Nerealistické renderování: kresby

Co je to NPR?

- NPR = všechno, co není fotorealistické renderování
- Painterly rendering, cartoons, ilustrace, netradiční perspektivy, vizualizace dat, ...
- Motivace: v určitých situacích není fotorealizmus žádoucí (př.: ilustrace v manuálu)
- Zpracování 2D obrazu i renderování 3D scén

Obsah

- Budeme se věnovat renderování **obrysových kreseb**
  - Extrakce obrysů z 2D obrazu se Z-bufferem a z 3D reprezentace scény
  - Kreslení čar
  - Vybarvování (cartoons), šrafování a stínování
  - Realtime varianty

Extrakce obrysů z 2D obrazu

- Vstupem je vyrenderovaný obrázek resp. jeho Z-buffer
- Kvalita omezena rozlišením vstupního obrazu, pro mnohé aplikace to ale nevadí
- Lze použít existujících nástrojů pro zpracování obrazu (detektory hran) => snadná implementace

Hranový detektor

- Nejjednodušší metoda: použít hranový detektor přímo na obraz a zobrazit výstup detektoru
- Nepotřebuje Z-buffer, lze aplikovat na nerenderovaná data
- Nevýhody:
  - HD zmatou textury
  - Stejnobarevné objekty, které v použité projekci sousedí

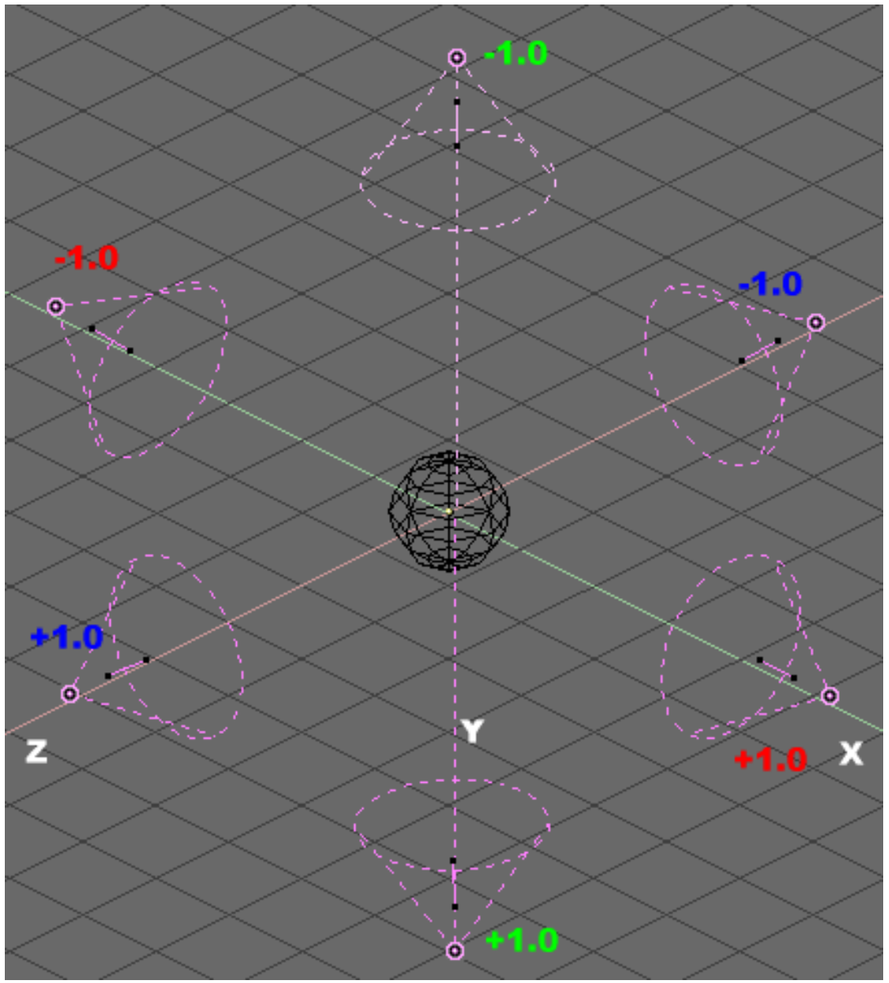
Hranový detektor v Z-bufferu

- Řešení: detekce hran se provede v Z-bufferu
- Pozorování: rozdíl hloubek sousedních pixelů je obvykle malý v rámci objektu, ale velký mezi 2 různými objekty
- Problémy:
  - nedetekuje hrany sousedících objektů ve stejné hloubce
  - Nedetekuje ohyby (nespojitosti první derivace)

Normálová mapa

- Normálová mapa:
- RGB hodnoty určují po řadě x,y,z složku normálového vektoru v daném pixelu
- Detekce hran v n.m. najde ohyby povrchu
- Jak generovat n.m.?
  - Přímo jako výstup raytraceru
  - Trikem: vyrenderování vhodně nasvícené scény

Vytvoření normálové mapy



- Bílý matný materiál objektu
- 6 směrových světel rovnoběžných se souřadnými osami
- Body objektu osvětleny úměrně k $\mathbf{n}.\mathbf{v}_{light}$
- Zápornou intenzitu lze nahradit 2 průchody

Detekce hran konvolucí

- Na tento typ obrázků stačí jednoduchý H.D., např. konvoluce Sobelovým filtrem + prahový operátor:

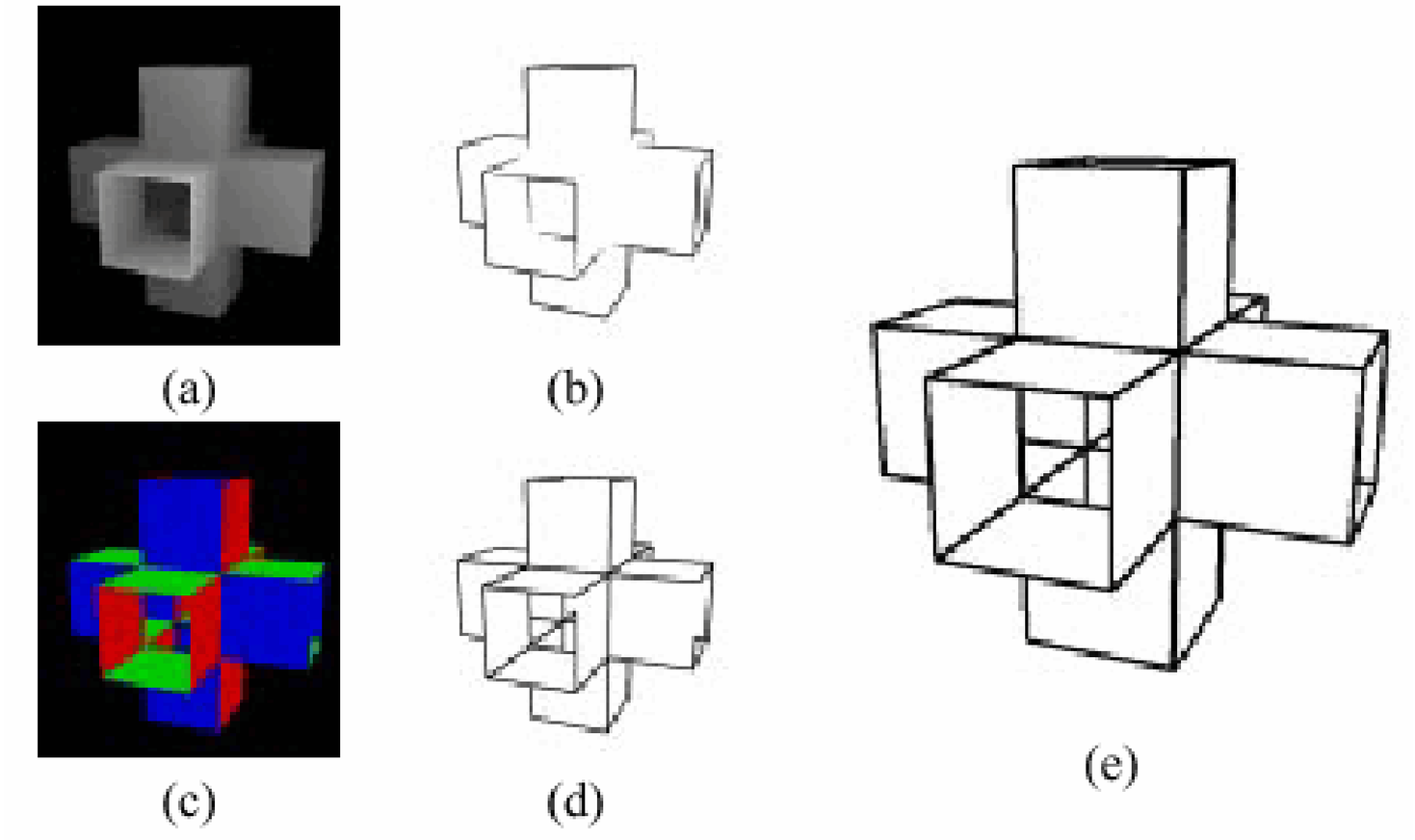
$$I(x,y)\otimes h(x,y)=\sum_{i=-k}^{k}\sum_{j=-k}^{k} I(i,j)h(x-i,y-j)$$

