Hardware pro počítačovou grafiku NPGR019

Textury v real-time grafice

Josef Pelikán
Jan Horáček

http://cgg.mff.cuni.cz/
MFF UK Praha

2012

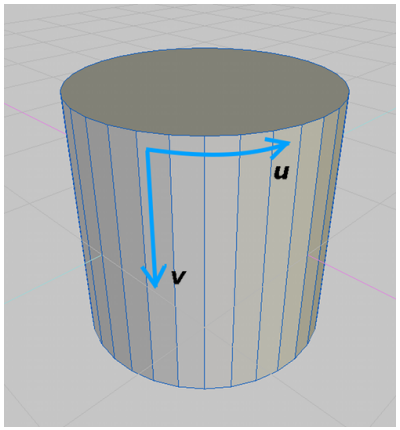
Obsah

Textury

- vylepšují **vzhled** povrchu těles
  - modifikace **barvy** (*bitmapa*)
  - dojem **hrbolatého povrchu** (*bump-texture*)
  - modulace **dalších parametrů**: průhlednost, odrazivost, lesk, ambientní osvětlení, ...
- definice textury:
  - **1D**, **2D** datovým polem (rastr, *bitmapa*)
  - **3D** datovým polem (*volume texture*)
  - **procedurálně** - algoritmus v každém bodu prostoru spočte hodnotu

Mapování textur

- textury se musí na povrch tělesa **mapovat**
  - **texturové souřadnice** $[u, v]$ (v OpenGL $[s, t]$) se zadávají v každém vrcholu
  - grafický akcelerátor je interpoluje do jednotlivých pixelů
  - v bitmapě se musí interpolovat mezi sousedními **texely** (= pixel textury)



Automatické *projektory*

- starší verze OpenGL měly módy pro tzv. **automatické generování** texturových souřadnic (glTexGen)
- dostupné projektory:
  - sférická projekce (singularita na pólu)
  - cylindrická projekce
  - lineární projekce (používá se často i pro 1D textury)
- dnes plně nahrazeno **vertex shadery**

Opakování textur

- **standardní** rozsah texturovacích souřadnic je $[0, 1]^D$
  - není předem jasné, jak se zobrazí údaje mimo tento interval
  - je třeba to explicitně specifikovat
- **cyklické** opakování (repeat, wrap, tile)
- **zrcadlové** opakování (mirror, flip)
  - každá druhá dlaždice se zrcadlově převrátí
  - spojitá funkce textury
- **nejbližší texel** (clamp, clamp to edge)
  - odolnost k numerickým chybám na okraji textury
- **explicitní okraj** (border, clamp to border)
  - zadán explicitně jedním řádkem/sloupcem textury

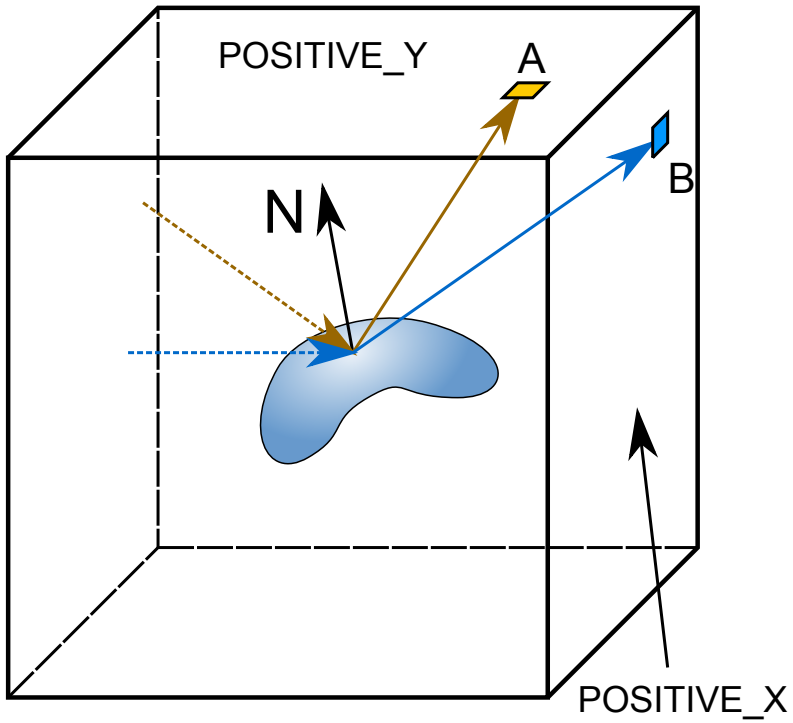
Transformace texturových souřadnic

- uživatelské transformační matice pro texturové souřadnice (mód GL_TEXTURE)
  - maticové transformace prováděné na GPU
  - obecná **homogenní matice** 4×4
  - vektor texturových souřadnic: $[s, t, r, q]$
  - **q** hraje roli homogenní složky (projektivní transformace)
  - **animace textur** na povrchu těles
- OpenGL: další zásobník transformačních matic
  - `GL_TEXTURE`
  - dá se nastavit individuálně pro každou texturovací jednotku = každá TU má svůj zásobník
- opět v novějších verzích plně nahrazeno **vertex shaderem**

