Řešení viditelnosti

Pavel Strachota

FJFI ČVUT v Praze

7. dubna 2020

Obsah

Obsah

Úvod

- situace:
  - daná scéna
  - daná specifikace pohledu
- problém - zobrazit pouze části viditelné z daného pohledu
  - určení viditelných čar/povrchů (*visible line/surface determination*)
  - odstranění **ne**viditelných (zakrytých) čar/povrchů (*hidden line/surface elimination*)

Základní rozdělení přístupů k řešení viditelnosti 1/2

Obrazový přístup (*image-precision* algorithms)

- řeší, který z objektů je vidět v každém pixelu obrazu

```
foreach(pixel in the image) {
  1. determine the object closest to the viewer
     in the direction of projection;
  2. draw the pixel in the appropriate color;
}
```

- *ray casting* pro řešení viditelnosti (více o něm viz přednáška o raytracingu)
- pracuje na úrovni rozlišení obrazu $\Longrightarrow$ aliasing, nutnost přepočítání např. při zoomování
- teoretická složitost naivního algoritmu je $O(np)$, kde $n =$počet objektů, $p =$počet pixelů

Základní rozdělení přístupů k řešení viditelnosti 2/2

Objektový přístup (*object-precision* algorithms)

- porovnává objekty mezi sebou a eliminuje celé objekty nebo jejich části, které nejsou viditelné

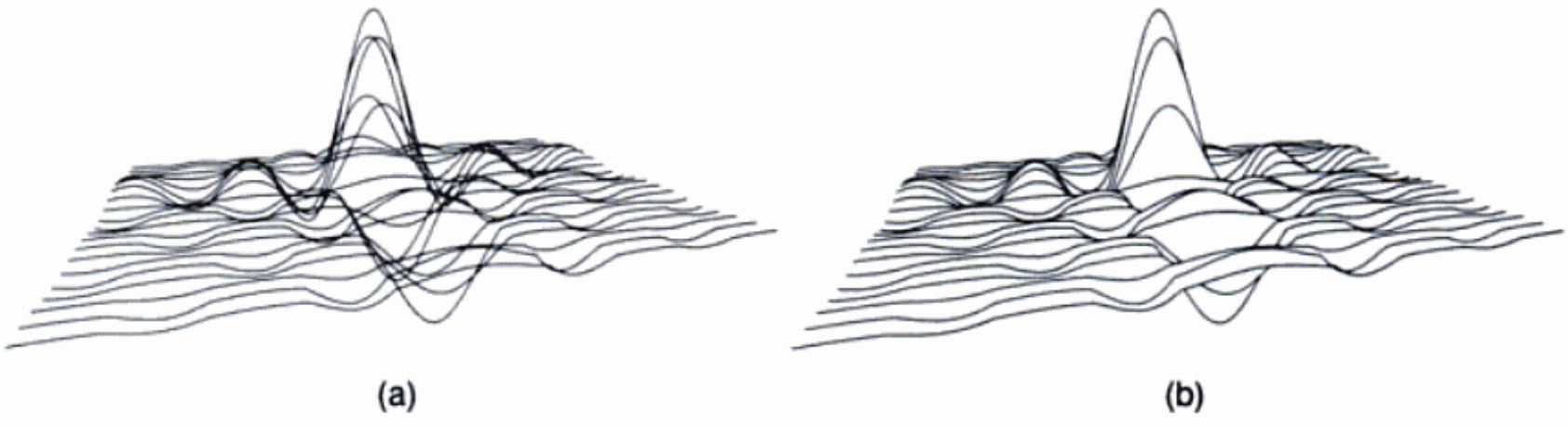
```
foreach(object in the scene) {
  1. determine the unobstructed parts of the
     object, either by other objects or other
     parts of the same object;
  2. draw those parts in the appropriate color;
}
```

- teoretická složitost je $O\left(n^2\right)$, ale i když $n \ll p$, tak kroky 1. a 2. jsou obvykle mnohem náročnější než u obrazového přístupu
- pracuje na úrovni datové struktury scény, nemusí se opakovat při přiblížení (zoomu) scény

Obsah

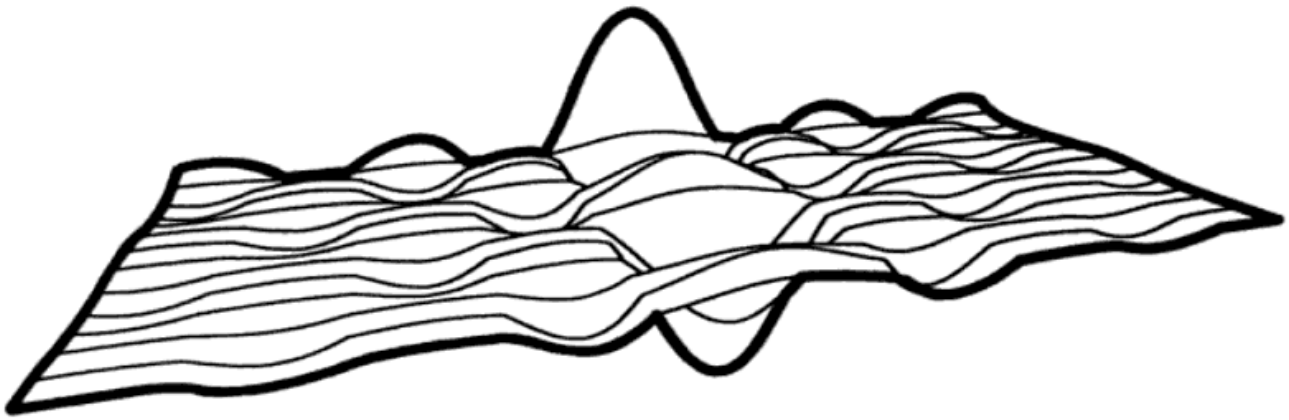
Grafy funkcí dvou proměnných

- nakreslit plochu $y = f(x,z)$
- síť $m \times n$ funkčních hodnot
- spojíme $f(x,z)$ lomenou čarou ve směru konstantního z, resp. x, pro každé x, resp. z - tzv. *drátěná mříž* (*wireframe*)