$$S_x=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}, S_y=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$$

$$I_x(x,y)=I(x,y)\otimes S_x, I_y(x,y)=I(x,y)\otimes S_y$$

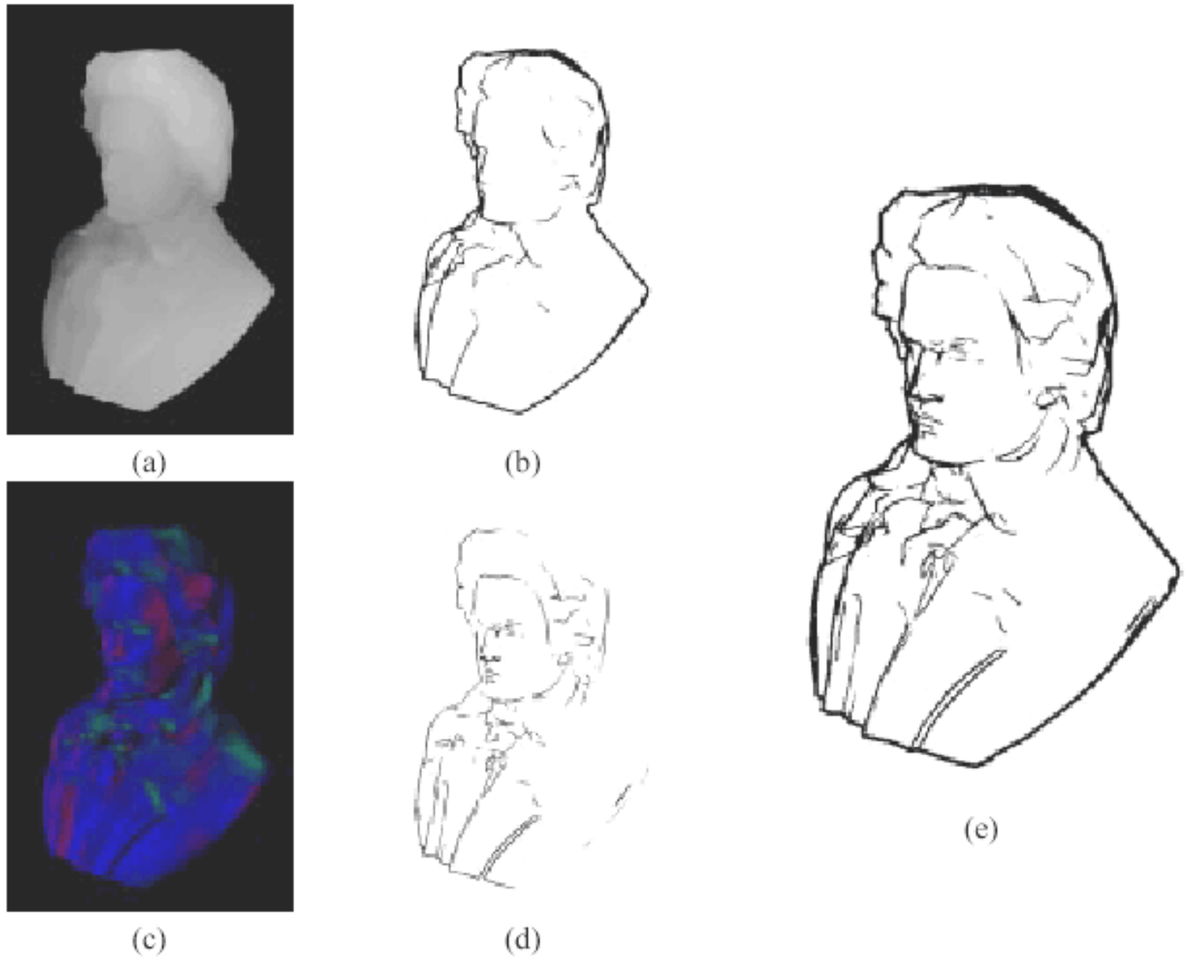
$$I(x,y)=\sqrt{I_x^2(x,y)+I_y^2(x,y)}$$

Výsledky detekce hran ve 2D



(a) z-buffer (b) hrany v z-bufferu (c) norm. mapa (d) hrany v n.m. (e) kombinace b a d

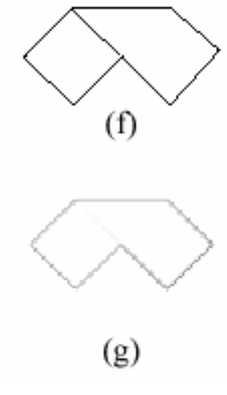
Výsledky detekce hran ve 2D



(a) z-buffer (b) hrany v z-bufferu (c) norm. mapa (d) hrany v n.m. (e) kombinace b a d

Problémy 2D extrakce hran

- Nelze změnit styl kresby, záleží na detektoru (konverze do hranové reprezentace je možná, ale obtížná)
- Nutné pro každý obrázek vyladit parametry h.d.
- Nemá k dispozici informace o scéně!! Důsledek: nelze šrafovat ani vybarvovat
- Patologické případy:

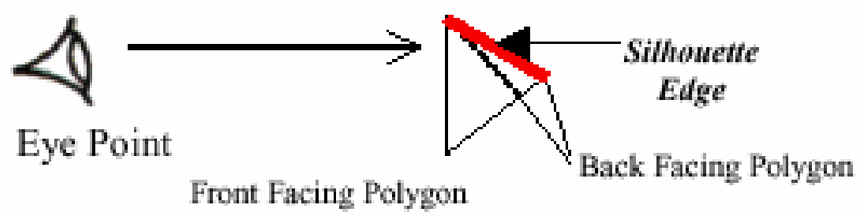


Extrakce obrysů z 3D modelu

- Složitější než 2D přístup, vyžaduje modifikovaný renderer, ale dává přesné výsledky
- Def.: Množina $M=\{x_i; \mathbf{n_i}.\bullet(x_i-C) = 0\}$, kde C je střed perspektivní projekce a n_i je normálový vektor v bodě x_i, se nazývá **obrys**.
- Pozn.: definice nezávisí na viditelnosti!

