ADVANCED TEXTUR MAPPING II (bump, parallax and displacement)

PETR FELKEL

DCGI FEL CTU PRAGUE, KN:E-413

felkel@fel.cvut.cz

https://cent.felk.cvut.cz/courses/PGR2

Version from 26.3.2015

Dnešní témata

- Vnímání tvarů v grafice a textury
- Tečný prostor (*tangent space*)
- Fake metody
  - *Bump mapping*
  - *Parallax mapping*
- Změna geometrie
  - *Displacement mapping*

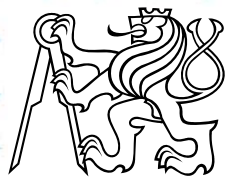
DCGI

Vnímání tvarů v grafice a textury

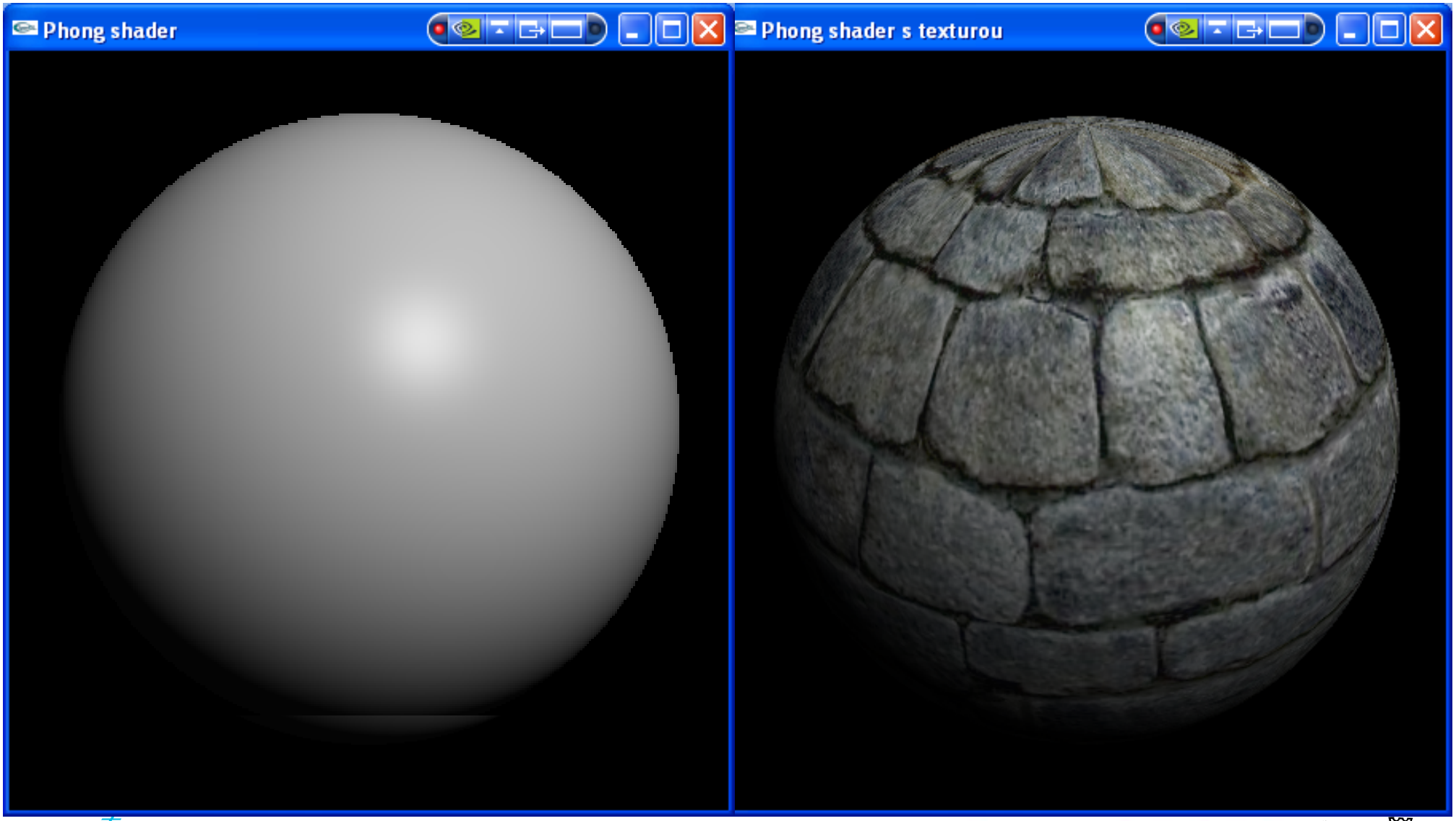
- Tvar povrchu vnímáme
  - Nahrubo ze zadané geometrie – podle hran a obrysů
  - Detaily vnímáme podle osvětlení
- Osvětlení závisí na normále povrchu
  - Ve fixním řetězci se počítá *ve vrcholech* (Phongův osv. model)
    - Pro fragmenty mezi vrcholy se při rasterizaci *interpoluje*
    - Interpoluje se výsledná barva – Gouraudovo stínování
  - V programovatelném řetězci (shadery)
    - Pro fragmenty mezi vrcholy se interpolují vektory (normála, směr ke světlu a k pozorovateli)
    - a osvětlení se počítá pro každý pixel (Phongovo stínování)
  - Menší detaily než zadaná geometrie se nezobrazí

Přidání detailů v textuře s lokálními změnami

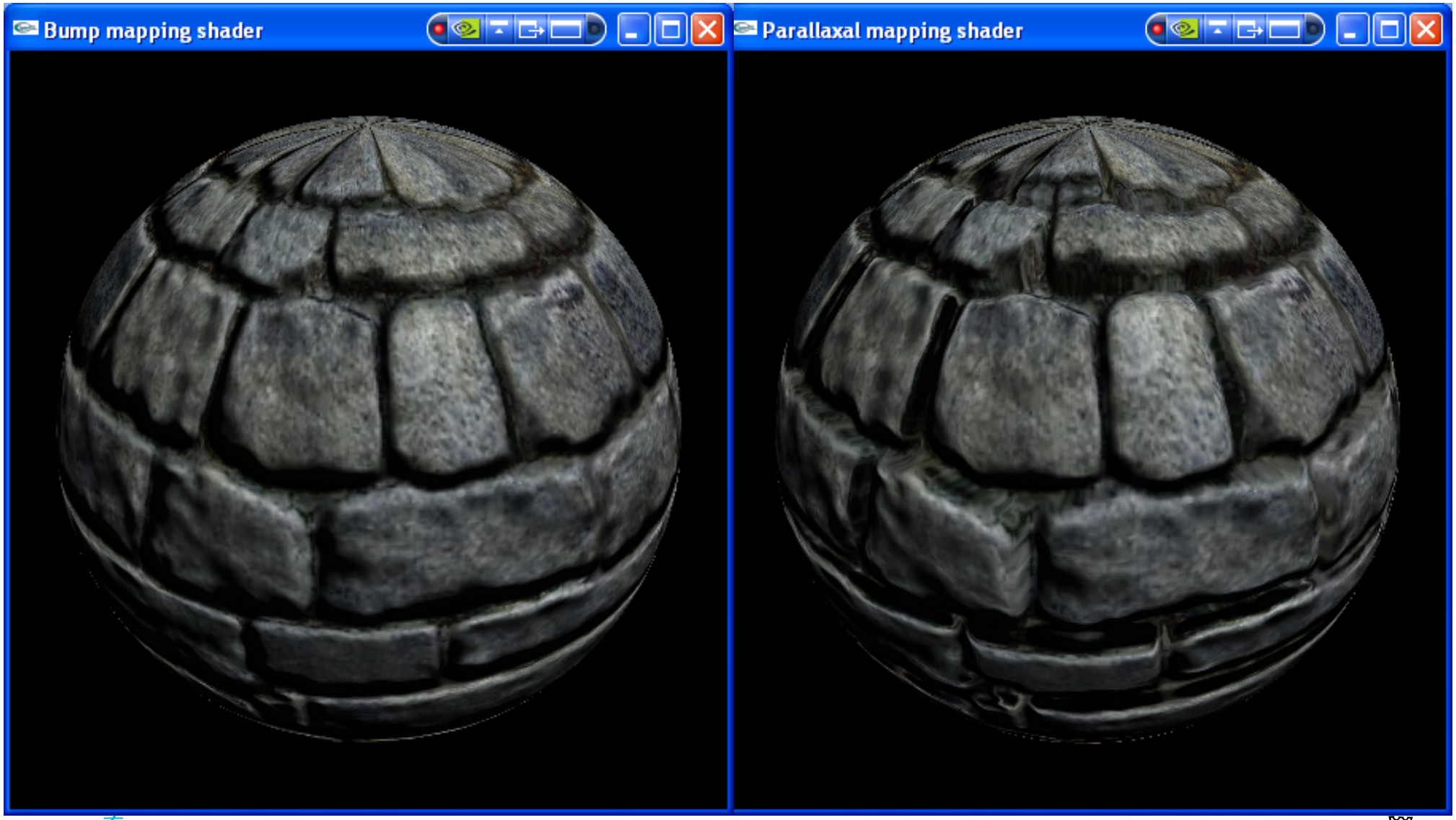
1. Modulovat vypočítané osvětlení texturou
   - Difúzní textura – „nalepený“ obrázek
2. Počítat osvětlení na základě informací z textury
   - Výškovou mapou – *bump mapping*
     - lokální posunutí geometrie (ve směru *n*)
     - z ní se odvodí jemnější normály
   - Normálovou mapou – *normal mapping*
     a) naklopené normály relativně vůči povrchu (v tečném prostoru - *tangent space*)
     b) textura s normálami kolem celého objektu (v objektovém prostoru - object space) – absolutní
3. Měnit přímo geometrii a posouvat vrcholy
   - Displacement mapping a výšková mapa



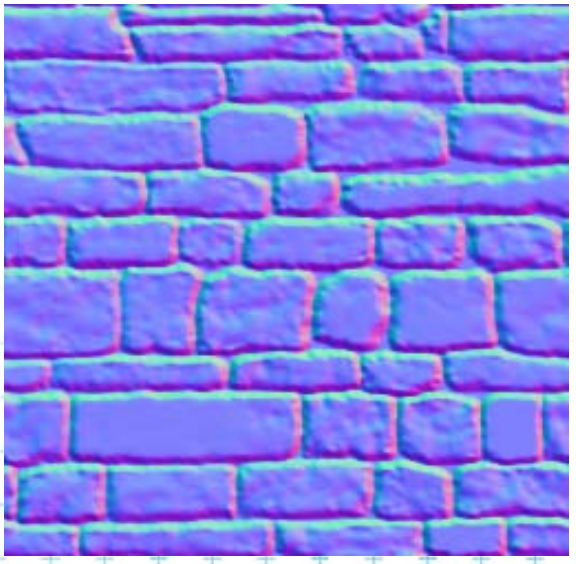
Phongovo stínování a difúzní textura



Bump mapping a parallax mapping



Difúzní textura, výšková a normálová mapa



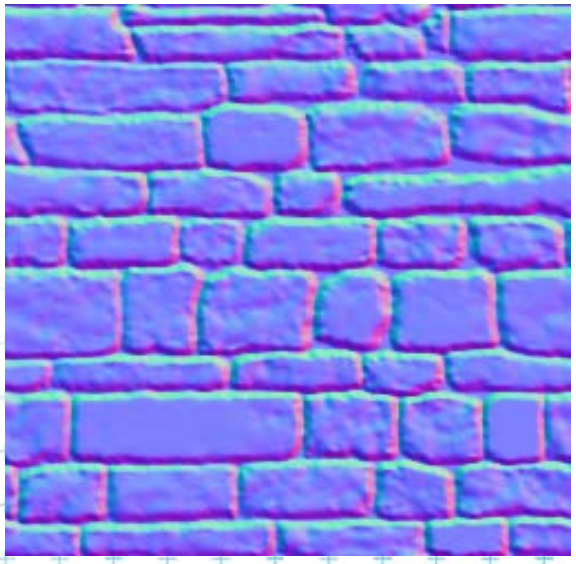
Výšková a normálová mapa

Výšková mapa

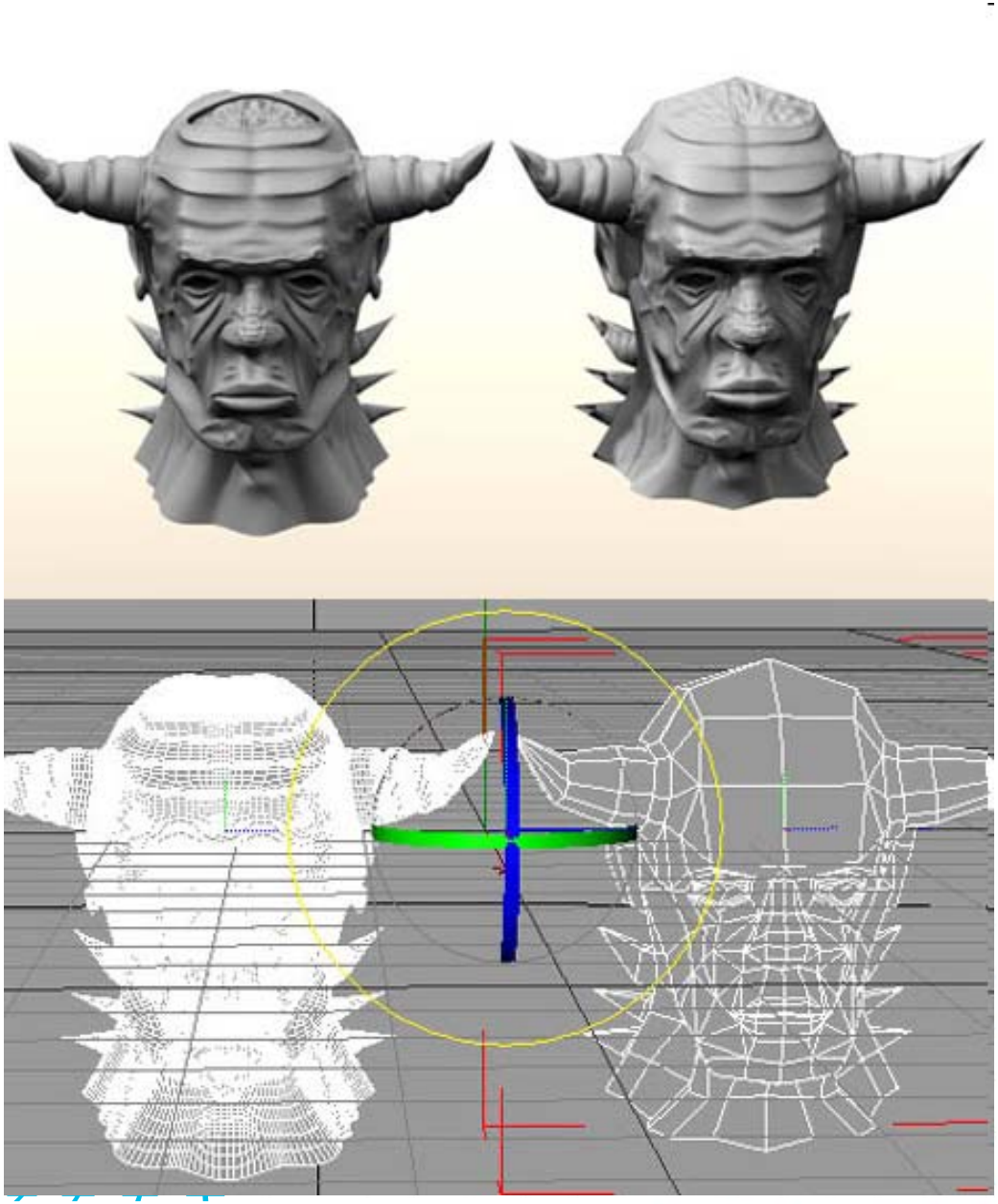
- Monochromatická textura
- Obsahuje výšku fragmentů nad rovinou