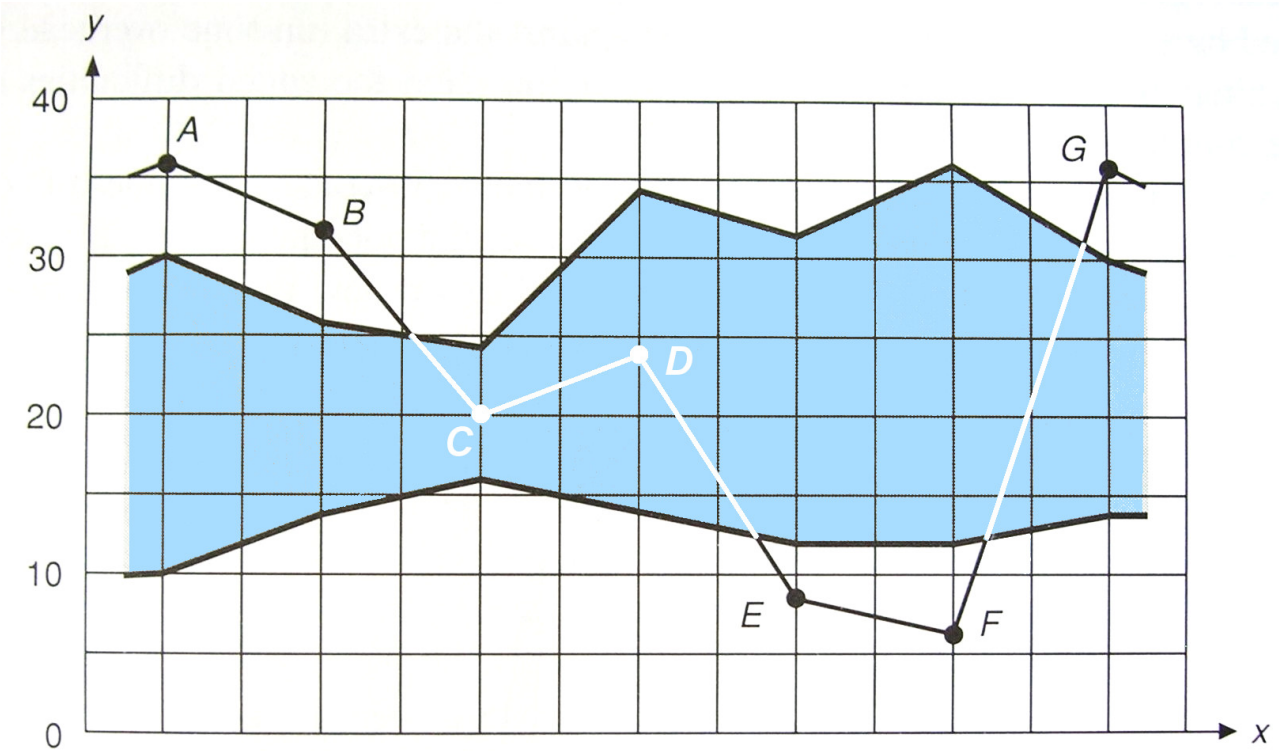
Algoritmus siluety 1/5

- *horizon line algorithm*
- každá lomená čára v **rovině** $z = z_i$, roviny rovnoběžné
  $\Longrightarrow$ žádná lomená čára nemůže být zakryta jinou, která leží v rovině (konstantního z) dále od pozorovatele
  $\Longrightarrow$ algoritmus vykreslování „odpředu dozadu“
- postupně kreslíme lomené čáry, udržujeme siluetu již nakresleného objektu: stačí pole *YMIN*, *YMAX* pro min. a max. **promítnutou** hodnotu y v závislosti na x
- toto pole má konečně mnoho prvků v závislosti na rozlišení obrazu (*image-precision* $\Longrightarrow$ nebezpečí aliasingu)



Algoritmus siluety 2/5

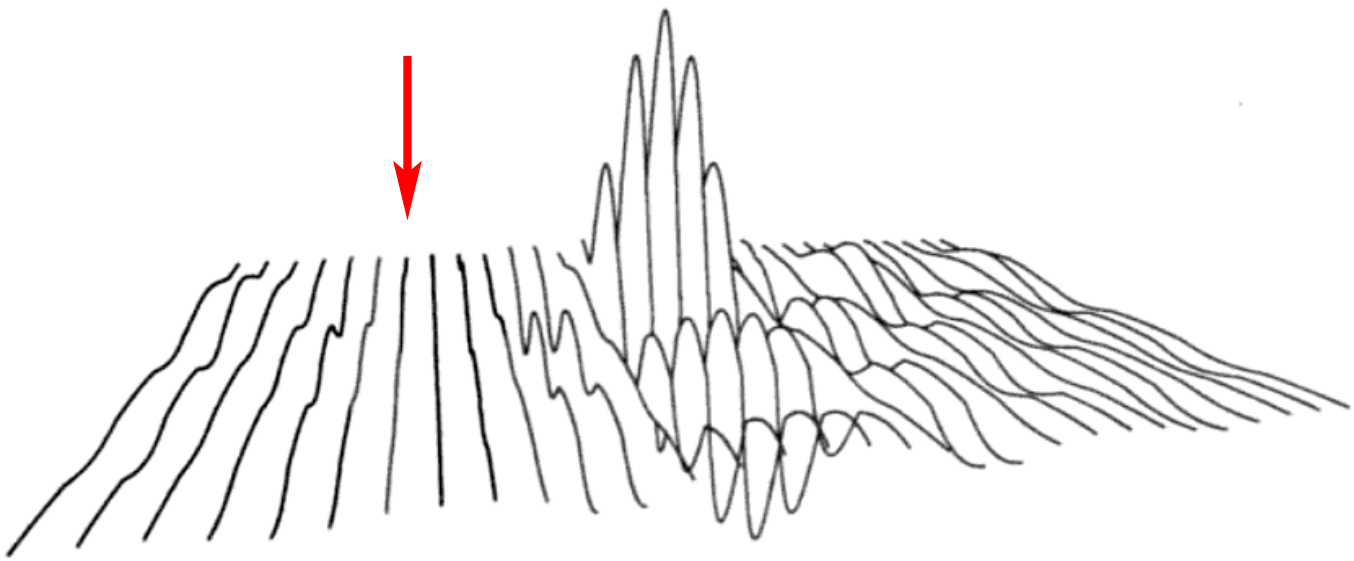
- z další (vzdálenější) lomené čáry nakreslíme jen části, kde promítnutá hodnota *y* leží mimo siluetu, a siluetu aktualizujeme



New *YMAX*	36	34	32	26	24	29	34	33	32	34	36	33	36
New *YMIN*	10	12	14	15	16	15	14	13	8	7	6	13	14
Line's *Y*	**36**	34	**32**	26	**20**	22	**24**	16	**8**	7	**6**	21	**36**
Old *YMAX*	30	28	26	25	24	29	34	33	32	34	36	33	30
Old *YMIN*	10	12	14	15	16	15	14	13	12	12	12	13	14

Algoritmus siluety 3/5

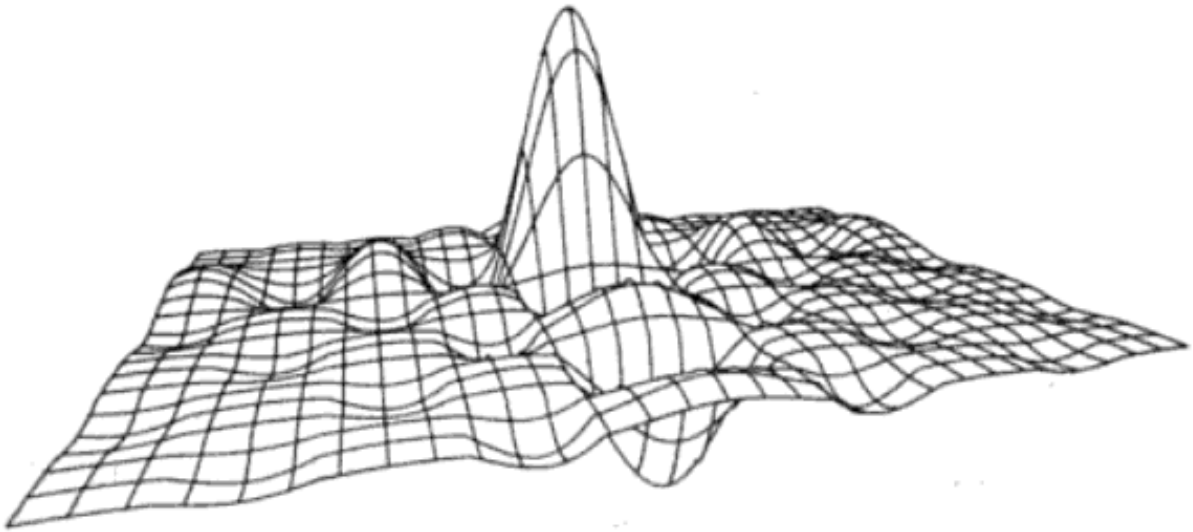
- pro lomené čáry ve směru konstantního x lze provést to samé
- pro jeden ze směrů (v našem pohledu je to x) může být nutné kontrolovat, v jakém směru se roviny vzdalují od pozorovatele (a vždy v tomto pořadí je zpracovávat)



