy naměřené pro případ *H* (*obr. 11h*) – soubor *inputH.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)H}$ [ms]	663	691	676	663	663	663	663	662	677	663	668
dif_{SH} [%]	-0,7	3,4	1,2	-0,7	-0,7	-0,7	-0,7	-0,9	1,3	-0,7	**668**
$t_{(C)H}$ [ms]	288	303	303	303	302	303	*332*	288	303	303	303
dif_C [%]	-4,9	0	0	0	-0,3	0	*9,6*	-4,9	0	0	**300**

V tomto případě z výpočtu vypustíme hodnotu 7 veličiny $t_{(C)H}$, získáme tak:

$$v_H = \frac{t_{(SH)H}}{t_{(C)H}} = \frac{668}{300} \doteq 2{,}23$$

Porovnáme-li tento výsledek s odhadovaným poměrem náročností v'_H, vidíme, že se jedná o hodnotu takřka poloviční. Opět tedy vyjádříme hodnotu zpomalení t_H dle vzorce (11):

$$t_H = \frac{t_{(SH)H} - t_{(C)H} v'_H}{1 - v'_H} = \frac{668 - 300 \cdot 5{,}24}{1 - 5{,}24} \doteq 213{,}21$$

Po přepočtu dle tab. 4 je tedy rozdíl oproti teoretické hodnotě více než 3400 cyklů. Je vidět, že chod algoritmu je zásadně zpomalován i jinými, než desetinnými operacemi.

Tabulka 6.9 – časy naměřené pro případ *I* (*obr. 11i*) – soubor *inputI.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)I}$ [ms]	230	231	*259*	230	231	230	240	230	231	240	235
dif_{SH} [%]	-2,1	-1,7	*10,2*	-2,1	-1,7	-2,1	2,1	-2,1	-1,7	2,1	**233**
$t_{(C)I}$ [ms]	303	302	303	*288*	303	302	319	319	319	319	308
dif_C [%]	-1,6	-1,9	-1,6	*-6,5*	-1,6	-1,9	3,6	3,6	3,6	3,6	**310**

S vypuštěním dvou nadlimitních odchylek – hodnoty 3 u měření $t_{(SH)H}$ a hodnoty u měření $t_{(C)H}$ – získáme:

$$v_I = \frac{t_{(SH)I}}{t_{(C)I}} = \frac{233}{310} \doteq 0{,}75$$

Opět můžeme porovnat s teoretickou hodnotou v'_I. Zde odchylka činí cca 18% v neprospěch nového algoritmu.

Tabulka 6.10 – časy naměřené pro případ *J* (*obr. 11j*) – soubor *inputJ.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)J}$ [ms]	774	774	774	774	774	729	773	728	728	728	756
dif_{SH} [%]	2,4	2,4	2,4	2,4	2,4	-3,6	2,2	-3,7	-3,7	-3,7	**756**
$t_{(C)J}$ [ms]	319	*365*	*349*	319	319	319	319	319	319	318	327
dif_C [%]	-2,4	*11,6*	*6,7*	-2,4	-2,4	-2,4	-2,4	-2,4	-2,4	-2,8	**319**

Pro abnormální odchylky vyřadíme hodnoty *2* a *3* veličiny $t_{(C)J}$:

$$v_J = \frac{t_{(SH)J}}{t_{(C)J}} = \frac{756}{319} \doteq 2{,}37$$

Porovnáme-li tento výsledek s hodnotou předchozího případu v_I, vidíme, že rychlost Sutherland-Hodgmanova algoritmu u trojúhelníka ležícího mimo okno silně závisí na pořadí jednotlivých hranic. Vyjádříme poměr v_{SH} časové náročnosti S-H algoritmu v případě *J* oproti případu *I*:

$$v_{SH} = \frac{t_{(SH)J}}{t_{(SH)I}} = \frac{756}{2{,}33} \doteq 3{,}24$$

Algoritmus je tedy v nejhorším možném případě oproti nejlepšímu více než třikrát pomalejší.