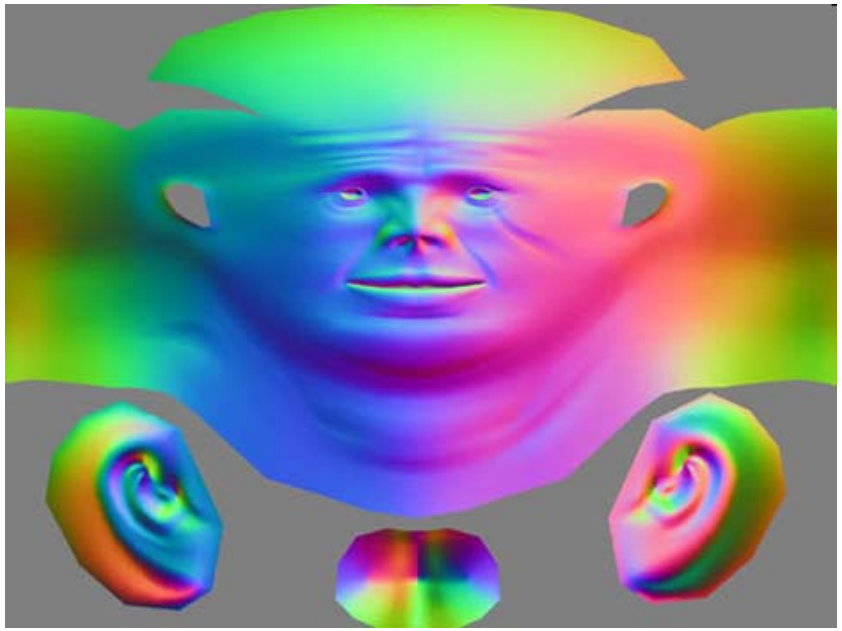
Normálová mapa – v tečném prostoru

- RGB textura
- Obsahuje normály fragmentů (odchylky od interpolovaných normál v rámci tečné roviny – *tangent space*)
- (0,0,1) = neodchýlená normála
- Proto je textura do modra

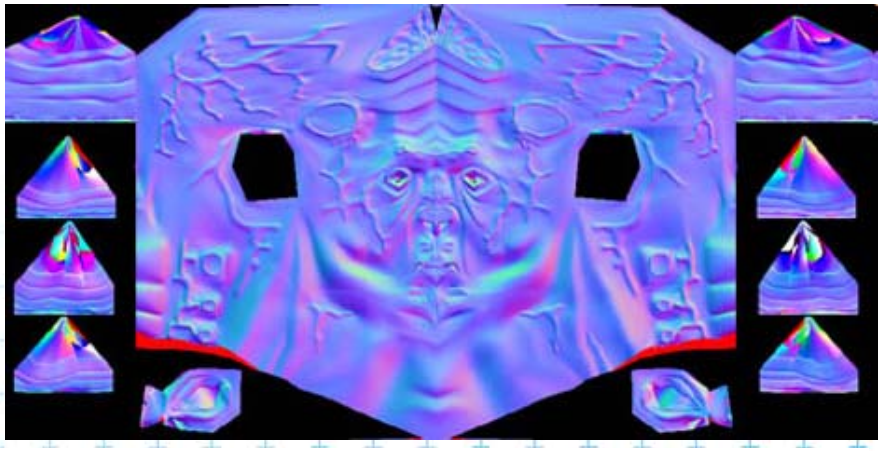


Složitý model a jednodušší s normálovou mapou





Normálová mapa v prostoru objektu



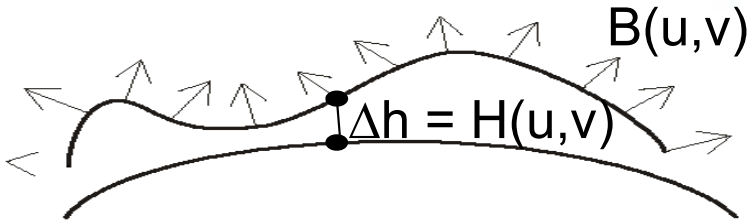
Normálová mapa v tečném prostoru

Princip metody *bump mapping*

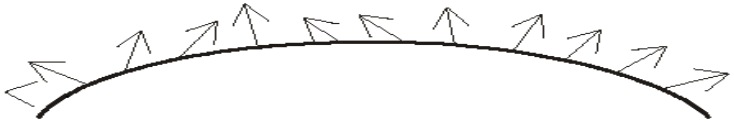
- **Výšková mapa** obsahuje posunutí vůči původnímu povrchu ve směru normály (skalární hodnoty Δh)
  - Z posunutí se vypočte naklonění normál
  - Vypočtené naklonění naklání normály - *bump mapping*
  - Nebo se normály nahradí normálami z textury - *normal mapping*
- Nakloněné normály se použijí při zobrazení modelu
- Nemění se geometrie modelu, ale jen normály fragmentů

1. původní povrch objektu O(u,v)

2. textura posunů (height field) H(u,v)
nebo textura s náklony normál B(u,v)



c) 3. B(u,v) mapováno na O(u,v)

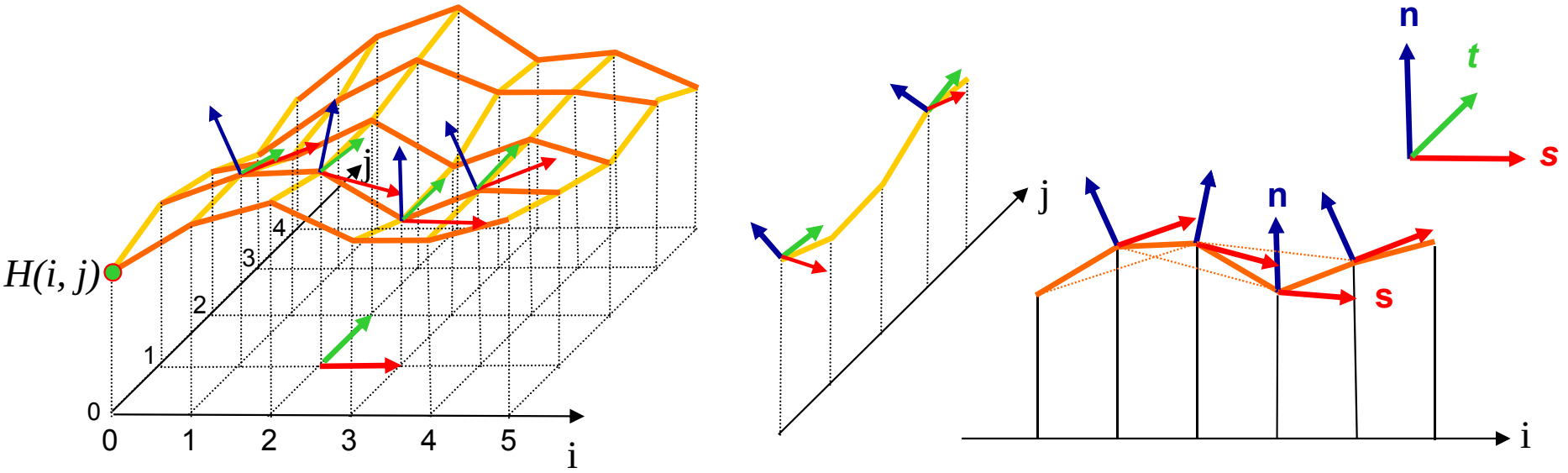


4. změněné normály
na původní geometrii

[MPG 2004]

Konstrukce normálové mapy z výškové

dle [Lengyel, 182]



Vstup: Výšková mapa $\boldsymbol{H(i,j)}$ = posunutí (*offset*) povrchu vůči rovině

$$\boldsymbol{s}(i,j) = \langle 1 \quad 0 \quad aH(i+1,j) - aH(i-1,j) \rangle$$ Vektor tangenty ve směru $\boldsymbol{s}$

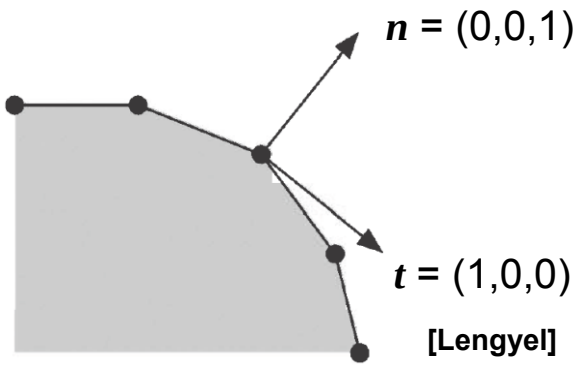
$$\boldsymbol{t}(i,j) = \langle 0 \quad 1 \quad aH(i,j+1) - aH(i,j-1) \rangle$$ Vektor tangenty ve směru $\boldsymbol{t}$

a = míra vychýlení normál

$$\boldsymbol{n}(i,j) = \frac{\boldsymbol{s}(i,j) \times \boldsymbol{t}(i,j)}{\|\boldsymbol{s}(i,j) \times \boldsymbol{t}(i,j)\|} = \frac{\langle -\boldsymbol{s}_z, -\boldsymbol{t}_z, 1 \rangle}{\sqrt{-\boldsymbol{s}_z^2 + \boldsymbol{t}_z^2 + 1}}$$

Vektor normály -> RGB a do normálové mapy N

Tečný prostor (*tangent space*)

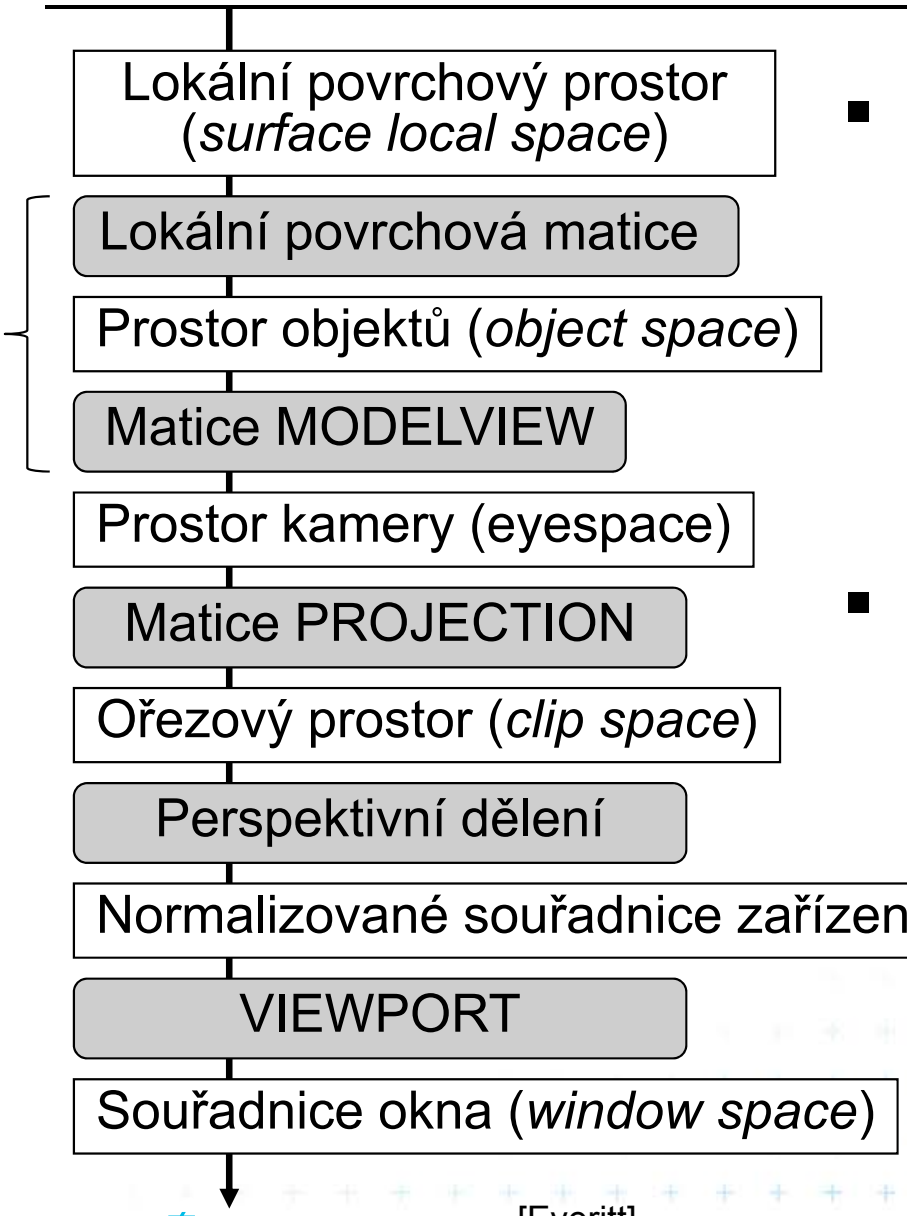


- Vektor (0,0,1) v normálové mapě odpovídá nenakloněné normále
- Aby odpovídal interpolované normále ve vrcholu, konstruujeme tečný prostor (*tangent space*)
- Směry ***s*** a ***t*** (x a y) musí odpovídat směrům *s* a *t* textury
- Leží v tečné rovině

Tečný prostor (*tangent space*) - výhody

- Normálové vektory v normálové mapě
  - Lze chápat jako perturbace skutečné normály (0,0,1)
  - Jsou nezávislé na souřadné soustavě objektu
  - Nemusíme proto normály a perturbované normály transformovat a následně normalizovat
  - Přímo je použijeme pro výpočet osvětlení
- Osvětlení počítáme v tečném prostoru
  - Ve VS transformujeme souřadnice vektorů $\boldsymbol{n}, \boldsymbol{t}, \boldsymbol{b}$ ve vrcholech z prostoru kamery do tečného prostoru
  - Rasterizátor interpoluje hodnoty vektorů $\boldsymbol{n}, \boldsymbol{t}, \boldsymbol{b}$
  - FS vypočítá osvětlení v místě fragmentu

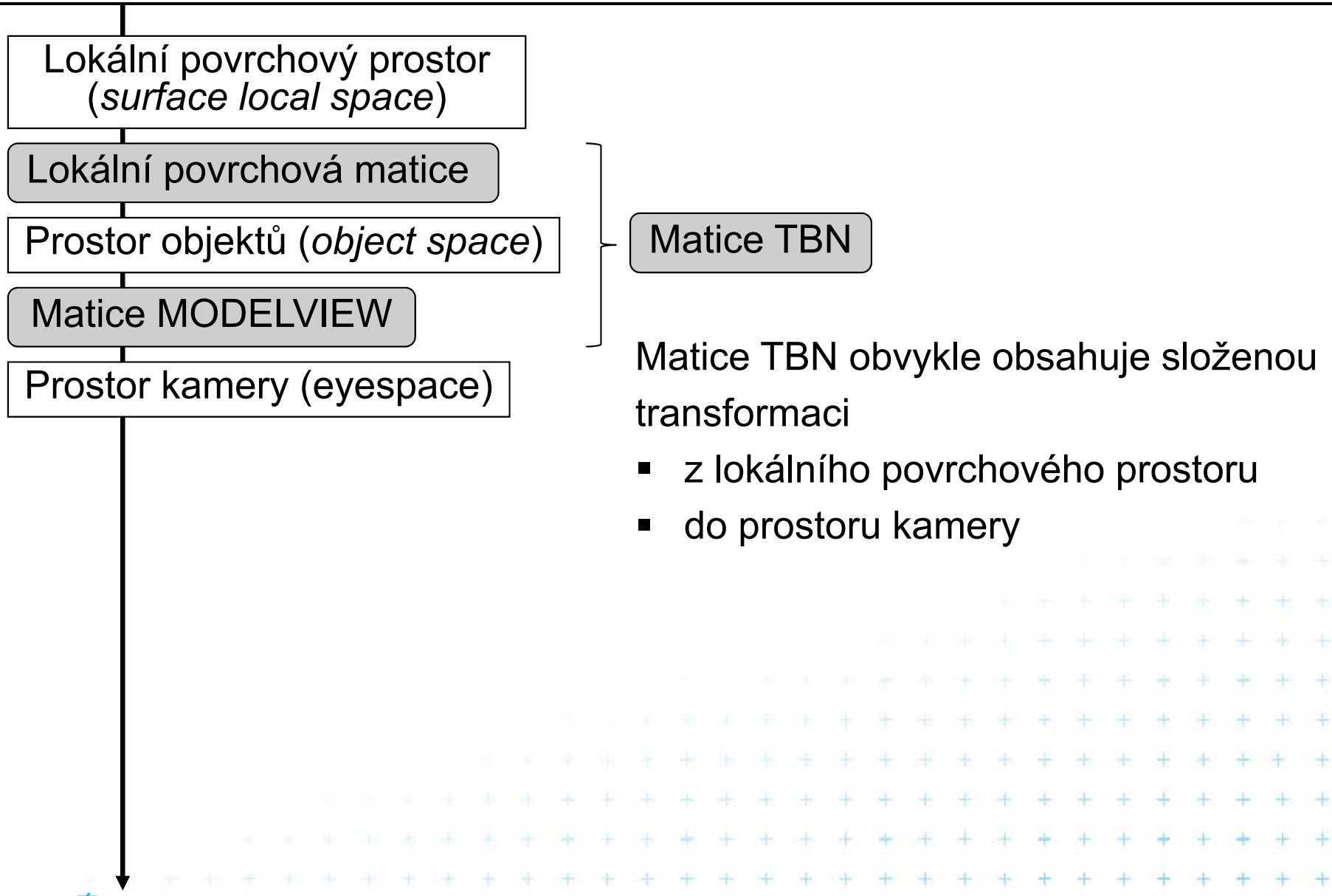
Lokální povrchový prostor



- Lokální povrchový prostor
  = třída prostorů, definovaných pro každý bod na povrchu
  - Prostor textur (s,t,r,q)
  - Tečný prostor (t,b,n)
- Tečný prostor – *tangent space*
  - Souřadné osy ***x*** a ***y*** jsou ortogonální a leží v tečné rovině
  - Souřadná osa ***z*** leží v normále k povrchu (k zadané geometrii)

