# Textury

**Petr Felkel, Jaroslav Sloup a Vlastimil Havran**

Katedra počítačové grafiky a interakce, ČVUT FEL
místnost KN:E-413 na Karlově náměstí
E-mail: felkel@fel.cvut.cz

Poslední změna: 16.6.2016

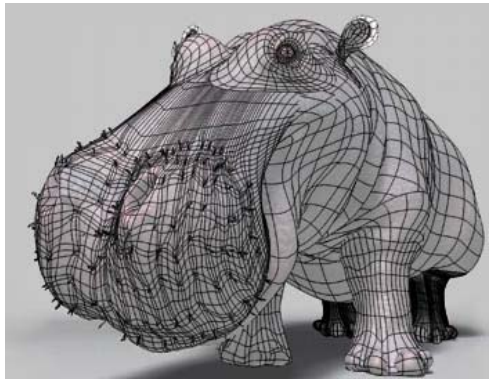
# Textury - osnova

- Co jsou textury a proč se používají?
- Nanášení textur – mapování a transformace souřadnic
- Kroky při definici textur v OpenGL
  - definice texturovacího objektu
  - předání obrázku textury
  - nastavení parametrů (zvětšení, zmenšení a wrap)
- Nanášení textur ve fragment shaderu
  - předání čísla texturovací jednotky shaderu
  - kombinace textury a osvětlení
  - více textur přes sebe (multitexturing)
- Generování texturovacích souřadnic
  - mapování okolního prostředí

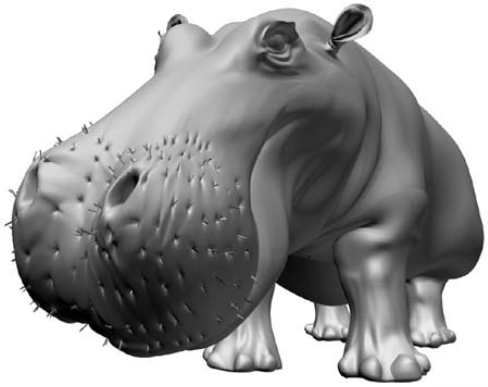
# Co jsou textury a proč se používají?

## Textura

- v obecném významu = vlastnost povrchu, popisuje drobné detaily (drobné změny geometrie) – plyš, omítka, pomeranč,…



*model*



*model + osvětlení*

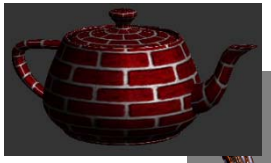
*Autor modelu*
*Jeremy Birn*

*model + osvětlení + textura*

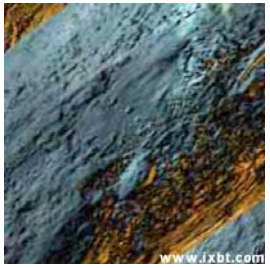
- v počítačové grafice = nástroj na vytvoření dojmu (zvýšení *vizuální kvality*) s minimálními náklady
  (barva, napodobenina odrazu okolí, normála, … )
  jednoduchý tvar VYPADÁ jako složitý, je to ale jen šikovný trik

# Druhy textur dle použití [Heckbert86, MPG'04]

**Textura moduluje některou vlastnost povrchu**



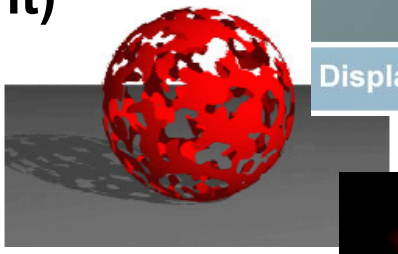
- **Barvu** – difúzní složku materiálu (cihly)
- **Odraz světla z okolí** – změna zrcadlové složky osvětlení (environment mapping)




- **Normálový vektor** – hrboly na hladké geometrii (bump mapping, pomeranč, stará podlaha z prken,…)

Displacement Mapping

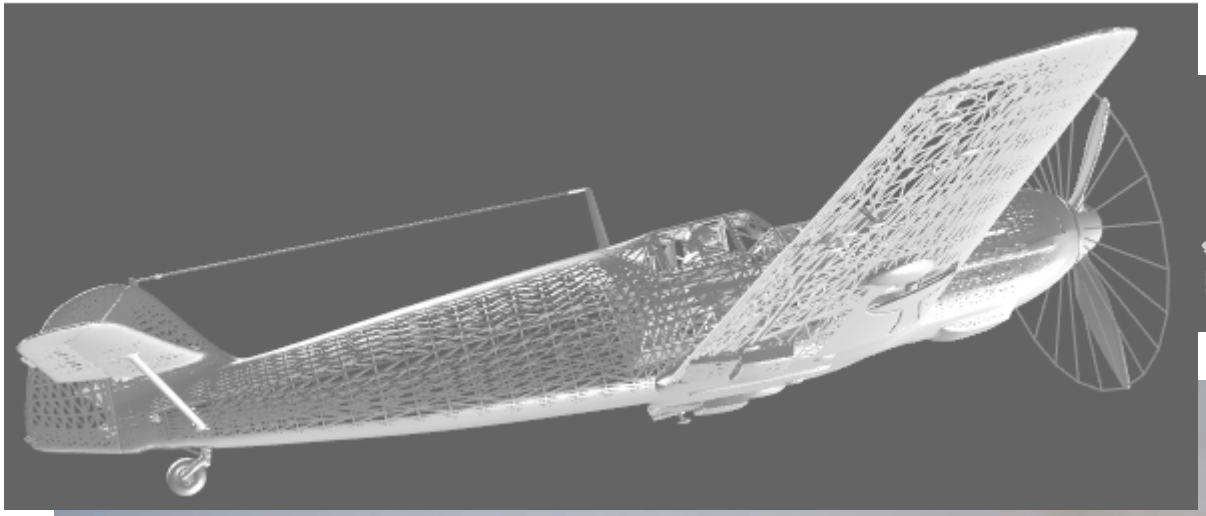
- **Geometrie** – **výšková mapa (displacement)** – posun bodu povrchu ve směru normály

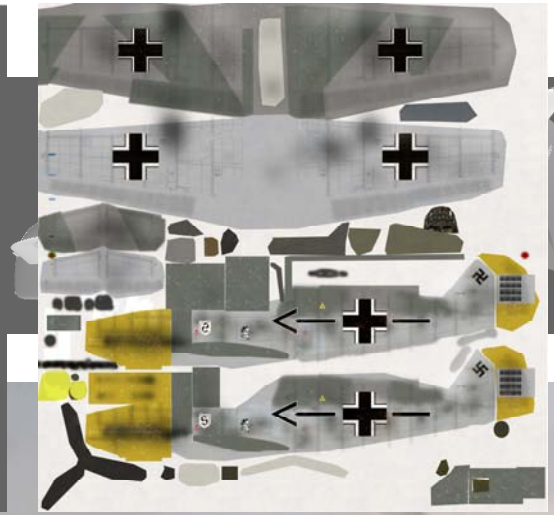


- **Průhlednost** – viz. děravé objekty Olbrama Zoubka

- **Hypertextura** – modeluje složité, nebo nejasné hranice (optické vlastnosti nad povrchem – hard-soft-outside) vlasy, oheň, tráva

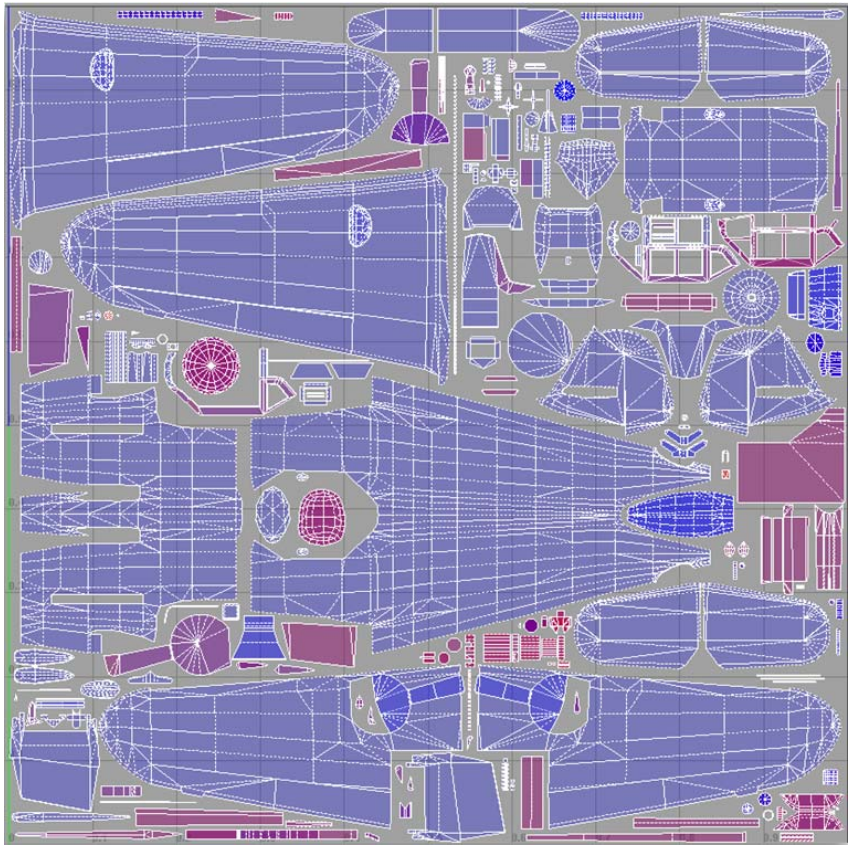
# Příklad letadla z 2. sv. války – Me109

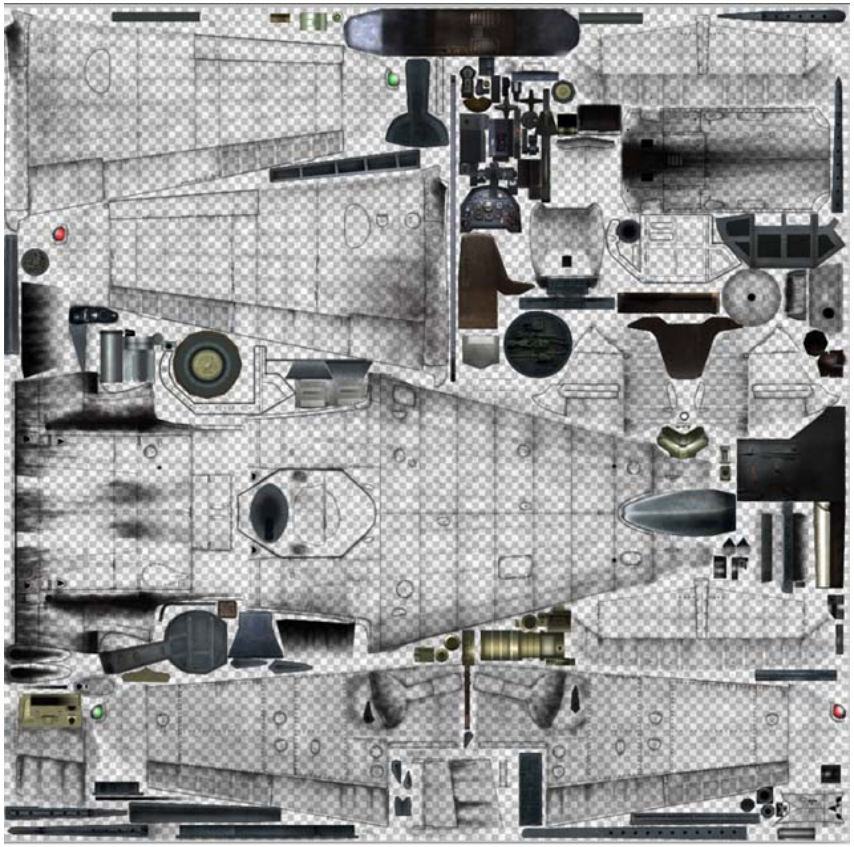




A. Lejczak, http://www.colacola.se/expo_me109.htm

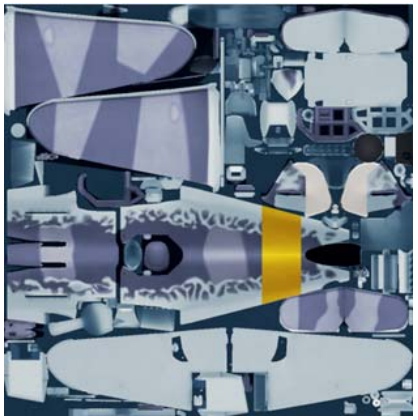
# Rozvinutá geometrie a základní textura

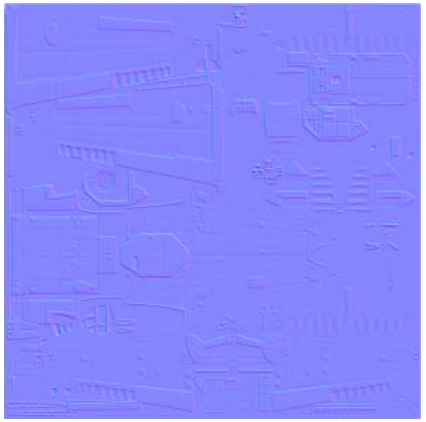




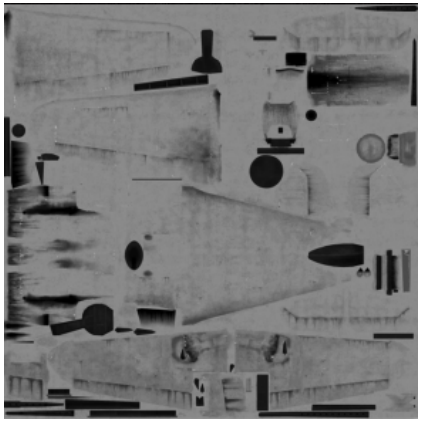
Baranenko: Digital Aircraft Design: How We Create Planes,
http://blog.worldofwarplanes.com/2014/03/13/digital-aircraft-design-how-we-create-planes/