Tabulka 6.11 – časy naměřené pro náhodné trojúhelníky – soubor *input.txt*

měření	1	2	3	*4*	*5*	6	7	8	9	10	∅
$t_{(SH)}$ [ms]	653	653	653	653	637	653	653	638	652	638	649
dif_{SH} [%]	0,6	0,6	0,6	0,6	-1,8	0,6	0,6	-1,7	0,5	-1,7	**649**
$t_{(C)}$ [ms]	441	441	441	425	441	426	441	441	441	441	438
dif_C [%]	0,7	0,7	0,7	-3	0,7	-2,7	0,7	0,7	0,7	0,7	**438**

V případě náhodných trojúhelníků jsou naměřené údaje vzácně konzistentní:

$$v = \frac{t_{(SH)}}{t_{(C)}} = \frac{649}{438} \doteq 1{,}48$$

Implementace

Součástí práce je implementace obou srovnávaných algoritmů. Jedná o konzolové aplikace pracující s daty ve vstupním textovém souboru, výstup, je-li povolen, je taktéž směrován do textového souboru (viz přílohy). Algoritmy jsou implementovány v jazyce C v prostředí Microsoft Visual Studio .NET 2003, později byly převedeny do Microsoft Visual C++ 2005 Express Edition; v obou případech bylo použito standardní nastavení.

Z využití kompilátoru Microsoft plyne, že aplikace nejsou napsány v souladu s normou ANSI C – pro její dodržení by byla nezbytná řada úprav, např. hlavičkového souboru, odstranění deklarací řídících proměnných cyklu přímo v místě užití, či některých dalších vlastností specifických pro překladač Microsoft C++, jako je využití speciálních verzí zabezpečených knihoven funkcí.

Cílem programů je praktické srovnání standardní verze Sutherland-Hodgmanova algoritmu (program *sh*) s algoritmem vektorového součinu (program *c*). Jsou tedy upraveny pro zvýšení rychlosti, následkem čehož značně utrpěla jejich přehlednost – jsou využívány inline funkce, části kódu se často s drobnými změnami opakují. Je to však daní za co nejobjektivnější zhodnocení poměru rychlostí realizovaných algoritmů.

Důležitým faktorem ovlivňujícím korektnost porovnání algoritmů je optimalizace kódu prováděná překladačem. Jeho zásahy jsou natolik rozsáhlé, že jakékoliv výsledky s povolenou optimalizací překladu jsou naprosto zavádějící. Proto je nezbytné, aby při překladu algoritmů byly optimalizace vypnuty. V projektech obsahujících zdrojové kódy aplikací je toto nastavení provedeno. Zásadní podmínkou, plynoucí z rozboru úlohy, je také skutečnost, že **zpracovávané trojúhelníky uložené ve vstupním souboru musejí být orientovány proti směru hodinových ručiček**, jinak program *C* nepodává konkrétní výsledky. Opačně orientovaný trojúhelník by doplňoval podle Algoritmu 1 (viz str. 16), tedy ve směru orientace okna.

Během vývoje těchto aplikací byly také podniknuty pokusy směřující k odstranění větvení prostřednictvím programové konstrukce switch, zde však nebyl zjištěn měřitelný výkonnostní rozdíl.

Detailní popis implementace algoritmů naleznete v příloze B této práce.

Závěr

Práce shrnuje a představuje základní principy algoritmů pro ořezávání grafických primitiv. Odtud vychází specializace pro případ ořezávání trojúhelníka normalizovanou oblastí a společně s aplikací vlastností prostoru homogenních souřadnic srovnává standardní implementaci Sutherland-Hodgmanova algoritmu s algoritmem založeným na zdrojích dodaných vedoucím práce – tzv. algoritmem s vektorovým součinem.