[Everitt]

Lokální povrchová matice TBN



Matice TBN obvykle obsahuje složenou transformaci

- z lokálního povrchového prostoru
- do prostoru kamery

Lokální povrchová matice TBN

= Matice pro přechod z *tečného prostoru* do prostoru *kamery*

- Pro každý vrchol je jiná matice

Matice po sloupcích

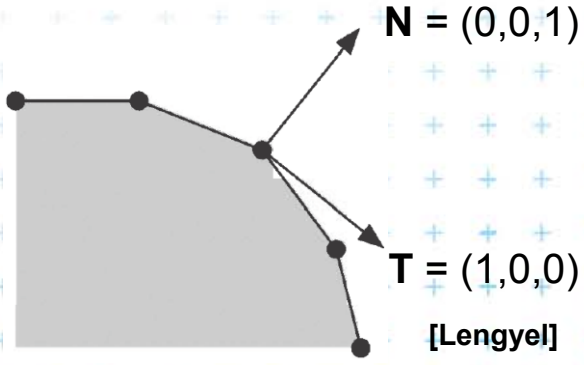
- Definují ji obrazy vektorů báze v souřadnicích kamery

$$F = \begin{bmatrix} t_x & b_x & n_x & p_x \\ t_y & b_y & n_y & p_y \\ t_z & b_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

t – tangenta (v jednom ze směrů *u* či *v*)
b – binormála, bitangenta (**b** = **n** × **t**)
n – normála ve vrcholu
t,b,n jsou v prostoru objektů
p – pozice vrcholu v prostoru objektů
(pro vektory se neprojeví)

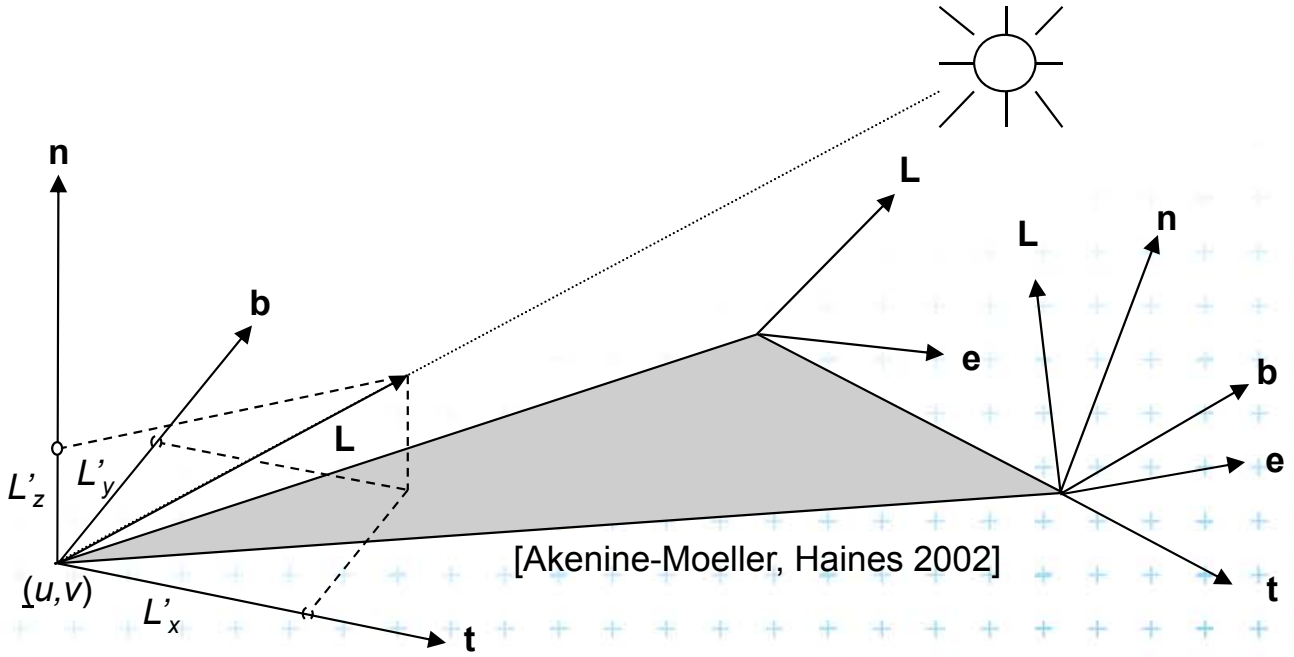
- Všechny vrcholy jsou v tečném prostoru stejné!

N = (0,0,1); pozice V = (0,0,0);



Transformace vektoru L do tečného prostoru

- Vektor z aktuálního vrcholu ke světlu L transformujeme do tečného prostoru a v tečném prostoru spočítáme osvětlení
  - Difúzní - z normály a transformovaného *L*
  - Spekulární - z normály a transformovaného *e* a *L*
- Normály se naklání v tečném prostoru



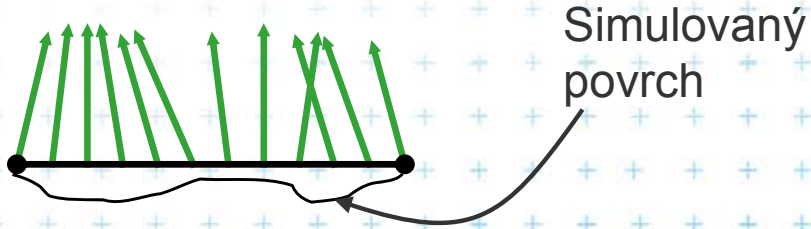
[Akenine-Moeller, Haines 2002]

Výpočet osvětlení lze i v prostoru kamery

- Pro osvětlení ve vrcholech
  - Transformuje se normála,
  - Vektory *e* = (0,0,0) a *L* jsou k dispozici

Normály per-vertex

- Při osvětlení per-pixel
  - by se všechny normály po přečtení z normálové mapy transformovaly do prostoru kamery a musely by se normalizovat!!! Větší zátěž fs.
  - Proto se osvětlení počítá v tečném prostoru

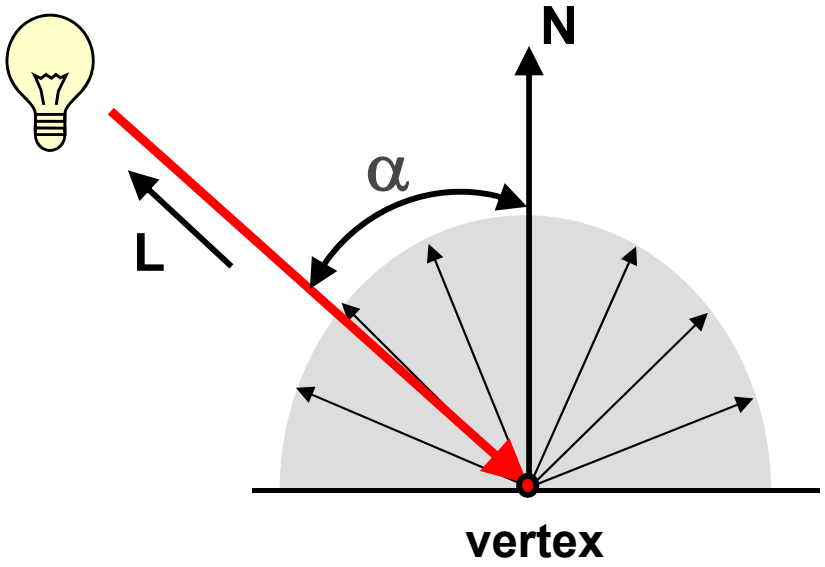


Normály per-pixel

[Everitt]

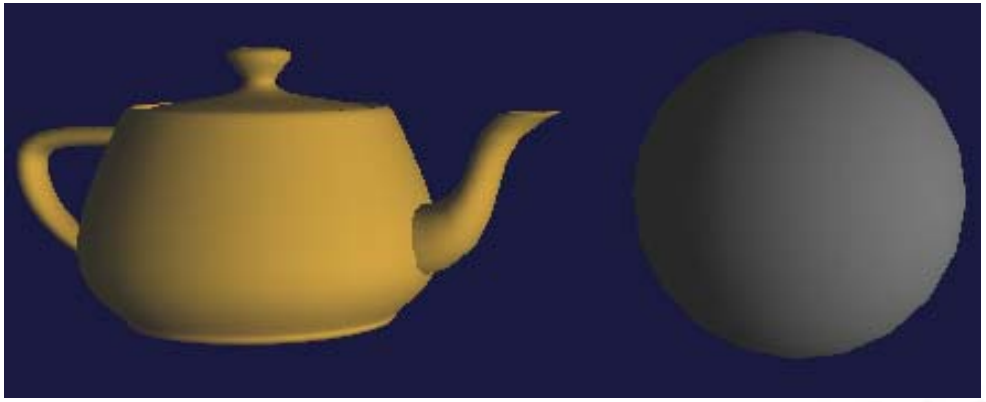
Složky odraženého světla

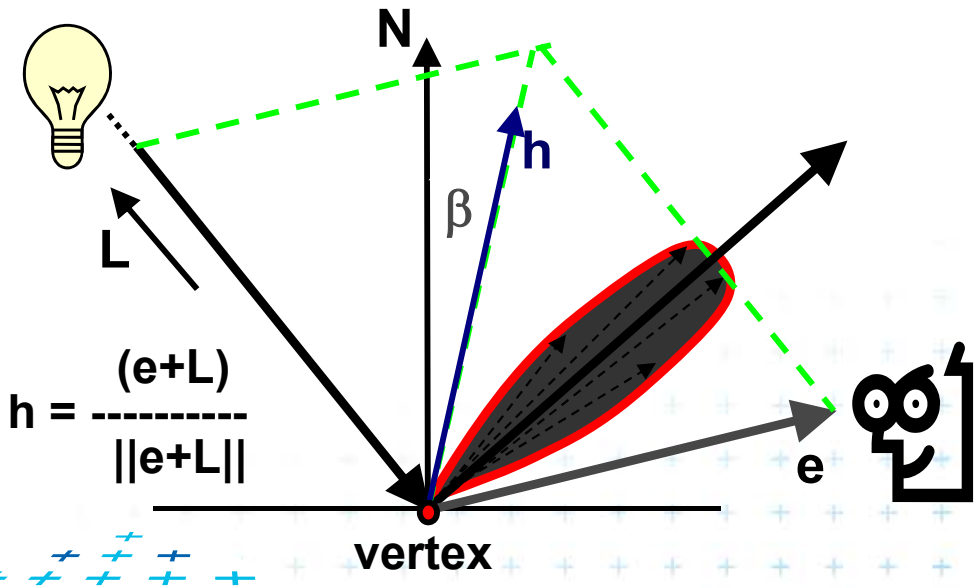
Platí pro normalizované vektory (délky 1)



- Difúzní složka světla

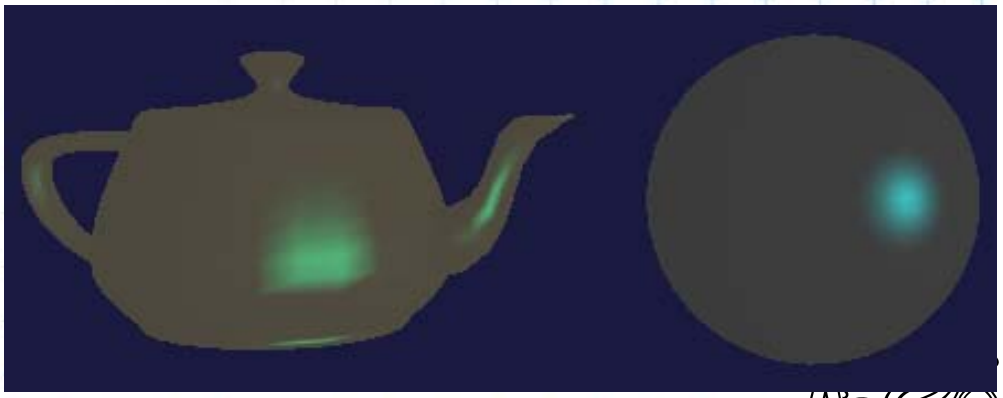
$$d_{ref} = (\max[N.L, 0]) * d_{light} * d_{material}$$





- Zrcadlová složka

$$s_{ref} = (\max[N.h, 0])^{sh} * s_{light} * s_{material}$$



Proč je matice někdy sloupcová a někdy řádková?

Matice F pro přechod z *tečného prostoru* do prostoru *kamery*

$$F = \begin{bmatrix} t_x & b_x & n_x & p_x \\ t_y & b_y & n_y & p_y \\ t_z & b_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matice po sloupcích

t -- object space tangent vector
b -- object space binormal (bitangent) vec.
n -- object space vertex normal
p -- object space vertex position
(pro vektory se neprojeví)

Matice F^{-1} pro přechod z prostoru *kamery* do *tečného prostoru*

$$F^T = \begin{pmatrix} t_x & t_y & t_z & -tp \\ b_x & b_y & b_z & -bp \\ n_x & n_y & n_z & -np \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

transformuje vektory ke světlu a k pozorovateli do tečného prostoru (tangent space)

pokud *t,b,n* ortonormální!!!
pak stačí jen transponovat lineární část

Matice po řádcích

Proč je matice někdy sloupcová a někdy řádková? 1/2

- Transformují se vektory $(x,y,z,0)$ => stačí matice 3x3
- Transformuje vektory z prostoru *kamery* do prostoru *textur*

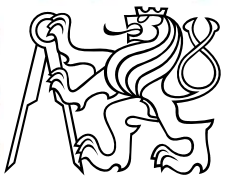
$$\begin{pmatrix} s_x \\ s_y \\ s_z \end{pmatrix} = \begin{pmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \\ n_x & n_y & n_z \end{pmatrix} \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} = \begin{bmatrix} o_x & o_y & o_z \end{bmatrix} \begin{pmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{pmatrix}$$

Tj. trojice skalárních součinů

$$s_x = t_x o_x + t_y o_y + t_z o_z = O.T = dot(O, T) = \text{dot}(T, O)$$

$$s_y = b_x o_x + b_y o_y + b_z o_z = O.B = dot(O, B) = \text{dot}(B, O)$$

$$s_z = n_x o_x + n_y o_y + n_z o_z = O.N = dot(O, N) = \text{dot}(N, O)$$



Proč je matice někdy sloupcová a někdy řádková? 2/2

*) podobně, jako v OpenGL

- Pozor na GLSL
- Matice se zadává *po sloupcích*[*)]

```
T = vec3 (…);
TBN = mat3( T, B, N );
```

$$\begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix}$$

- Místo TBN^T se prohodí pořadí násobení vektoru maticí

```
s = o * TBN;
```

$$\begin{bmatrix} s_x & s_y & s_z \end{bmatrix} = \begin{bmatrix} o_x & o_y & o_z \end{bmatrix} \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix}$$

Zadáme vektory po složkách sloupcově

Zadání matice po sloupcích

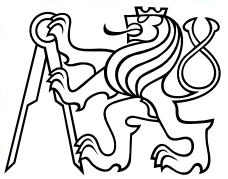
```
TBN = mat3( T, B, N );
```

$$\begin{pmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{pmatrix}$$

Zadání matice po řádcích

```
TBN = mat3(
          T.x, B. x, N.x,
          T.y, B. y, N.y,
          T.z, B. z, N.z
);
```

$$\begin{pmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \\ n_x & n_y & n_z \end{pmatrix}$$

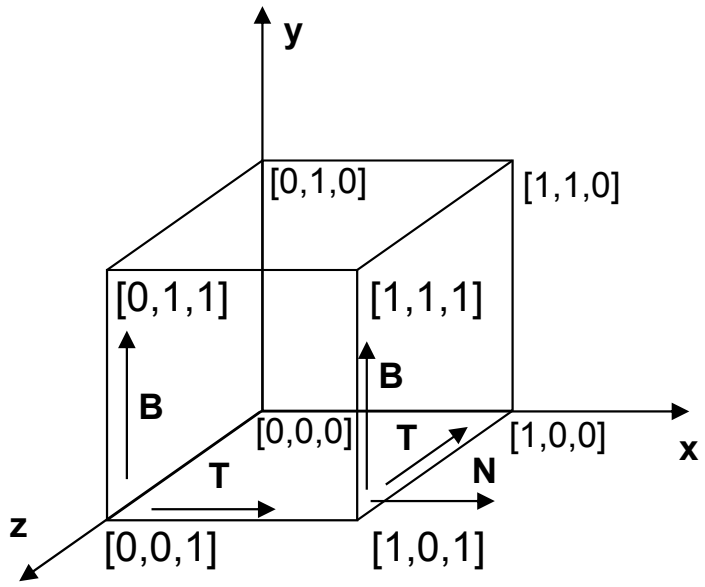


Získání matice TBN v praxi

- Vektory T,B,N se předpočítají na CPU
- a předají se vertex shaderu jako *atributy*