# Barevné varianty + další textury

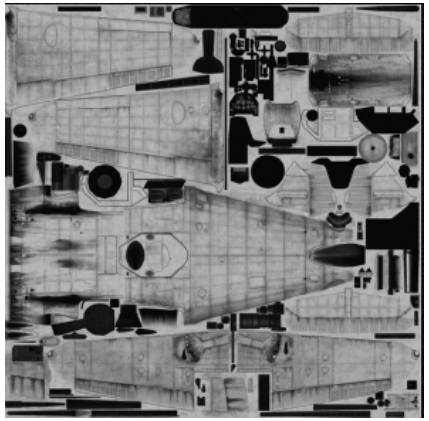




Normal map



Light map – gloss layer



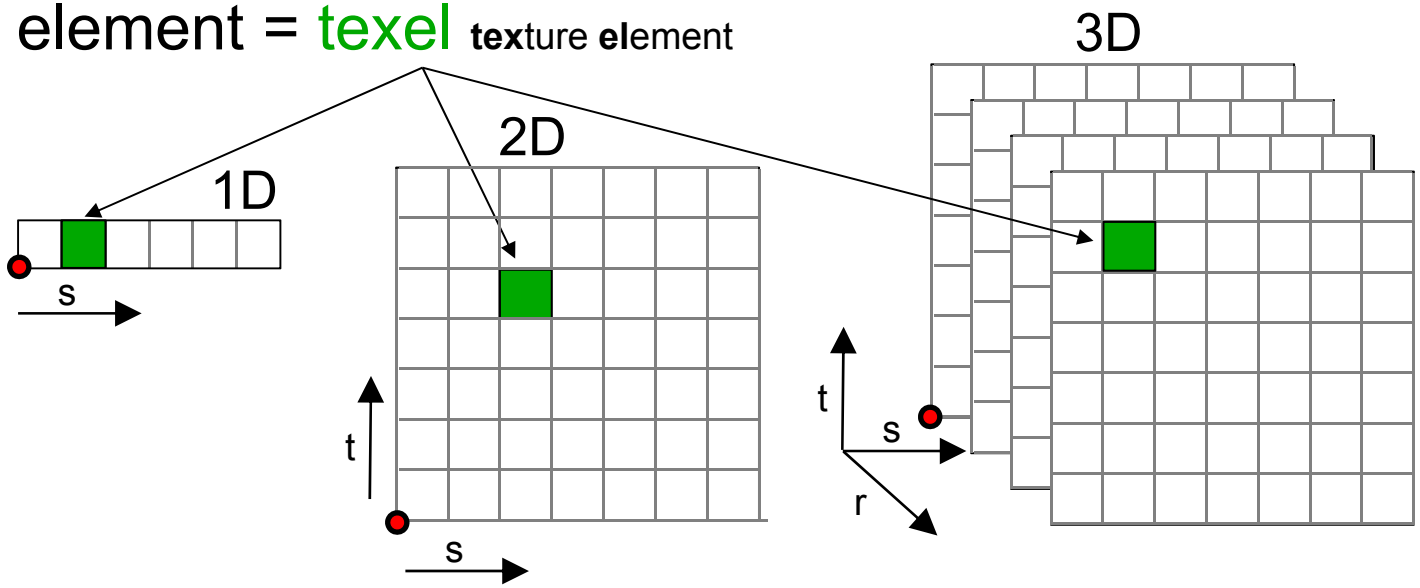
Light map – specular layer


Details

Baranenko: Digital Aircraft Design: How We Create Planes,
http://blog.worldofwarplanes.com/2014/03/13/digital-aircraft-design-how-we-create-planes/

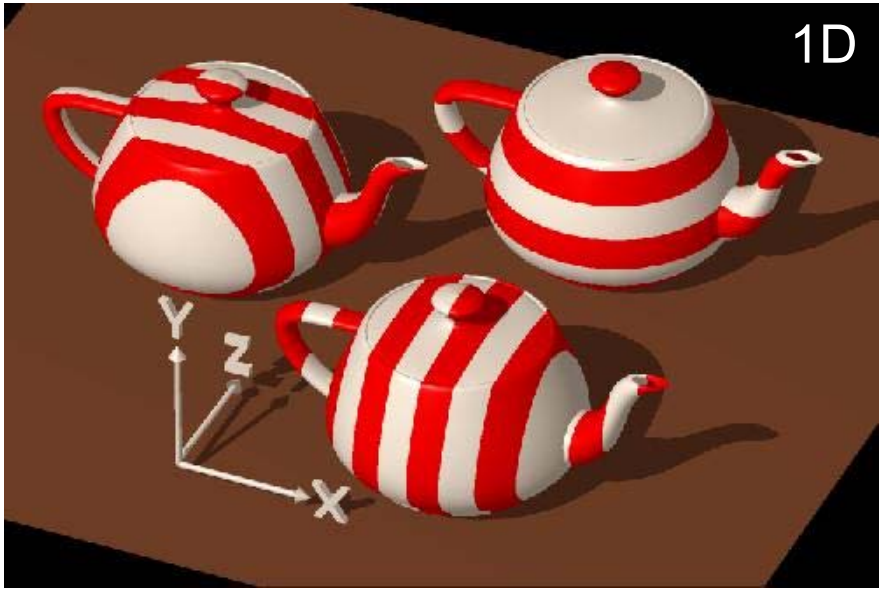
## Textura v počítačové grafice

- pole hodnot (1D-tabulka LUT, 2D-obrázek, 3D-objem)
- element = texel **tex**ture **el**ement

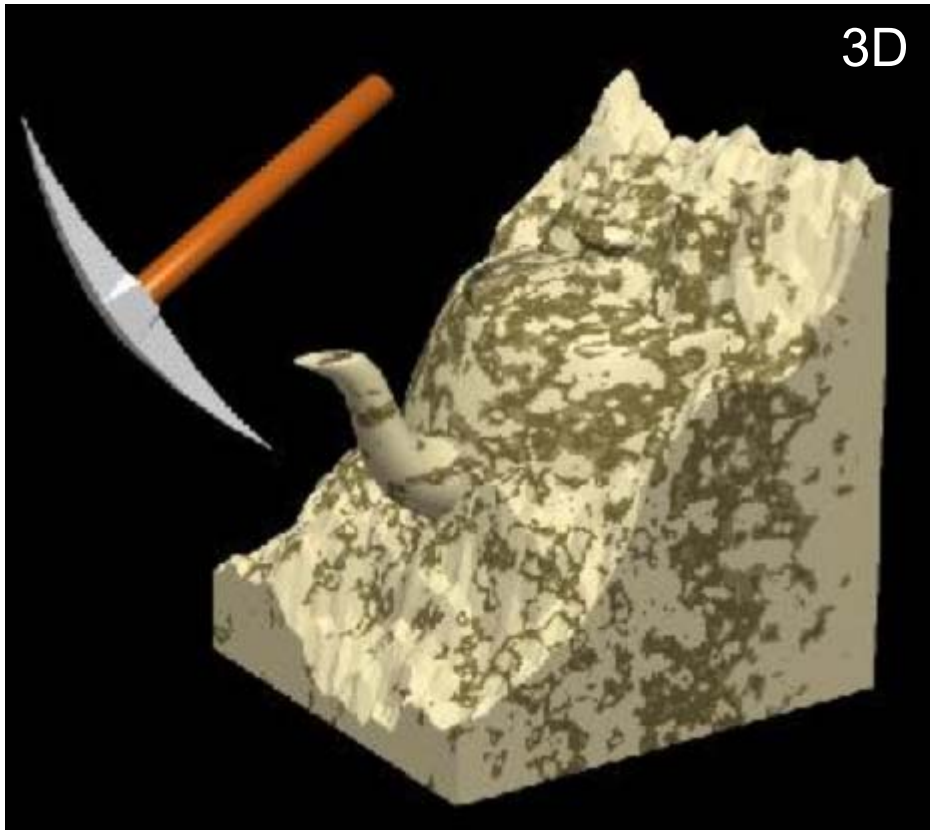



- Popisuje detaily povrchu
- Hodnoty ze souboru, nebo procedurálně

# Příklady aplikace textur různých dimenzí




[TT97]




[TT97]

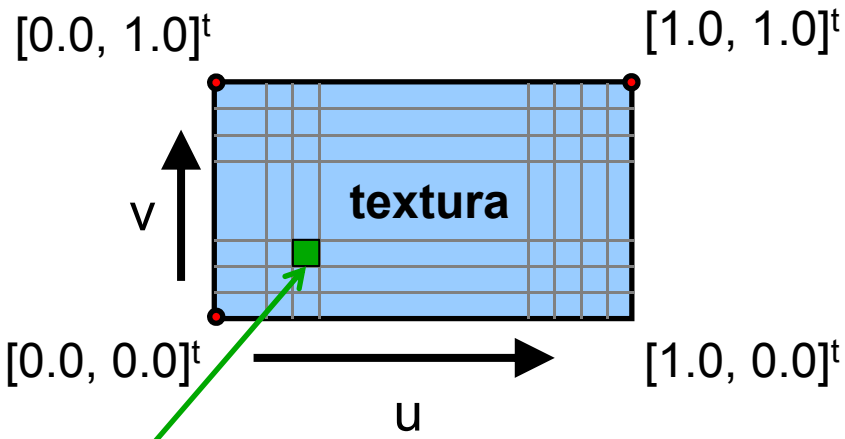

[TT97]

Mapování textury = nanášení textury na objekt

# Souřadnice textur – příklad ve 2D

- Textura je pravoúhelník s rozměry $m \times m$ nebo $m \times n$
- Přistupuje se k ní pomocí texturovacích souřadnic u,v, (s, t)

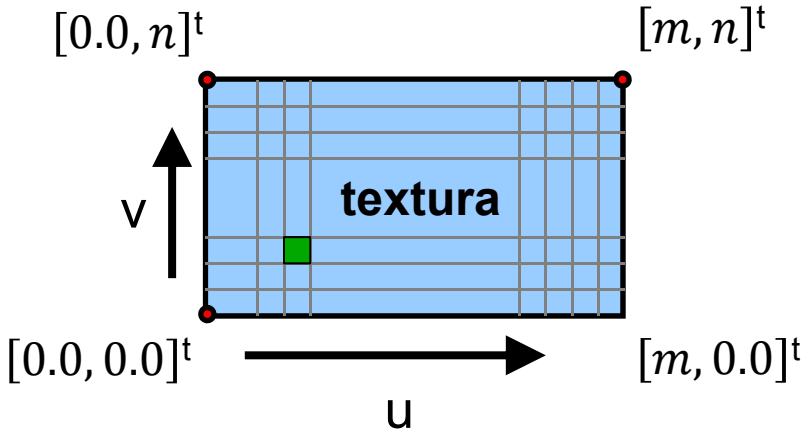
**Normalizované souřadnice**
$\langle 0.0 \,..\, 1.0 \rangle^2$




Použití: běžné textury
Souřadnice: $0.0 \,..\, 1.0$
**GL_TEXTURE_2D**
*g*sampler2D

texel = **tex**ture **el**ement

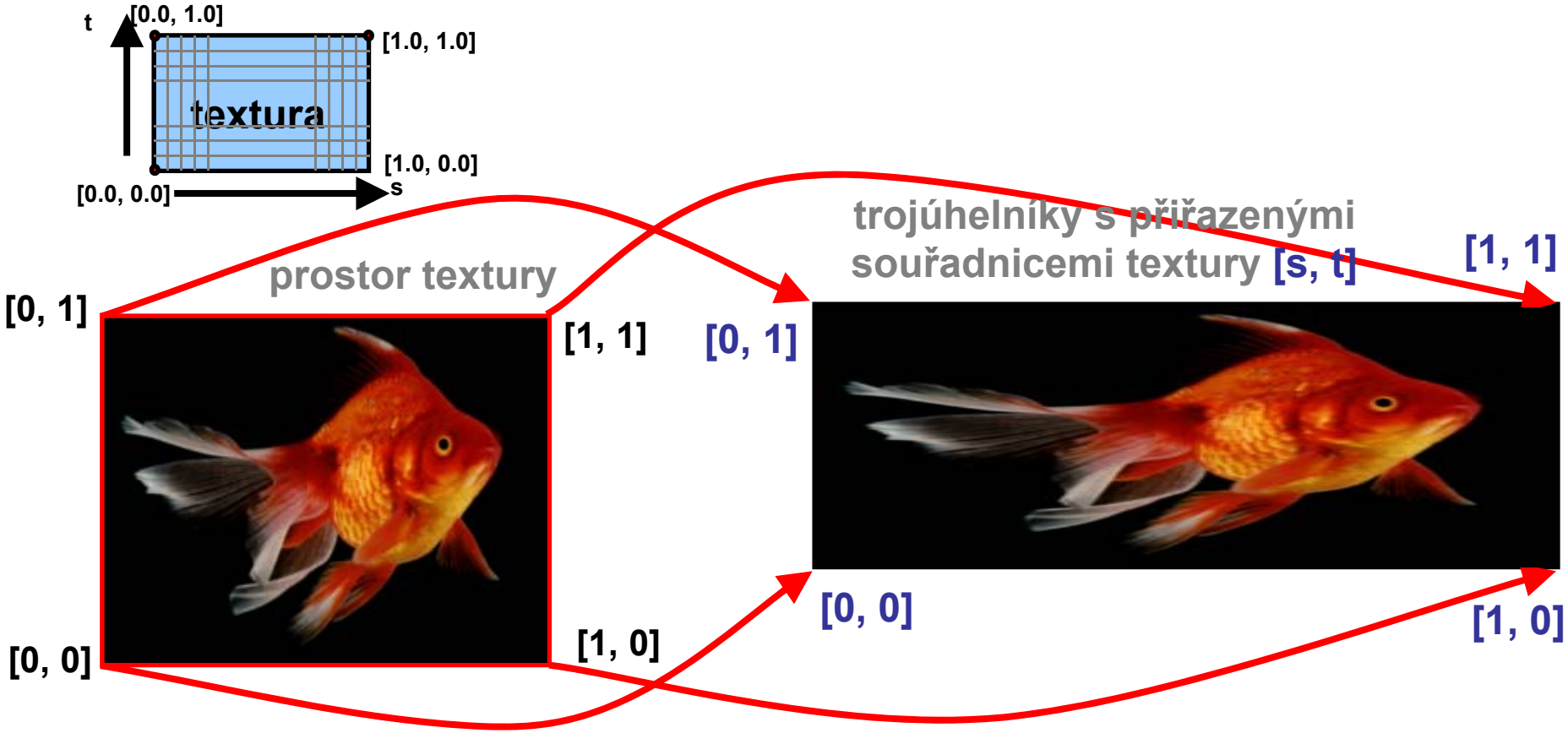
**Souřadnice v prostoru textury**
$\langle 0.0 \,..\, m \rangle \times \langle 0.0 \,..\, n \rangle$




Použití: video, tabulka hodnot
Souřadnice: $0.0 \,..\, velikost$
**GL_TEXTURE_RECTANGLE**
*g*sampler2DRect

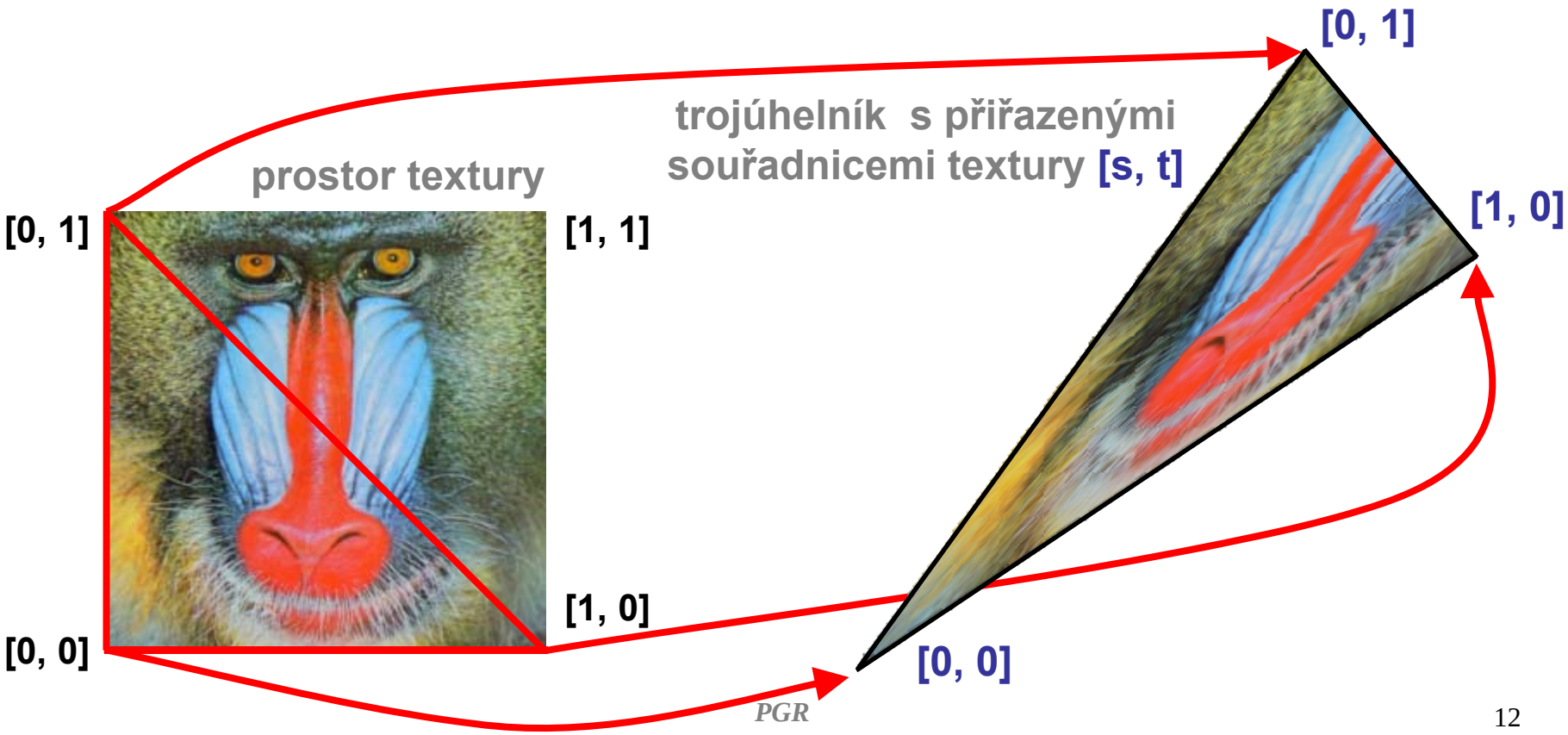
# Příklad nanášení textur

Příklad: Přiřazení souřadnic textury vrcholům polygonu.