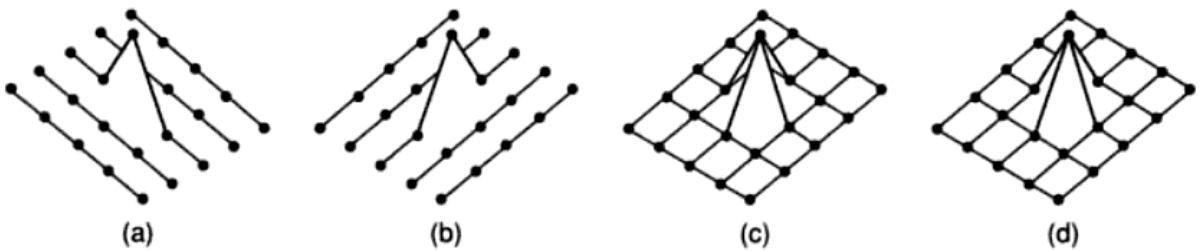
- vlevo od šipky se roviny vzdalují směrem doleva, vpravo od šipky směrem doprava

Algoritmus siluety 4/5

- chceme-li nakonec síť v obou směrech

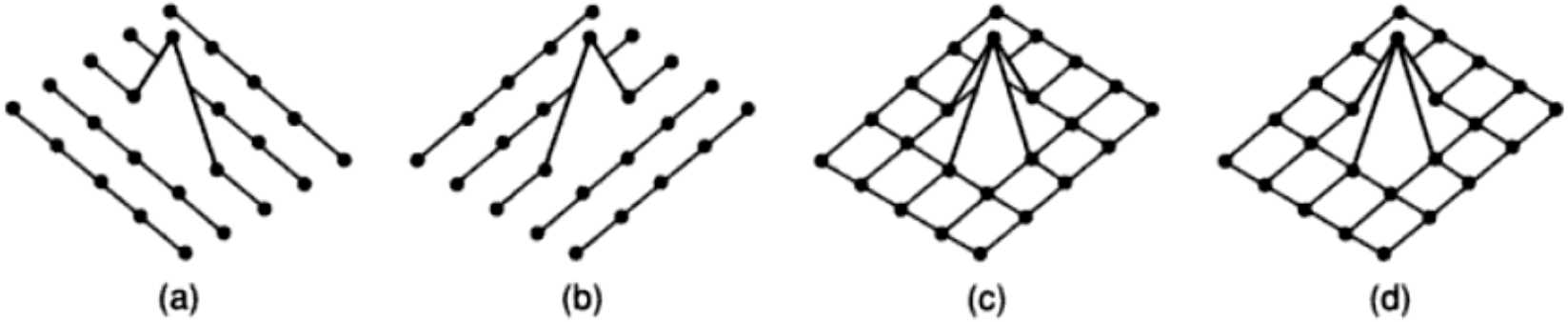


tak nestačí složení obou obrazů - nedá správný výsledek:



(a) Lines of constant z. (b) Lines of constant x. (c) Superposition of parts (a) and (b). (d) The correct solution.

Algoritmus siluety 5/5

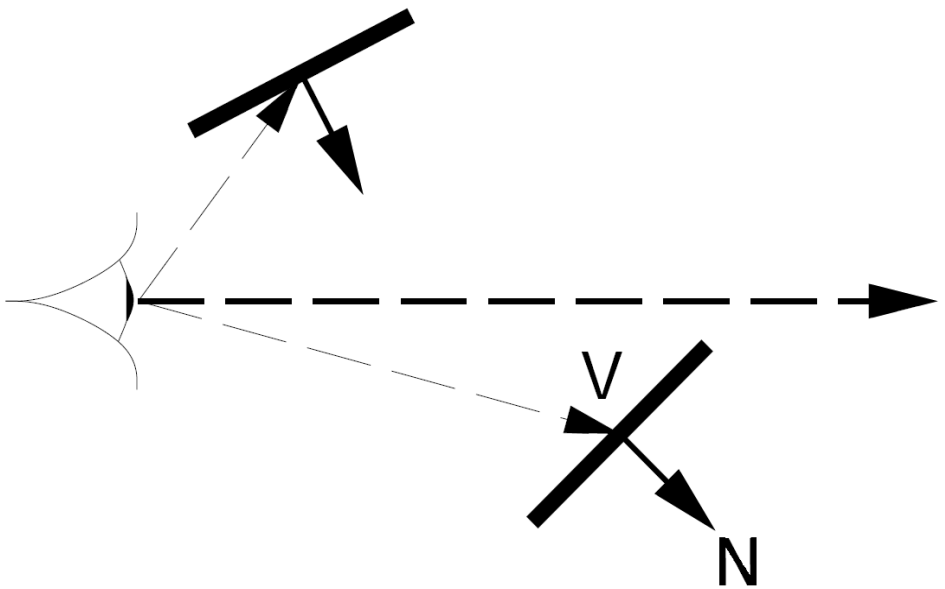


(a) Lines of constant z. (b) Lines of constant x. (c) Superposition of parts (a) and (b). (d) The correct solution.

- **Řešení:** nakreslit základní lomenou čáru ve směru konstantního z a pak vždy jednu čáru ve směru konst. z + úseky všech čar ve směru konst. x, které se se vejdou mezi dvě poslední čáry ve směru konst. z
- ve všech krocích dodržovat pořadí kreslení směrem od pozorovatele

Back-face culling 1/2

- „*culling*“ = redukce, odstřel
- **odstranění odvrácených ploch**
- pro objekty složené z mnohostěnů $\Longrightarrow$ objem ohraničen polygonálními stěnami
- předpoklad: nedíváme se dovnitř objektu (objektem neprochází přední ořezávací rovina)
- polygon, jehož normála míří od pozorovatele, je odvrácený a je plně zakryt jinými bližšími polygony.



Back-face culling 2/2

- nechť $\boldsymbol{n}$ je vnější normála polygonu, $\boldsymbol{v}$ vektor od pozorovatele k polygonu
- polygon je odvrácený, pokud $\boldsymbol{n} \cdot \boldsymbol{v} > 0$ ($\Leftrightarrow \cos(\measuredangle \boldsymbol{nv}) > 0$)
- algoritmus nelze bez dalšího zpracování použít, pokud:
  - objekt není konvexní
  - objekt obsahuje díry
  - máme více než 1 objekt
- **výhoda** - lineární složitost vzhledem k počtu stěn (vhodným předzpracováním lze dosáhnout sublineární složitosti)
  $\Longrightarrow$ používá se jako předstupeň k dalším (úplnějším) algoritmům na řešení viditelnosti