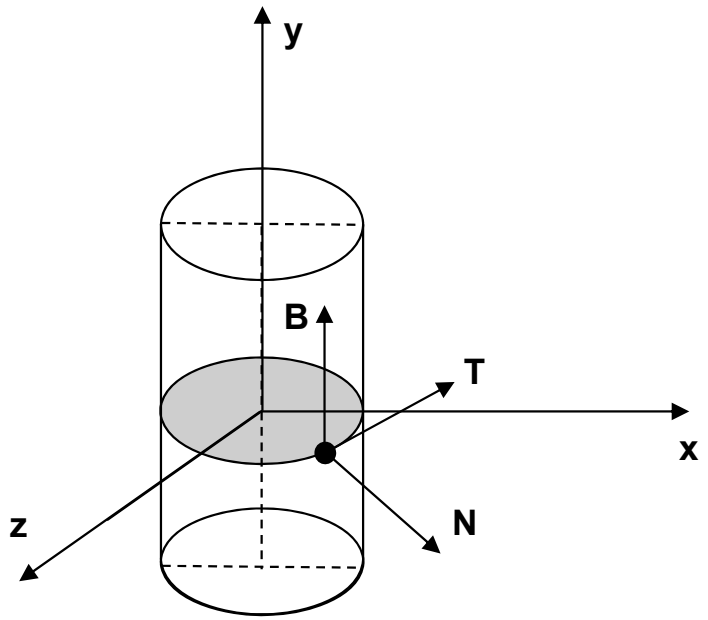
- Pro analyticky zadaná tělesa spočítáme derivace
- Pro jednoduchá tělesa zadáme přímo – viz dále krychle a válec
- Popis výpočtu pro libovolná tělesa se zdrojovým kódem viz [Lengyel a Terathon]

T,B,N pro krychli – v prostoru objektů



- Přední stěna
  - **T** = (1,0,0)
  - **B** = (0,1,0)
  - **N** = (0,0,1)
- Pravý bok
  - **T** = (0,0,-1)
  - **B** = (0,1,0)
  - **N** = (1,0,0)
- …

T,B,N pro plášť válce



- Obvodová stěna
  - **B** = (0,1,0)
  - **N** = (x,0,z) / ||x+z||
    [x,0,z] bod kružnice
  - **T** = **B** x **N**
- **Horní stěna**
  - **B** = (-x,0,-z) / ||x+z||
    [x,0,z] bod kružnice
  - **N** = (0,1,0)
  - **T** = **B** x **N**

Normalizace

Poznámka: **Vektorový součin T = B x N**

Výsledkem vektorového součinu dvou vektorů je vektor na ně kolmý, jeho délka je rovna orientovanému obsahu rovnoběžníka …

Pro ortonormální vektory **B** a **N** tudíž není nutno **T = B x N** normalizovat

Optimalizace

- Příspěvky světla lze počítat odděleně
- Co se nehýbe, lze předpočítat a uložit do textury
  - Difúzní osvětlení od nepohyblivých světel
  - Spekulární složka, pokud se nehýbe pozorovatel
- Co se hýbe, počítá se
  - Pohyblivé světlo
  - Spekulární osvětlení

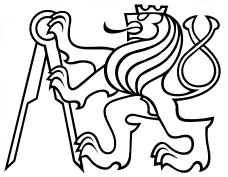
Parallax mapping with offset limiting

Co je Parallax Mapping?

- metoda **aproximace** hrbolatých povrchů prostřednictvím modifikace souřadnic textury

- zjednodušení, které vyvolá dojem hrbolatého povrchu bez nutnosti měnit geometrii (a zjemňovat ji)

- technika snadno realizovatelná na GPU



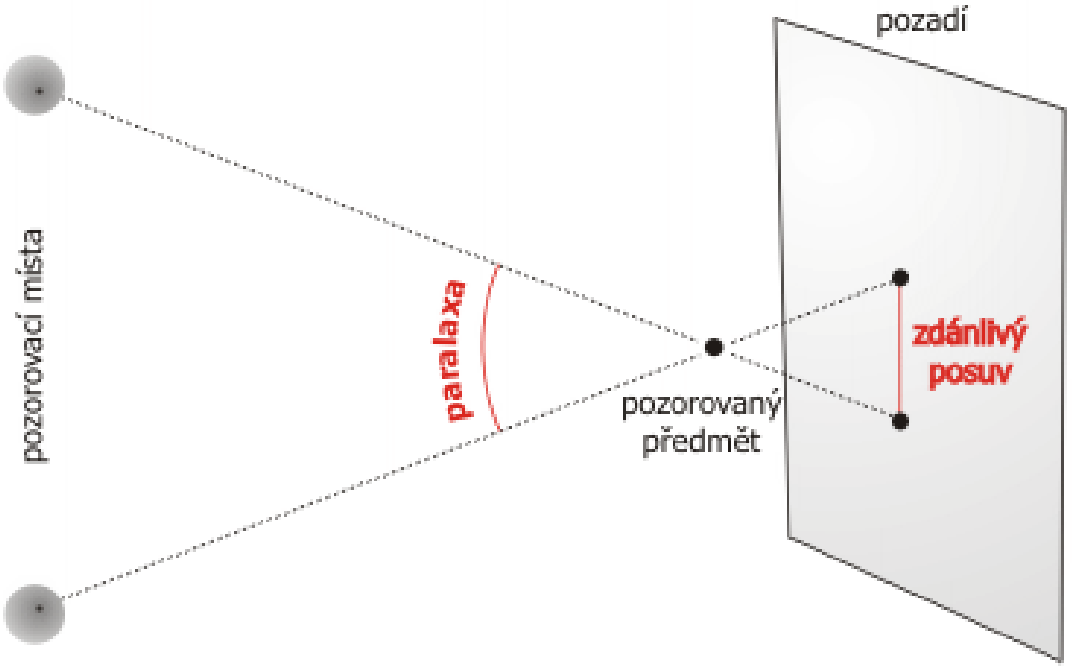
Co je Parallax Mapping?

[Welsh04]

Co je paralaxa?

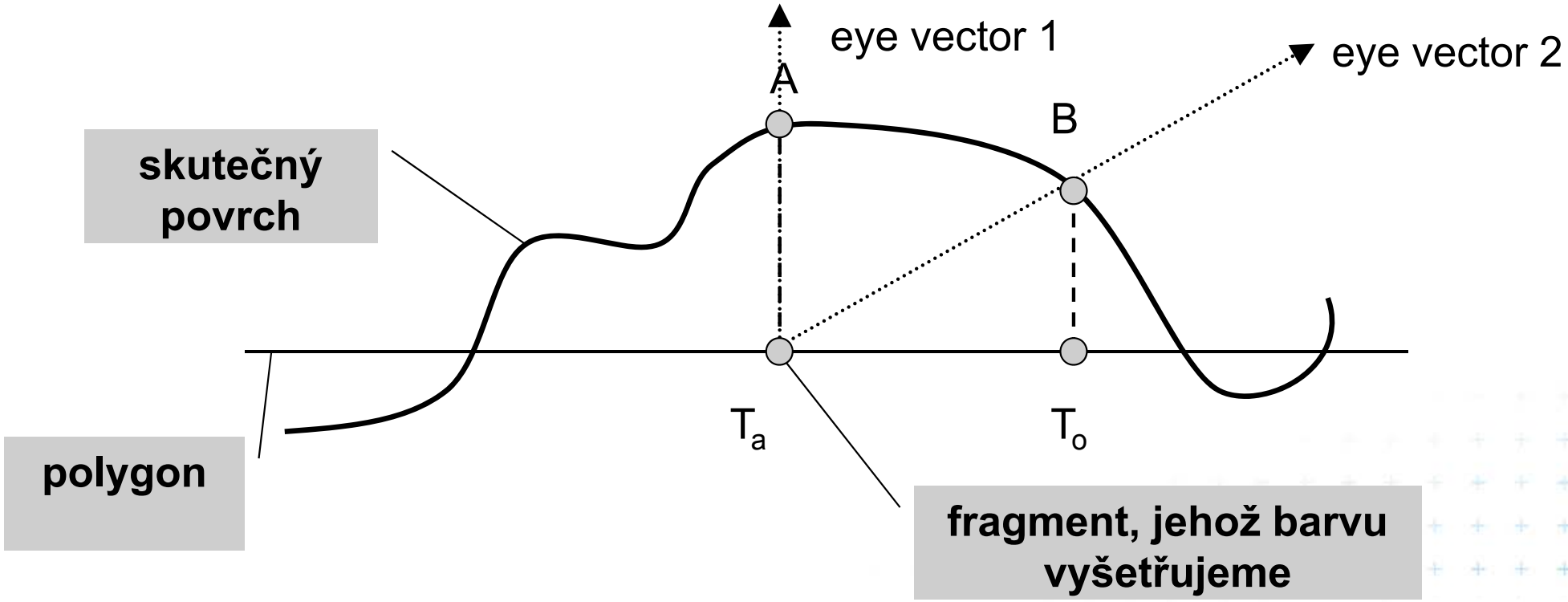
Paralaxa je úhel, který svírají přímky vedené ze dvou různých míst v prostoru k pozorovanému bodu.

Paralaxa je zdánlivý rozdíl polohy bodu vzhledem k pozadí při pozorování ze dvou různých míst.



[cs-wikipedia]

U polygonu s texturou – paralaxa chybí



Povrch aproximován polygonem s texturou

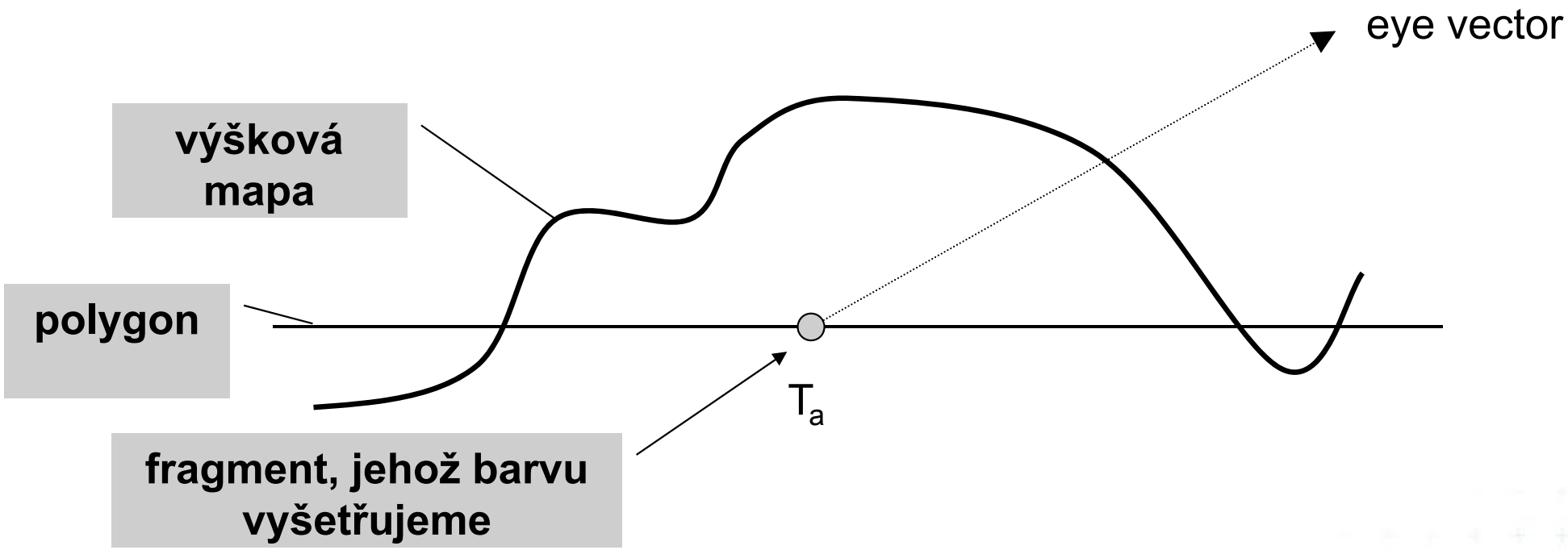
1. Při pohledu kolmo na polygon vidíme správně texel T_a
2. Při pohledu šikmo opět vidíme texel T_a, přestože bychom měli vidět texel T_o, který odpovídá směru pohledu a zakřivení povrchu.

Parallax Mapping

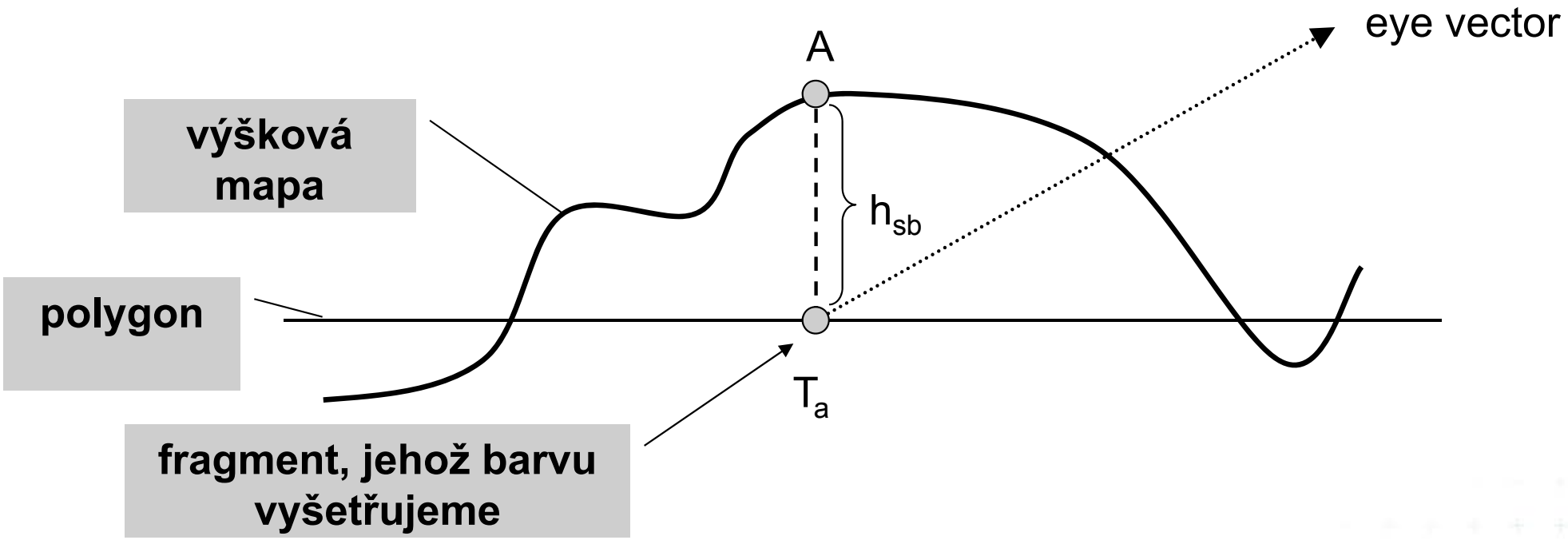
Parallax mapping

- Aproximuje paralaxu posunem textury o offset
- Pro každý pixel je vstupem
  - počáteční poloha v textuře T_a
  - výška povrchu *h*
  - vektor od počátku ke kameře v tečných souřadnicích (eye_{str})
    (transformovaný z prostoru kamery pomocí matice TBN)
- Výstupem je posunutá texturovací souřadnice T_a

Parallax Mapping – výpočet(1)



Parallax Mapping – výpočet(2)



Z výškové mapy přečteme pro původní souřadnici $\mathbf{T_a}$. výšku **h**.