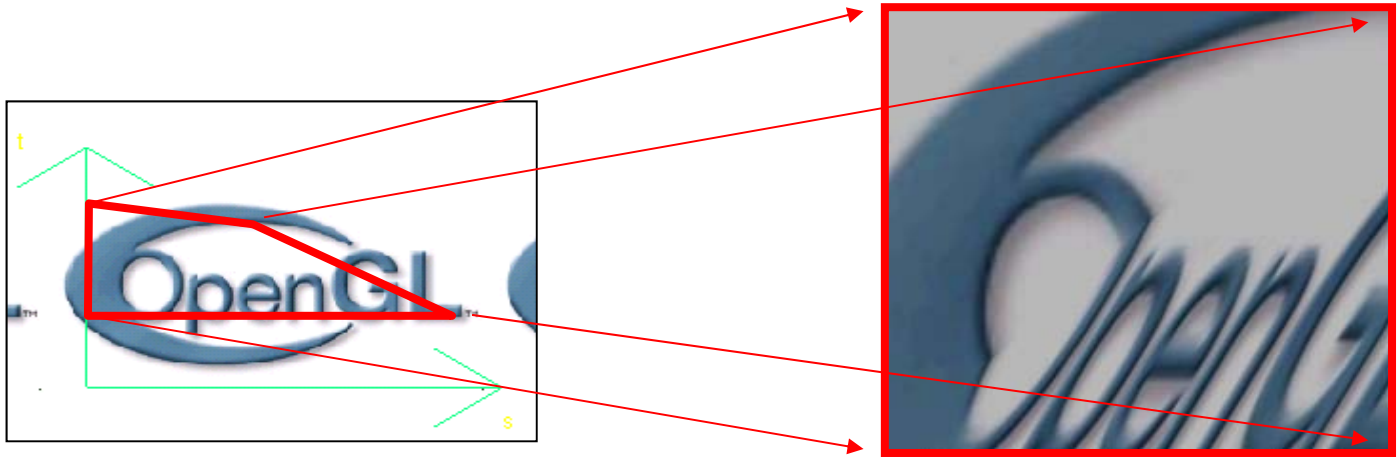
# Příklad nanášení textur

Příklad: Přiřazení souřadnic textury vrcholům trojúhelníka



# Příklad nanášení textur

Příklad: Pozor na přílišnou deformaci textury



# Transformace souřadnic textury

**Souřadnice textury lze před aplikací transformovat**
(například maticí 3x3 nebo
4x4 ve vertex shaderu)

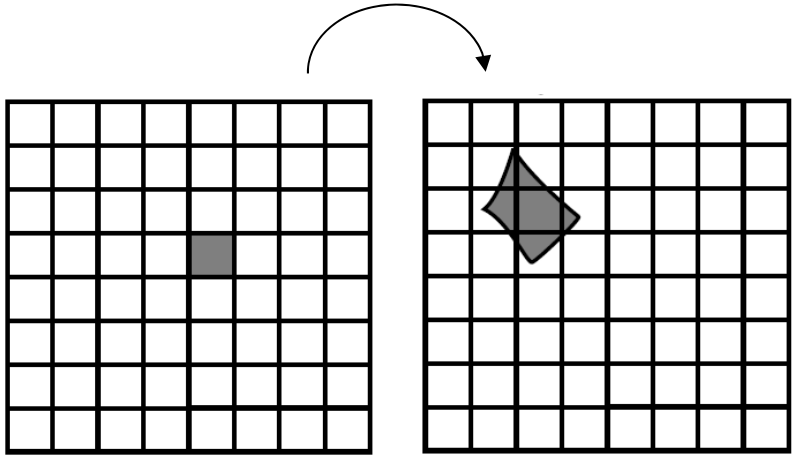
Lze tak pohybovat texturou po objektu nebo lépe nastavit texturu na pozici, kterou chceme dostat.

# Textury - osnova

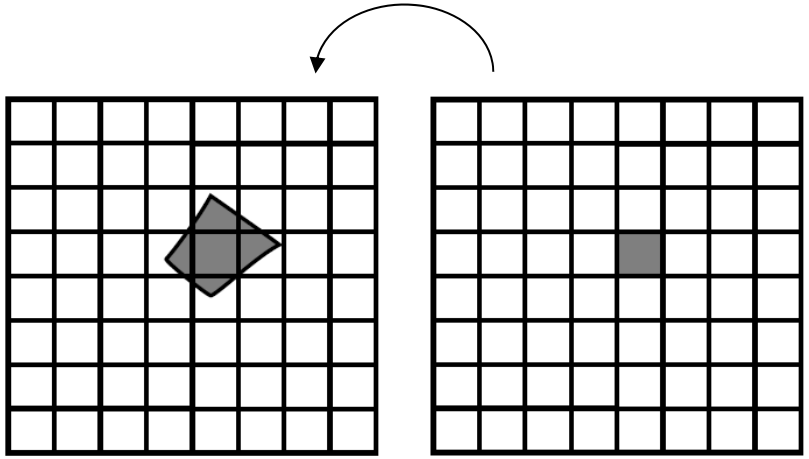
- Co jsou textury a proč se používají?
- Nanášení textur – mapování a transformace souřadnic
- Kroky při definici textur v OpenGL
  - definice texturovacího objektu
  - předání obrázku textury
  - nastavení parametrů (zvětšení, zmenšení a wrap)
- Nanášení textur ve fragment shaderu
  - předání čísla texturovací jednotky shaderu
  - kombinace textury a osvětlení
  - více textur přes sebe (multitexturing)
- Generování texturovacích souřadnic
  - mapování okolního prostředí

# Dopředné a zpětné mapování

Dopředné mapování



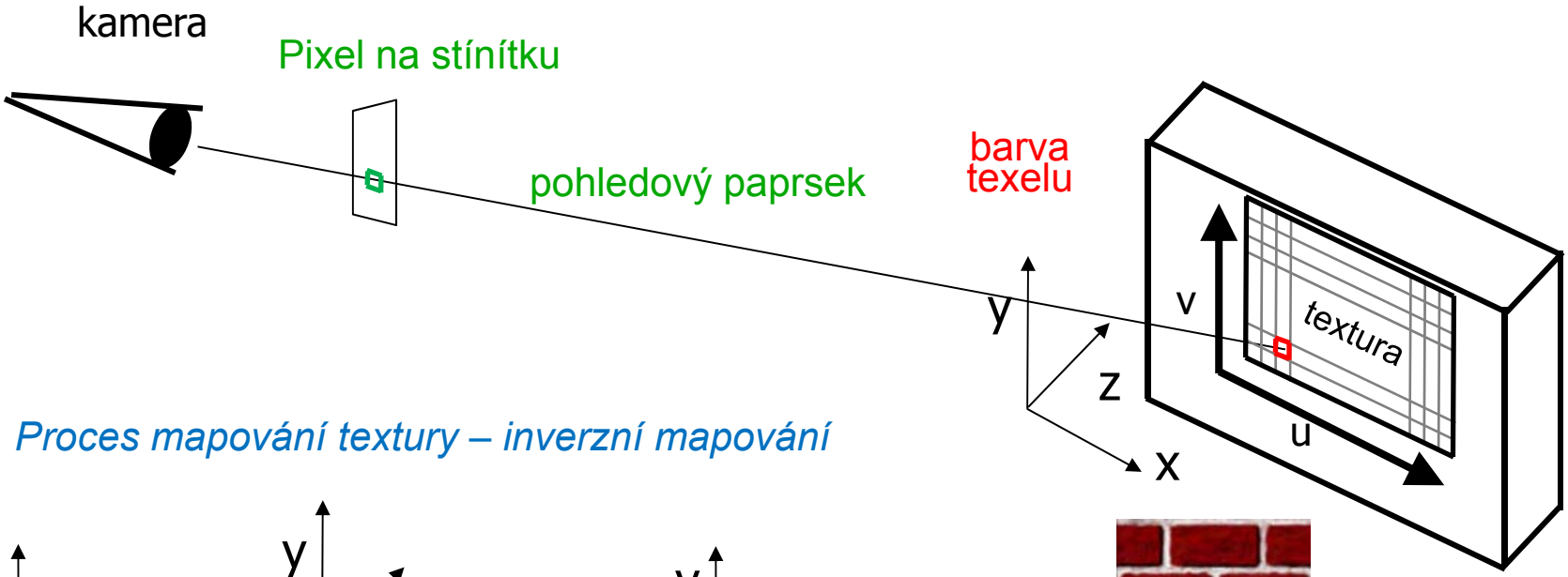
Zpětné mapování



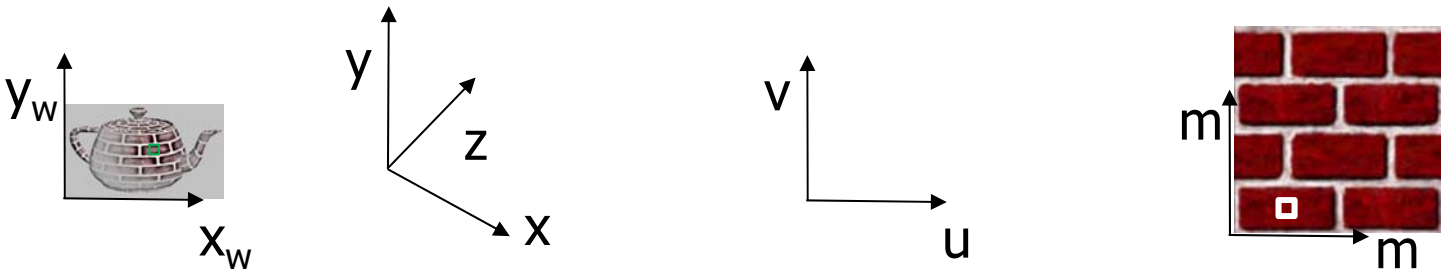
Procházíme pixely textury
- Hledáme umístění na obrazovce
- Zvětšení i zmenšení
  – mohou vznikat díry

Procházíme pixely na obrazovce
- Hledáme umístění textury
- Zvětšení i zmenšení
  – zasáhne mnoho texelů či žádný

# Princip inverzního mapování textur




*Proces mapování textury – inverzní mapování*




$(x_w,y_w)$ —*inverzní mapování*→ $(x,y,z)$ —*inverzní mapování*→ $(u,v)$ —*texturovací jednotka (sampler)*→ **textura**

| Souřadnice pixelu na stínítku | Bod v objektových či světových souřadnicích | Parametrické texturovací souřadnice | Souřadnice v rámci obrázku textury |
|---|---|---|---|

# Inverzní mapování textury

**Inverzní mapování textur** – určuje ze souřadnice fragmentu ve scéně texel, který se do něj promítne.

| Definice (rovinné) textury | $\circ$ | Inverzní mapování 2D obrazu na 3D povrch tělesa |
|---|---|---|
| $T$: $[u\ v]^t$ –> Barva | | $M$: $[x\ y\ z]^t$ –> $[u\ v]^t$ |

$$T \circ M: [x\ y\ z]^t \text{ –> } [u\ v]^t \text{ –> Barva}$$

Nutná apriorní znalost geometrie tělesa
(proto vhodné pro jednoduché tvary)

# Inverzní mapování textury

**Inverzní mapování textur** – určuje ze souřadnice fragmentu ve scéně texel, který se do něj promítne.

| Definice (rovinné) textury | $\circ$ | Inverzní mapování 2D obrazu na 3D povrch tělesa | | Inverzní mapování 3D povrchu tělesa na 2D souř. fragmentu |
|---|---|---|---|---|
| T: $[u\ v]^t$ –> Barva | | M: $[x\ y\ z]^t$ –> $[u\ v]^t$ | | S: $[x_w\ y_w\ z_w]^t$ –> $[x\ y\ z]^t$ |

$$T \circ M \circ S: [x_w\ y_w\ z_w]^t \rightarrow [x\ y\ z]^t \rightarrow [u\ v]^t \rightarrow \text{Barva}$$

Nutná apriorní znalost geometrie tělesa
(proto vhodné pro jednoduché tvary)

# Inverzní mapování textury - upřesnění

Při inverzním mapování (matice M) M: $[x\ y\ z]^t \rightarrow [u\ v]^t$

- Hledáme souřadnici texelu $[u\ v]^t$ (texturovací souřadnice ) pro daný bod $[x\ y\ z]^t$ (vrchol, fragment)
- $[x\ y\ z]^t$ nemusí být nutně jen v objektových souřadnicích
- Příklady souřadnic bodu pro různé situace
  - šmouhy na hledáčku kamery – souřadnice kamery
  - projektor – světové souřadnice
  - nálepka na objektu – modelové souřadnice