Cube mapping

- **Greene**, 1986
- jednodušší než dosud používané sférické mapování
- textura je uložena v **6 čtvercových bitmapách**
  - `POSITIVE_X`, `NEGATIVE_X`, `POSITIVE_Y`, ...
  - snadné pořízení dat - možnost dynamické syntézy přímo na GPU
- snadné adresování bitmap, není potřeba *normalizace*, jenom *dělení*
  1. výběr složky s **maximální** absolutní hodnotou $\Rightarrow$ konkrétní čtverec
  2. výpočet souřadnic ve čtverci (**dělení** zbývajících složek tou maximální)

Cube mapping 2



$$A : s = \begin{pmatrix} \frac{x}{y} \\ t = \frac{z}{y} \end{pmatrix}$$

$$B : s = \begin{pmatrix} \frac{y}{x} \\ t = \frac{z}{x} \end{pmatrix}$$

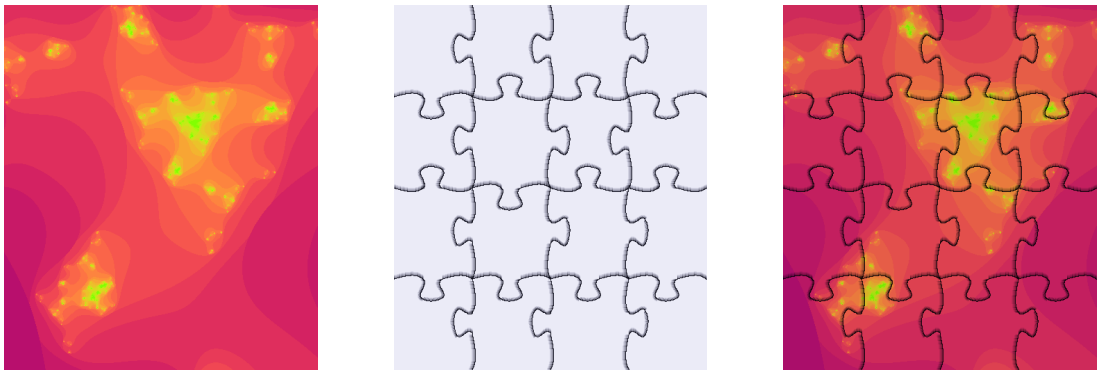
Další typy směrového mapování

- **sphere mapping**(Miller, Hoffman: 1984)
  - přímá implementace odrazu světla na **perfektní kouli** (*light probe*)
  - data jsou v *jediné textuře*, ale část je nevyužita
  - může se objevit tzv. *black-hole* artefakt

©1999 Paul Debevec

- **paraboloid mapping** (Heidrich, Seidel: 1998)
  - **dva snímky** z přesně opačných směrů
  - bez zkreslení na okrajích, dobré využití paměti
  - rovnoměrnější vzorkování prostorového úhlu

Kombinace textur



- moderní GPU umějí v jednom pixelu **kombinovat** několik textur
- tzv. **multitexturing**
  - globální (pomalu se měnící) základ + detailní textura
  - předem spočítané **osvětlení** (*lightmap*)
  - odlesk okolí (*environment map*)
  - ...