Algoritmy byly srovnány jak teoreticky, tak i praktickým testováním na obsáhlých množinách vstupních dat. Výsledky praktické se neshodují zcela s teoretickými hodnotami. Na základě naměřených výsledků byly potvrzeny předpoklady o spodní hranici výkonnosti algoritmu s vektorovým součinem, zdaleka však nebylo dosaženo teoretických údajů o horní hranici výkonnosti. Je zde tedy prostor pro optimalizaci implementace tohoto algoritmu. I tak jsou ale výsledky tohoto algoritmu velmi dobré – v nejlepším změřeném případě je Sutherland-Hodgman algoritmus zhruba 2,5násobně pomalejší. Ve srovnání výkonnosti zpracování náhodných dat, které se může blížit řadě praktických aplikací, vyžaduje S-H algoritmus o téměř 50% více času. Jedinou odhalenou slabinou je specifický případ, kdy je algoritmem S-H trojúhelník ihned zamítnut testem vůči první hranici okna. Zde nový algoritmus dosahuje pouze zhruba 75% rychlosti algoritmu S-H.

Pro Sutherland-Hodgmanův algoritmus byla předvedena jeho významná slabina, zásadní vliv pořadí aplikace jednotlivých hranic okna zejména při specifických případech ořezávání. Časová náročnost nejhoršího případu vzhledem k nejpříznivějšímu je více než trojnásobná.

Při výpočtu polohy vrcholů ořezaných trojúhelníků se mohou vyskytovat drobné chyby vzniklé kvůli nezbytné nepřesnosti procesoru při dělení, nejsou však významné, protože se nacházejí na řádu 10^{-17}. V obou programech byla pro co nejpřesnější srovnání výkonnosti variant Sutherland-Hodgmanova algoritmu provedena řada úprav pro zvýšení rychlosti jejich běhu.

V budoucnu připadá v úvahu rozšíření algoritmu do další dimenze na základě probíhajících výzkumných aktivit vedoucího práce. Rovněž by bylo vhodné přistoupit k další optimalizaci implementace algoritmu s vektorovým součinem a dále se tak přiblížit k výsledkům teoretické části práce.

Použitá literatura

[Žára04] J. Žára, B. Beneš, J. Sochor, P. Felkel. *Moderní počítačová grafika*. Computer press, 2. vydání, 2004

[Bui99] D.H.Bui, Algorithms for Line Clipping and Their Complexity – dissertation thesis, 1999

[Spro68] R.F. Sproull, I.E. Sutherland. A clipping divider. *AFIPS Conference Proceedings*, vol.33, p765-776, 1968

[Cyru83] M.Cyrus, J. Beck. Generalized two- and three dimensional clipping. *Computers and Graphics*, 12(1):23-28, 1978

[Lian83] Y.-D. Liang, B.A. Barsky. An analysis and algorithm for polygon clipping. *Communications of ACM*, 26(11):868-877, 1983

[Bars84] Y.D. Liang, B.A. Barsky. A New Concept and Method for Line Clipping. *ACM Transactions on Grapgics (TOG)*, 3(1):1-22, 1984

[Suth74] I.E. Sutherland, G.W. Hodgman. Reentrant polygon clipping. *Communications of ACM*, 17(1):32-42, 1974

[Weil77] K.Weiler, P. Atherton. Hidden Surface Removal Using Polygon Area Sorting. *SIGGRAPH '77 Conference Proceedings*, 214-222, 1977

[Skal98] D.H. Bui, V. Skala. New Fast Line Clipping Algorithm in E2 with O(lgN) Complexity. *SCCG'99 Int.Conf. proceedings*, 221-228, 1998

[Zhan02] M. Zhang, Ch.L. Sabharwal. An Efficient Implementation Of Parametric Line And Polygon Clipping Algortihm. *Proceedings of the 2002 ACM symposium on Applied computing*, 796-800, 2002

[Mail92] P.-G. Maillot. A new, fast method for 2D polygon clipping: analysis and software implementation. *ACM Transactions on Graphics (TOG)*, 11(3):276-290, 1992

[Grei98] G. Greiner, K. Hormann. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics (TOG)*, 17(2):71-83, 1998

[Vatt92] B.R. Vatti. A generic solution to polygon clipping. *Communications of ACM*, 35(7):56-63, 1992

[Blin78] J.F. Blinn, M.E. Newell. Clipping using homogeneous coordinates. *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, 245-251, 1978