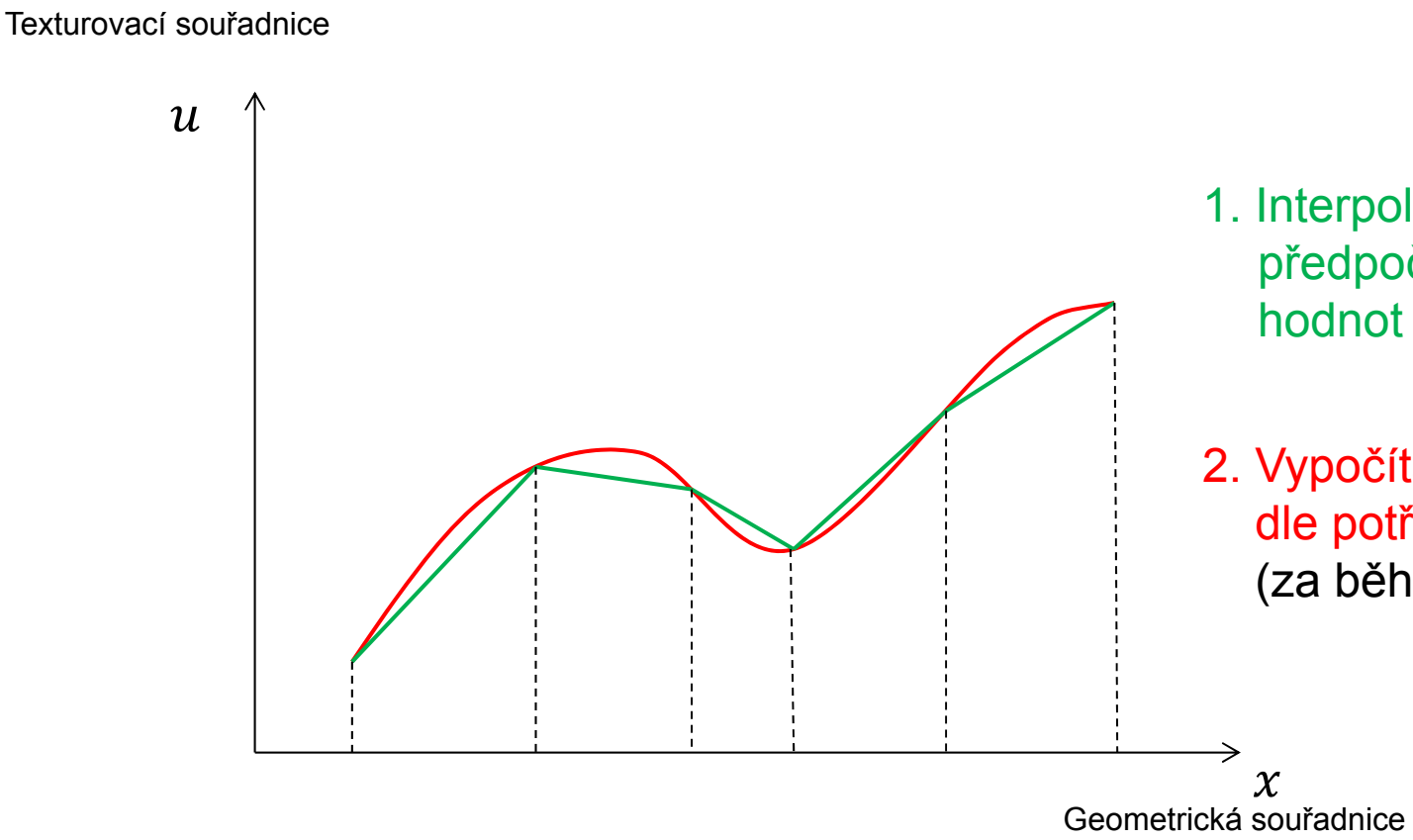
# Realizace inverzního mapování v RT apl.

**2 základní způsoby získání souřadnic textury při vykreslování**

1. Interpolací souřadnic předpočítaných ve vrcholech při vytvoření modelu
   - + rychlé za běhu aplikace (výpočet 1x při vytváření modelu)
   - – je to aproximace z diskrétních vzorků
2. Výpočtem během vykreslování
   - + přesně pro libovolný bod objektu či scény
   - – obtížně pro složitá tělesa

# Porovnání realizací inverzního mapování

Zjednodušeně v 1D




1. Interpolace předpočítaných hodnot

2. Vypočítané hodnoty dle potřeby (za běhu)

# 1. Inverzní mapování textury - interpolací

I. Předpočítat při vytváření modelu (tabulka, obrázek,…)

- inverzní mapování provede autor objektu v modeláři (1x)
- v modelovacích programech jako je blender a Maya existují speciální funkce pro výpočet texturovacích souřadnic (mapování na různá tělesa, viz http://wiki.blender.org/index.php/Doc:Manual/Textures/Options/Map_Input)
- model exportovat včetně texturovacích souřadnic

II. Interpolovat za běhu

- model načíst včetně texturovacích souřadnic
- pro každý vrchol předat $[u, v]$ jako atribut do vertex shaderu (jako normálu či barvu vrcholu) a předávat dále do FS
- $[u, v]$ perspektivně správně interpolovat při rasterizaci (kvalifikátor smooth u výstupu VS a vstupu FS)
- ve fragment shaderu použít pro přístup do textury

Inverzní mapování tedy nahrazuje perspektivně správnou interpolací

# Výpočet inverzního mapování

**2 základní způsoby získání souřadnic textury**

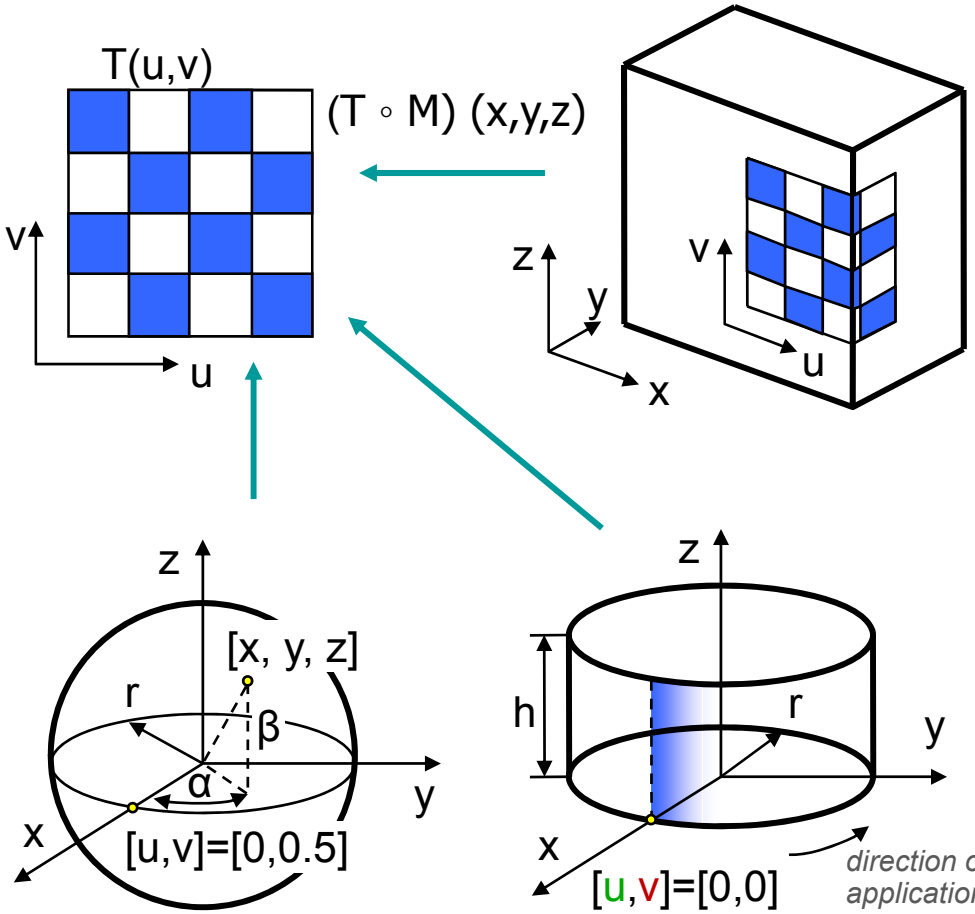
a) analyticky

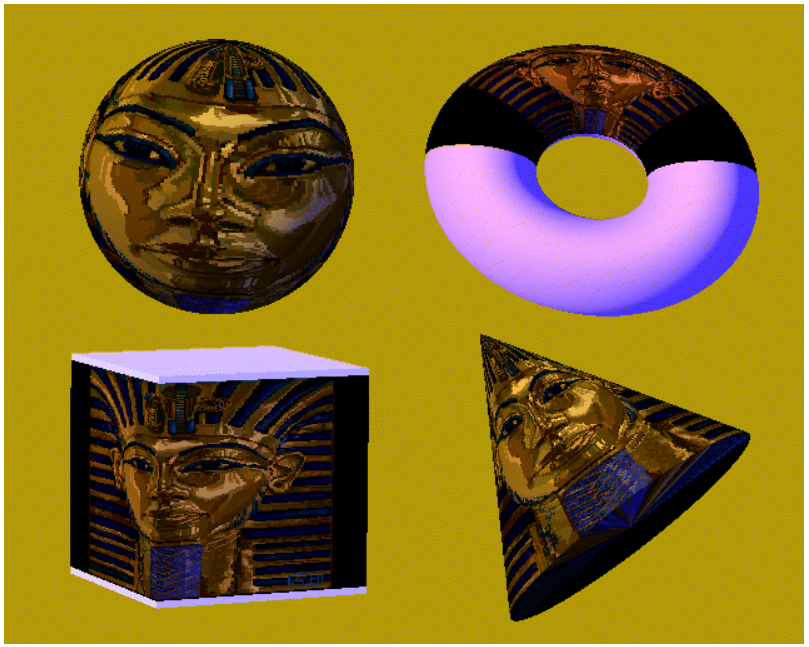
- pro jednoduché 3D objekty, jako je koule a válec, lze spočítat texturovací souřadnice analyticky – inverzní mapování ze 3D povrchu v objektových souřadnicích do 2D $[x_o, y_o, z_o] \rightarrow [u, v]$
- pro projektor či vržené stíny – sestavíme inverzní projekční matici ze světových souřadnic $[x_w, y_w, z_w] \rightarrow [u, v]$

b) přes pomocný objekt – jako je mapa okolí, viz dále

# a) Inverzní mapování textury - analyticky

Koule, toroid, krychle, kužel, válec apod.



válec (*parametrická rovnice*):

x = r*cos(2*π*u)

y = r*sin(2*π*u)

z = v / h

u = …

v = z*h

## b) Inverzní mapování textury – přes objekt

- Objekt se „uzavře“ do pomocného tělesa (rovina, válec, koule, kostka, …)

- Zvolí se mapovaný parametr
  - Normála obalujícího tělesa
  - Normála k povrchu v bodě
  - Spojnice bodu s těžištěm
  - Odražený paprsek (pro daný směr pohledu a normálu povrchu)

# 2b. Dvoustupňové mapování na rotační těleso pro různý pomocný povrch

Rovina

Válec

Koule

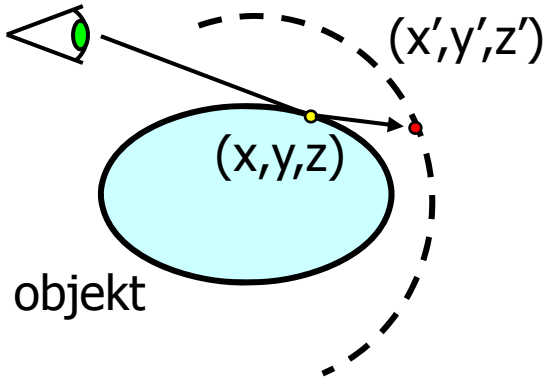
Kostka

[Watt, Policarpo]

# b) Inverzní mapování textury – přes objekt

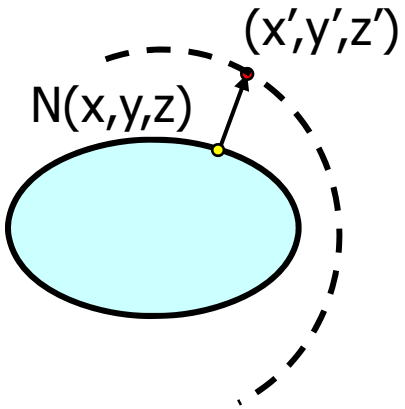
Pro složité objekty se textura nalepí na pomocné obalující těleso