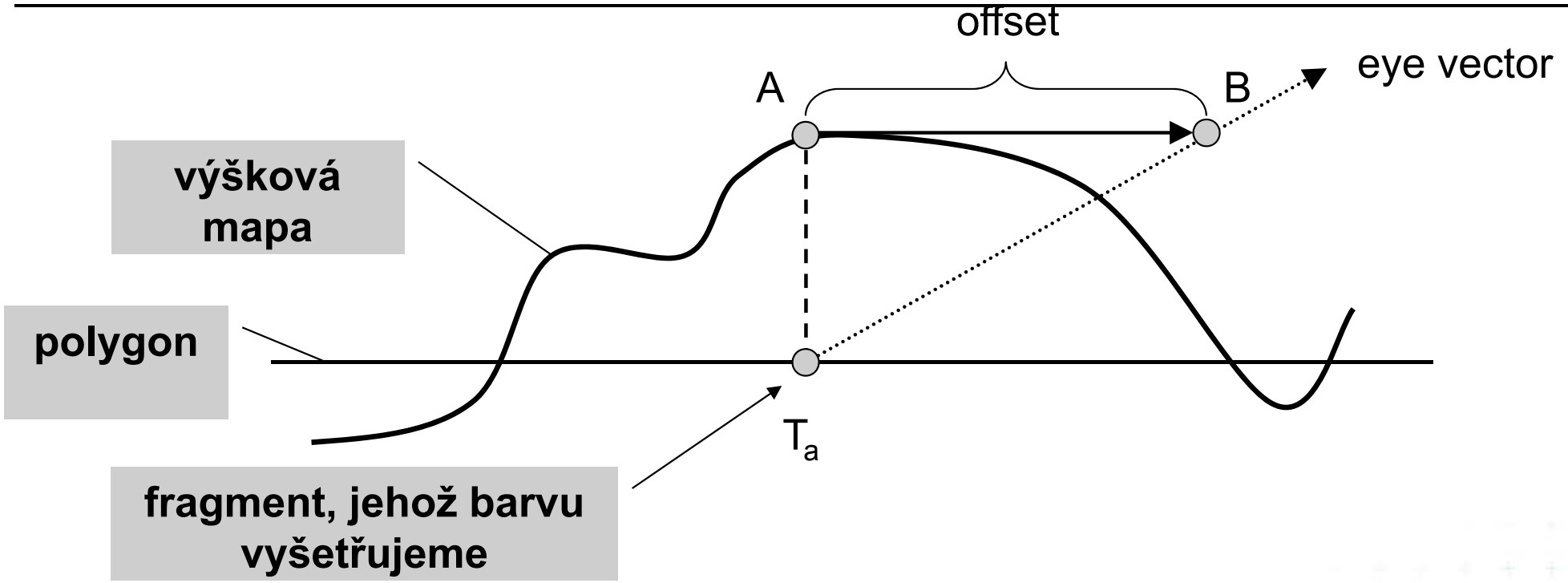
- Výška **h** je v rozsahu ⟨0.0, 1.0⟩
- Aby výška odpovídala poměru velikosti detailů povrchu a rozměru textury, je upravena scale faktorem **s** a bias složkou **b**

$$\mathbf{h_{sb} = h * s + b}$$

Vhodné je zajistit, aby b ~ -0,5s

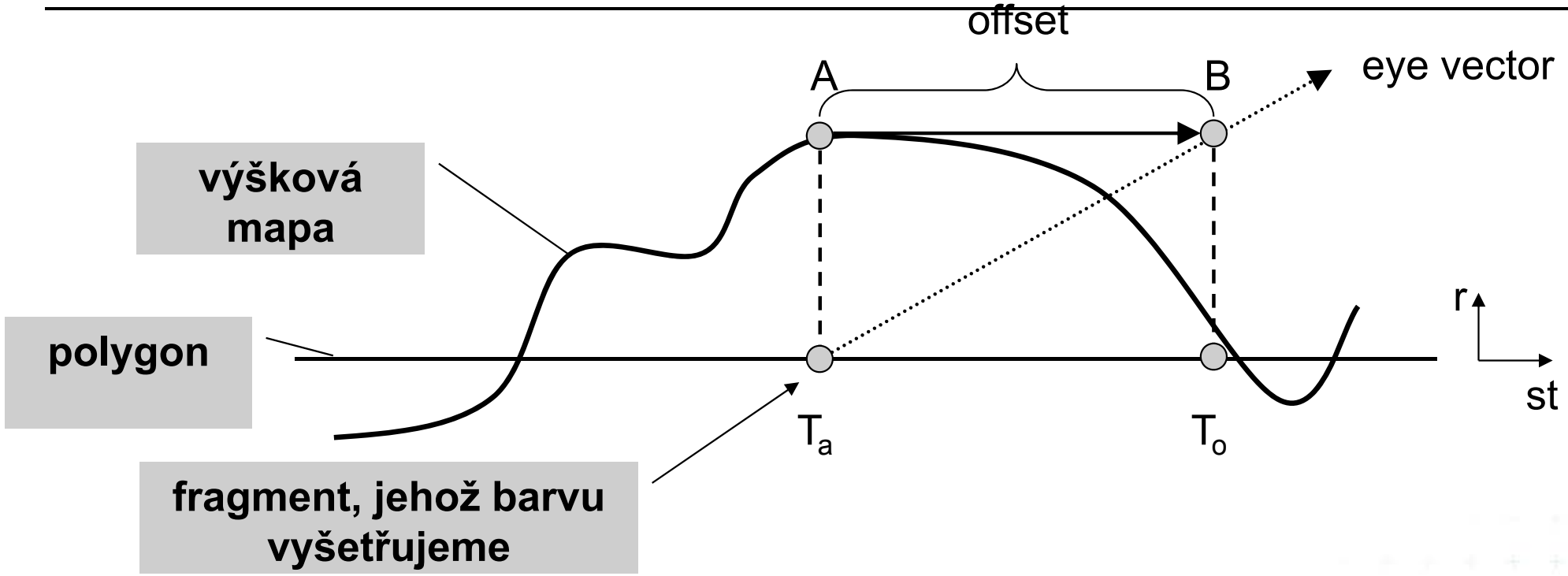
(Welsh: s = 0.04, b = -0.02)

Parallax Mapping – výpočet(3)



Vedeme rovnoběžku s rovinou polygonu a najdeme bod B, kde se tato rovnoběžka kříží s vektorem pohledu.

Parallax Mapping – výpočet(4)



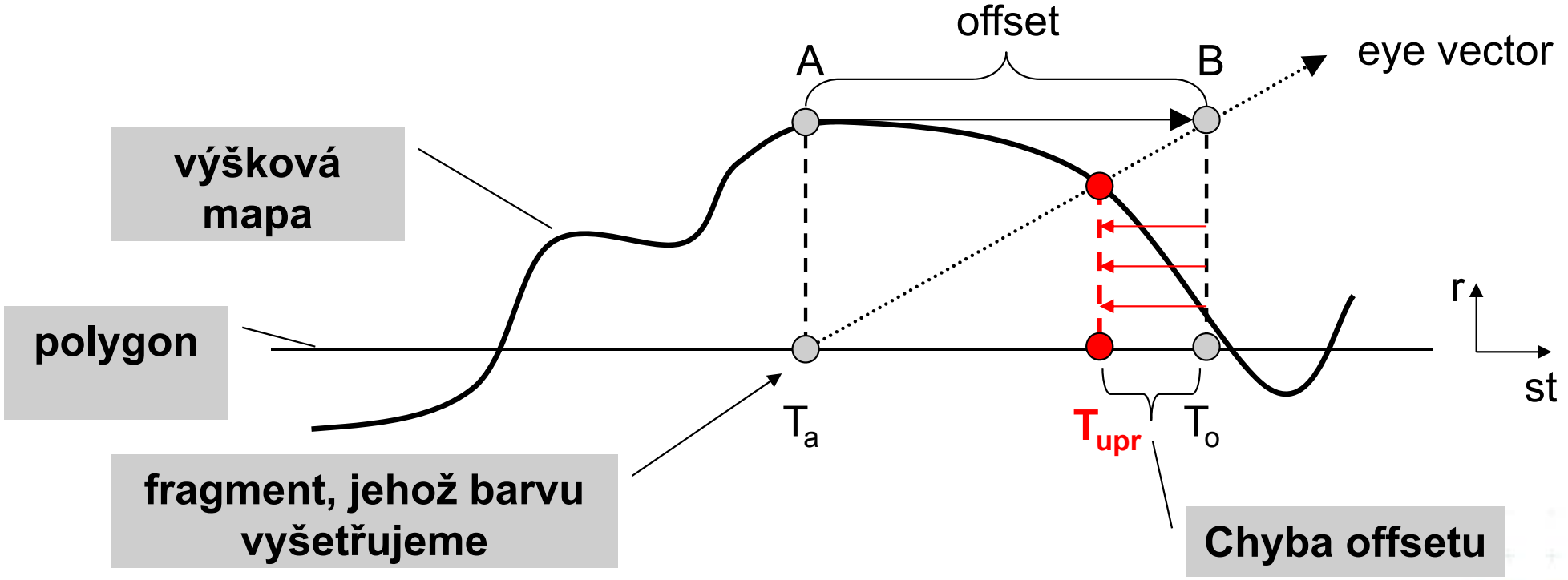
Pro bod B nalezneme odpovídající texel textury T_o povrchu, jehož barvu fragment získá.

$$T_o = T_a + (h_{sb} * eye_{st} / eye_r)$$

eye je normalizovaný vektor – má délku 1

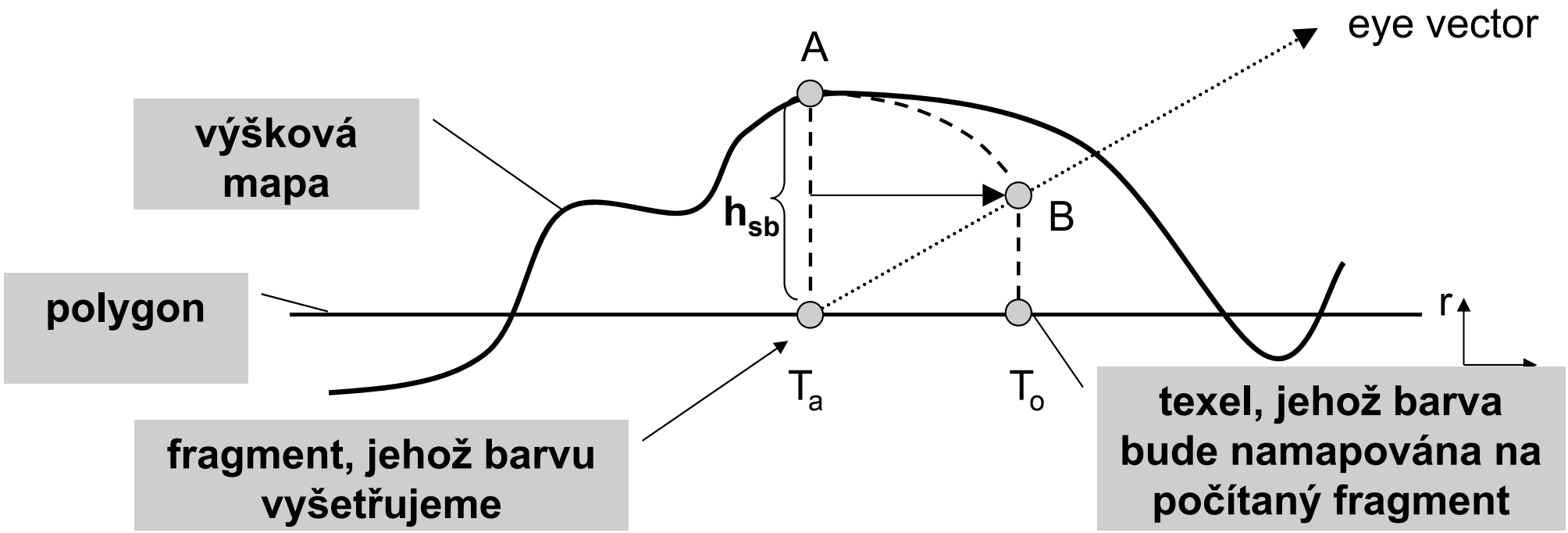
Poměr eye_{st} / eye_r může růst k ∞

Parallax Mapping – problém



Problémem je chyba offsetu – texel T_o má stejnou výšku jako T_a. Ve skutečnosti jsou výšky rozdílné, vektor pohledu protíná reálný povrch v jiném místě a korektním texelem by měl být tedy v tomto případě T_{upr} !! Čím menší úhel bude svírat eye vector s rovinou polygonu, tím větší chyba offsetu nastane (offset → ∞).

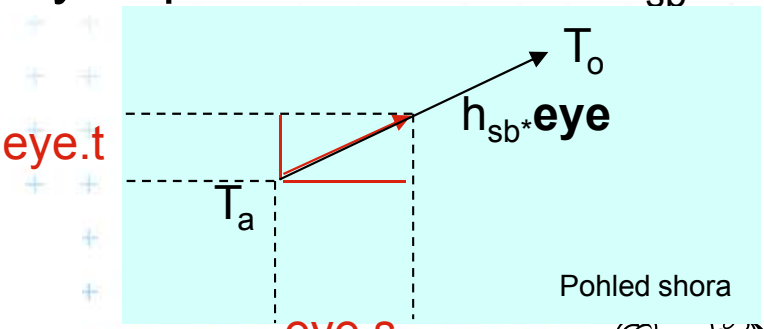
Omezení chyby ofsetu - Offset Limiting



Offset Limiting – jednoduché aproximované řešení (heuristika)

=> Offset omezíme tak, že nikdy neporoste déle než h_{sb}.