[Blin77] J.F. Blinn. A homogeneous formulation for lines in 3 space. *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, 237-241, 1977

[Skal05] V. Skala. A new approach to line and line segment clipping in homogeneous coordinates. *Visual Computer* 21:905-914, Springer Verlag 2005

Seznam obrázků, tabulek a algoritmů

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Příloha A – Uživatelská příručka

Spuštění programů

Jednotlivé programy se v dodaném stavu spouští příkazy:
program ***SH***: `sh.exe [vstupní_soubor]`
program ***C***: `c.exe [vstupní_soubor]`

Pokud není přítomen parametr pro vstupní soubor, doplní je program sám standardní hodnotou definovanou v hlavičkovém souboru.

K replikování výsledků je přiložen dávkový soubor `run.bat`, který spustí oba algoritmy nad všemi datovými množinami. Jeho výstup je vhodné směrovat do textového souboru.

Pokud jsou programy přeloženy bez deklarace `NOTEXTOUT` (viz přílohu B), přijímají navíc ještě další volitelný parametr – jméno výstupního souboru. Pokud není uveden, je opět nahrazen standardní hodnotou z hlavičkového souboru. Obsah výstupního souboru není přepisován, ale výstup je připojován na konec souboru.

Formát souborů

Formát vstupního souboru
Jedná se o textový soubor, který na každé řádce obsahuje data jednoho trojúhelníka. Každý řádek musí obsahovat tři trojice reálných čísel, oddělených libovolnou sekvencí znaků daných konstantou `SEPARATORS` definovanou v hlavičkovém souboru. Každá trojice představuje homogenní souřadnici jednoho z vrcholů trojúhelníka, v pořadí x, y, w. Doporučený formát je následující:
$(x_1, y_1, w_1); (x_2, y_2, w_2); (x_3, y_3, w_3)$
Na konci souboru by měla být prázdná řádka.

Pro vstup programu ***C*** je nezbytné, aby vrcholy trojúhelníka byly orientovány shodně s ořezávacím oknem, tj. proti směru hodinových ručiček. V opačném případě algoritmus nepodává korektní výsledky.

Formát výstupního souboru
zachovává doporučený formát vstupního souboru. Jelikož jsou ovšem výstupem ořezávání obecně polygony namísto prostých trojúhelníků, může jedna řádka obsahovat i více trojic reprezentujících souřadnice vrcholů.

Příloha B – Popis implementace

Algoritmy jsou implementovány v prostředí Microsoft C++ 2005 Express Edition. Ve starší verzi Microsoft .NET Studia nelze řešení otevřít. Zahrnuty jsou dva projekty – jeden pro Sutherland-Hodgmanův algoritmus (***SH***), druhý pro algoritmus s výpočtem průsečíku metodou vektorového součinu (***C***).

Všechny potřebné funkce obsahují soubory `sh.cpp`, resp. `c.cpp`, potřebné deklarace konstant a typů jsou obsaženy v hlavičkových souborech `sh.h`, resp. `c.h`.

Hlavičkové soubory

sh.h

Konstanty

`CLIP_LEFT`, `CLIP_RIGHT`, `CLIP_BOTTOM`, `CLIP_TOP` slouží k reprezentaci jednotlivých ořezávacích rovin. Musí se jednat o čísla v rozmezí 0–3. LINE_LENGTH určuje maximální délku řádky ve vstupním souboru.

Pole `SEPARATORS[]` obsahuje znaky oddělovačů pro získání hodnot ze vstupního souboru. `STD_INFILE[]`, `STD_OUTFILE[]` specifikují implicitní název vstupního, resp. výstupního souboru.