Mapování prostředí

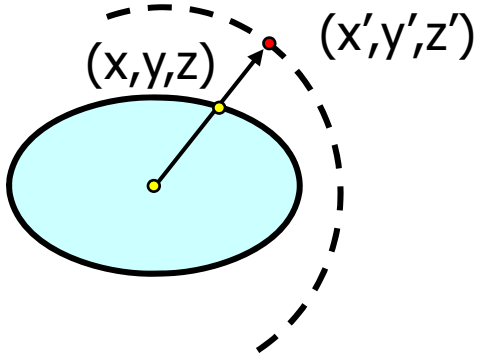



- Odražený paprsek

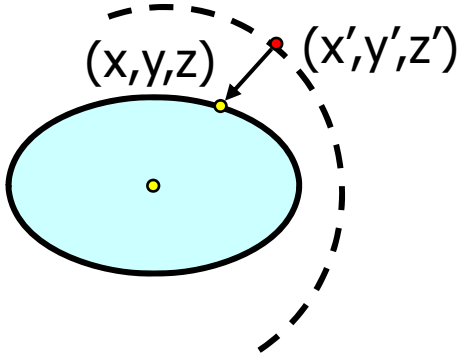
Dvoustupňové mapování




- Normála povrchu objektu

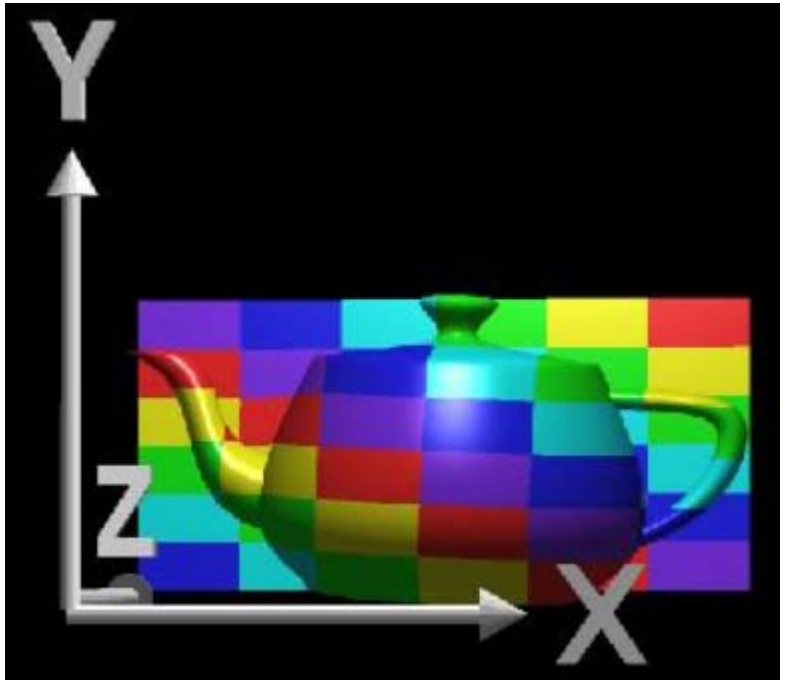


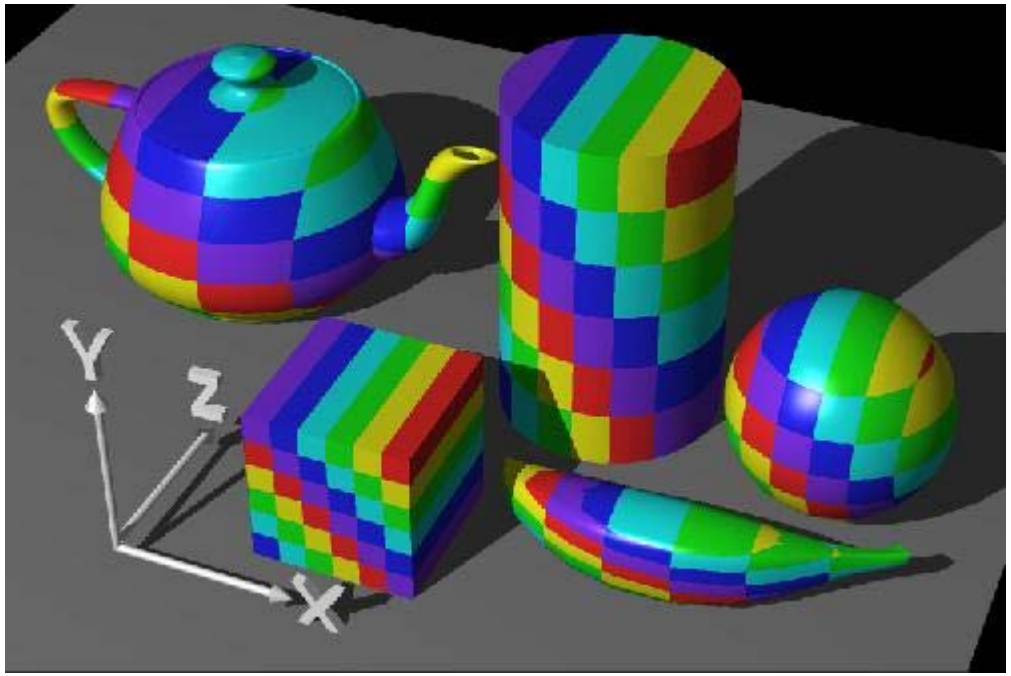

- Těžiště objektu



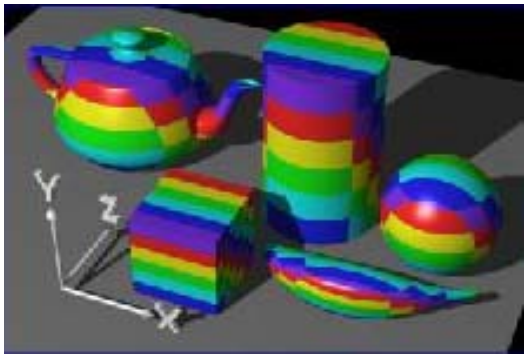

- Normála pomocného povrchu

# Pomocné těleso je rovina

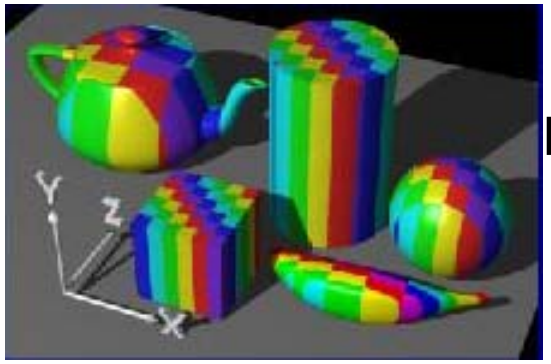



Rovina XY



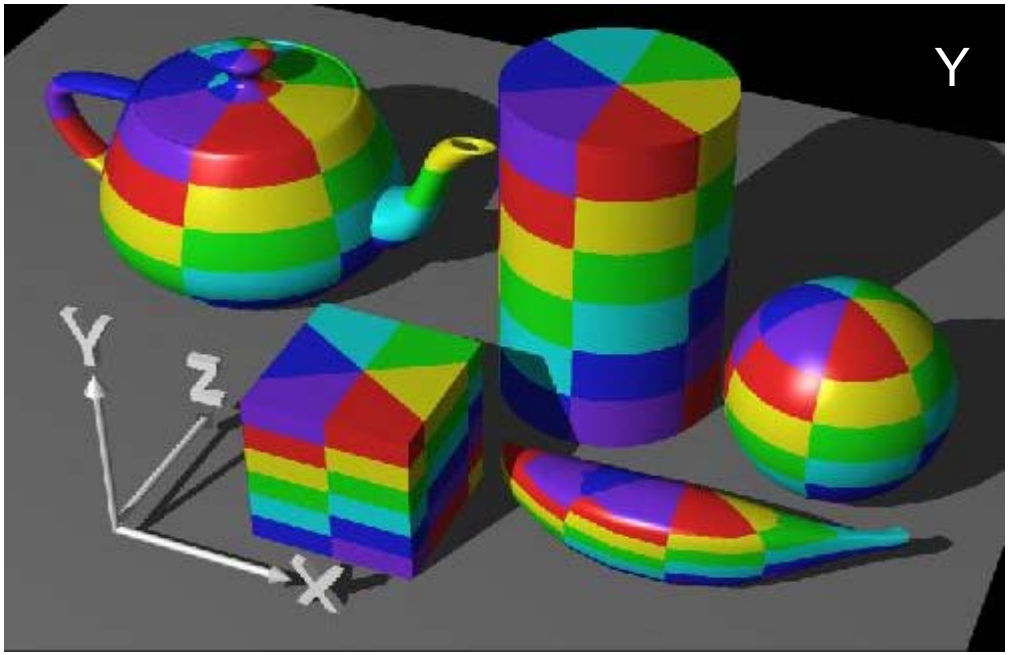
Rovina ZY

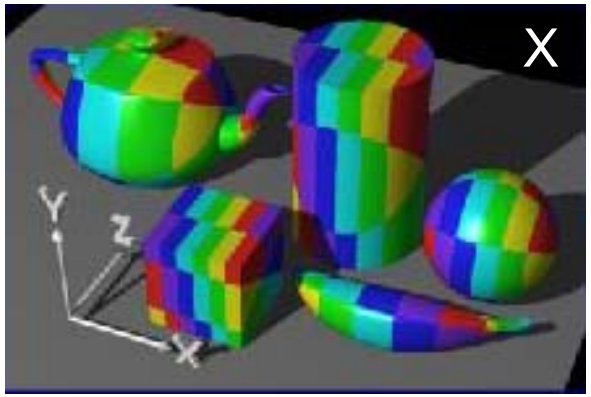


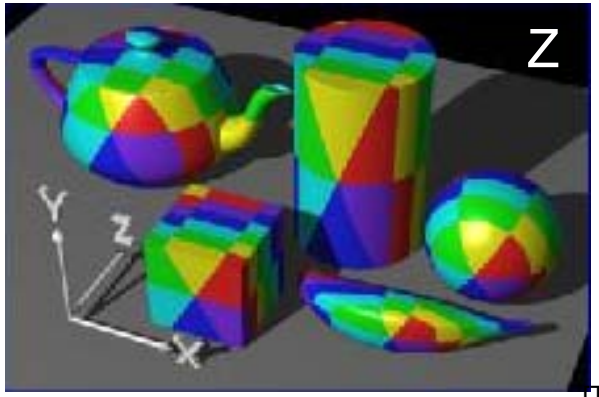
Rovina XZ

[TT97]

# Pomocné těleso je válec

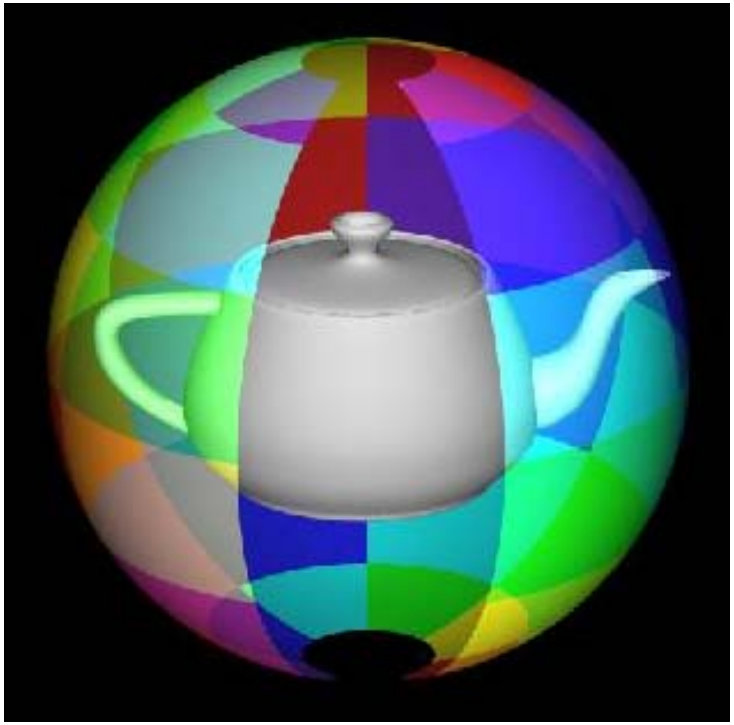


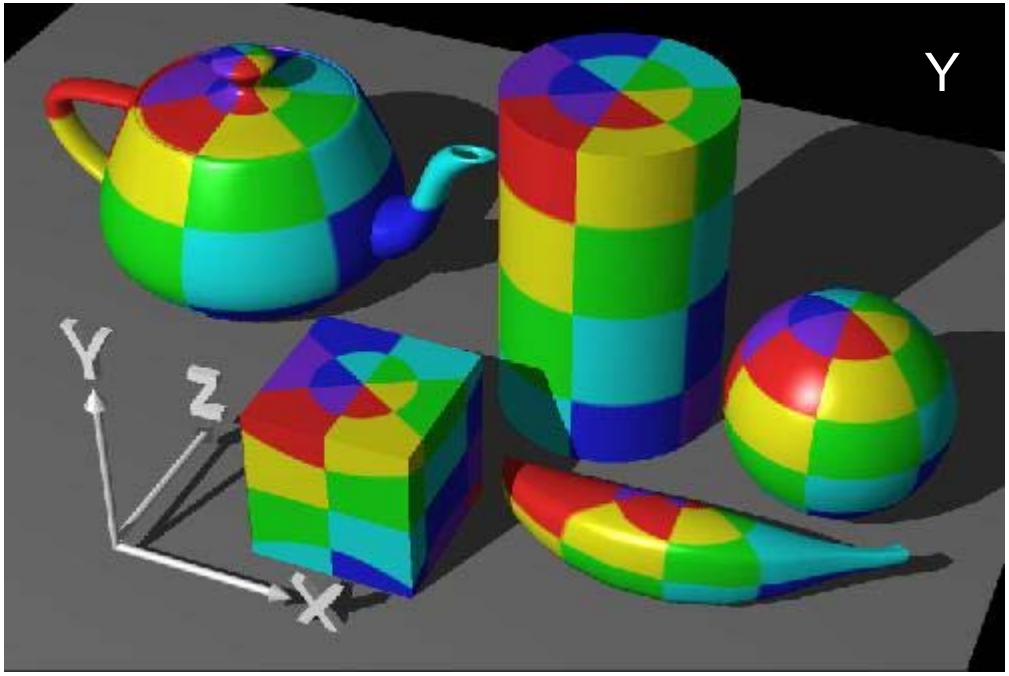



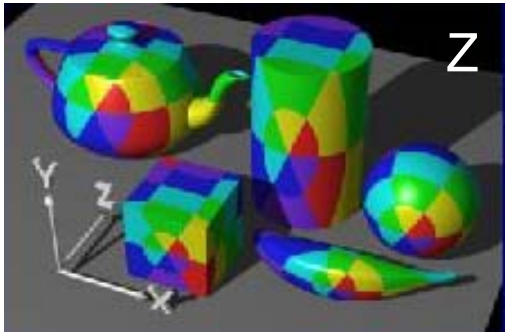



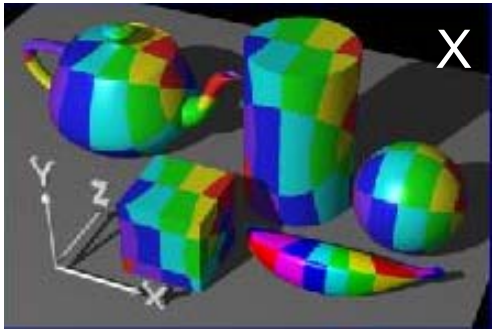

[TT97]

# Pomocné těleso je koule





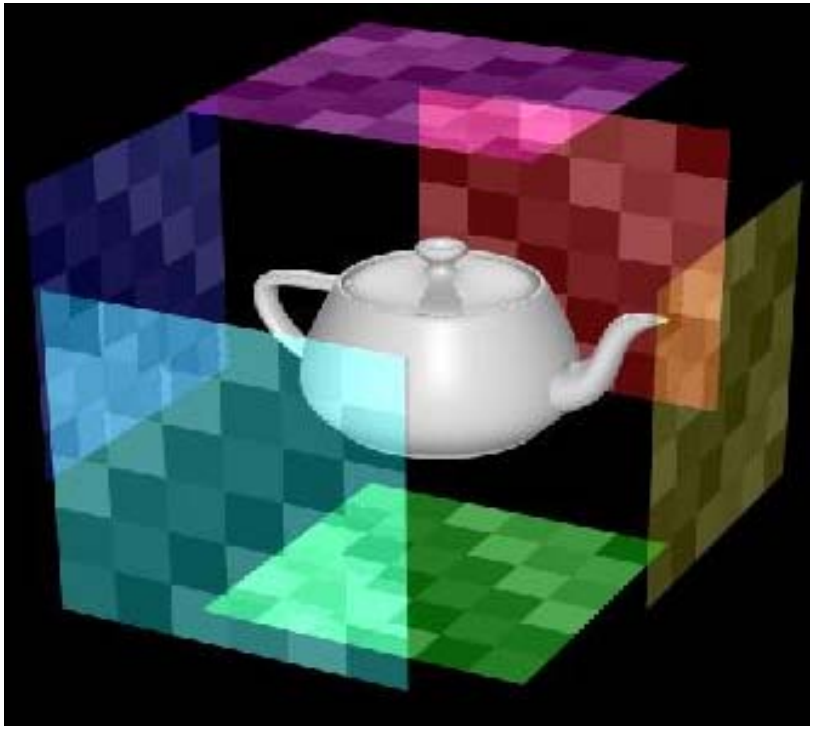



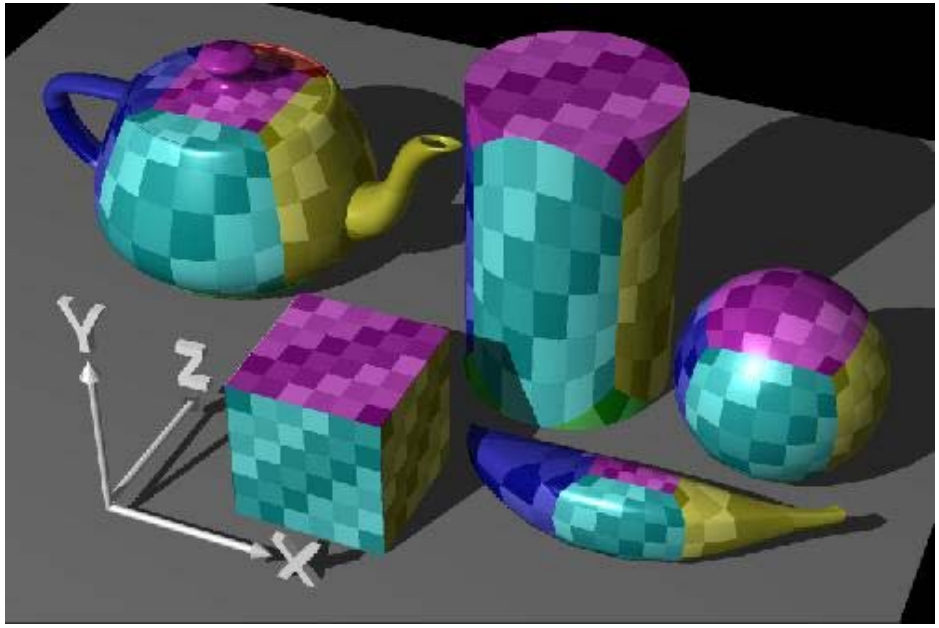



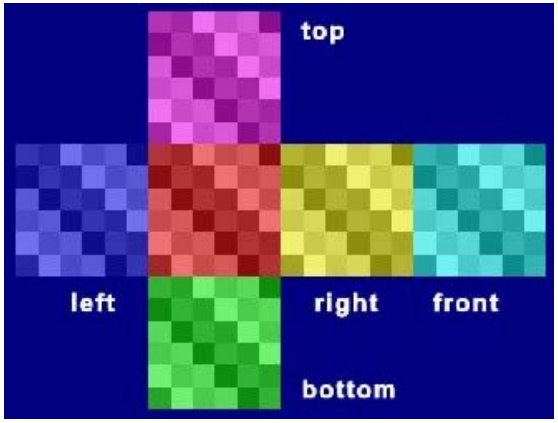

(x,y,z) to spherical coords
latitude to s, longitude to t

[TT97]