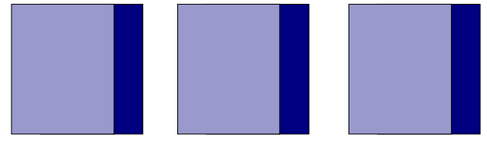
Extrakce obrysů z 3D modelu

- Pozn.: pro ortogonální projekci se použije test $\mathbf{n}_i \bullet \mathbf{v}=0$, kde $\mathbf{v}$ je směr pohledu
- V případě polygonových modelů (mesh) je obrys množina hran spojujících přivrácený a odvrácený polygon



Algoritmus nalezení obrysu polygonového modelu

```
def FindOutline(EyeVect, Edges):
 Outline = {}
 for E in Edges do:
  dotp1 = E.Face1.Normal•EyeVect
  dotp2 = E.Face2.Normal•EyeVect
  if dotp1 * dotp2 <= 0:
   Outline += E
return Outline
```



Obrys hladkých ploch

- Naivní přístup: aplikovat předchozí algoritmus na aproximaci plochy trojúhelníkovou sítí
- Nedává dobré výsledky (viz koule)
- Problém: obrys může probíhat uvnitř polygonu ve zvolené aproximaci
- Ukážeme algoritmus, který nalezne „dobrý“ obrys

Předpoklady

- Plocha je spojitá
- Umíme trojúhelníkovou aproximaci plochy
- Možnost libovolně zjemnit aproximaci (není nezbytně nutné)
- NURBS, subdivision surfaces, ...

Algoritmus pro hladké plochy

1) Vytvoříme trojúhelníkovou aproximaci plochy
2) Pro každý její vertex spočteme

$$d_i = \frac{n_i \cdot (x_i - C)}{|n_i| \cdot |x_i - C|}$$

$$s_i = 1, d_i \geq 0$$
$$s_i = -1, \; jinak$$

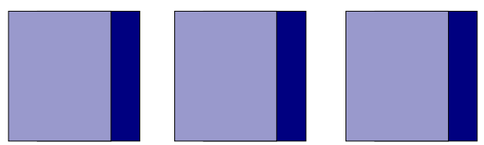
Platí: $E=\langle x_i, x_j \rangle$ hrana & $s_i \neq s_j \Rightarrow \exists\, x \in E$ bod obrysu

Algoritmus pro hladké plochy

3) Pro každou hranu jejíž vertexy mají různé hodnoty s_i lineárně interpolujeme polohu obrysového bodu:

$$x = \frac{|d_j|}{|d_i| + |d_j|} x_i + \frac{|d_i|}{|d_i| + |d_j|} x_j$$

5) Pokud má trojúhelník 2 obrysové body, přidáme jimi určenou úsečku do obrysu (pozn.: možné jsou pouze případy 2 nebo žádný obrys. bod)



Obrys v aproximační síti

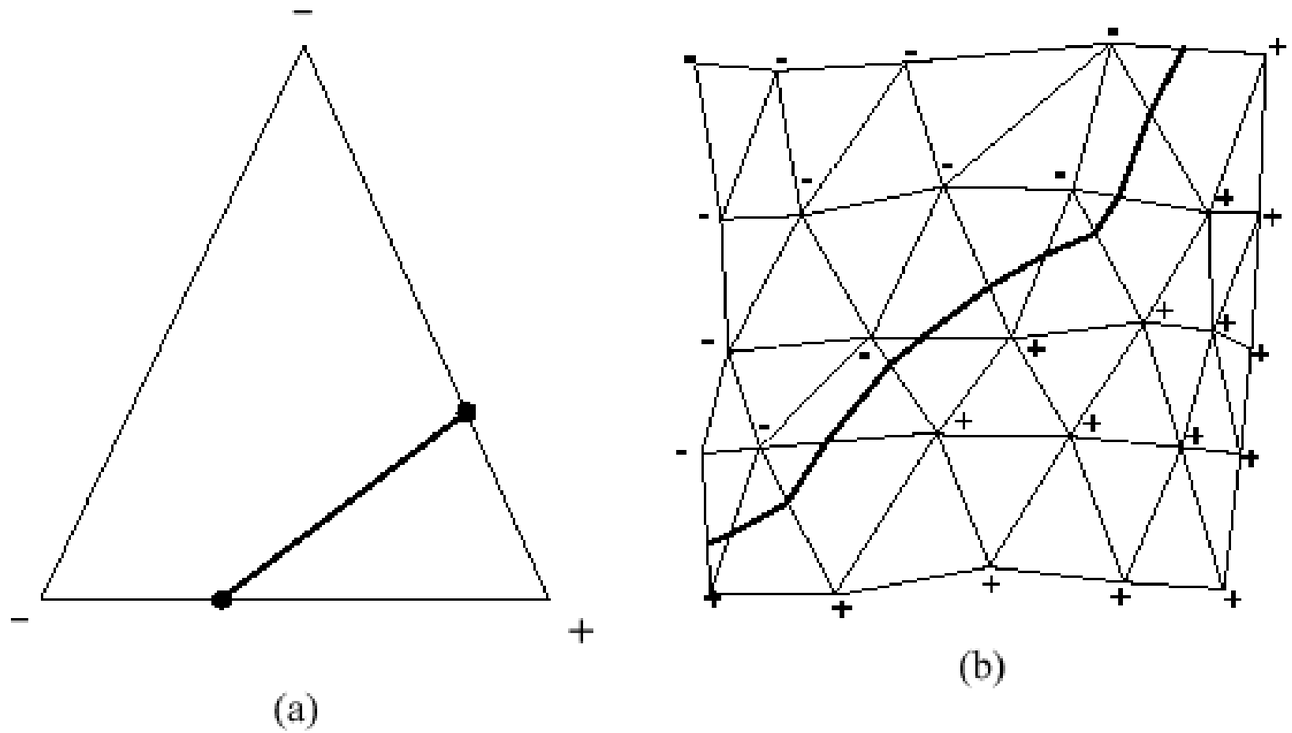


Figure 6: Approximating the silhouette curve for a smooth surface. The smooth surface is represented by a triangle mesh that is close to the surface. (a) Silhouette points are computed on each edge by linear interpolation. A line segment is created by connecting the points. (b) Line segments are connected to form a piecewise-linear curve.

Hladké plochy – poznámky

- Čím jemnější aproximace, tím lepší výsledek
- Při příliš hrubé síti může algoritmus část obrysu úplně vynechat!
- Řešení: adaptivní zjemňování

Viditelnost obrysu

- Úseky obrysu mohou být viditelné, neviditelné nebo jen částečně viditelné
- Je potřeba spočítat jejich průsečíky a rozdělit je na elementární úseky, které jsou celé (ne)viditelné
  - V O(k2) nebo O(k.log(k)) (sweep-line algorithm)
  - Pozor na singularity (vertexy na okrajích nebo ležící současně na přivrácené a odvrácené hraně)
  - Viditelnost se počítá pro každý úsek zvlášť vrháním paprsku nebo pomocí Appelova algoritmu

Realtime hledání obrysu

- Pozorování: Většina hran není obrysová, testují se zbytečně
- Pro RT aplikace existuje pravděpodobností algoritmus:
  - je rychlý (~5x rychlejší)
  - negarantuje nalezení celého obrysu
- Pokud nalezneme jednu obrysovou hranu, lze „stopováním“ triviálně získat celý příslušný obrys

Realtime hledání obrysu

- Pokud má typický obrys 100 hran, pravděpodobně ho detekujeme už při otestování 1 % všech hran
- Zlepšení: setřídíme hrany podle úhlu φ svíraného příslušnými polygony a přiřadíme jim pravděpodobnosti rostoucí s klesajícím φ
- (Pravděpodobnost, že hrana je obrysová, je úměrná π- φ

Realtime hledání obrysu

- Časová koherence: obrys se mezi následujícími snímky skoro nemění
- Vždy se na příslušnost k obrysu testují všechny obrysové hrany z předchozího snímku
- Navíc se z několika málo bodů minulého obrysu prohledávají hrany v (omezeném) okolí:
  - Směrem ke kameře, pokud výchozí bod ležel na odvráceném povrchu, a od kamery, pokud na přivráceném

Realtime hledání obrysu

- K výpočtu viditelnosti hran lze použít:
  - optimalizovaný Appelův algoritmus, který snižuje počet vrhaných paprsků
  - Z-buffer
    - Nemusí se počítat průsečíky, viditelnost řeší HW
    - Vyžaduje kreslení čar pomocí texturovaných polygonů (viz dále)
    - Neumí kreslit neviditelné části obrysu