Typy

Typ `VERTEX3` představuje bod s vektorem polohy o třech složkách:

```
typedef struct {
      double x; double y; double w;
} VERTEX3;
```

Typ `TRIANGLE3` je pole pro uložení vrcholů trojúhelníka

```
typedef VERTEX3 TRIANGLE3[3];
```

Typ `POLYGON3` je struktura pro uložení dat obecnějšího polygonu. Pole `data` sloužící k uchování jednotlivých vrcholů má kapacitu stanovenou na 7, což je maximální počet vrcholů vzniklý ořezáním trojúhelníka. Položka `length` uchovává počet platných vrcholů polygonu.

```
typedef struct {
      VERTEX3 data[7];
      int length;
} POLYGON3;
```

K práci s více trojúhelníky slouží typ `TRLIST`, který reprezentuje jeden prvek dynamického seznamu trojúhelníků.

```
typedef struct t_TRLIST{
      TRIANGLE triangle;
      t_TRLIST *next;
} TRLIST;
```

c.h

Konstanty

Význam konstant `LINE_LENGTH`, `SEPARATORS[]`, `STD_INFILE[]` a `STD_OUTFILE[]` je shodný se souborem *sh.h*.

Novými konstantami jsou `NONE` a `NA`, reprezentující prázdnou, resp. vyloučenou položku tabulky protínaných hran (viz tab. 1). Třetí novou konstantou je `INSIDE`, která je využívána k uchování informace o poloze vrcholu ořezaného polygonu uvnitř okna, tj. mimo jeho hranice.

Typy

Typ `VERTEX3c` je rozšířenou verzí typu `VERTEX3` ze souboru *sh.h*. Položkou navíc je celočíselná hodnota `code`, sloužící k uchování vektoru polohy bodu vůči hranicím okna. Tato pozměněná implementace vrcholu pochopitelně implikuje i změnu typu pro trojúhelník `TRIANGLE3` a struktury polygonu `POLYGON3`.

Typ `VECTOR3` představuje prostý trojrozměrný vektor desetinných čísel. Stejně jednoduchá je i struktura `LINE3`, sloužící k uchování koncových bodů ořezávané úsečky.

Programové kódy

sh.cpp

Důležitou položkou ovlivňující chod programu jsou deklarace `#define INLINE` a především `#define NOTEXTOUT` . Obě jsou standardně aktivní. První slouží k povolení překladu s nuceným vkládáním těl funkcí ořezávání do místa volání, druhá pak vypíná výstup do textového souboru.

Funkce

```
int freeTrList(TRLIST *list);
```
Slouží k uvolňování paměti alokované pro seznam trojúhelníků odkazovaný ukazatelem `list`. Návratovou hodnotou je počet odstraněných položek seznamu, tato se však používá pouze pro ladicí účely.

```
TRLIST *readTrList(char *filename);
```
Načte ze souboru s názvem `filename` data a vrátí je jako seznam trojúhelníků. Provádí kontrolu formátu souboru, v případě neplatných dat ignoruje celý příslušný trojúhelník, závažnější chyby vedou k přerušení načítání a vrácení prázdného seznamu.

```
int writePoly(FILE *f, POLYGON3 p);
```
Do souboru odkazovaného handle `f` zapíše ve standardním formátu (viz výše) vrcholy polygonu `p`. Tato funkce slouží pro kontrolu správnosti, v běžném stavu je deklarací `NOTEXTOUT` výpis do souboru potlačen.

`bool inside(VERTEX3 v, int clipEdge);`
Testuje, zda bod `v` leží v polorovině příslušné ořezávací rovině dané parametrem `clipEdge`. Hodnota tohoto parametru by měla odpovídat jedné z hodnot konstant `CLIP_*`, tedy ležet v intervalu <0, 3>, v opačném případě je vrácena vždy hodnota `false`. K určení hodnoty využívá (standardně inline) funkcí `insideLeft`, `insideRight`, `insideTop` a `insideBottom`, které pro jednotlivé případy volá. Tělo všech těchto funkcí obsahuje pouze prosté porovnání.