$$T_o = T_a + (h_{sb} * eye_{st})$$



eye má délku 1

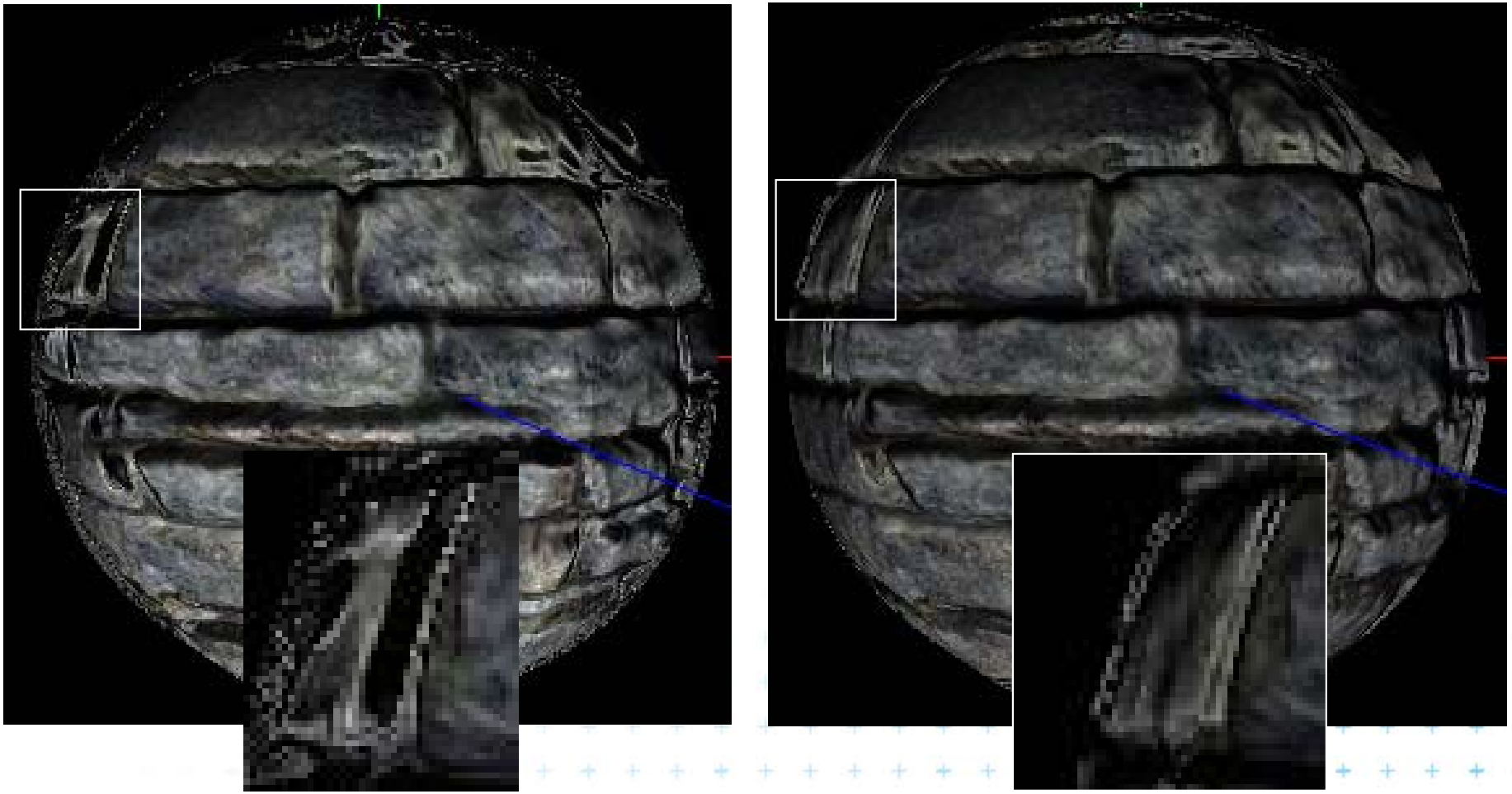
=> $eye_{st} \leq 1$

=> $(h_{sb} * eye_{st}) \leq h_{sb}$

Porovnání verzí Parallax Mapping

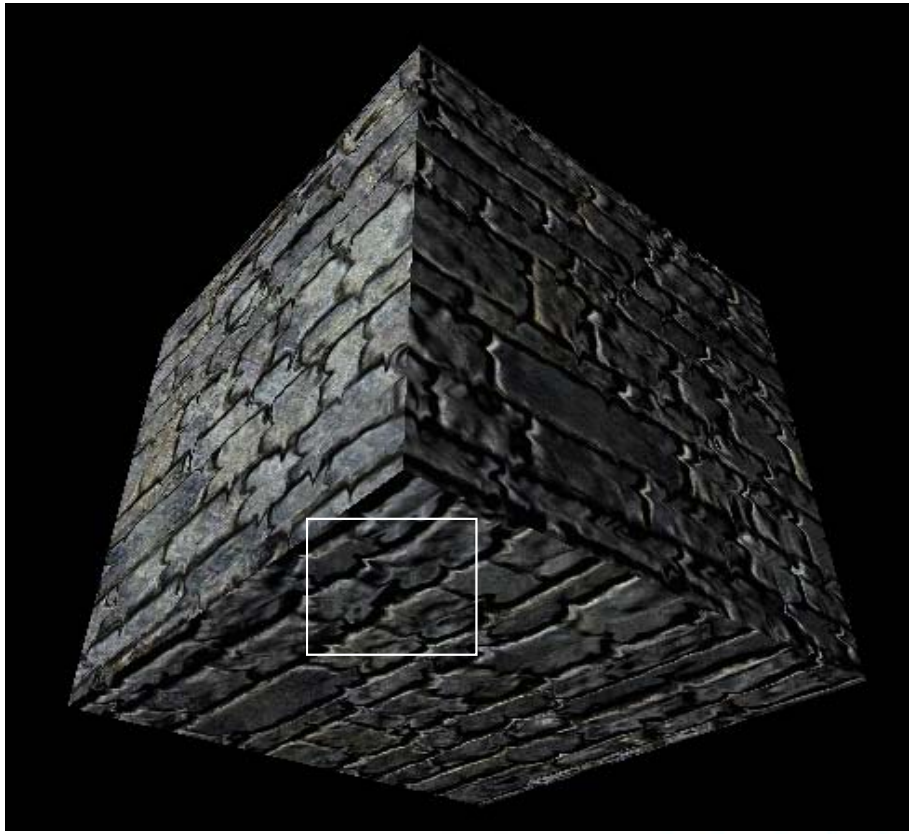
Bez omezení posunutí

S omezením posunutí – Offsett limiting

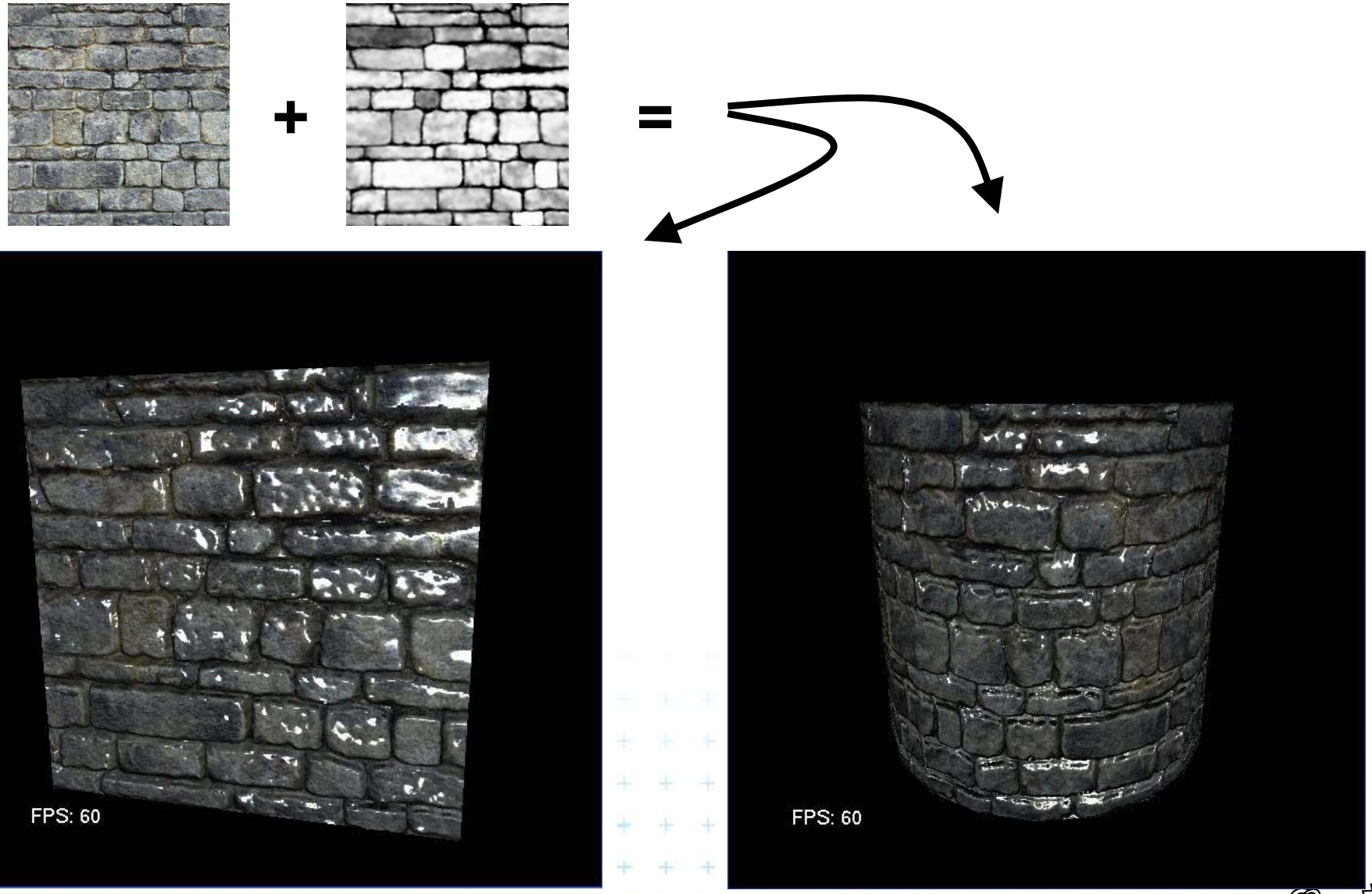


Porovnání verzí Parallax Mapping

Bez omezení posunutí a s nenormalizovanou normálou v matici TBN

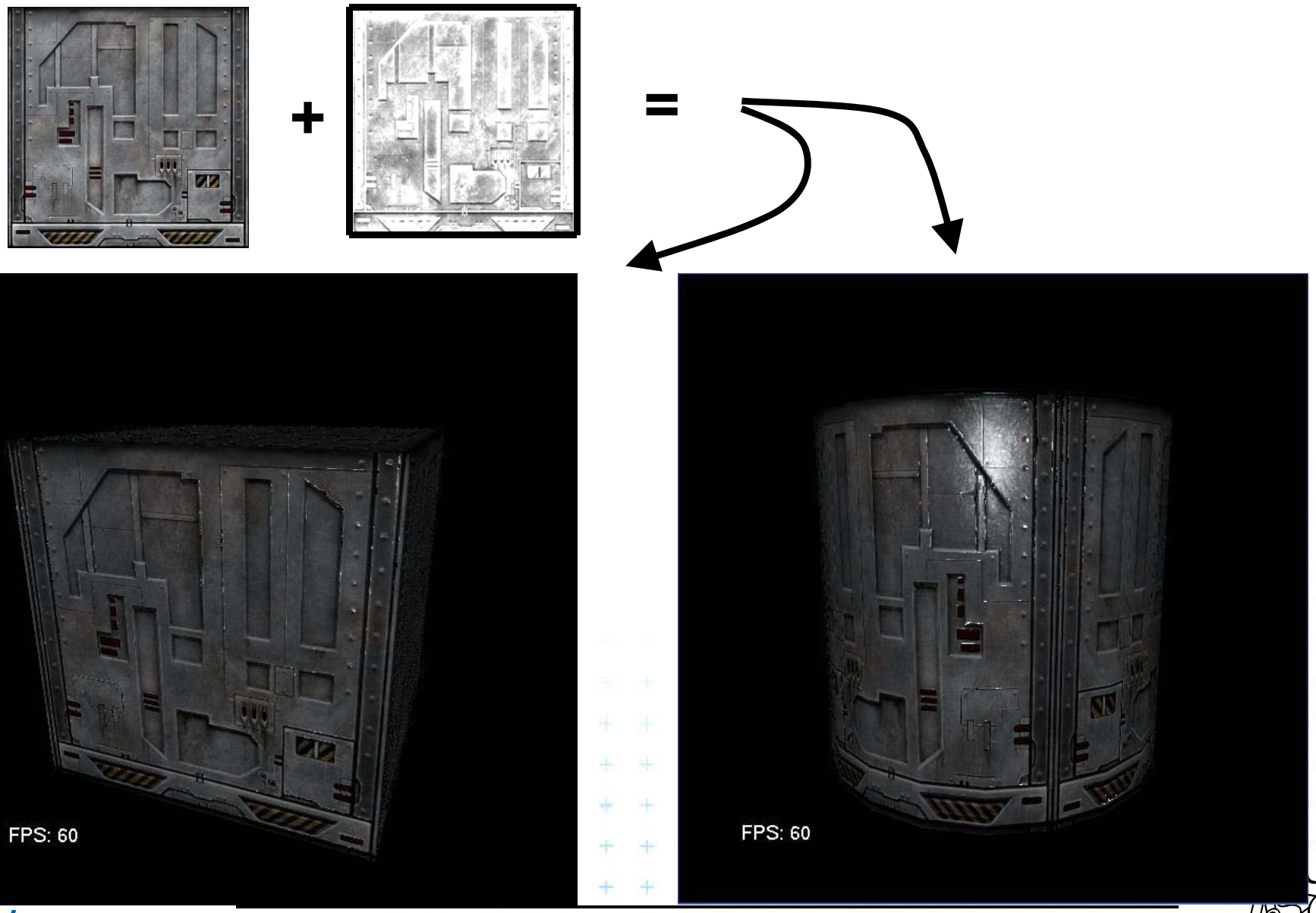


Ukázky



[4]

Ukázky



[4]

Parallax vs. Bump mapping(1)

Parallax mapping

Bump mapping

Parallax vs. Bump mapping(2)

Parallax mapping

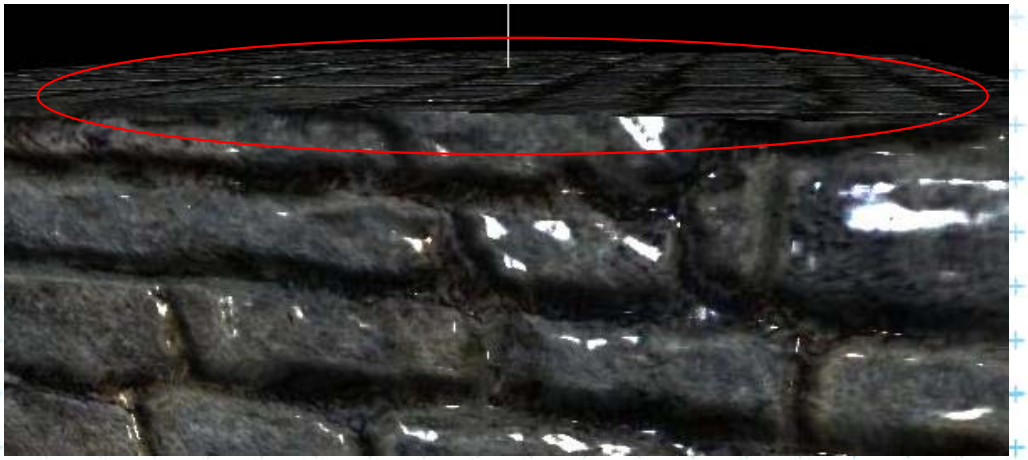
Bump mapping

[4]

Parallax mapping – výhody a nevýhody

- nevýhody základního algoritmu s omezením offsetu
  - nereprezentuje přesně hloubku povrchu (heuristika - omezeni na h_{sb})
  - plovoucí pixely (artefakty) při malých úhlech
  - zploštění objektů pro malé úhly
  - ostré obrysy
  - nedetekuje zakrytí (vlastní stíny)

- výhody
  - HW velmi „levné“
  - vhodné pro malé sklony textur



Nové algoritmy - příklad

Parallax Occlusion Mapping

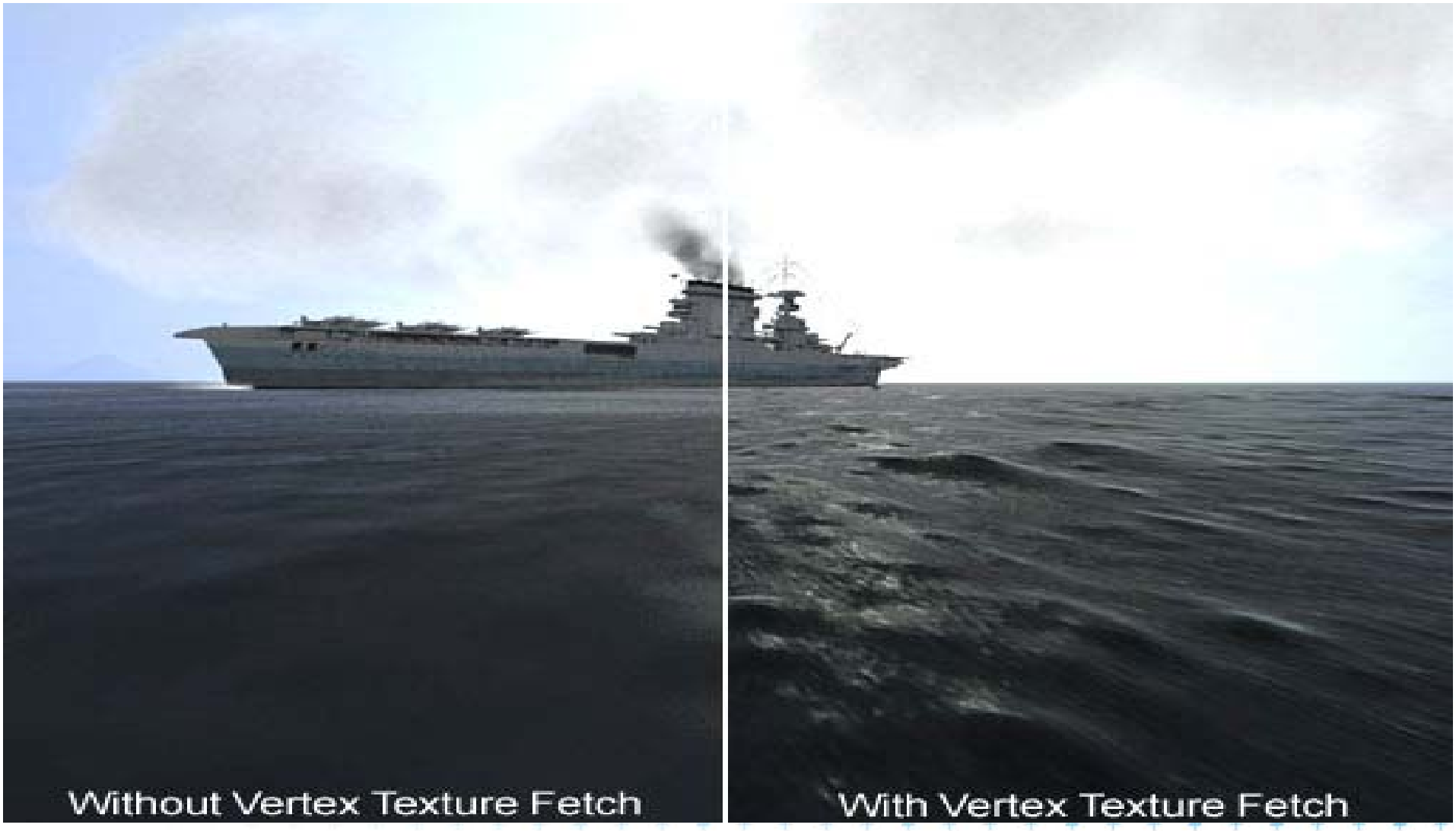
- Normálová mapa obsahuje výšku ve složce alfa
- Alg. nalezne přesně bod na povrchu iterativně

```
NB = texture2D(normalBumpMap, offsetCoord);
while (NB.a < height) {
  height -= step;
  offsetCoord += delta;
  NB = texture2D(normalBumpMap, offsetCoord);
}
```

- Přesněji najde offset odpovídající bodu B na povrchu
- Shadows, self-shadows ,…