# Pomocné těleso je kostka

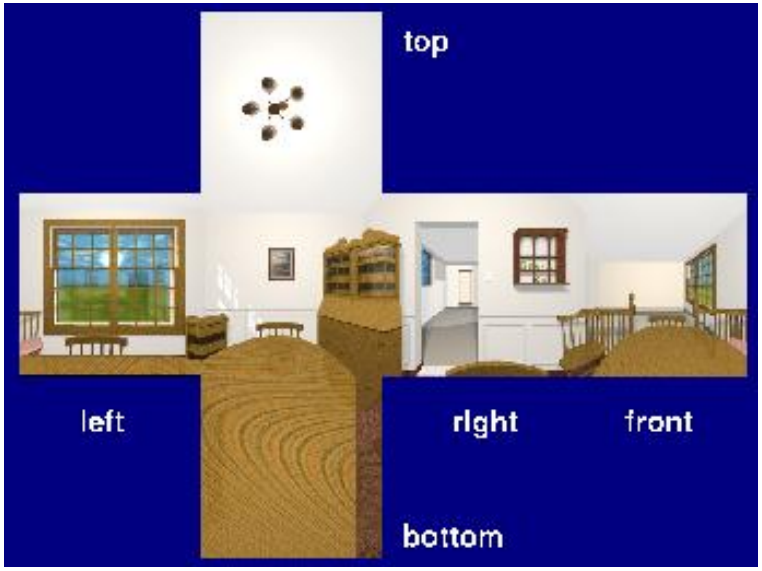





[TT97]

# Pomocné těleso je kostka

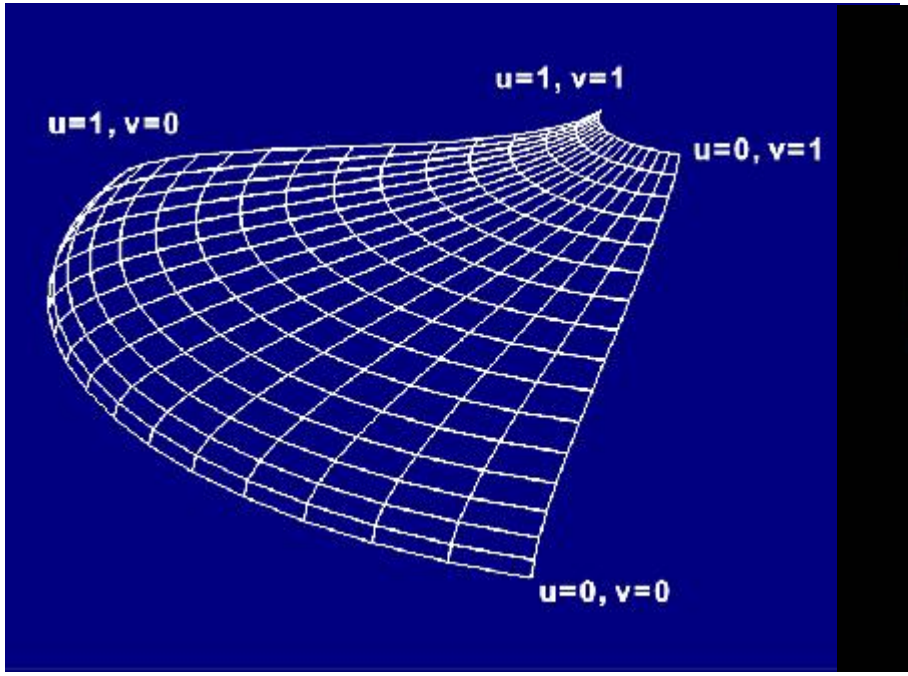


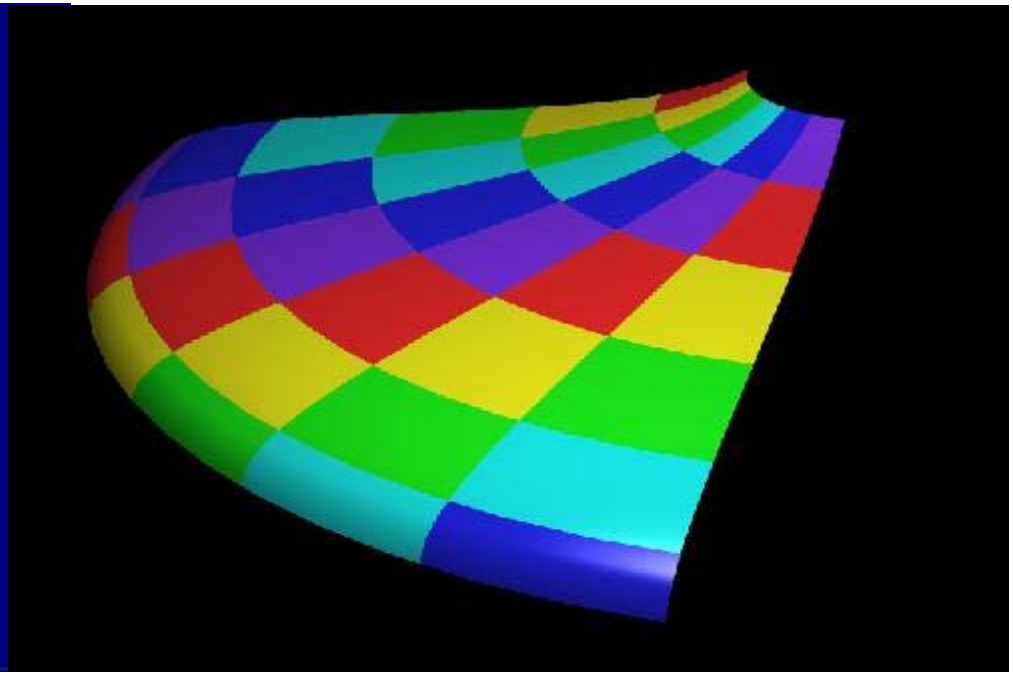

Textura prostředí je nanesena na čajník.
Toto není mapování prostředí – zde není pozorovatel.

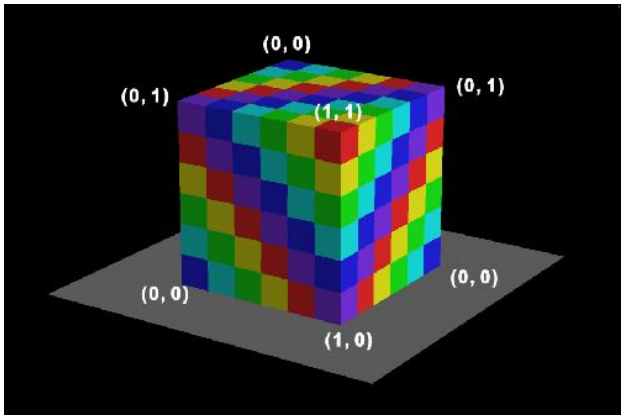


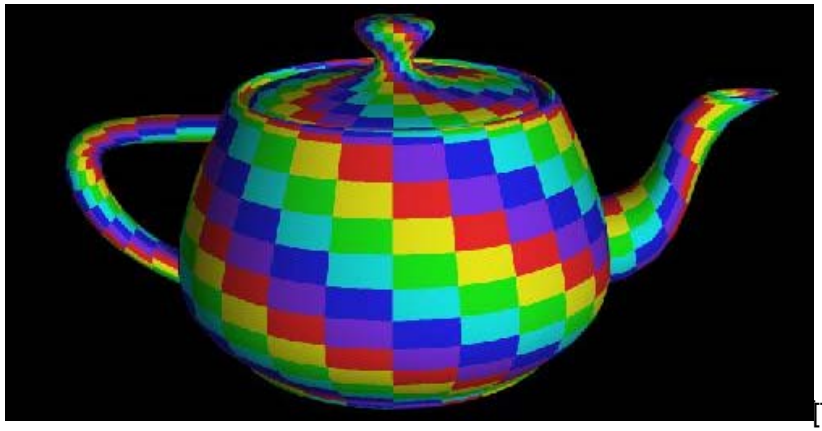
[TT97]

# Parametrické plochy – přímo parametr







[TT97]

# Textury v OpenGL

# Textury - osnova

- Co jsou textury a proč se používají?
- Nanášení textur – mapování a transformace souřadnic
- Kroky při definici textur v OpenGL
  - definice texturovacího objektu
  - předání obrázku textury
  - nastavení parametrů (zvětšení, zmenšení a wrap)
- Nanášení textur ve fragment shaderu
  - předání čísla texturovací jednotky shaderu
  - kombinace textury a osvětlení
  - více textur přes sebe (multitexturing)
- Generování texturovacích souřadnic
  - mapování okolního prostředí

# Kroky při texturování v OpenGL

- Příprava (inicializace v aplikaci)
  - vytvoří se texturovací objekt (gen, bind)
  - předá se mu pole hodnot texelů (image)
    - OpenGL nemá funkce pro načtení obrázku ze souboru
    - Nutno použít externí knihovnu (např. DevIL)
  - nastaví se parametry texturování (filter – min / mag, wrap, …)
- Použití textury (vytvořeného texturovacího objektu)
  - v aplikaci (OpenGL)
  - v shaderu (GLSL)

# Použití texturovacího objektu

- Použití textury v aplikaci (OpenGL)
  - připojí se texturovací objekt (bind)
  - zvolí se aktivní texturovací jednotka (activeTexture)
  - získá se odkaz na sampler v shaderu (getUniformLocation)
  - číslo zvolené jednotky se předá shaderu (uniform(texID, …))
  - vykreslí se objekt (tj. souřadnice vrcholů i textur)
- Použití textury v shaderu (GLSL)
  - v shaderu se definují proměnné
    - typu sampler – odkazující na texturovací jednotku s texturou
    - typu vector – pro předání texturovacích souřadnic
  - přečte se hodnota textury na souřadnicích (texture)
  - hodnota z textury se použije ve vzorci
    (viz kombinace textur a barvy povrchu - dále)

# Vytvoření texturovacího objektu

- v paměti lze mít více textur najednou a přepínat je **glBindTexture()**
- přepínání je rychlejší než nahrávání pomocí **glTexImage2D()**
- textura s veškerým nastavením se uloží do texturovacího objektu (**texture object)**
- texturovací objekt identifikován jménem

Použití více textur:

1. **glGenTextures(**GLsizei **n,** GLuint ***textureNames);**
   vygeneruje se pole jmen pro více texturových objektů

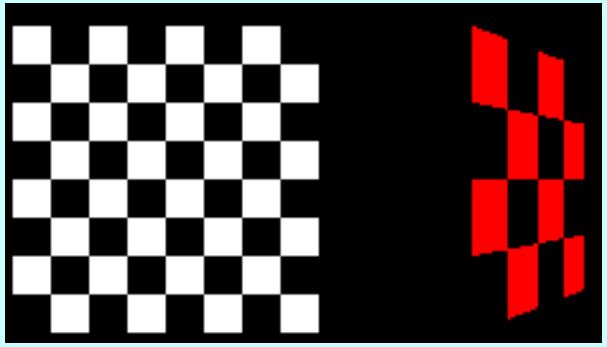
2. **glBindTexture(**GLenum **target,** GLuint **textureName);**
   Poprvé se texturovací objekt vytvoří
   a  případně spojí s texturou (s texely i nastavením)
3. Dalším vyvoláním se přepíná mezi texturovacími objekty

# Příklad – přepínání mezi dvěma texturami

```
 … v inicializaci …
GLuint texName[2];          /* místo pro dvě jména */
glGenTextures(2, texName); /* generuj dvě jména */

/* první textura */
glBindTexture(GL_TEXTURE_2D, texName[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
             checkImageWidth, checkImageHeight, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, checkImage);

/* druhá textura */
glBindTexture(GL_TEXTURE_2D, texName[1]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
             otherImageWidth, otherImageHeight, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, otherImage);
```

# Příklad – přepínání mezi dvěma texturami

```
… při kreslení …

/* první textura */
glBindTexture(GL_TEXTURE_2D, texName[0]);
drawRectangle0();


/* druhá textura */
glBindTexture(GL_TEXTURE_2D, texName[1]);
drawRectangle1();
```



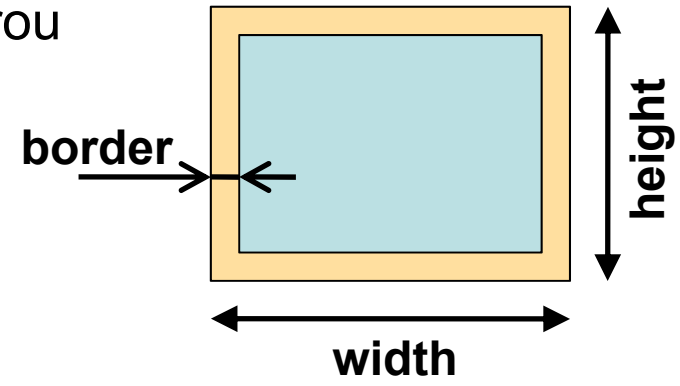
# Určení textury příkazem glTexImage2D 1/5

- OpenGL používá 1D, 2D a 3D textury
- Zde se soustředíme pouze na 2D textury

```
glTexImage2D(
        GLenum target, GLint level, GLint internalFormat,
        GLsizei width, GLsizei height, GLint border,
        GLenum format, GLenum type, const GLvoid *texels );
```

Definuje dvojrozměrnou texturu (**target = GL_TEXTURE_2D**) a načte ji do paměti textur

- **width**, **height** – rozměry obrázku s texturou
- **border** – okraj
  - v OpenGL ≥ 3.0 musí být 0
  - nahrazen jednou barvou



# Určení textury příkazem glTexImage2D - Příklad



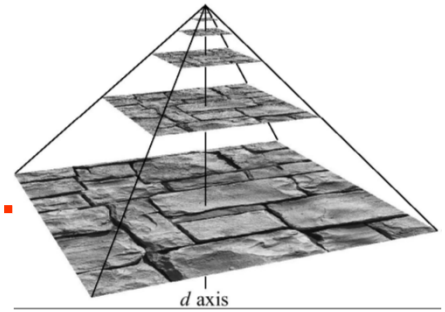
Textura

- Ručně vygenerovaný obrázek šachovnice
- 64x64 pixelů, každý má 4 složky (RGBA), každá složka v 8 bitech

```
static GLubyte checkImage[64][64][4];      /* místo pro uložení textury */
/* zarovnání řádek pixelů v paměti */
/* 1 (na byty), 4 (na slova), atd.  */
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexImage2D(GL_TEXTURE_2D,   0,                    4,    64, 64,   0,
             GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
```

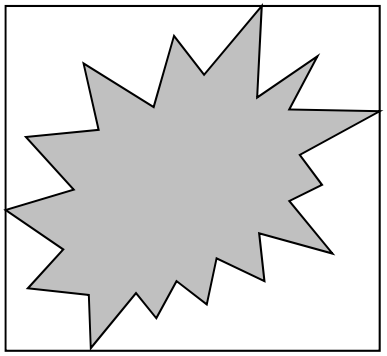
target, level, Components (internal format), width, height, ~~border~~, format, type, texels

# Určení textury v příkazu glTexImage2D 3/5

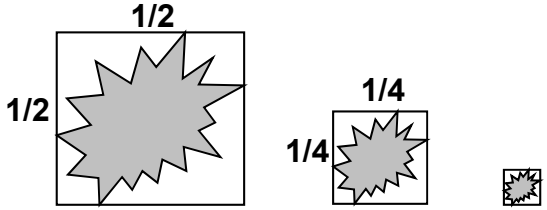



- **level** = úroveň detailu (mipmap pro LOD)
  - 0 pro jedno rozlišení
  - pro vzdálené textury, neb rozlišení klesá se vzdáleností k pozorovateli

**mipmaps**



**původní textura**
**(level = 0)**




**před-filtrované textury**
**(levels=1,2, … )**

**poslední úroveň mipmap má velikost 1x1 texel**

| size | level |
|---|---|
| 64 | 0 |
| 32 | 1 |
| 16 | 2 |
| 8 | 3 |
| 4 | 4 |
| 2 | 5 |
| 1 | 6 |

- **glGenerateMipmap(GLenum target)**
  - vytvoří pyramidu textur (mipmaps) až do velikosti 1x1
  - upraví měřítko textury, aby byla velikost mocninou 2

# Určení textury v příkazu glTexImage2D 4/5

## internalFormat

- popis složek textury (RGBA, hloubka, luminance (jas), intenzita)
  – jak budou poskládány za sebou v paměti textur
- V závorce postup určení složek [R,G,B,A]) + bitová hloubka
  - 6 základních **vnitřních formátů**

    ALPHA (0,0,0,A), DEPTH_COMPONENT,
    RGB(R,G,B,1), RGBA (R,G,B,A)
    OpenGL je uloží po svém dle vnitřního formátu
  - Řada **vnitřních formátů s danou velikostí**
    ALPHA{4|8|12|16}, DEPTH_COMPONENT{16|24|32}
    R3_G3_B2, RGB{4,5,8,10,12,16}, RGBA4, RGB5_A1,
    RGBA{2,4,8,12,16}, .....