Kombinace textur

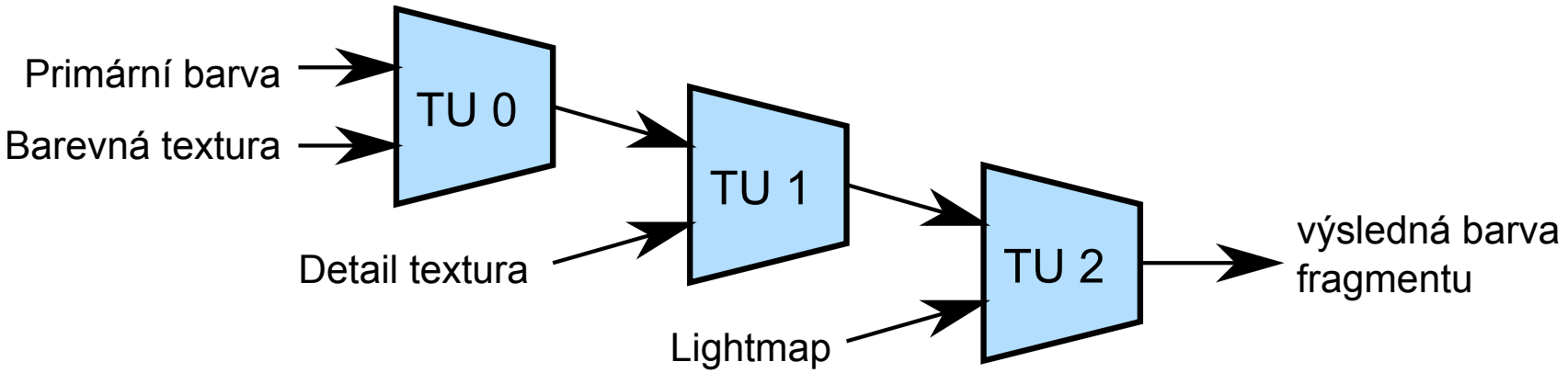
- běžně implementované **kombinační funkce**
  - `REPLACE` - ignoruje se původní barva
  - `MODULATE` - násobení (jen snižuje původní barvu)
  - `DECAL` - poloprůhledná textura na barevný povrch
  - `DOT_3_RGB[A]` - skalární součin složek pro 3 kanály
  - `ADD, ADD_SIGNED, SUBTRACT, BLEND, ...`
- programovatelné GPU mají potenciálně neomezené možnosti

Texturovací jednotky

- jedna **texturovací jednotka** (TU):
  - vybírá hodnotu z **jedné** textury (i s filtrováním)
  - realizuje zadanou **kombinační funkci**
  - výsledek se zapisuje do výstupního fragmentu
- **vstupy** (argumenty) kombinační funkce:
  - *předchozí barva* (výstup předcházející TU)
  - *data z textury* (přečtený/filtrovaný texel
  - *konstantní barva* (`GL_TEXTURE_ENV_COLOR`)
  - primární barva (před texturováním - interpolovaná barva z rasterizační jednotky)
  - ...

Texturovací kaskáda

- příklad lineárního zapojení texturovacích jednotek:



- s příchodem **pixel shaderů** téměř libovolné možnosti kombinací včetně *závislého* čtení textur
  - **hodnota** získaná z jedné textury se použije při výpočtu **souřadnic** pro čtení z jiné textury

Komprese textur - S3TC

- první použití: S3 v DirectX 6 (1998)
  - DXTC, v OpenGL: S3TC, DXT1
- **pevný kompresní poměr**
  - výhodné pro plánování spotřeby paměti
  - 4:1 až 6:1 **ztrátová** komprese
- rozklad textury na **bloky** (*tiles*) velké 4×4
  - každý blok je reprezentován **dvěma 16-bitovými barvami** a **16 2-bitovými indexy** (celkem tedy 4bpp)
  - 2 extrémní barvy (R5G6B5), mezi nimi další 2 mezibarvy (nebo 1 mezibarva a černá)
  - každý originální pixel je nahrazen *odkazem* na jednu barvu
- DXT2, DXT3: ukládají navíc 4bpp alpha kanál, barva podobná jako DXT1 (bez spec. případu černé)
- DXT4, DXT5: alpha uložena do 64bitů jako $2\times$ 8-bit hodnota a 4×4 3-bit LUT, barva jako u ostatních

Komprese textur - další metody

- nVidia **VTC** (Volume Texture Compression)
  - 3D varianta S3TC
  - bloky dat $4 \times 4 \times 1$, $4 \times 4 \times 2$ nebo $4 \times 4 \times 4$
- 3Dfx komprese **FXT1** (1999)
  - 8×4 texelů se uloží do **128 bitů** paměti (tzn. 4bpp)
  - 4 různé formáty bloků (rozhoduje o nich kodér)
    - CC_HI: 2× R5G5B5, 32× 3-bit kód (5 mezi a 1 průhledná)
    - CC_CHROMA: 4× R5G5B5, 32× 2-bit kód (pův. barvy)
    - CC_MIXED: 4× R5G6B5, 32× 2-bit kód (2 pro každý 4×4 čtverec, komplikované sub-formáty)
    - CC_ALPHA: 3× R5G5B5A5, 32× 2-bit kód (pro komplexní průhlednost)