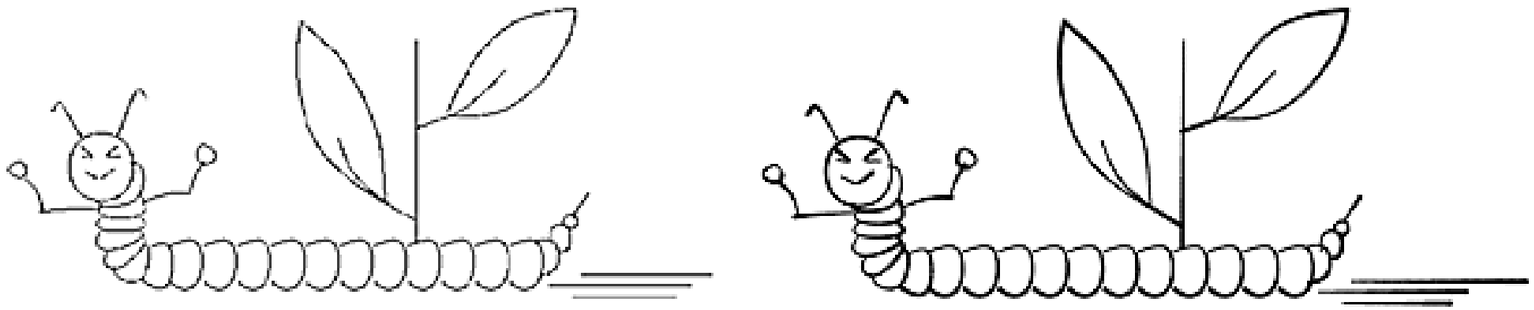
Kreslení čar – linestyles

- Kreslené křivky jsou parametrizovány negeometrickými atributy, které mají vliv na vzhled kreslené čáry
  - Tlak – mapuje se na tloušťku
  - Jas – projeví se na stupni šedi čáry
- Nemusí být (a nebývá) konstatní na celé čáře
- Závisí na informacích ze 3D reprezentace

Příklady stylů

- Lineární změna tlaku na tužku během tahu
- Slabší čáry ve větší vzdálenosti od kamery
- Neviditelné části obrysu kreslené tečkovaně
- Osvětlení: nasvícené hrany tence, odvrácené tlustou čárou (efekt stínu)
- Randomizace parametrů i geometrie