# Určení textury v příkazu glTexImage2D 5/5

## Informace o poli texelů texels

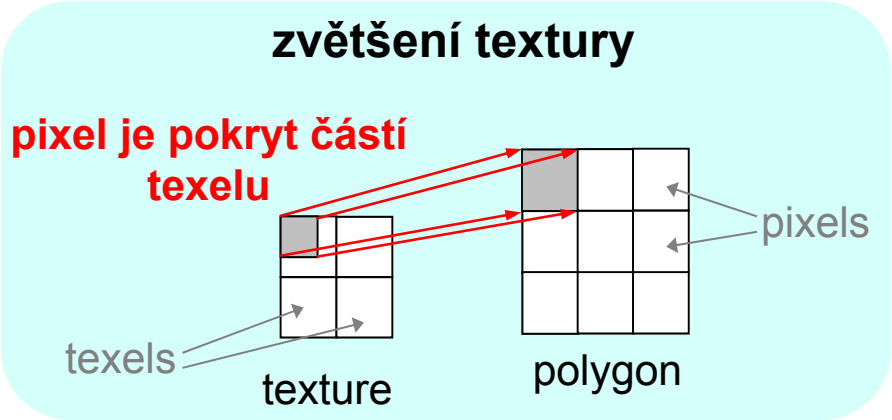
- **format** – složky zadaných texelů a jak jdou za sebou v poli texels
  GL_RGB, GL_BGR, GL_RGBA, GL_BGRA, GL_RED, GL_RG
- **type** – typ složek (v poli texels)
  GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, or GL_FLOAT a další formáty – jak jsou uloženy pixely v paměti na adrese danou ukazatelem *texels*
- **texels** – data s texturou a případně s hranicí

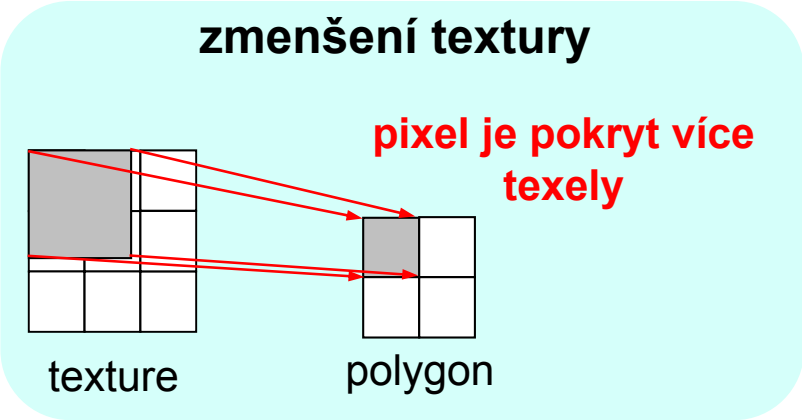
# Nastavení parametrů textury

- Filtrování textur (jak ji zvětšovat a zmenšovat)
- Wrap (co je okolo?)
- Detaily o obdélníkových texturách (texture rectangle)

# Filtrování textur

- čtvercové texely se mapují na čtvercové pixely a **málokdy se kryjí!!!**
- pixel na obrazovce je pokryt částí texelu => zvětšení (magnification)
  (Textura je „menší“ než objekt a musí se natáhnout)
- pixel je pokryt více texely (textura je „větší“) => zmenšení (minification)




**Problém: Jak vypočítat hodnotu texelu pro fragment?**

⇒ OpenGL poskytuje metody pro získání správných hodnot texelů - filtrování (kvalita versus rychlost)

⇒ filtr na zvětšování a na zmenšování samostatně

# Filtrování textur

**glTexParameteri(GL_TEXTURE_2D,** GLenum **param,** GLenum **filter);**

**param** je **GL_TEXTURE_MAG_FILTER** zvětšování
nebo **GL_TEXTURE_MIN_FILTER** zmenšování

NUTNÉ !!!

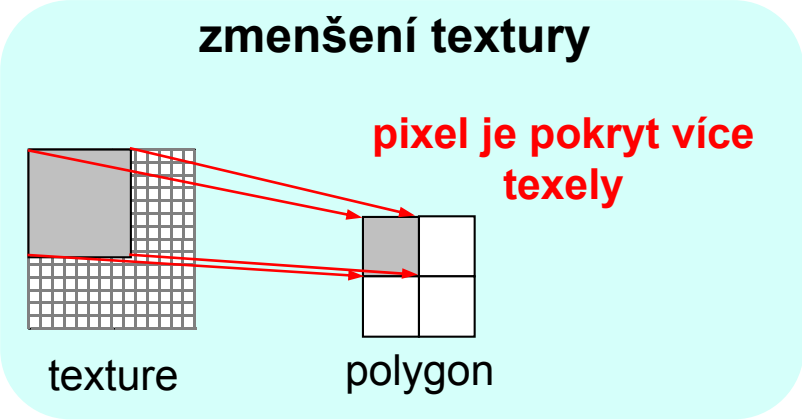
**filtr GL_NEAREST**

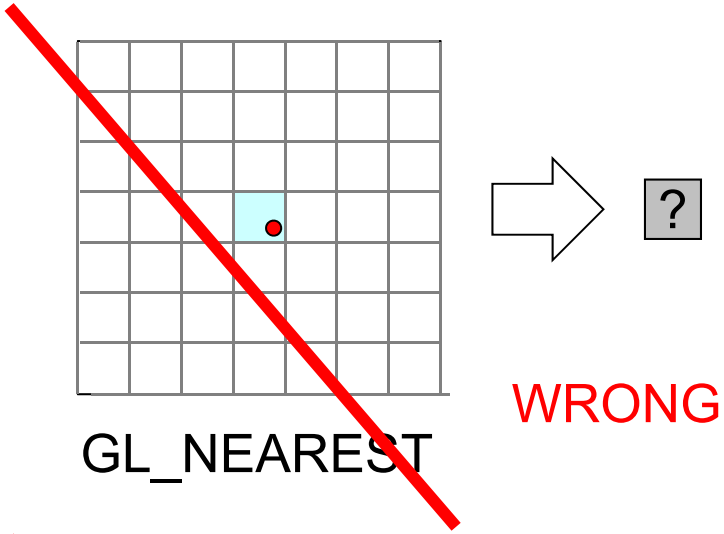
- texel se souřadnicemi nejblíže středu pixelu
- aliasing, rychlé

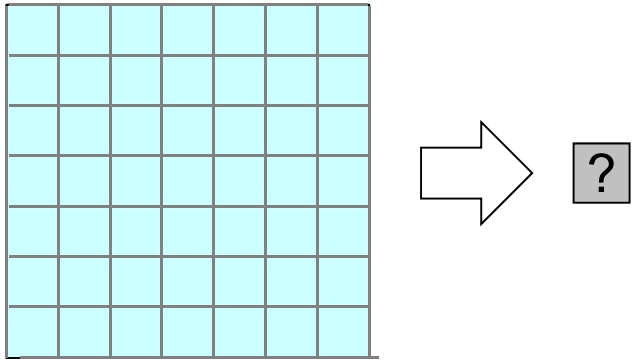
**GL_LINEAR**

- vážená lineární kombinace 2x2 texelů nejblíže středu pixelu
- hladké, ale pomalejší

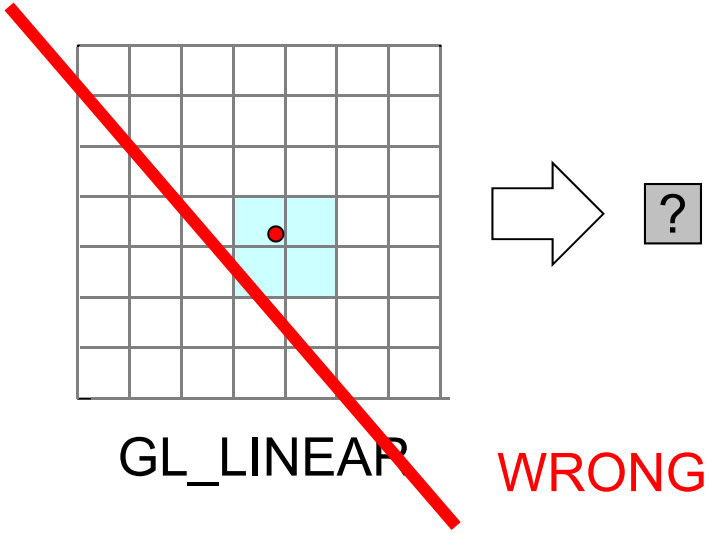
# Problém při zmenšení



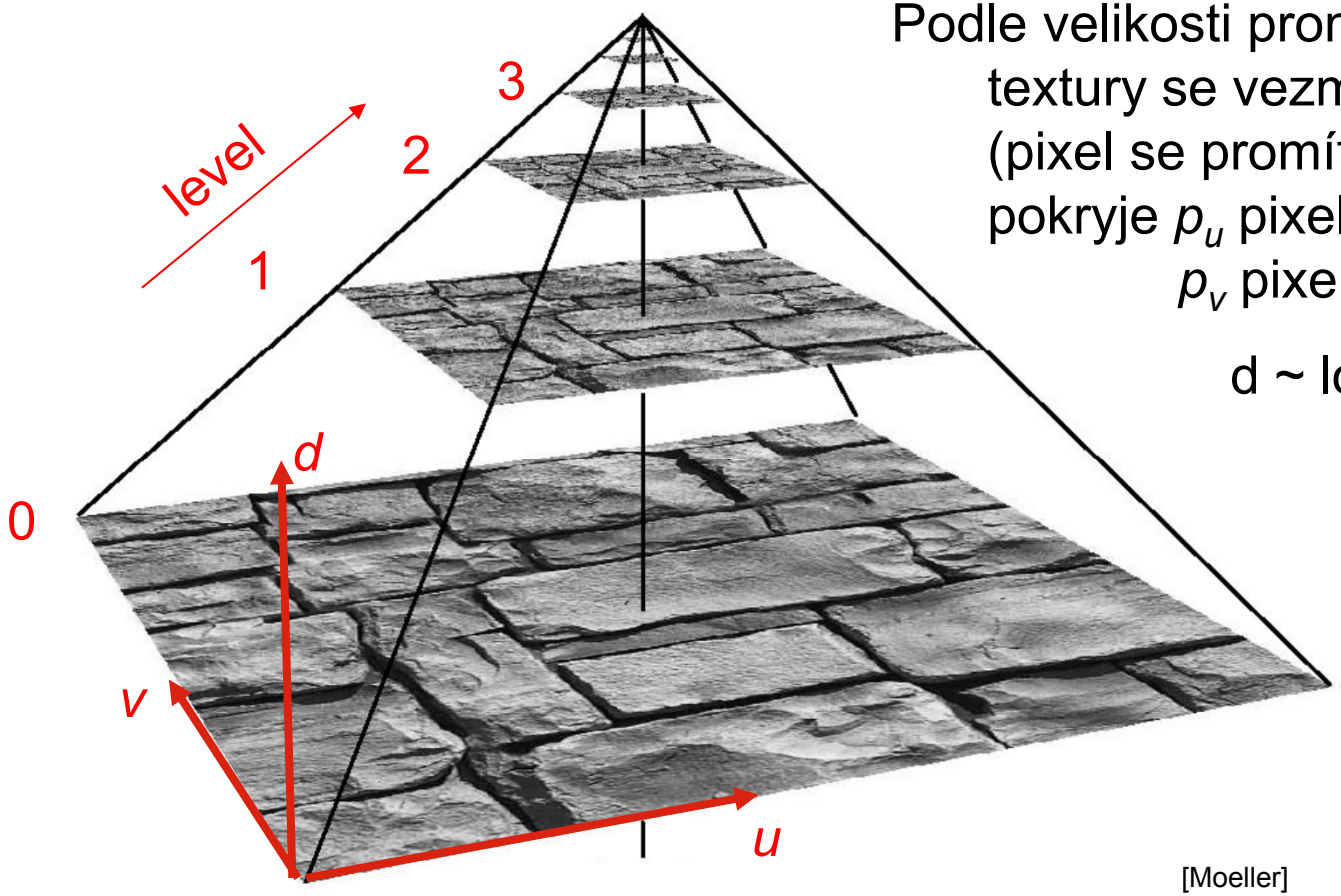




HOW to do it CORRECTLY ??????

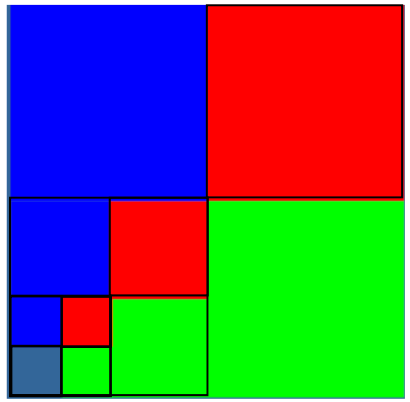


# Mipmapy – pyramida zmenšených textur




Podle velikosti promítnutého pixelu do textury se vezme zmenšený obrázek (pixel se promítne do úrovně 0, pokryje $p_u$ pixelů ve směru $u$ a $p_v$ pixelů ve směru $v$)

$$d \sim \log_2(\max(p_u, p_v))$$



[Moeller]

Jen o 1/3 více paměti

GL_....._MIPMAP_....