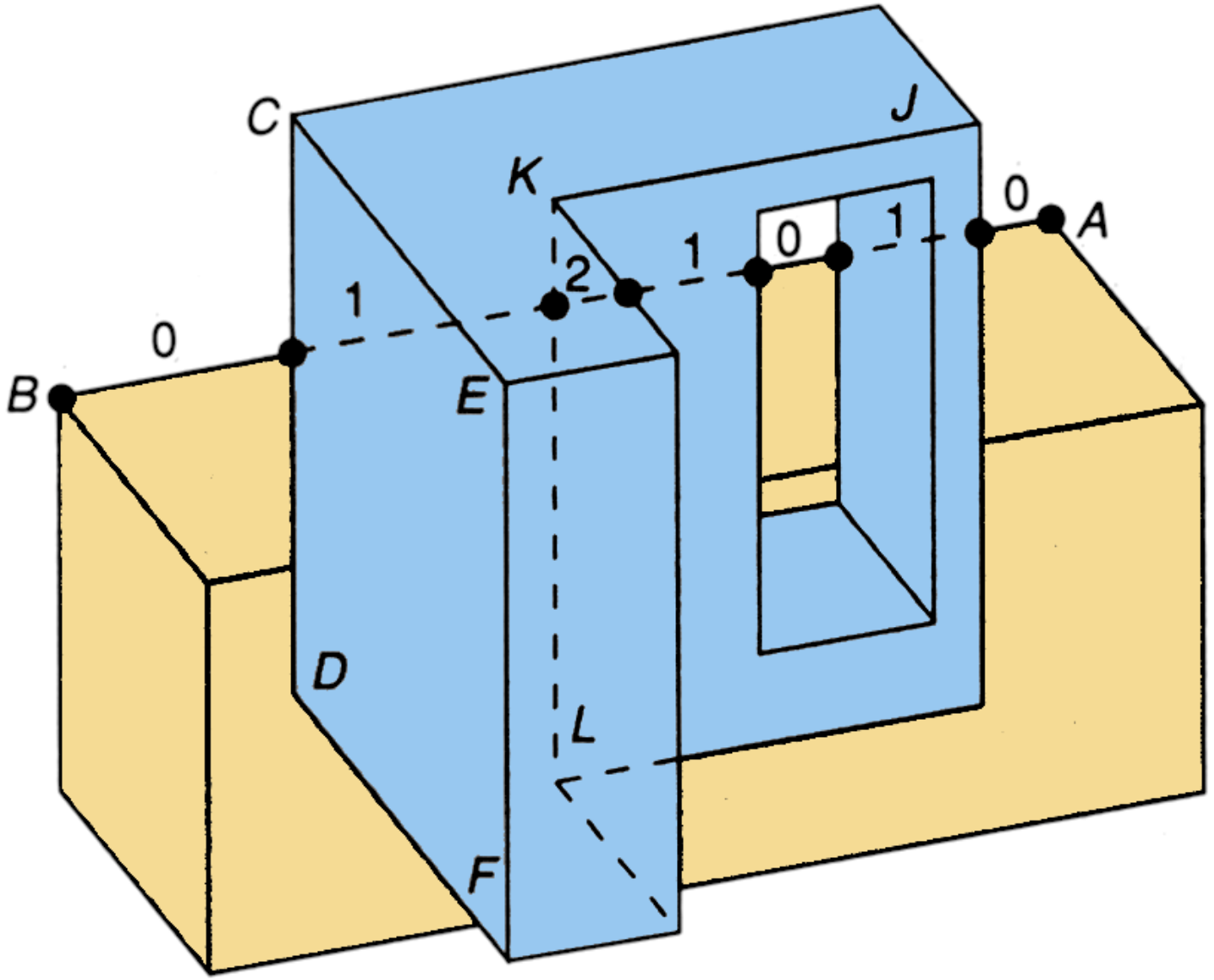
Appelův algoritmus 1/4

- algoritmus pro **určení viditelných úseček** (nikoliv např. ploch $\Longrightarrow$ pouze obrysy)
- obecnější, požaduje, aby úsečky byly součástí polygonálních ploch, ne nutně mnohostěnů

kvantitativní neviditelnost bodu na úsečce:

- pokud úsečka vstupuje za přivrácený polygon, zvýší se kvant. neviditelnost o 1
- pokud úsečka vystupuje zpoza přivráceného polygonu, kvant. neviditelnost se o 1 sníží
- vykreslují se jen části úseček s kvantitativní neviditelností rovnou 0

Appelův algoritmus 2/4



Appelův algoritmus 3/4

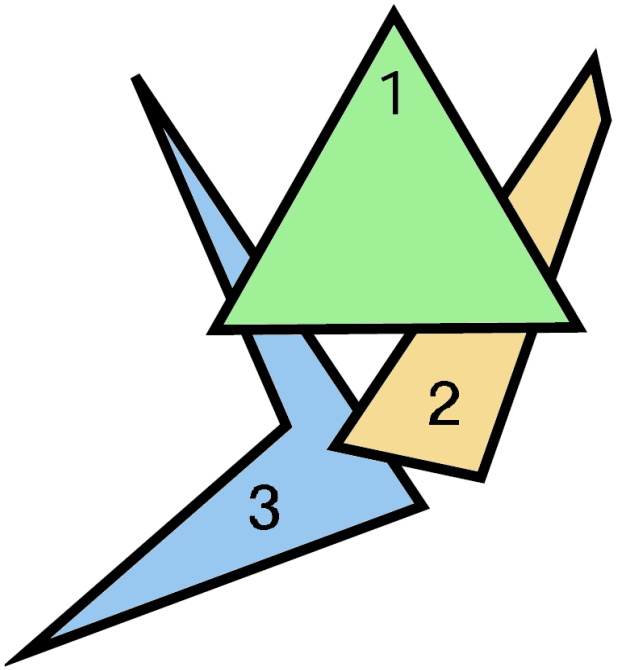
- pokud nemáme navzájem se prostupující polygony, kvantitativní neviditelnost se mění jen při přechodu za tzv. *obrysovou* čáru
- obrysová čára
  - hrana sdílená přivráceným a odvráceným polygonem
  - hrana přivráceného polygonu, který není částí uzavřeného mnohostěnu
- hrana sdílená dvěma přivrácenými polygony nemění neviditelnost
- obrysová čára prochází **před** hranou AB, jestliže protíná $\triangle ABE$, kde E je místo pohledu (oko - *eyepoint*, resp. střed promítání (COP - *center of projection*) - viz přednáška *Promítání*
- $\Longrightarrow$ lze využít algoritmy pro průsečík úsečky a polygonu

Appelův algoritmus 4/4

- nejprve se určí kvantitativní neviditelnost jednoho výchozího bodu (*seed vertex*)
  - výpočet s pomocí „hrubé síly“, kdy se otestují průsečíky všech přivrácených polygonů s paprskem od pozorovatele do výchozího bodu
- poté se využívá „koherence“ hran, tj. vlastnosti, že tvoří polygony, resp. polygonovou síť. Rekurzivně:
  - po všech hranách z výchozího bodu se dojde na jejich konec, kvant. neviditelnost se mění podle dříve popsaného postupu
  - tato nová hodnota kvant. neviditelnosti = výchozí hodnota pro další hrany vycházející z koncových bodů původních hran

Algoritmus depth-sort 1/6

- zjednodušená verze - tzv. *malířův algoritmus*
- kreslení polygonů postupně do tzv. *frame bufferu* - offline grafické paměti (pro barvu pixelů), kterou poté najednou zobrazíme, až když je scéna připravená
- kreslíme polygony od nejvzdálenějšího k nejbližšímu, postupně se překreslují