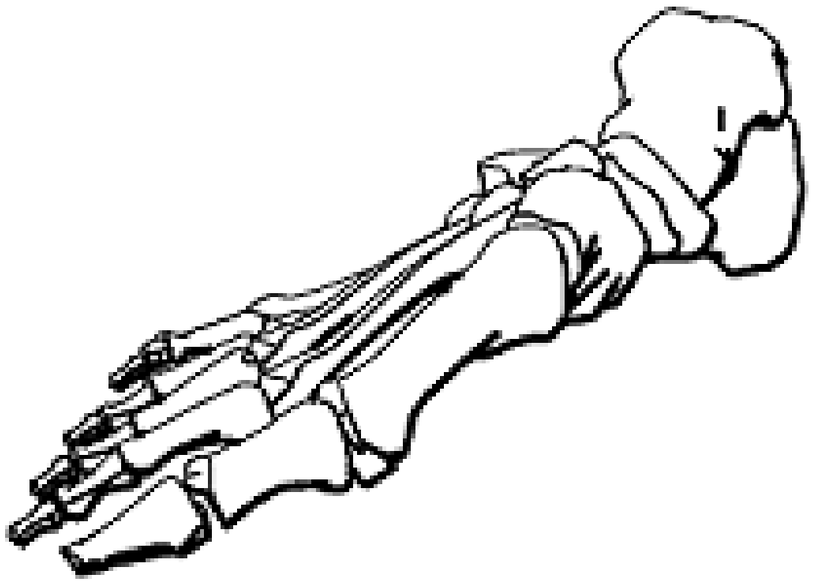
Příklad – lineární změna tlaku



Nalevo: konstatní tlak

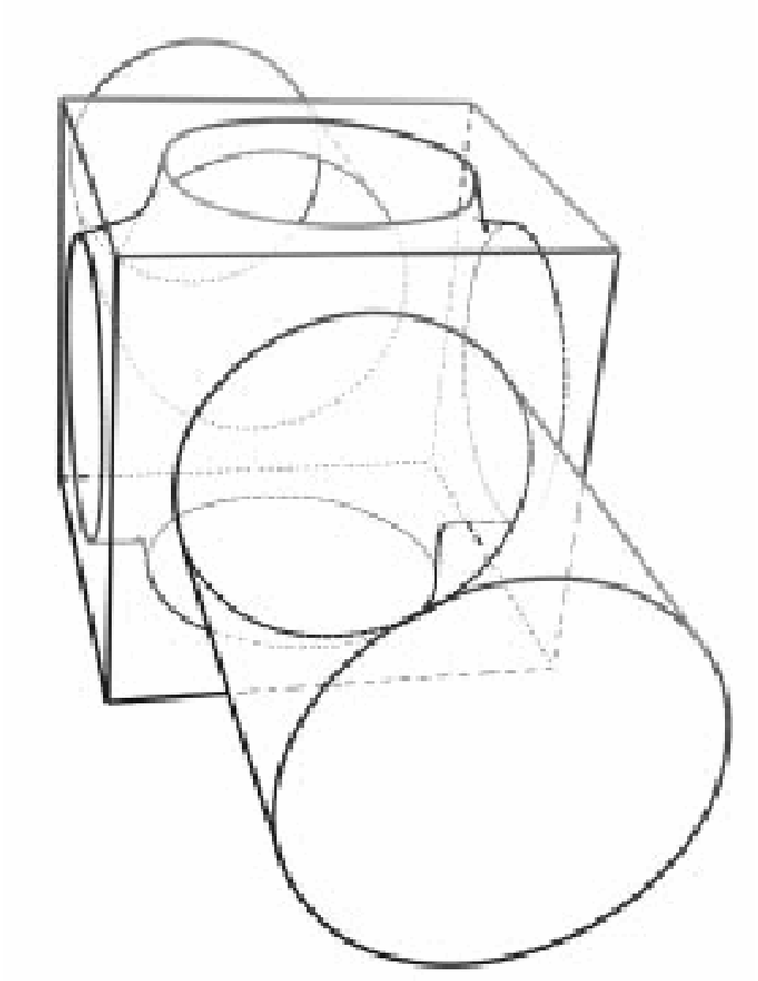
Vpravo: tlak během tahu lineárně klesá

Příklad – osvětlení



Simulace osvětlení zleva shora pomocí tlaku

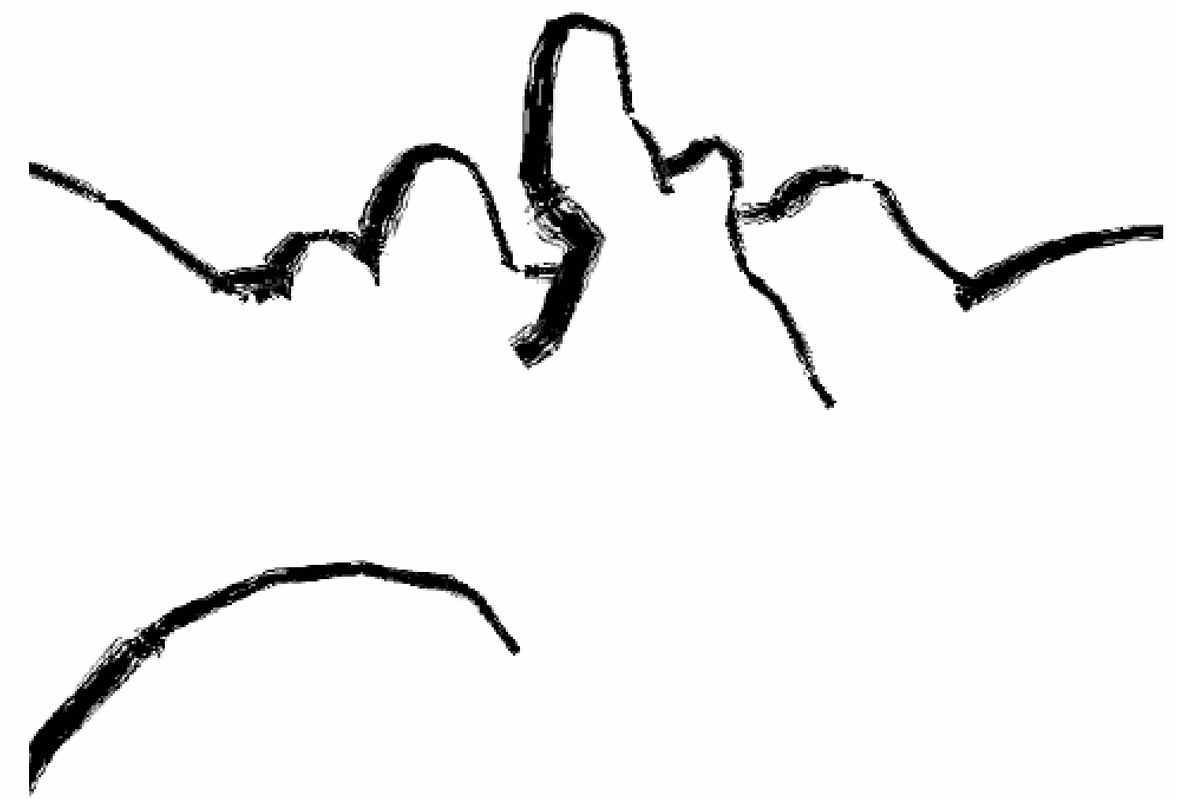
Příklad – skryté hrany



Kreslení pomocí textur

- Podél úseku obrysu (lomené čáry) se vytvoří 2D mesh:
  - V každém vrcholu lomené čáry se vztyčí kolmá příčka
  - Tloušťka (popř. zvolená textura) závisí na osvětlení
  - Textura se namapuje tak, aby se (cca) zachoval její stranový poměr
- S využitím HW je to realtime technika

Textury – příklad



Stylizované hrany

- Cíl: odstranit příliš rovné hrany, nahradit je „přirozenými“ křivkami
- Textura se vybírá z těchto tří:

Kritériem je úhel mezi hranou a jejím následníkem:

$E1 \bullet E2 \leq -\cos(d)$ => použij levotočivou texturu

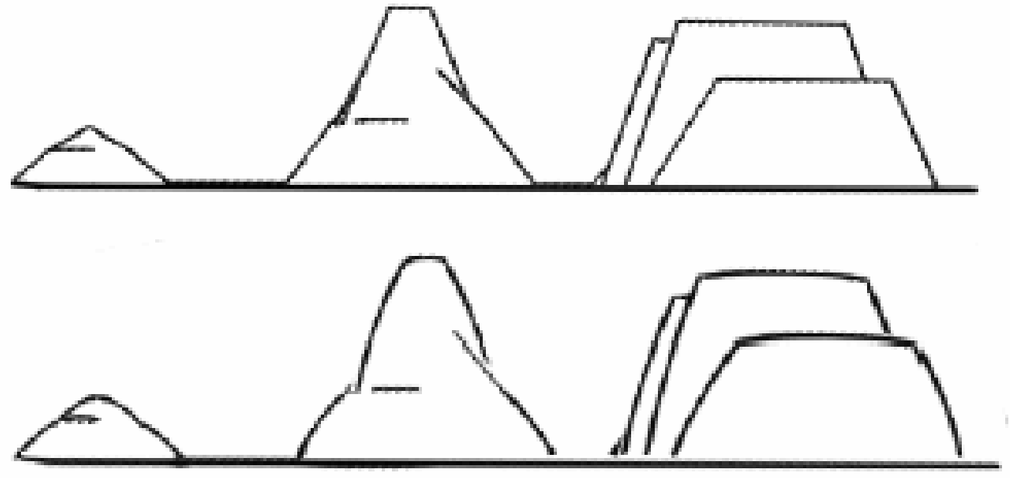
$E1 \bullet E2 \in (-\cos(d), \cos(d))$ => použij rovnou texturu

$E1 \bullet E2 \geq \cos(d)$ => použij pravotočivou texturu

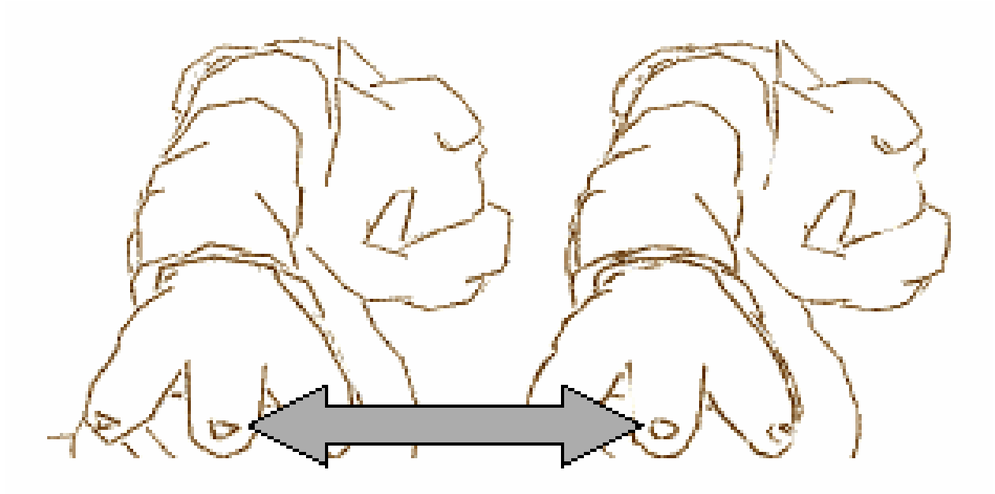
Stylizované hrany

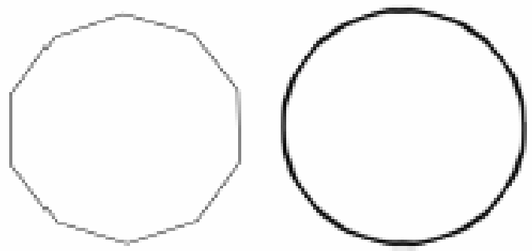
- Dalších efektů lze dosáhnout randomizací tahů
- Realtime technika (textury se vyberou předem)
- Problémy:
  - Mění se geometrie křivek => při použití s vybarvováním nebo šrafováním vznikají artefakty
  - U tlustých čar a hodně zakřivených hran může polygon tvořící čáru prolínat sám sebe

Stylizované hrany – příklad



Stylizované hrany – příklady





Vybarvování a šrafování

- Cílem vizualizovat viditelné povrchy – barvou nebo naznačením jejich tvaru
- Potřebuje informace o 3D scéně
- Předchází před renderováním obrysů
- Mnoho různých přístupů

Cartoons – vlastnosti

- Potlačené stínování, používá se jenom málo barev
- Jednolité barevné plochy
- Ostré přechody

Cartoons – světelný model

- Algoritmus vychází ze zjednodušeného světelného modelu:

$$C = a_g a_m + a_l a_m + d_l d_m max(\vec{L} \cdot \vec{n}, 0)$$

(**a_g** koeficient globálního ambientního světla, **a_l** a **d_l** ambientní a difuzní koeficient světla, **a_m** a **d_m** ditto pro materiál, **L** normovaný vektor od světla k vertexu)

Cartoons – stínování

- Součin **L•n** se interpetuje jako index do 1D textury (gradientu) – typický rozměr je 2x1 pixelu