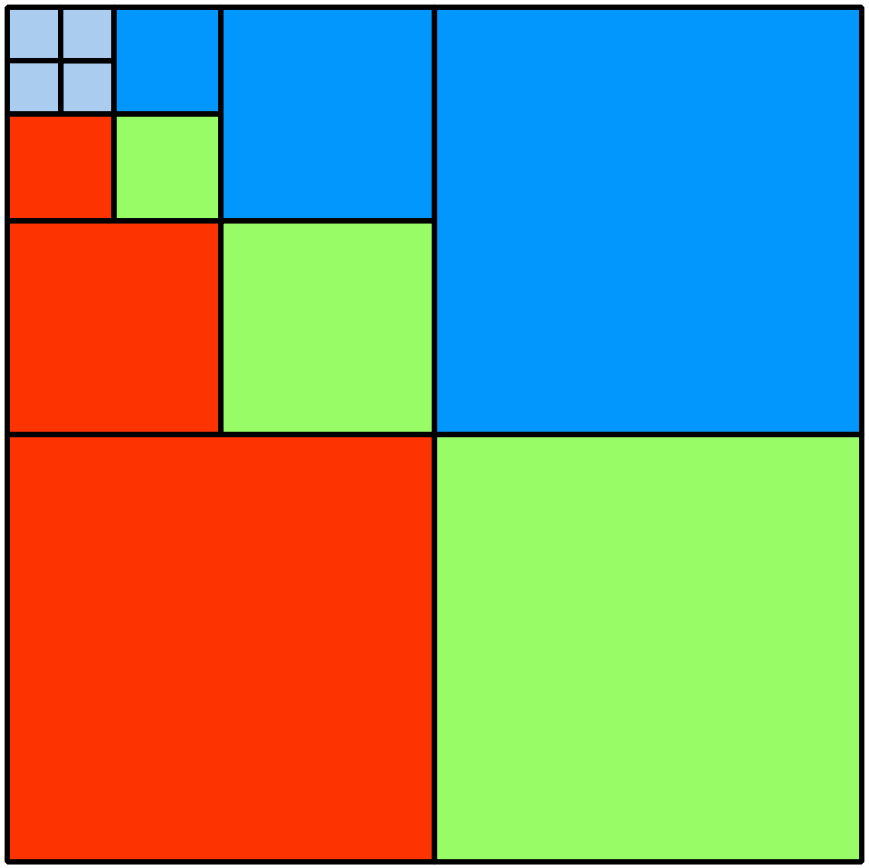
Filtrování textur

- textury pozorované z velké dálky se také musí filtrovat (průměrovat)
  - jinak by se objevil **alias** - zrněním při pohybu kamery
- existuje několik technik, jak si zmenšené textury předem připravit
  - **MIP-map** (*multum in parvo*) asi nejznámější metoda, rekurzivní průměrování 2×2 texelů
  - **ripmap** (Hewlett-Packard) obsahuje i neizotropní zmenšeniny
  - **anizotropní** filtrování - dynamicky se počítá, v jakém poměru se má textura zmenšit, sčítá se jen pár texelů
  - **součtové tabulky** - předem sečtené UL obdélníky

MIP-mapping

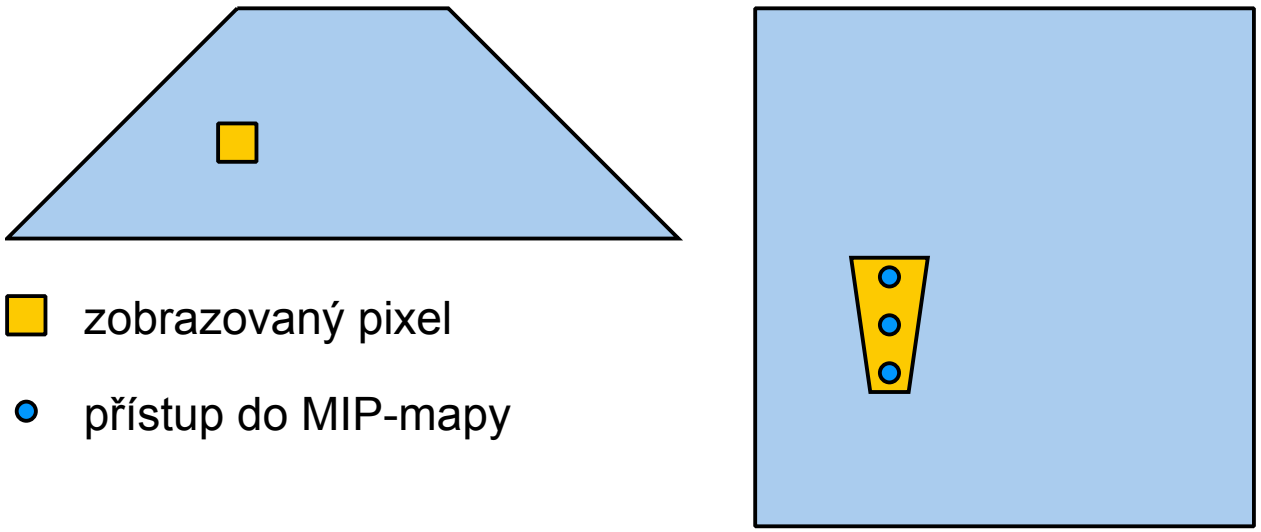
- předem se textura zmenší na binární zlomky ($\frac{1}{4}$, $\frac{1}{16}$, ...), dá se generovat na GPU
  - při zmenšování se použije **průměrování** barev v sousedních pixelech
  - pro **3-kanálovou** bitmapu (např. RGB) existuje šikovné uspořádání MIP-mapy - všechny zmenšeniny se vejdou do chybějící 4. složky
- použití MIP-mapy
  - určí se úroveň a z ní se přečte texel(rychlost)
  - interpoluje se **mezi 2 úrovněmi** MIP-mapy, případně ještě mezi čtyřmi sousedy v nich (kvalita)