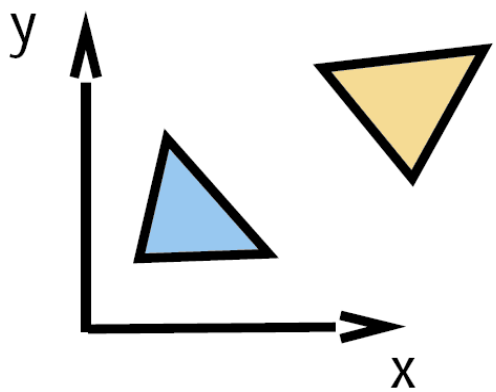


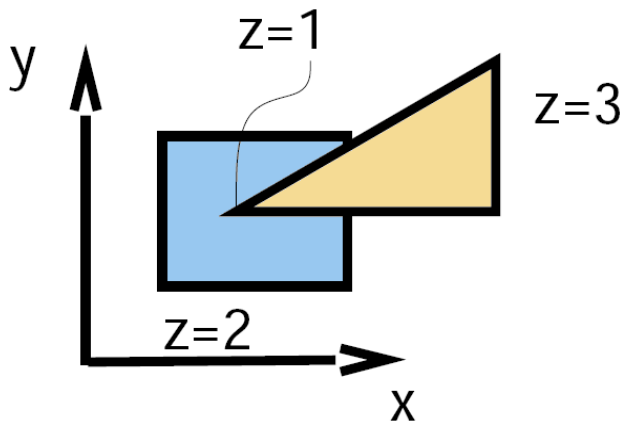
Algoritmus depth-sort 2/6

Zjednodušený postup

1. seřadit polygony podle jejich nejmenší (či největší) souřadnice z (souřadnice z již v projekci, tj. vzdálenost od pozorovatele (*depth*))
2. vyřešit nejednoznačnosti (když se rozsahy z překrývají $\Longrightarrow$ možná nutné dělení polygonů)
3. vykreslit polygony v pořadí od nejvzdálenějšího

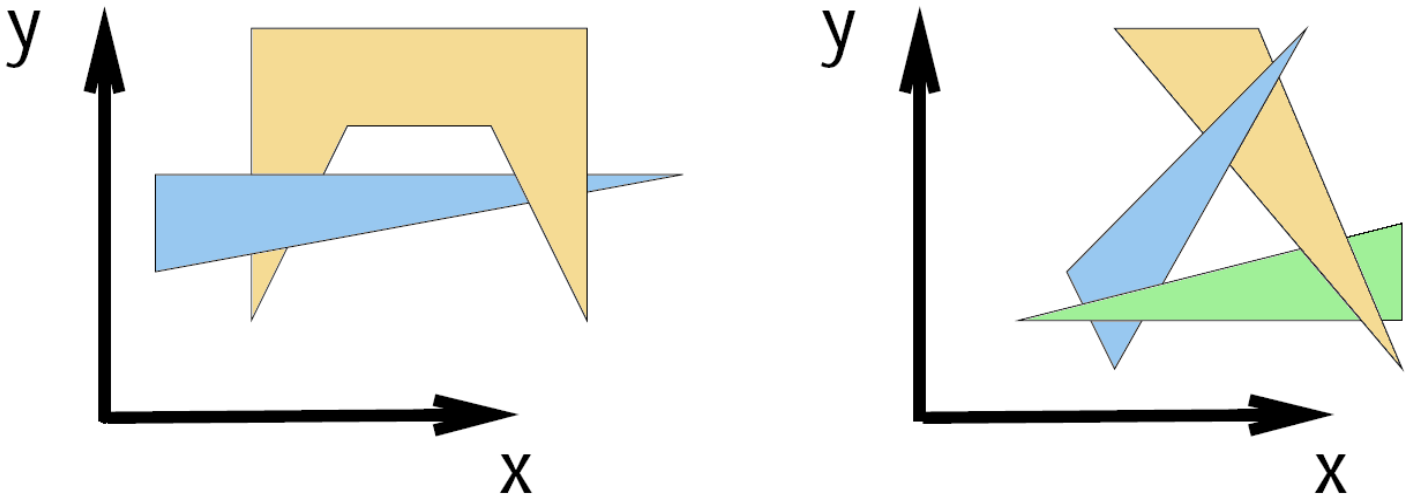
- **malířův algoritmus** neobsahuje bod 2 $\Longrightarrow$ vyřeší jednoduché případy, kde každý polygon je v rovině $z = konst$.





Algoritmus depth-sort 3/6

Možné případy překryvu



- složitější situace si vynutí další rozhodování
- nechť P je polygon na konci setříděného seznamu
- nechť Q_i jsou všechny polygony, jejichž rozsah z se překrývá s rozsahem z polygonu P
- než vykreslíme P, musíme ověřit, že nezakryje žádný z Q_i

Algoritmus depth-sort 4/6

Testování překryvu

- provádíme 5 testů se zvyšující se obtížností mezi P a Q_i. Úspěch libovolného z nich $\Longrightarrow$ P nezakryje Q_i.

1. jsou rozsahy souřadnice x polygonů P a Q_i disjunktní?
2. jsou rozsahy souřadnice y polygonů P a Q_i disjunktní?
3. leží celý P za rovinou polygonu Q_i ve směru pohledu?
4. leží celý Q_i před rovinou polygonu P ve směru pohledu?
5. jsou průměty obou polygonů disjunktní?