`VERTEX3 intersect(VERTEX3 s, VERTEX3 p, int clipEdge);`
Nalezne a vrátí průsečík úsečky (resp. přímky) dané koncovými body `s` a `p` s hranicí `clipEdge`. Tato funkce předpokládá, že přímka určená těmito body není s ořezávací rovinou rovnoběžná, vyžaduje tedy před voláním testy polohy bodu (funkcí `inside`), jinak dojde k dělení nulou. Pro poslední parametr platí shodná omezení jako u funkce `inside`. Stejně jako tato funkce také pouze „volá“ další (inline) funkce, `intersectLeft`, `intersectRight`, `intersectTop` a `intersectBottom`, jejichž náplní je samotný výpočet průsečíku s konkrétní hranicí.

`POLYGON3 clipEdge(POLYGON3 p, int clipEdge);`
Je jádrem samotného algoritmu. Provede oříznutí polygonu `p` hranicí `clipEdge`. Přímo implementuje Sutherland-Hodgmanův algoritmus – postupně kontroluje polohu bodů jednotlivých hran polygonu, generuje případné průsečíky a výstupní body. Vrací výsledný ořezaný polygon.

`POLYGON3 *clip(POLYGON3 p) {`
Ořeže polygon `p` standardním oknem intervalu <`-1`, `1`>. Jedná se vlastně o zastřešující funkci pro `clipEdge`, kterou volá pro polygon vzniklý ze vstupního trojúhelníku a postupně vznikající částečně ořezané polygony. Vrací kompletně ořezaný polygon.

`int main(int argc, char* argv[]);`
Hlavní funkce se stará o kontrolu parametrů příkazové řádky a volá další výše zmíněné funkce zajišťující činnost programu.

c.cpp

Obsah souboru je z velké části obdobný zdrojovému kódu Sutherland-Hodgmanova algoritmu *sh.cpp*. Proto jsou zde pouze zdůrazněny rozdíly.

Globální proměnné

Narozdíl od *sh.cpp* obsahuje tento zdrojový kód globální proměnné:

```
int TAB1[] = {NONE, 0, 1, 1, 2, NA, 2, 2, 3, 0, NA, 1, 3, 0, 3, NONE};
int TAB2[] = {NONE, 3, 0, 3, 1, NA, 0, 3, 2, 2, NA, 2, 1, 1, 0, NONE};
int MASK[] = {NONE, 1, 4, 4, 2, NA, 2, 2, 8, 1, NA, 4, 8, 1, 8, NONE};
VERTEX3c VERTS[] = {{1, -1, 1, 1}, {1, 1, 1, 2}, {-1, 1, 1, 3},
{-1, -1, 1, 0}};
```

První tři implementují prostými vektory tabulku pro identifikaci hran průsečíků (viz tab.1 v hlavní části práce).
Poslední pak zajišťuje pohodlné doplňování průsečíků získaných ořezáním úseček o vrcholy okna, tedy na kompletní ořezaný polygon.

Funkce

`int getPositionCode(VERTEX3c v);`
Jedná se o zcela novou funkci, která pro vrchol `v` určí Cohen-Sutherlandův kód polohy vůči hranicím okna.

`int getBitLength(int code);`
Určí bitovou délku hrany s kódem `code`.

Následují funkce `intersect`*, které jsou obdobou stejnojmenných funkcí v *sh.cpp*. Rozdíl je pochopitelně ve vnitřní implementaci výpočtu průsečíku. Naopak obdoby funkcí `inside`* chybí, neboť jsou nahrazeny jednoduchými bitovými testy v hlavní funkci algoritmu. Tou je funkce:

`LINE3 *clipLine(LINE3 line);`
Tato funkce ořízne úsečku `line` standardním ořezávacím oknem a vrátí odkaz na ořezanou úsečku, resp. `NULL`, pokud ořezávaná úsečka leží mimo toto okno. Samotný algoritmus je popsán v hlavní části práce jako Algoritmus 4.

`POLYGON3 clipTriangle(TRIANGLE3 t);`
Ořeže trojúhelník `t` standardním ořezávacím oknem. Postupně vytváří z vrcholů trojúhelníka jeho hrany, které předává k ořezání funkci `clipLine`. Z výsledných ořezaných úseček a případně vrcholů ořezávacího okna sestavuje výsledný polygon, implementuje tak Algoritmus 1 popsaný v hlavní části práce.