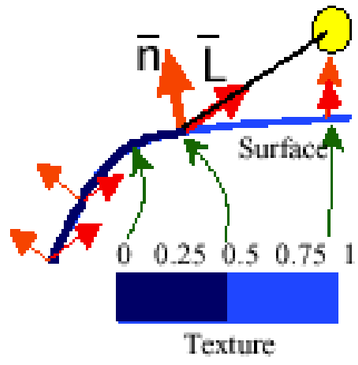


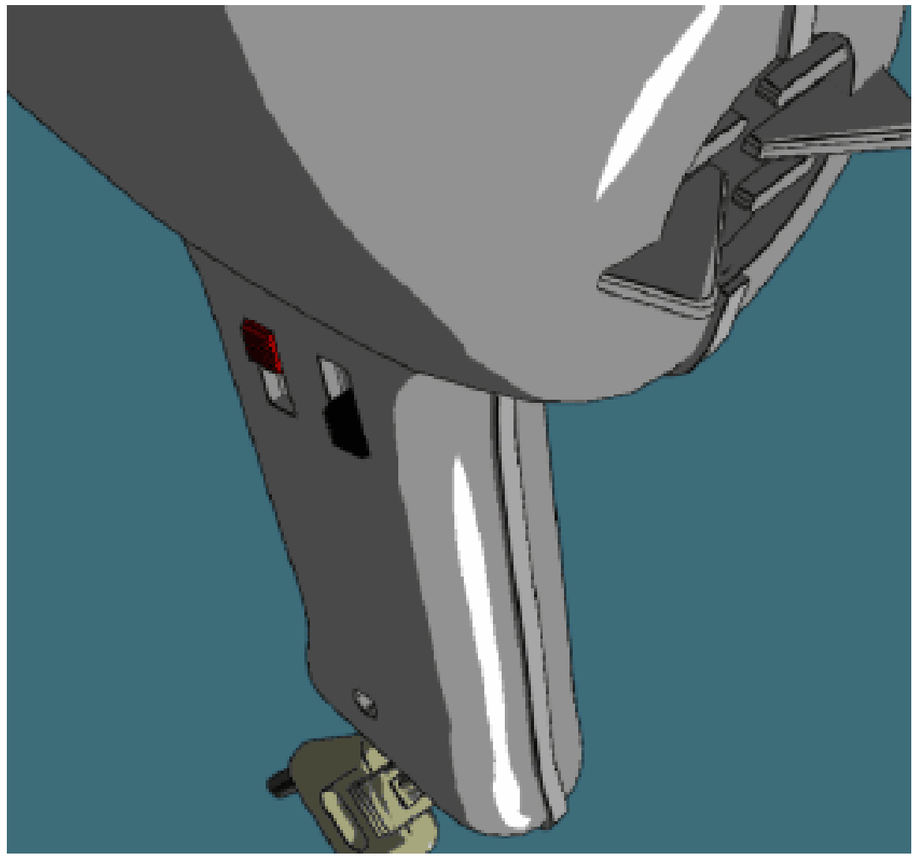
Figure 2: Generation of texture coordinates from $\overline{L} \cdot \overline{n}$. In this case, the shadow boundary occurs at the point where $\overline{L} \cdot \overline{n}$ equals 0.5.

- Vhodnou volbou gradientu lze dosáhnout různých vizuálních stylů
- Kombinuje se s následným vyrenderováním obrysu

Realtime cartoons

- Gradienty lze přímo použít jako textury a renderovat pomocí 3D akcelerátoru
- Problémy, pokud je mesh příliš hrubý:
  - Při blízkém pohledu jsou vidět „zuby“ na přechodech světlo/stín
  - Lze odstranit lineární interpolací, ale přijdeme o ostrost obrazu
  - Pokud je sít dostatečně jemná, tak je interpolace naopak vhodná

Příklad – shadow & highlight



Textura 16x1, 8 texelů barva stínu, 6 nebo 7 texelů difuzní barva osvětleného materiálu, 2 nebo 1 texel highlight (difuzní barva vynásobená highlight faktorem)

Příklad – dark comics

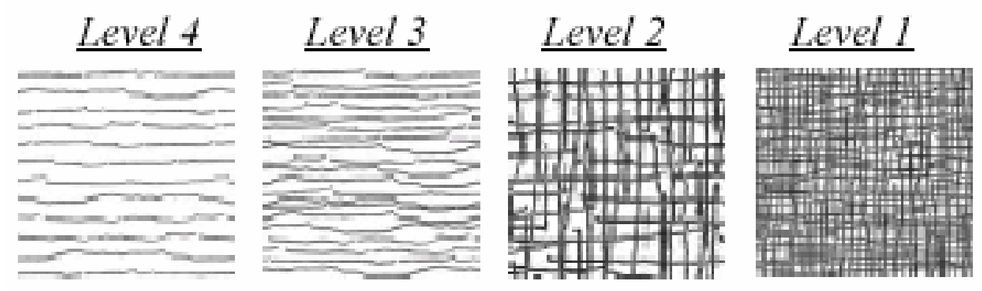
Tělo – textura 8x1, 7 texelů černých, 1 bílý

Oči a náramky – shadow & highlight

Příklad – komplexní scéna

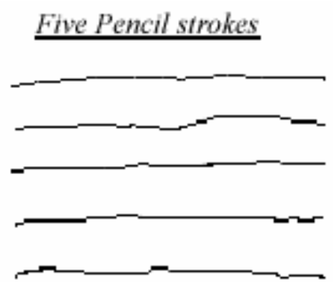
Realtime skicování

- Rozšíříme cartoons renderer tak, aby produkoval výstup působící dojmem kresby tužkou
- Pro každý vertex se opět spočítá **L•n**. Bude sloužit jako index do tabulky 2D textur – na méně osvětlené oblasti se použije hustší textura



Realtime skicování

- Textury v tabulce se vygenerují tak, že se opakovaně náhodně vybírá tah z textury tahů a nanese se v náhodných intervalech



- Pomocí multitexturingu se nejprve nanese textura papíru

Realtime skicování

- Obě textury jsou namapované rovnoběžně s obrazovkou:

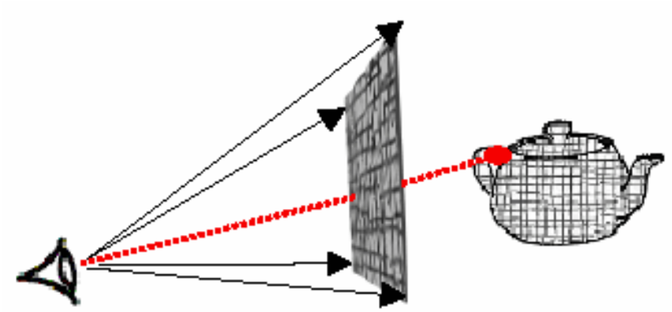
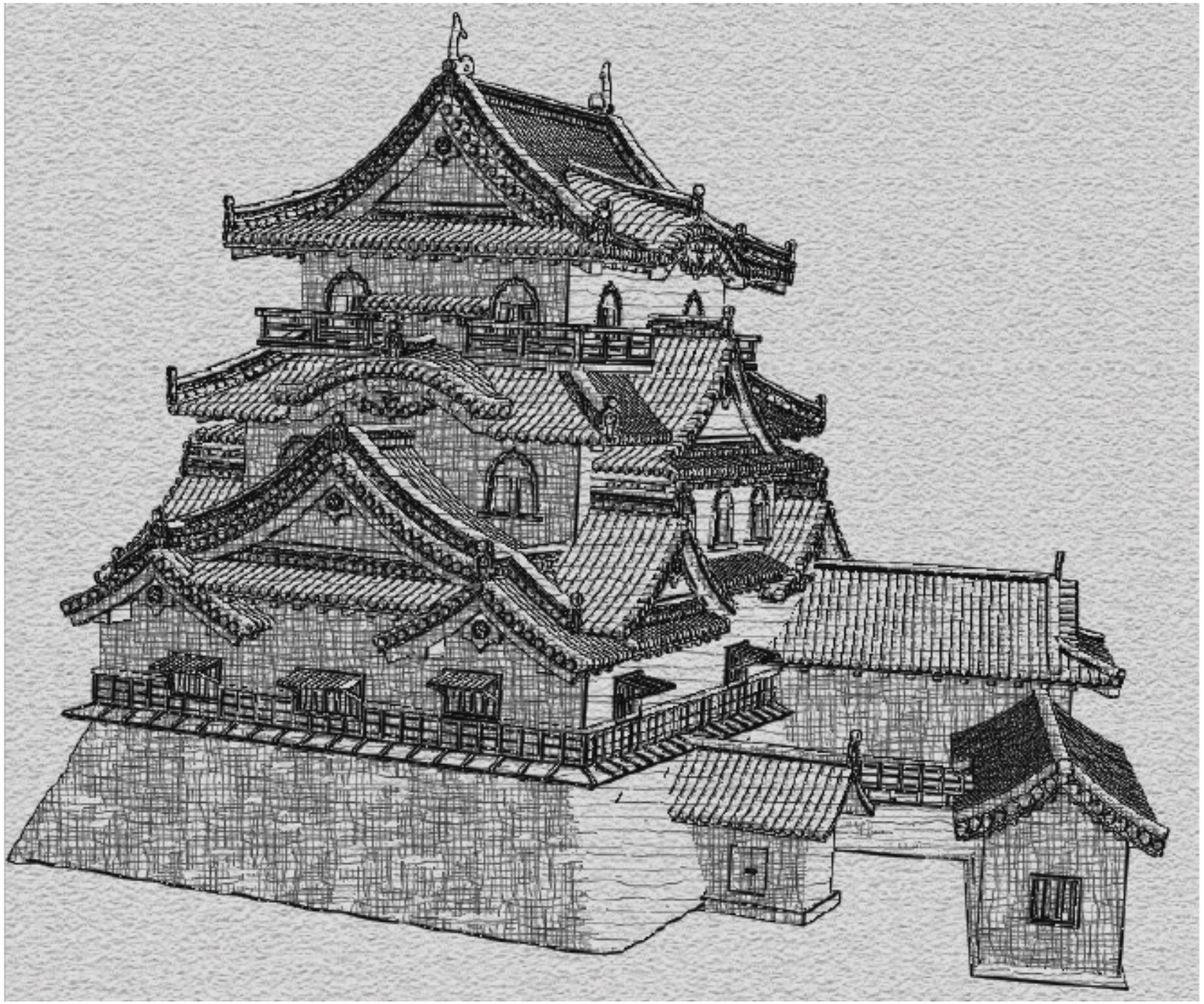


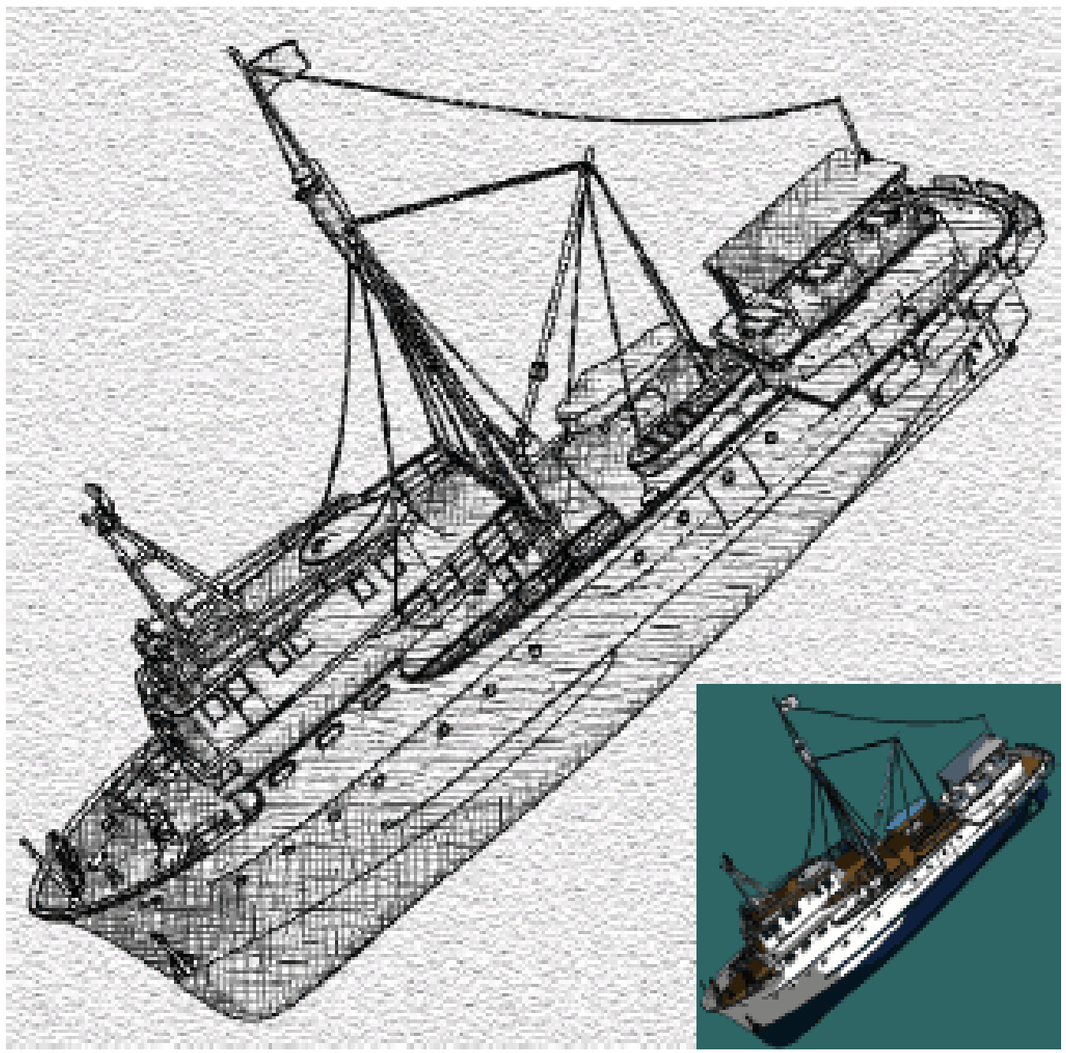
Figure 5: To generate texture coordinates, each texture is projected from the viewport onto the model.

- Důsledek: nelze použít pro animaci (artefakty; model „plave“ napříč texturou)
- Neumí znázornit zakřivení povrchu
- Ale je to rychlé...

Příklad skicy

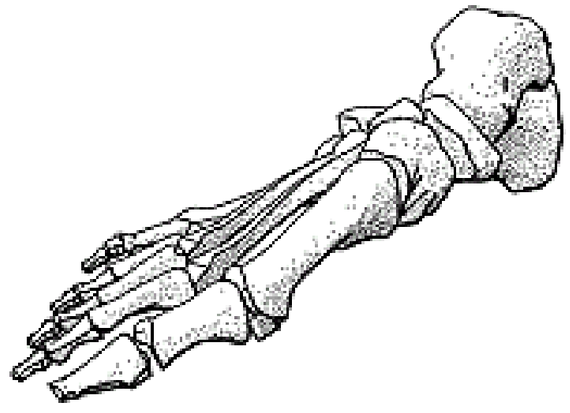


Příklad skicy



Tečkování

- Jednoduchý přístup k šrafování: použije se pouze tečkování
- Implementace: model se vyrenderuje v odstínech šedi (bílý matný materiál) a během postprocessingu se kvantuje do B/W (random dithering)