DCGI

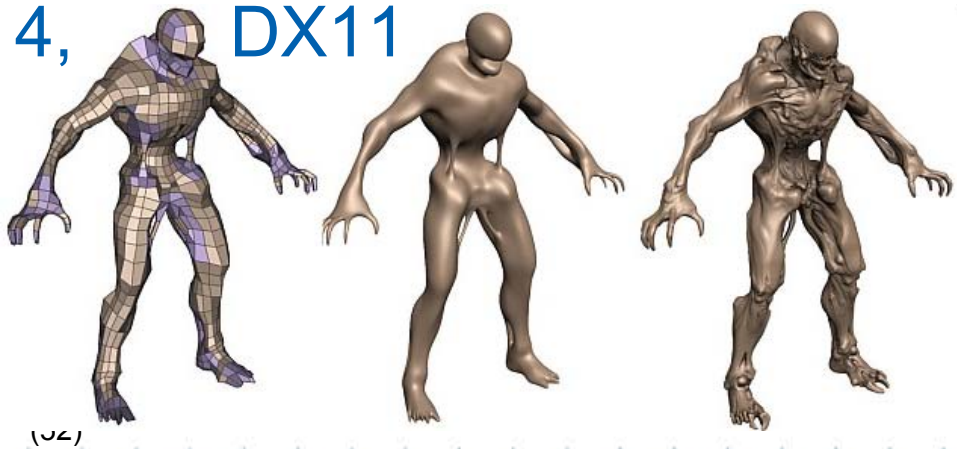
Displacement Mapping



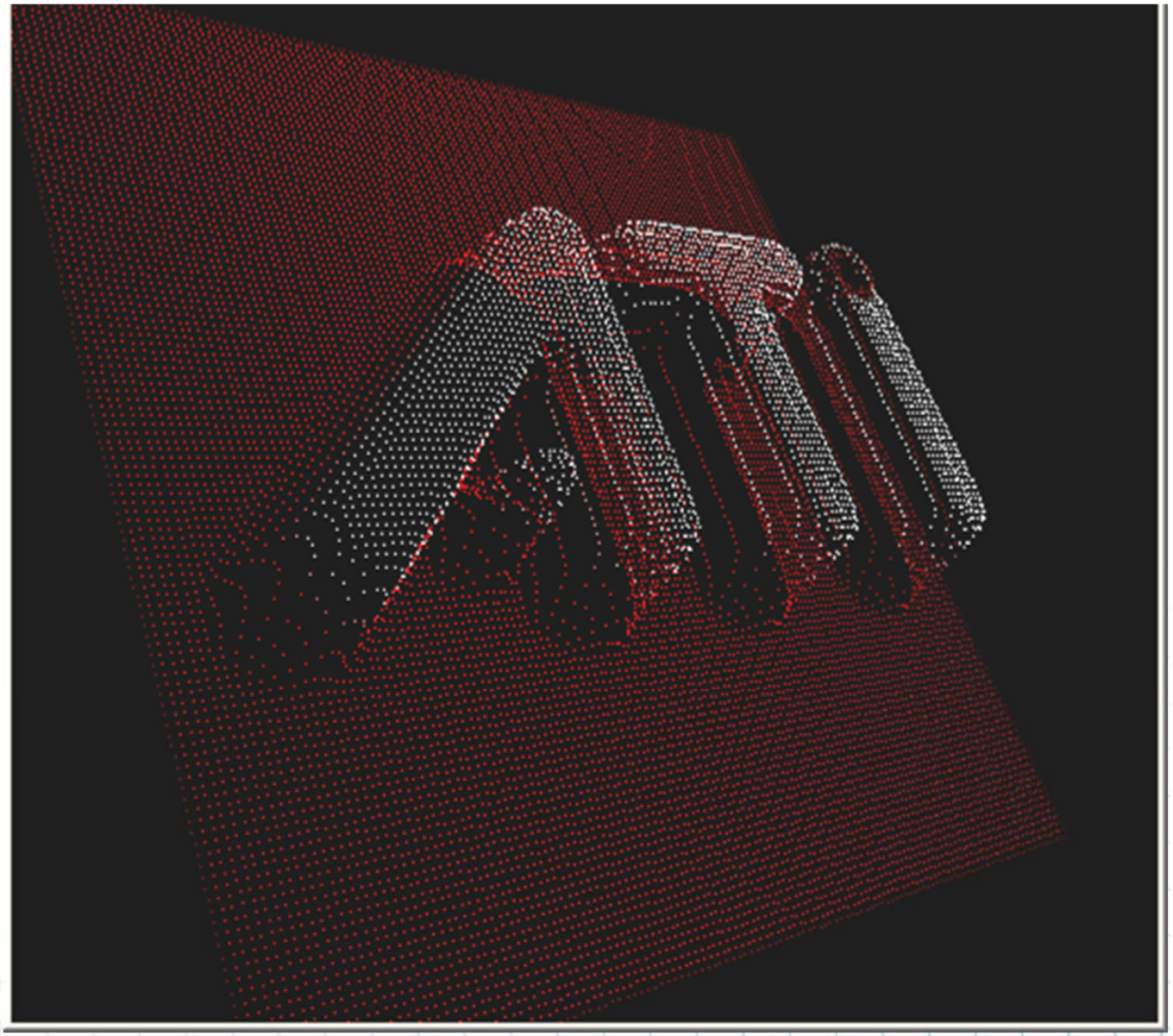
http://developer.nvidia.com/object/using_vertex_textures.html

Displacement Mapping

- Simulace členitých povrchů modifikací geometrie (změnou poloh vrcholů ve vertex shaderu)
- Protipól „fake“ algoritmům (Bump, Parallax Mapping,…)
- Vertex shader má přístup k textuře (Displacement) – posunuje vrcholy podél normály
- Shader model 3.0 – Vertex proc. neumí vytvářet vrcholy
  - Nutná detailní geometrie, kterou modifikuje,
  - plus textura s mapou posunutí (displacement map)
- SM4 – umožňuje vytvářet vrcholy v geometry shaderu
- SM5 – Teselátory - OpenGL 4, DX11
  - Méně geometrie na CPU
  - Nižší nároky na sběrnice do GPU



Př.: Displacement mapping I



Použité textury

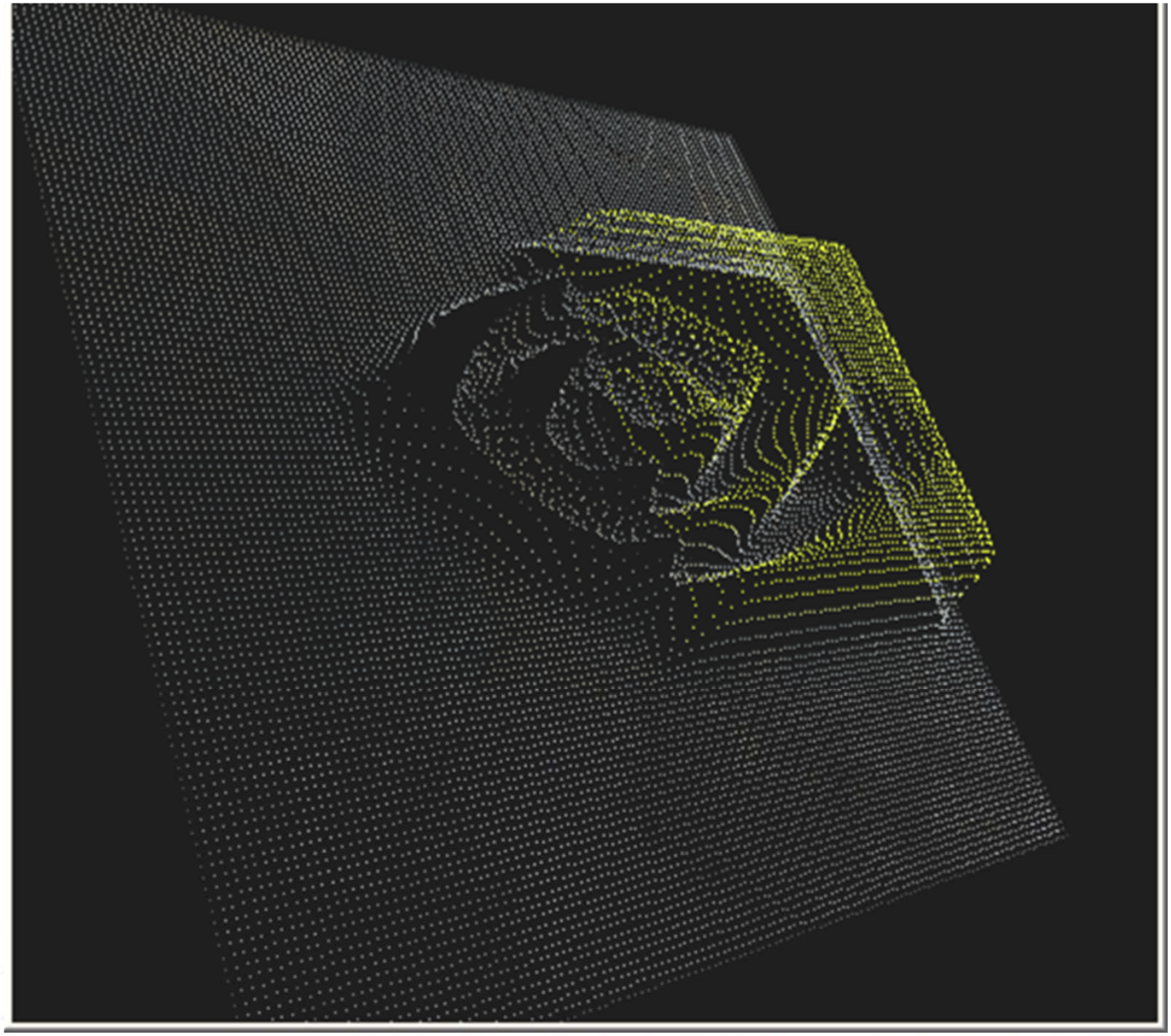
Barevná textura

256x256x24b

Posunutí (*displacement*)

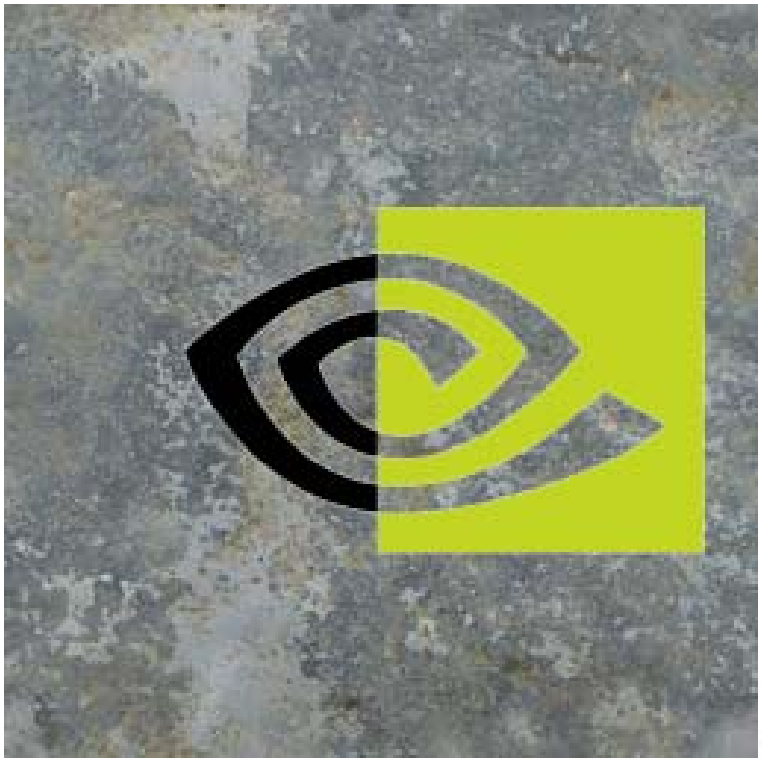
256x256x8b

Př.: Displacement mapping II



Použité textury

Barevná textura



256x256x24b

Posunutí (*displacement*)

256x256x8b

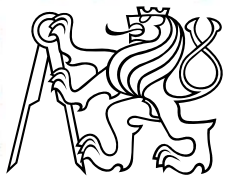
Princip implementace displacement Mapping

- Vygeneruje se hustá síť vrcholů
- Pokud může VS přistupovat do textur (≥SM 3)
  - Posune se pozice vrcholu ve směru normály:

    float offset = texture(displaceTex, texCoord);

    vec4 pos = vertexPos + normal * offset* SCALE;

    position = PVM * pos;
- Pokud VS nemůže přistupovat do textur (ES)
  - na dva průchody (FBO a PBO-> VBO)
    1. průchod: vygeneruje novou texturu o rozměrech sítě vrcholů a překopíruje ji do pixel buffer objektu (PBO)
    2. průchod: přemapuje PBO na VBO a použije jako další atribut vrcholu

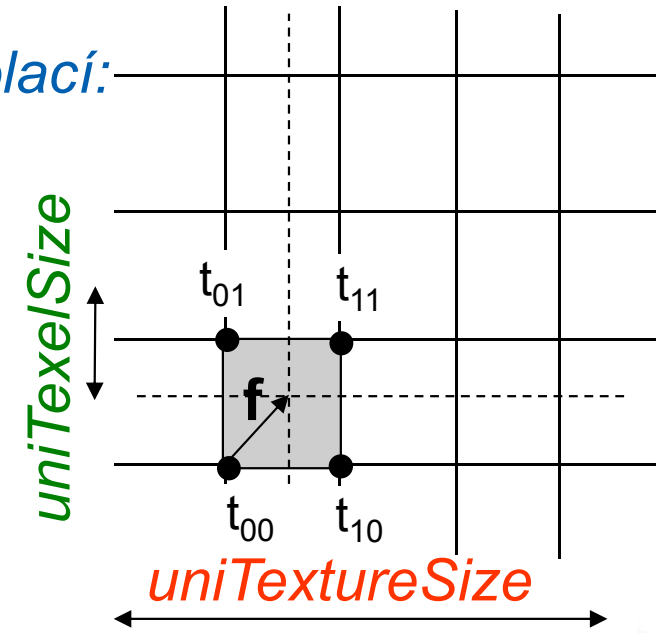


Bilineární filtrace ve vertex shaderu

Pokud VS neumí přístup do Tex s bilineární interpolací:

```
uniform float uniTextureSize; // tess+1
uniform float uniTexelSize;    //1.0

vec4 texture2D_bilinear( sampler2D tex, vec2 uv )
{
      vec2 f = fract( uv * uniTextureSize );
      vec4 t00 = texture2D( tex, uv + vec2( 0.0, 0.0 ));
      vec4 t10 = texture2D( tex, uv + vec2( uniTexelSize, 0.0 ));
      vec4 tA = mix( t00, t10, f.x );
      vec4 t01 = texture2D( tex, uv + vec2( 0.0, uniTexelSize ) );
      vec4 t11 = texture2D( tex, uv + vec2( uniTexelSize, uniTexelSize ) );
      vec4 tB = mix( t01, t11, f.x );
      return mix( tA, tB, f.y );
}
```



GL_NEAREST

VTF Fragment Shader

Barva displace.fs

```
uniform sampler2D texnVidia;


void main()
{
    vec2 base = gl_TexCoord[0].st;
    gl_FragColor =  texture2D(texnVidia, base);
}
```

GL_LINEAR_MIPMAP_NEAREST

Závěr: VTF x R2VB

- **Vertex Texture Fetch**
  - GeForce6+
- **Pros**
  - Flexibilní
  - Jednoduše se implementuje
- **Cons**
  - Pro omezený počet formátů
  - Pomalejší než R2VB
  - Potřeba dvojnásobného počtu texturovacích jednotek než pro R2VB
  - Funguje pouze na některých kartách nVidia

- **Render 2 Vertex Buffer**
  - Radeon 9500+,GF6+
- **Pros**
  - Rychlé
  - Libovolný formát textur
  - Funguje i na některých starších GPU ATI
  - A v OpenGL ES
- **Cons**
  - Použitelné jen pro čtvercové povrchy nebo s dobře organizovanými mapami posunutí (displacement maps)
  - Potřebuje dva vykreslovací průchody (two rendering passes)

DCGI

Konec

Odkazy pro parallax a displacement mapping

[1] Terry Welsh.: *Parallax Mapping with Offset Limiting: A Per-Pixel Approximation of Uneven Surfaces*, 2004
http://www.cs.ualberta.ca/~keith/610/papers/parallax_mapping.pdf

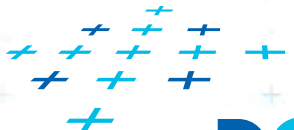
[2] TyphoonLabs.: *OpenGL Shading Language Course*, www.typhoonlabs.com

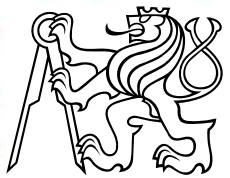
[3] Addison-Wesley.: *GPU Gems 2*, 2005

[4] J. Drahokoupil, M. Dušek: *Parallax Mapping with Offset Limiting*, GSY 2007

[5] Natalya Tatarchuk.: *Those Delicious Texels Dynamic Image-Space Per-Pixel Displacement Mapping with Silhouette Antialiasing via* ***Parallax Occlusion Mapping***, http://ati.amd.com/developer/gdc/Tatarchuk-ParallaxOcclusionMapping-FINAL_Print.pdf, GDC 2005