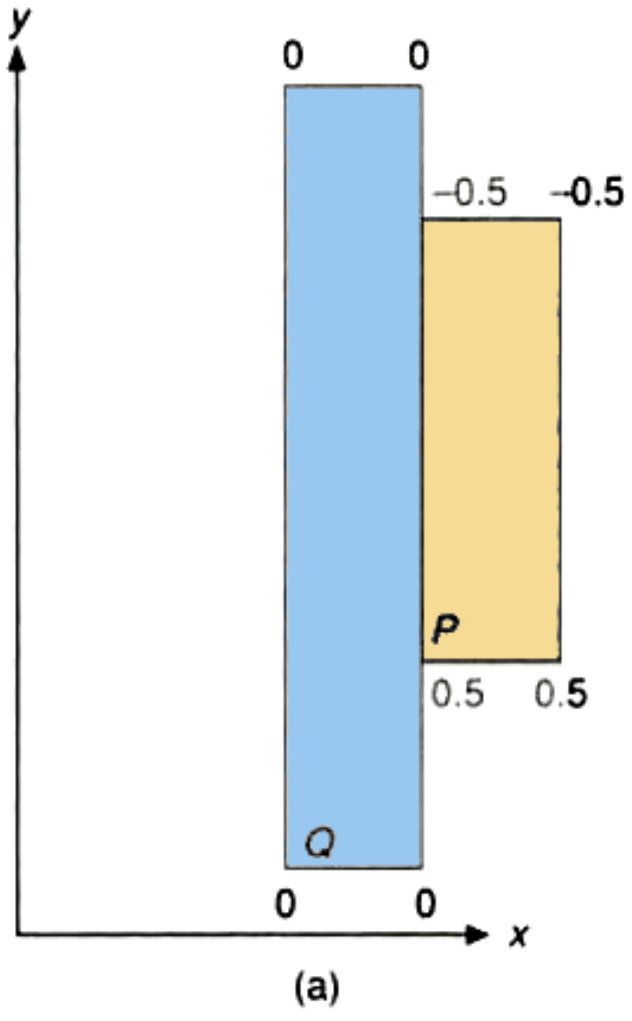
Algoritmus depth-sort 5/6

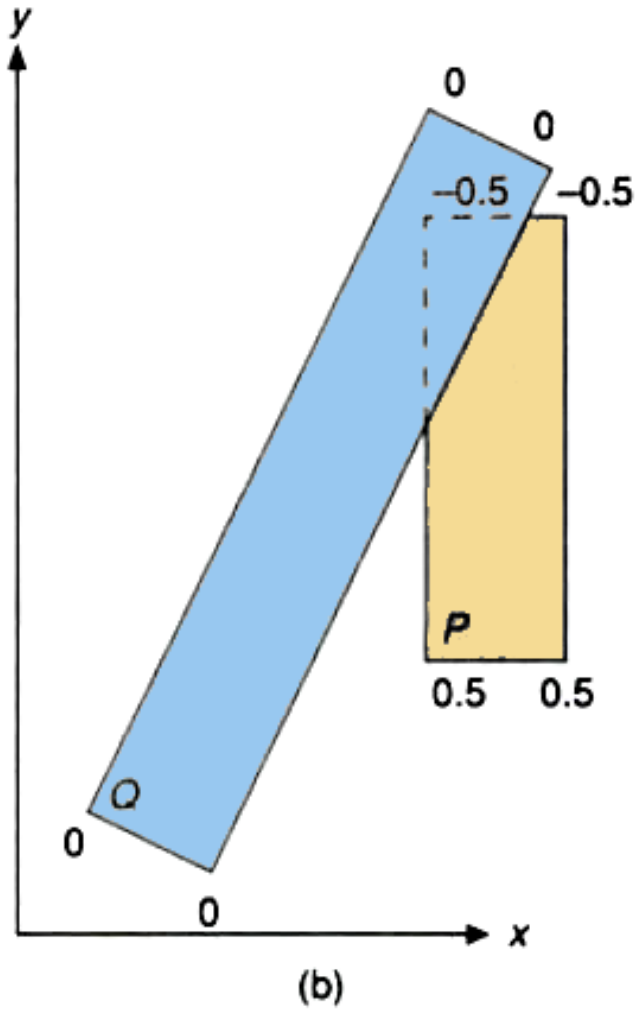
Výsledky testování překryvu

- pokud nějaký test uspěje, testujeme další Q_i
  - pokud test projde $\forall i$, nakreslíme P
- pokud testy neuspějí, prohodíme pro účel testování P a Q_i a provedeme znovu testy 3,4 (ostatní netřeba opakovat).
  - pokud nyní uspějeme, zařadíme Q_i na konec seznamu místo P
- pokud ani „prohozený“ test neuspěje, je nutné rozdělit jeden polygon rovinou druhého, původní zahodit, správně seřadit podle (nejmenší) hodnoty z a zařadit do seznamu
- potřeba ošetřit možnost zacyklení a další detaily

Algoritmus depth-sort 6/6

Zbytečné dělení polygonů (kdy všech 5 testů selže i bez vzájemného překryvu)





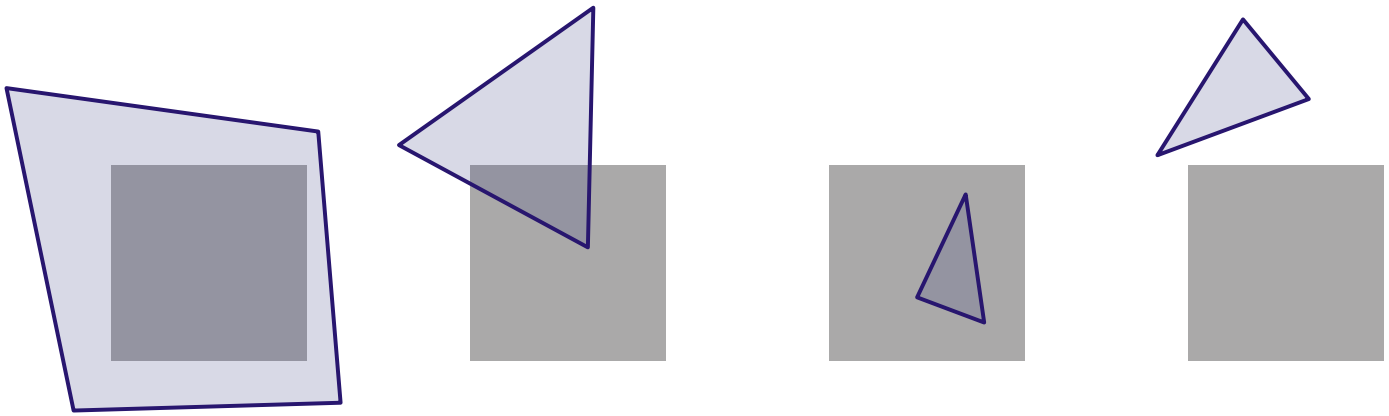
Warnockův algoritmus 1/6

- obrazový (image-precision) algoritmus typu „rozděl a panuj“ - rekurzivní dělení oblasti
- určuje případy, které lze snadno zvládnout a ihned vykreslit
- v ostatních případech dělí obdélníkovou „oblast zájmu“ (tj. okno v průmětně) na 4 menší a pokračuje rekurzivně v každé z podoblastí

Warnockův algoritmus 2/6

4 případy vztahu polygonu k oblasti zájmu

1. obklopující polygon - jeho projekce plně obsahuje oblast zájmu
2. polygon protínající oblast zájmu
3. polygon plně obsažený v oblasti zájmu
4. disjunktní polygon zcela mimo oblast zájmu



Warnockův algoritmus 3/6

4 snadné případy, které lze ošetřit rovnou bez dělení oblasti

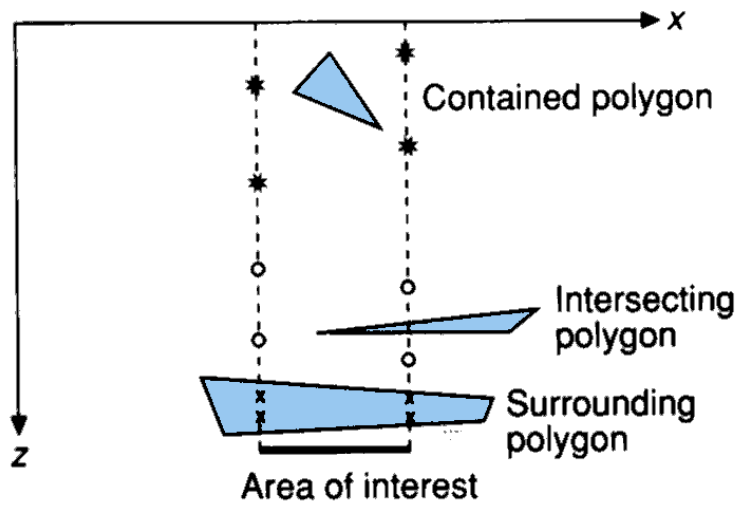
1. všechny polygony disjunktní $\Longrightarrow$ oblast vyplněna barvou pozadí
2. jediný protínající, resp. obsažený polygon $\Longrightarrow$ oblast vyplněna barvou pozadí, poté promítnut a vykreslen daný polygon, resp. jeho část
3. jediný obklopující polygon, žádné další $\Longrightarrow$ oblast vyplněna barvou daného polygonu
4. několik obklopujících, protínajících a obsažených polygonů, ale jeden z nich je obklopující, který je nad všemi ostatními (nejblíže pozorovateli) $\Longrightarrow$ viz 3. bod