MIP-mapping - příklad pro 3 kanály



Anizotropní filtrování

- **zpětně promítnutý pixel** má na textuře tvar deformovaného čtverce
  - z mipmapy se vybere taková úroveň, aby měla kratší strana toho čtyřúhelníku délku cca jednoho texelu
  - podél **delší strany** čtyřúhelníku se průměruje

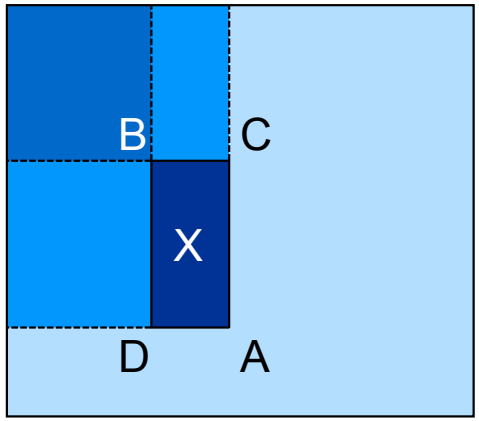


Součtové tabulky

- pro **rychlé počítání součtu** (a tedy i průměru) libovolného obdélníku
  - předem se připraví součty všech obdélníků začínajících v počátku textury
  - je potřeba větší přesnost (minimálně 24 bitů místo 8)

- A, B, C, D ... levé horní součty

$$X = A + B - C - D$$



Efekt textury

- přímá **modifikace barvy**, modulace barvy
  - skládání několika barev (detaily, vyšší variace, šum, . . .)
  - různé operace (viz. *multitexturing*), **RGB**, α
- efekt **obtisku** (*decal*)
  - aplikace poloprůhledného obtisku na povrch tělesa
  - *billboards*, *imposters*
- **hrbolatý povrch** (*bumpmapping*)
  - modifikace normálového vektoru texturou
- **modifikace materiálu**
  - efektu je dosaženo např. násobením dané složky světla skalární texturou (*Luminance* formát)

Efekt textury 2

- **nanášení světla a stínu** (*lightmapping, shadowmapping*)
  - předem spočítané světlo nebo stín se aplikuje texturou
  - násobení difuzní složky světla texturou
  - *lightmapping* - většinou označuje statické mapy s libovolně předpočítaným osvětlením (soft shadows, GI, . . .)
  - *shadowmapping* - většinou se používá pro označení dynamicky počítaných stínů uložených do textury
- **lesklý odraz** okolí (*environment mapping*)
  - napodobení ideálního lesklého odrazu okolí scény
  - GPU musí počítat odražený paprsek a adresovat texturu směrovým vektorem
  - textury se dnes dají **dynamicky** aktualizovat, poměrně realistický odraz scény
  - mapování pomocí **sférických souřadnic** nebo **cube mappingu**

Literatura

- Tomas Akenine-Möler, Eric Haines: **Real-time rendering, 2nd edition**, A K Peters, 2002, ISBN:1568811829
- OpenGL Architecture Review Board: **OpenGL Programming Guide: The Official Guide to Learning OpenGL**, Addison-Wesley, nejnovější vydání (aktuálně 8. vydání pro OpenGL 4.1)
- David Eberly: **Rotation Representations and Performance Issues**, `http://www.geometrictools.com/Documentation/RotationIssues.pdf`, 2002-2008
- Wikipedia: `http://en.wikipedia.org/wiki/Rotation_matrix`, ...