[6] M. Hapala: *Displacement mapping*, GSY, 2007





Další odkazy

[MPG2004]
J.Žára, B. Beneš, J. Sochor, P. Felkel, *Moderní počítačová grafika* (2. vydání), Computer Press, 2005, ISBN 80-251-0454-0

[AM2002]
T. Akenine Moeller, E. Haines: Realtime Rendering (2nd ed), A. K. Peters, 2002, ISBN 1-56881-182-9

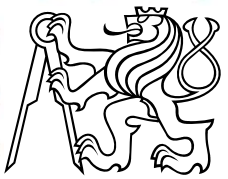
[AGPU] T. McReynolds, D. Blythe: Advanced Graphics Programming Using OpenGL. Morgan Kaufmann, 2005

[Lengyel] E. Lengyel. Matjematics for 3D Game Programming & Computer Graphics, 2nd ed., Charles River Media, 2004

[Terathon] Lengyel, Eric. "Computing Tangent Space Basis Vectors for an Arbitrary Mesh". Terathon Software 3D Graphics Library, 2001. http://www.terathon.com/code/tangent.html

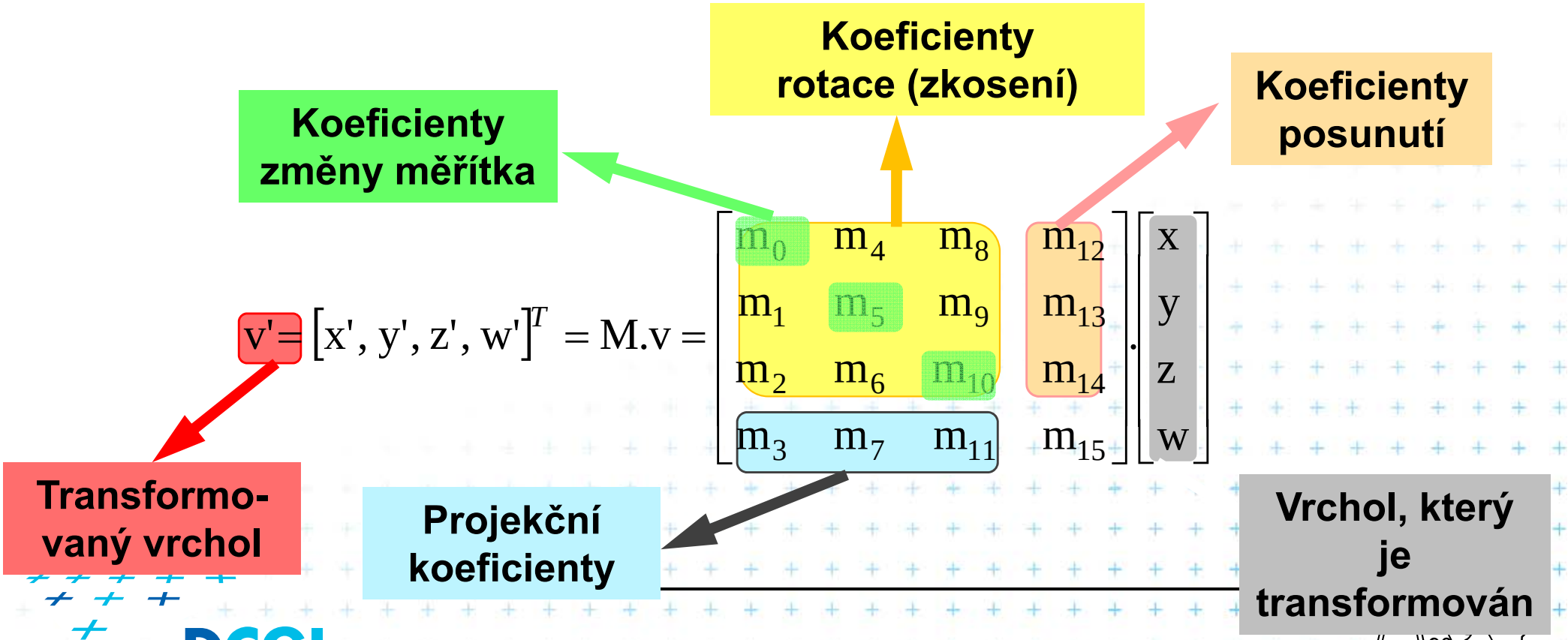
[Everitt] Cass Everitt. Mathematics of Per-Pixel Lighting, nVidia. http://developer.nvidia.com/object/mathematicsofperpixellighting.html

[teselace] Lukáš Fiala. Hardwarová teselace je cestou k lepší 3D grafice http://www.extrahardware.cz/hardwarova-gpu-teselace-truform-directx-11-displacement-mapping-tesselation



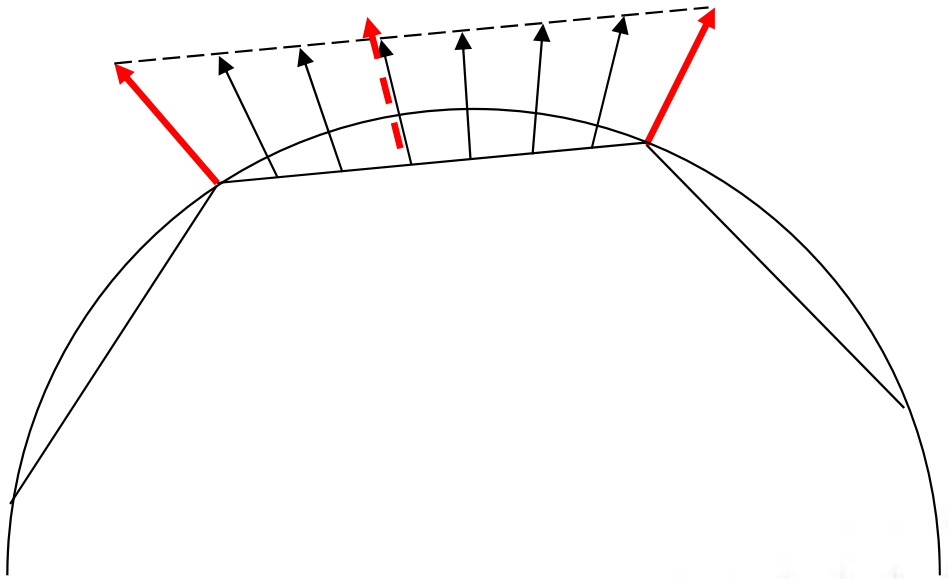
Součásti transformační matice

- **vrcholy** jsou reprezentovány **sloupcovými vektory v = [x, y, z, w]^T**
- **transformace** jsou zadány **4x4** maticí **M**
- transformace se provede vynásobením matice **M** a vektorem **v**.
- na vektory nemá vliv posunutí (mají w=0)



Zkrácení normál při interpolaci

- Při interpolaci mezi vrcholy při rasterizaci
- Interpolované vektory jsou kratší



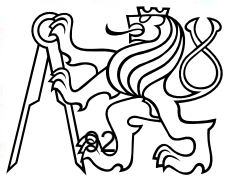
=> **normály nutno před použitím normalizovat**

V shaderu: N = **normalize** (interpolatedNormal);

N = **normalize(**Matrix * interpolatedNormal**);**

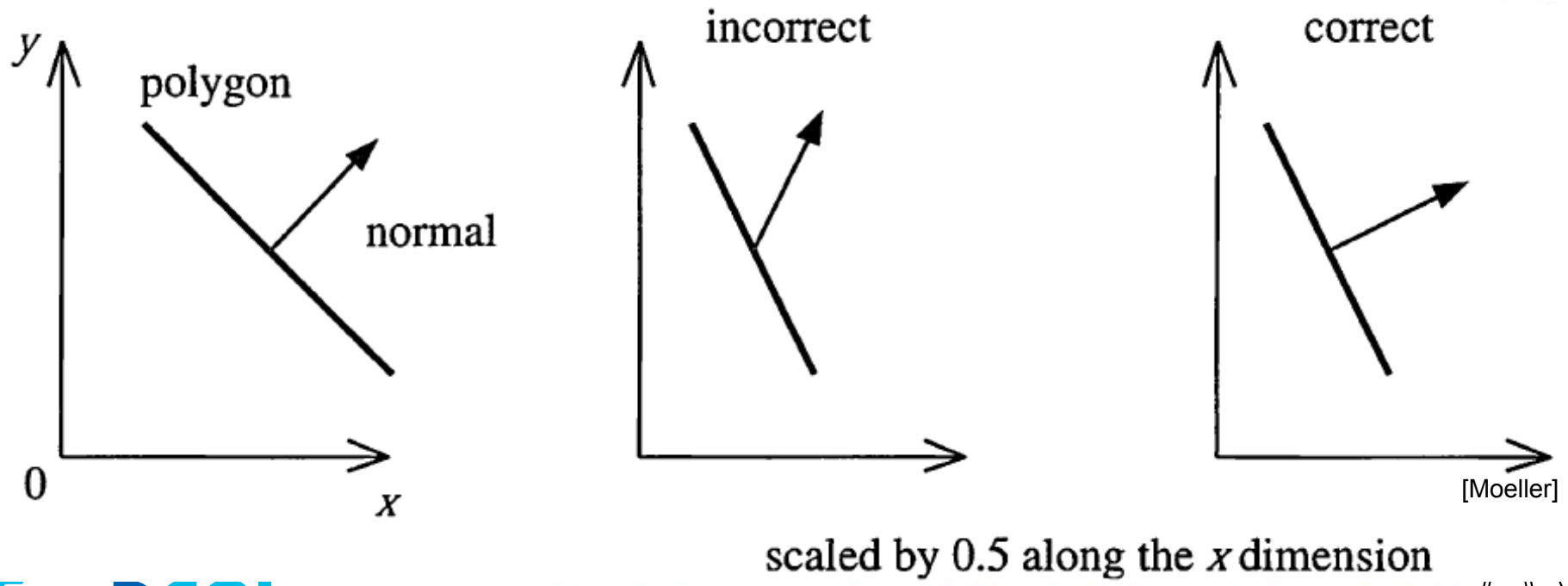
Změna délky normály při změně měřítka

- Při změně měřítka ve všech směrech stejné *k*-krát se *k*-krát prodlouží i normála
  - Stačí vydělit složky normály hodnotou *k*
  - Není třeba počítat celý vzorec pro normalizaci



Změna sklonu normály

- Nesymetrická změna měřítka, či jiná nerigidní afinní modelovací transformace M
- Transformovaná normála přestane být kolmá k povrchu (vektor **Mn**, o kterém si myslíme, že je to normála, už normálou není) => pro normály použít $\mathbf{N} = (\mathbf{M}^{-1})^T$



[Moeller]

Odvození transformace pro normály 1

- Vrchol **x** je transformován maticí **M** násobením touto maticí:

  x' = M x matici **M** známe – je to modelovací matice
- Stejně je transformována tečna t (rozdíl dvou povrchových vrcholů)

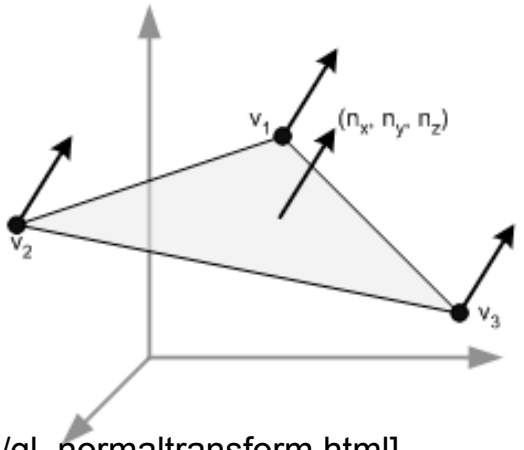
  t' = M t
- Normála **n** je transformována maticí **N** tak:

  n' = N n matici **N** hledáme
- Předpoklad: normála **n** je kolmá na tečnu **t** před transformací i po transformaci geometrického objektu maticí **M**:

  před **n . t** = 0- zapsáno jako skalární součin

  $n^T t$ = 0 - dtto jako vektor n krát t

  po **n'. t'** = 0



Felkel P./ Programování grafiky 2 [http://www.sgi.com/software/opengl/gl_normaltransform.html]

Odvození transformace pro normály 2

Hledáme neznámou matici **N**

- Vyjdeme z rovnice: $\mathbf{n}^T\mathbf{t} = 0$ Trik: $\mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$

  $\Rightarrow \mathbf{n}^T\mathbf{M}^{-1}\mathbf{M}\,\mathbf{t} = 0$

  $\Rightarrow (\mathbf{n}^T\mathbf{M}^{-1})(\mathbf{M}\,\mathbf{t}) = 0$

$$\begin{pmatrix} n_x & n_y & n_z & n_w \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = 0$$

$$\underbrace{\begin{pmatrix} n_x & n_y & n_z & n_w \end{pmatrix} M^{-1}}_{\text{normal}^T} \underbrace{M \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}}_{\text{vertex}} = 0$$

- Transformace vektoru **t** maticí **M** je tangenta

  $\mathbf{t}' = \mathbf{M}\,\mathbf{t}$

  $\Rightarrow (\mathbf{n}^T\,\mathbf{M}^{-1})\,\mathbf{t}' = 0$

- Transformovaná normála a tangenta jsou na sebe kolmé.

  $\mathbf{n}'^T\,\mathbf{t}' = 0$

  $\Rightarrow \quad \mathbf{n}'^T = (\mathbf{n}^T\mathbf{M}^{-1})$

$$\begin{pmatrix} nx_{eye} \\ ny_{eye} \\ nz_{eye} \\ nw_{eye} \end{pmatrix}^T = \begin{pmatrix} nx_{obj} & ny_{obj} & nz_{obj} & nw_{obj} \end{pmatrix} M^{-1}$$

$\Rightarrow$ Algebraickou úpravou $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$ dostaneme

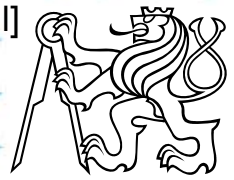
$\mathbf{n}' = (\mathbf{n}^T\,\mathbf{M}^{-1})^T = (\mathbf{M}^{-1})^T\,\mathbf{n} = \mathbf{N}\mathbf{n}$

$\Rightarrow$ transformační matice **N** pro normálu je rovna

$\mathbf{N} = (\mathbf{M}^{-1})^T$

$$\begin{pmatrix} nx_{eye} \\ ny_{eye} \\ nz_{eye} \\ nw_{eye} \end{pmatrix} = (M^{-1})^T \begin{pmatrix} nx_{obj} \\ ny_{obj} \\ nz_{obj} \\ nw_{obj} \end{pmatrix}$$

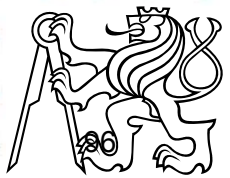
[http://www.songho.ca/opengl/gl_normaltransform.html]



Výpočet inverzní matice

- Pro sekvenci jednoduchých transformací
  - Invertujeme dosazením invertovaných tr. a opačným pořadím
    např. pro $\boldsymbol{M} = \boldsymbol{T}(\boldsymbol{t})\boldsymbol{R}(\propto)$ je $\boldsymbol{M}^{-1} = \boldsymbol{R}(-\propto)\boldsymbol{T}(-\boldsymbol{t})$
- Pokud je $\boldsymbol{M}$ ortogonální – např. sada rotací
  - je $\boldsymbol{M}^{-1} = \boldsymbol{M}^{T}$
- Je-li transformace neznámá, vypočteme inverzní matici
  - Pro vektory stačí inverzní matice 3x3
  - Proto není nutno dělit determinantem původní matice
  - Stejně je nutno vypočtenou normálu následně normalizovat
  - Místo inverzní matice proto stačí
    adjungovaná (reciproká) matice
    = transponovaná matice algebraických doplňků
    (transposed cofactor matrix of M = adjoint of M)
  - Navíc ji transponujeme ještě jednou => cofactor matrix

DCGI



Inverzní matice pomocí algebraických doplňků

$$\mathbf{M}_1 = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$$

$$N = (M^{-1})^t$$

$$M^{-1} = \frac{1}{\det \mathrm{M}} \text{ (adjoint of M)}$$

$$N = (M^{-1})^t = \frac{1}{\det \mathrm{M}} \text{ (adjoint of M)}^{\mathrm{t}}$$

N = normalize (cofactor matrix of M)

Matice algebraických doplňků

$$\mathbf{N} = \begin{bmatrix} m_{22}m_{11} - m_{12}m_{21} & m_{12}m_{20} - m_{10}m_{22} & m_{10}m_{21} - m_{20}m_{11} \\ m_{02}m_{21} - m_{22}m_{01} & m_{22}m_{00} - m_{02}m_{20} & m_{20}m_{01} - m_{00}m_{21} \\ m_{12}m_{01} - m_{02}m_{11} & m_{10}m_{02} - m_{12}m_{00} & m_{00}m_{11} - m_{10}m_{01} \end{bmatrix}$$