- vyšetření 4. případu viz dále

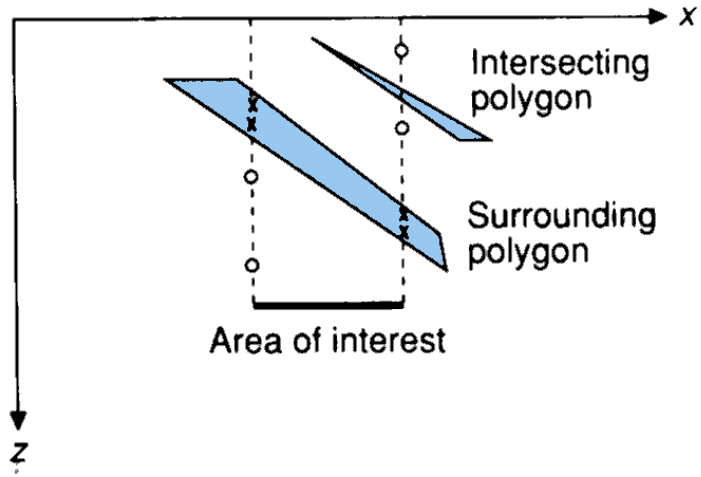
Warnockův algoritmus 4/6

Vyšetření 4. „snadného“ případu

- pro každý polygon se určí souřadnice z roviny, ve které je obsažen, ve 4 rozích oblasti zájmu
  (z průsečíků roviny s hranami příslušného pohled. objemu)
- pokud všechny průsečíky (**x**) náležící jednomu obklopujícímu polygonu jsou nad průsečíky rovin všech ostatních polygonů (obklopujících, protínajících (∘) a obsažených (★)), je rozpoznán případ 4.
- na obr. **b)** není případ 4 rozpoznán $\Longrightarrow$ dělení oblasti



(a)



(b)

Warnockův algoritmus 5/6

Zastavení dělení

- dosud pracoval algoritmus jako objektový (object-precision)
- algoritmus je ale obrazový kvůli **kritériu zastavení dělení**:

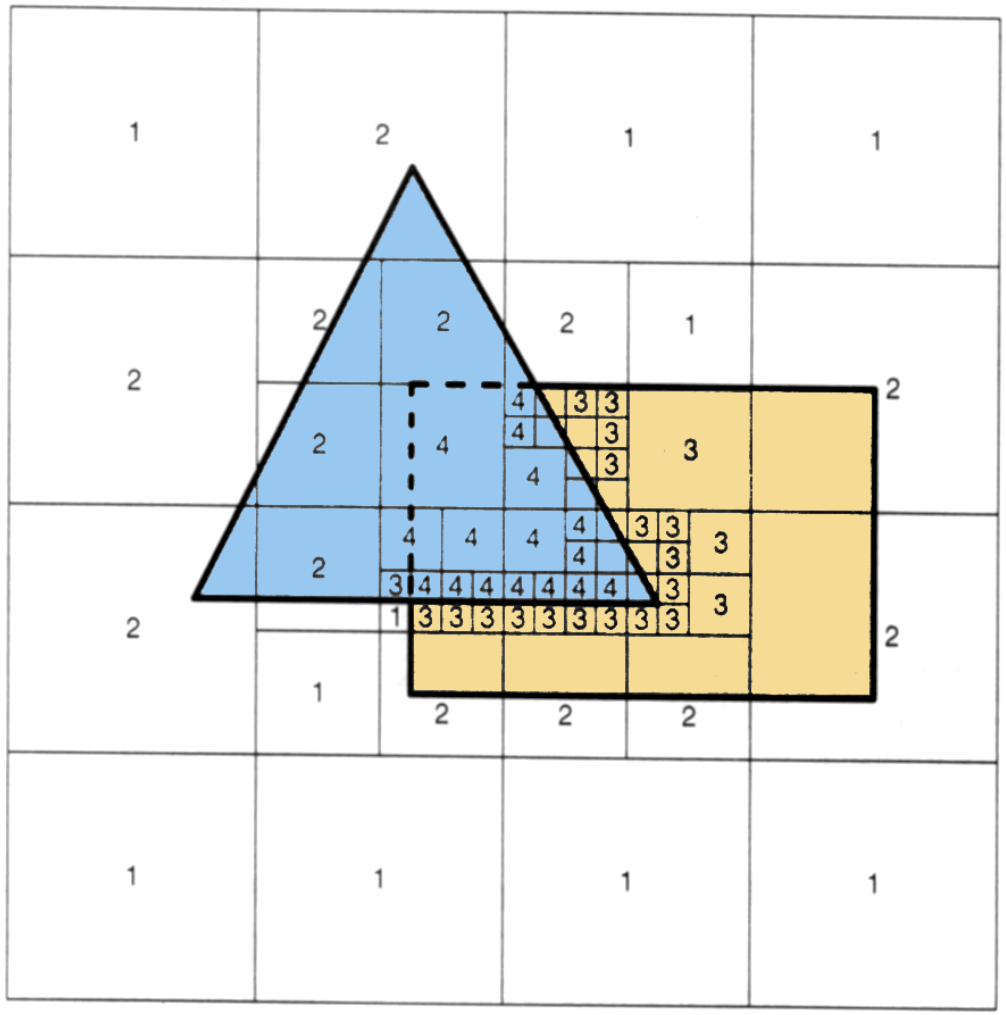
oblast dosáhne velikosti pixelu (a stále nenastanou případy 1-4) $\Longrightarrow$ vyplníme barvou polygonu, který je nejblíže pozorovateli v daném bodě (středu oblasti) - *ray casting*

- pro odstranění aliasingu možné nadvzorkovat na velikost menší než 1 pixel

Warnockův algoritmus 6/6

Příklady dělení

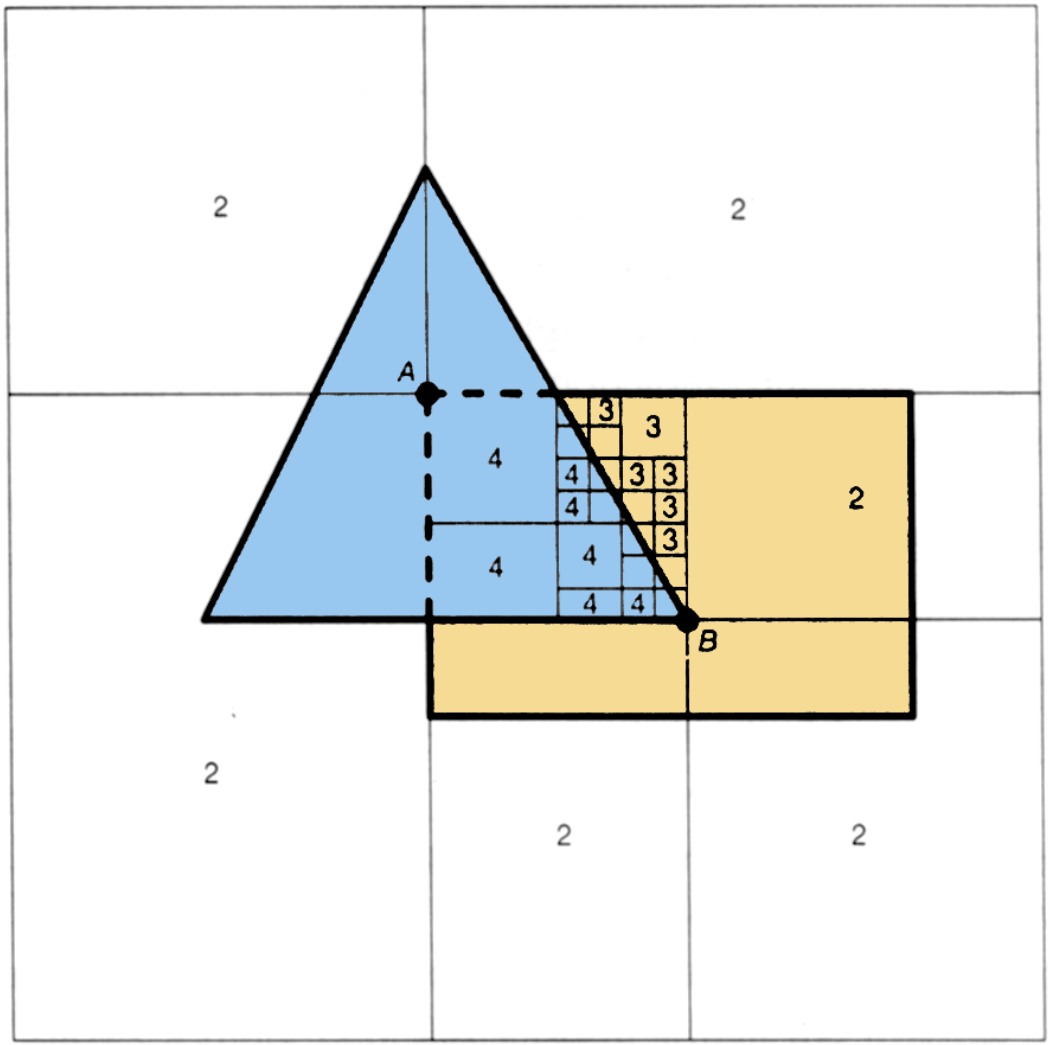
- dělení na čtverce
- neuniformní dělení (první v bodě A, druhé v bodě B)