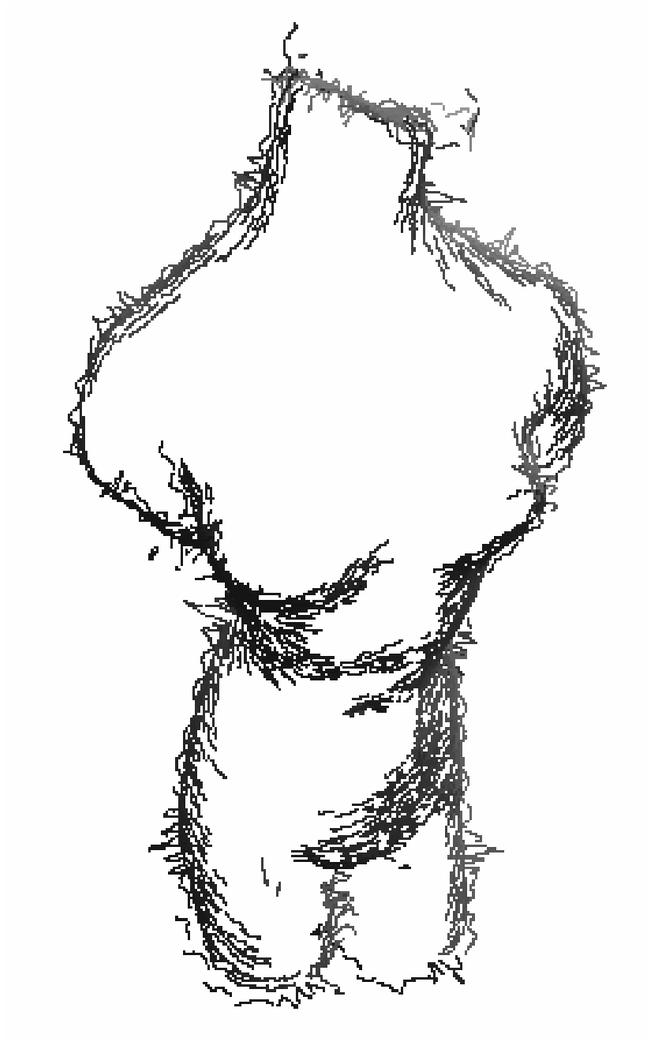
Realtime šrafování

- Předpoklad: světlo je umístěné na kameře
- Viditelné povrchy se rovnoměrně pokryjí částicemi (např. do každého středu trojúhelníku jedna)
- Po perspektivní transformaci jsou díky předpokladu částice husté právě v málo osvícených oblastech

Realtime šrafování

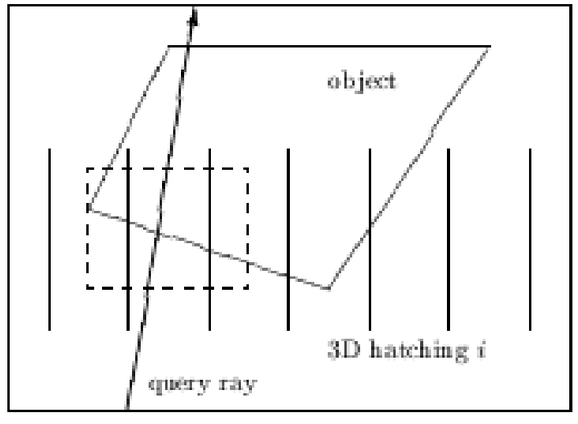
- Do každé částice (mimo těch na světlých místech povrchu) se umístí šrafovací čára konst. velikosti (světové souř.) ve směru $(x_i\text{-}C)\times\mathbf{n_i}$
- Pokud je čára v obrazových souř. příliš velká, nahradí se lomenou čarou, jejíž každý úsek se zorientuje stejným způsobem (např. podle svého středu)
- Čáry se vyrenderují libovolnou technikou

Příklad RT šrafování

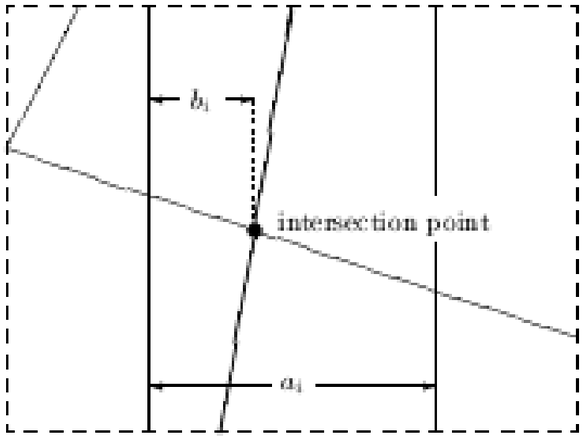


Šrafování raytracingem

- Přístup využívající prostorové textury a postprocessing
- Základem je klasický raytracer
- Renderovaný objekt je proložen rovnoběžnými rovinami – pláty:



Šrafování raytracingem



- Nechť $t=<t_x,t_y,t_z>$ je průsečík paprsku s objektem. Spočítáme normovanou vzdálenost od plátů b_i a použijeme ji jako barvu:

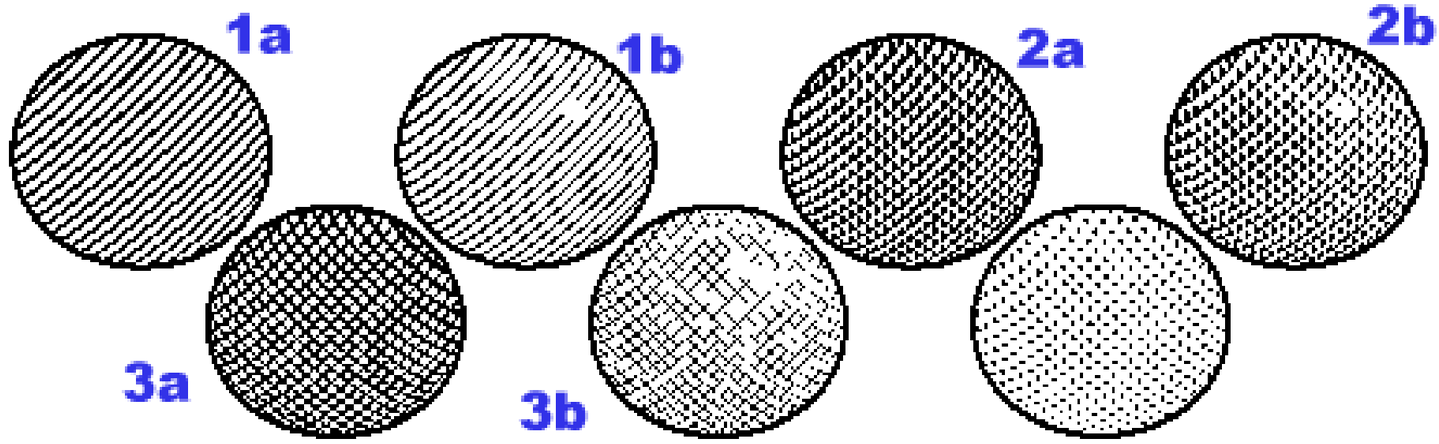
$$f_i = e_{i,1}t_x + e_{i,2}t_y + e_{i,3}t_z + e_{i,4}$$

$$b_i = \begin{cases} (f_i \bmod a_i)/a_i, & f_i \geq 0 \\ (a_i - f_i \bmod a_i)/a_i, & f_i < 0 \end{cases}$$

Šrafování raytracingem

- Postprocessing: prahový operátor
- Lze kombinovat několik plátových bloků dohromady – $\sum wibi$ nebo $\max\{bi\}$
- Vhodné udělat vážený součet s jasem povrchu => na světlých plochách bude méně šrafování
- Metoda se hodí jenom pro jednoduché objekty

Příklad



a) bez přidání jasové složky; b) s přidáním jasu

1) šrafování

2) dva směry šrafování, je vidět trojúhelníkový vzor

3) dva směry šrafování s použitím max

Zdroje

Aaron Hertzmann's papers (http://www.mrl.nyu.edu/hertzmann)

http://www.red3d.com/cwr/npr/

ResearchIndex: The NECI Scientific Literature Digital Library (http://citeseer.nj.nec.com/)

SIGGRAPH papers