Warnockův algoritmus 6/6

Příklady dělení

- dělení na čtverce
- neuniformní dělení (první v bodě *A*, druhé v bodě *B*)



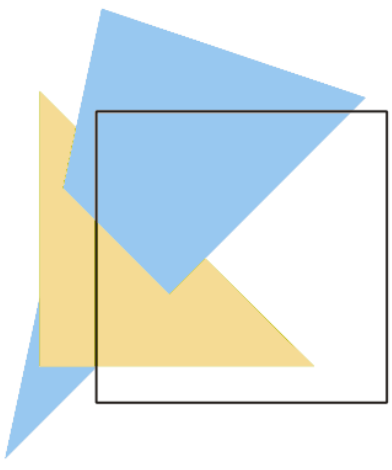
z-buffer 1/3

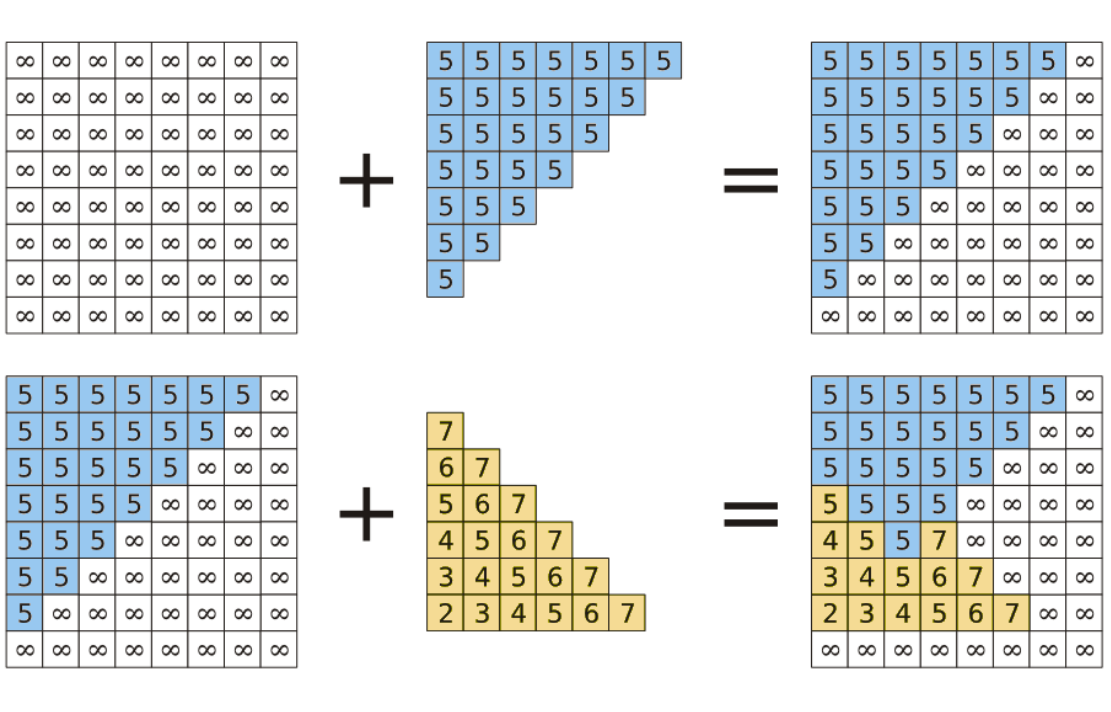
- jinak také *depth-buffer*
- obrazový (image-precision) algoritmus
- využívá frame buffer a vedle něj *depth-buffer* (paměť hloubky) stejné velikosti
- jednotlivé objekty (ne nutně jen polygony) se v libovolném pořadí vykreslují do průmětny (frame bufferu) a současně se aktualizuje paměť hloubky

- paměť hloubky na začátku nastavena na $+\infty$, průmětna na barvu pozadí
- Každý pixel objektu se zakreslí na průmětnu jen tehdy, je-li jeho souřadnice *z* (hloubka, vzdálenost od pozorovatele) menší než hodnota uložená v paměti hloubky na dané pozici. V tom případě se paměť hloubky aktualizuje na tuto novou hodnotu.

z-buffer 2/3

Schéma vykreslení dvou polygonů





z-buffer 2/3

Vlastnosti

- rychlý, jednoduchý, snadný na implementaci v HW
- obecný (není omezen jen na polygony)

Aliasing způsobený konečnou přesností z-bufferu

- z-buffer v HW je obvykle 32 bit integer
- pro hloubku se používá pseudo-vzdálenost (viz *Promítání*)

$$PD(z) = \frac{z(f+n) - 2fn}{z(f-n)},$$

kde $z = n$ a $z = f$ jsou přední a zadní ořezávací roviny.

- pro $f \gg n$ a velká z je $PD(z)$ blízko 1 ($PD : [n, f] \mapsto [-1, 1]$)
- konečná přesnost $\Longrightarrow$ různě (ale hodně) vzdálené objekty mohou získat stejnou hodnotu hloubky $\Longrightarrow$ prolnutí vzdálených objektů

Další algoritmy

- algoritmy pro BSP (binary space partitioning) stromy (Schumacker)
  - efektivní, když se hýbe kamera, ale nemění se scéna
- algoritmy pro octree
  - uspořádání oktantů odzadu dopředu
- algoritmy pro řešení viditelnosti zakřivených povrchů
  - dělení na polygonální síť + algoritmus pro polygony
  - přímé zpracování zakřivených povrchů (Weiss, Woon, Mahl, Levin, Sarraga)
  - algoritmy pro bikubické povrchy (Blinn, Whitted)
- ...

Literatura


J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, K. Akeley: *Computer Graphics: Principles and Practice (3rd ed.)*, Addison Wesley, 2014.


M.E. Newel et al.: *A Solution to the Hidden Surface Problem.* Proceedings of the ACM National Conference 1972, 443450. *(algoritmus depth-sort)*


A. Appel A.: *The notion of quantitative invisibility and the machine rendering of solids.* Proceedings of the 1967 22nd national conference, 1967, 387393.