# Filtrování textur a mipmapy

Nanesená textura zrní a poblikává při animaci

– v rámci jedné úrovně (viz předchozí slide)

– mezi úrovněmi mipmapy (přepínání úrovní při zmenšování)

- **GL_NEAREST_MIPMAP_NEAREST** – zvolí mipmapu nejbližší velikosti pixelů a v ní hledá metodou GL_NEAREST
- **GL_LINEAR_MIPMAP_NEAREST** - zvolí mipmapu nejbližší velikosti pixelů a v ní hledá metodou GL_LINEAR
- **GL_NEAREST_MIPMAP_LINEAR** - zvolí dvě mipmapy nejbližší velikosti pixelů, v obou najde metodou GL_NEAREST hodnotu a tyto hodnoty váženě interpoluje, výsledek je hodnotou textury
- **GL_LINEAR_MIPMAP_LINEAR** - zvolí dvě mipmapy nejbližší velikosti pixelů, v obou najde váženou lineární interpolací (GL_LINEAR) hodnotu v rámci mipmap. Tyto hodnoty váženě interpoluje, výsledek je hodnotou textury (trilineární interpolace)

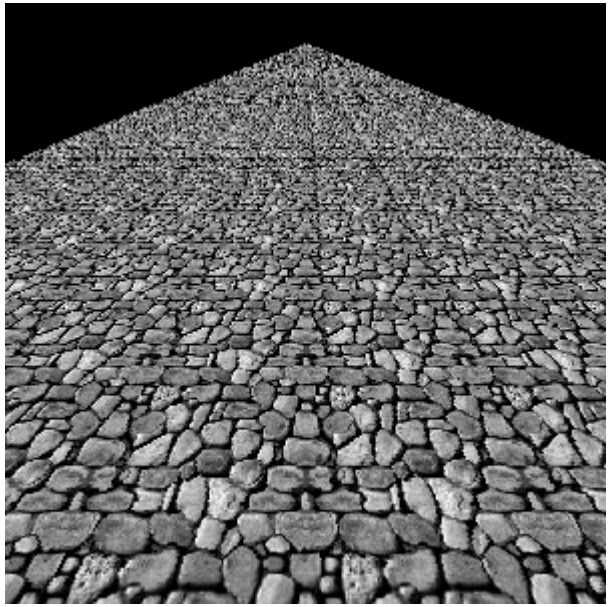
# Filtrování textur a mipmapy

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D, **GL_TEXTURE_MIN_FILTER**, G**L_LINEAR_MIPMAP_LINEAR**);

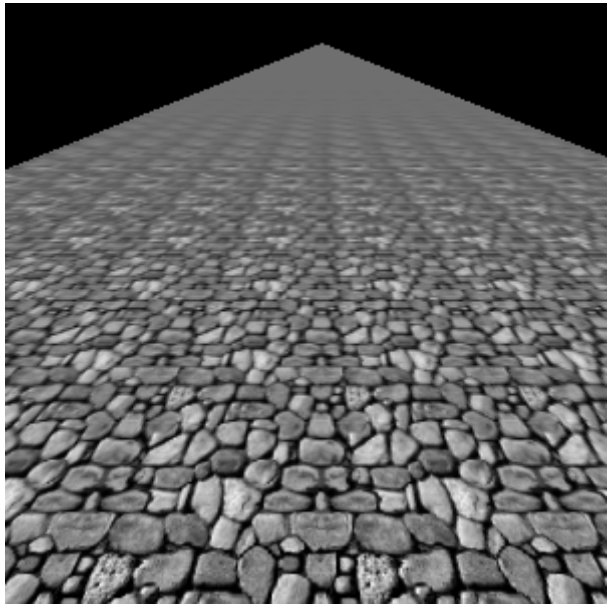
mipmaps.cpp

**Bez mipmapy**

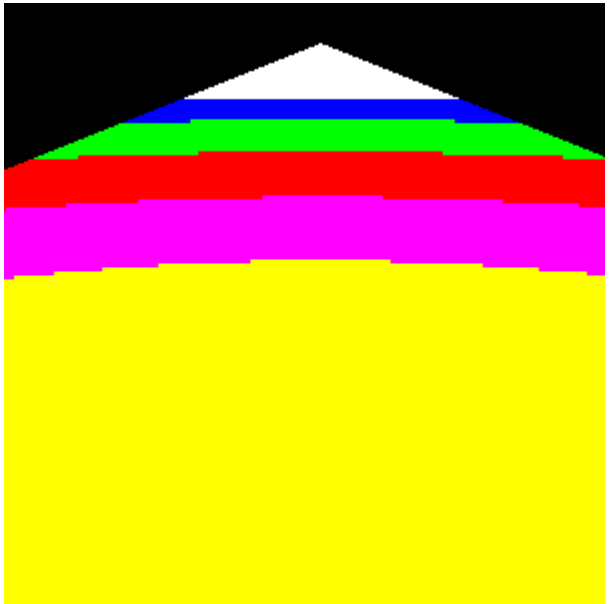


**GL_LINEAR**

**Použití mipmapy, trilineární interpolace**

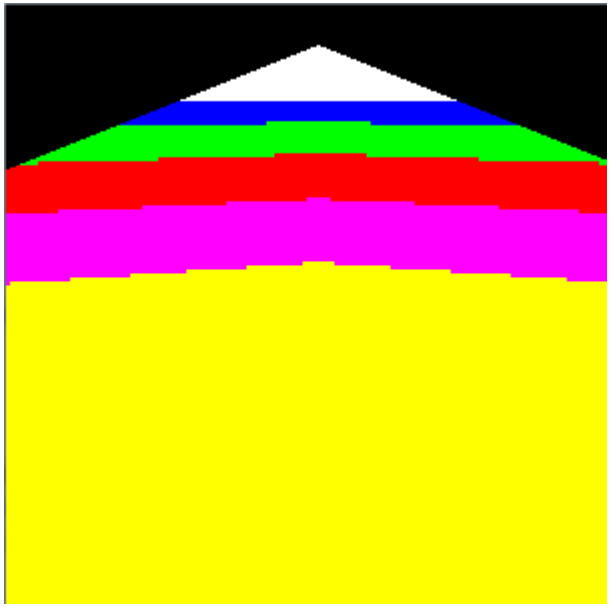


**GL_LINEAR_MIPMAP_LINEAR**

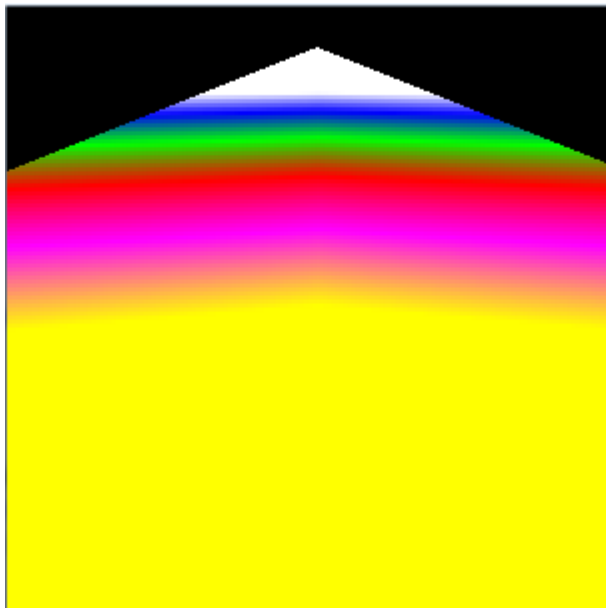


**místa přepnutí mipmap**

# Rozdíl mezi MIPMAP_NEAREST a LINEAR



GL_LINEAR_MIPMAP_NEAREST



GL_LINEAR_MIPMAP_LINEAR

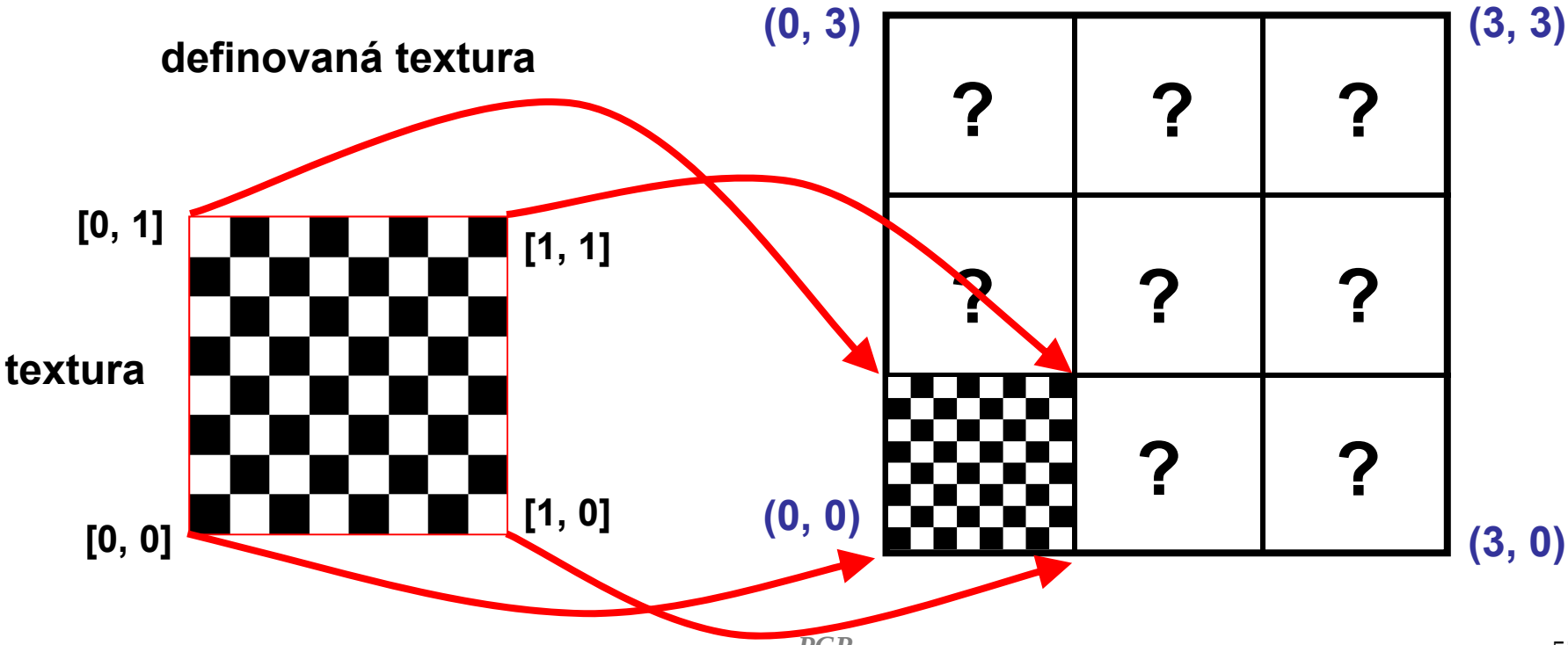
# Opakování a omezení textury

Jak na souřadnice
mimo interval [0..1]?

clampRepeat

texturovací souřadnice
přiřazené vrcholům
polygonu

definovaná textura



# Opakování a omezení textury

- textura je definována v rozsahu souřadnic [0.0, 1.0]
- zadat ale můžeme i souřadnice mimo interval [0.0, 1.0]
  - **textura se opakuje** (**REPEAT, MIRRORED_REPEAT**) – adresuje desetinnou částí texturovacích souřadnic (pozor, aby okraje textury navazovaly - levý na pravý, horní na dolní)
  - **textura je omezena na danou velikost** (**CLAMP**) – hodnoty texturovacích souřadnic mimo interval [0.0, 1.0] nahrazeny
    - Barvou – GL_CLAMP_TO_BORDER, GL_TEXTURE_BORDER_COLOR
    - Okrajovými pixely – GL_CLAMP_TO_EDGE

**glTexParameter{if}{v}(**GLenum ***target*****,** GLenum ***pname*****,** TYPE ***param*****);**

- *target* je **GL_TEXTURE_nD**, **GL_TEXTURE_CUBE_MAP,…**
- *pname* určuje, která texturová souřadnice se bude definovat **GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T** nebo …**_R**
- *param* je hodnota či ukazatel na pole hodnot, s parametry

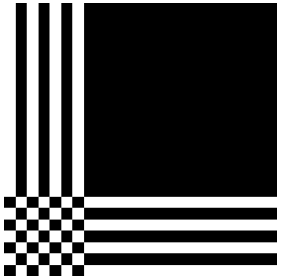
# Opakování a omezení textury

**Chování textury v místech, kde není definována (texturovací souřadnice S a T) ?**

**GL_TEXTURE_WRAP_S a GL_TEXTURE_WRAP_T, param:**

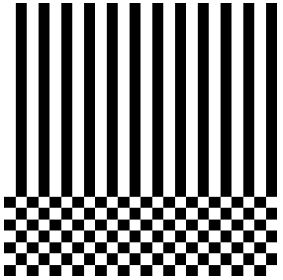
S: GL_CLAMP_TO_EDGE

T: GL_CLAMP_TO_EDGE



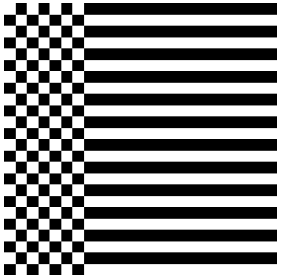
S: GL_REPEAT

T: GL_CLAMP_TO_EDGE



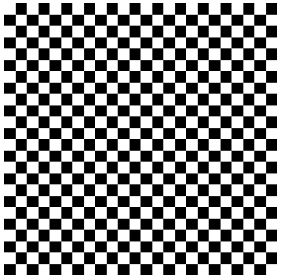
S: GL_CLAMP_TO_EDGE

T: GL_REPEAT



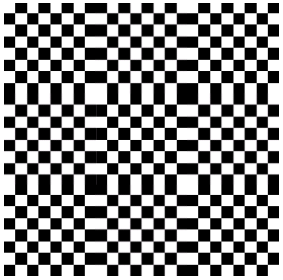
S: GL_REPEAT

T: GL_REPEAT



GL_MIRORRED_REPEAT

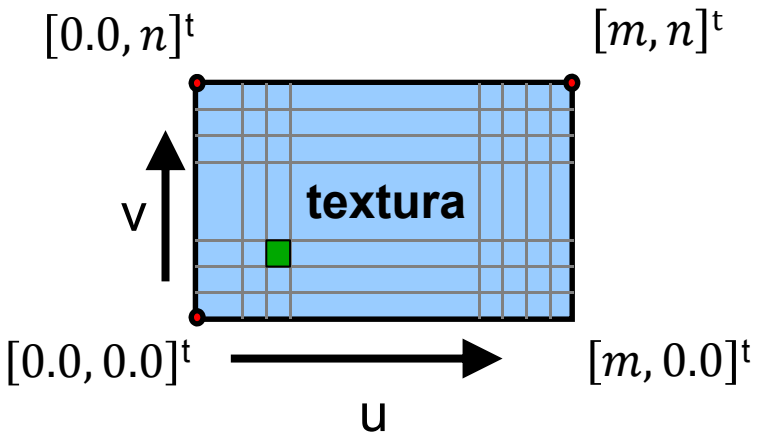
GL_MIRORRED_REPEAT



# Obdélníkové textury (rectangle textures)

- Speciální 2D textury typu GL_TEXTURE_RECTANGLE
  Rozměr $m \times n$
- Souřadnice nenormalizovány
- Umožňují adresovat přímo v souřadnicích pixelů, včetně lineární interpolace
  (9.5 je pozice mezi texelem 9 a 10)
- Omezení
  - Nemají mip-mapy
  - Filtry jen GL_NEAR, GL_LINEAR
  - Wrap jen GL_CLAMP_TO_EDGE a GL_CLAMP_TO_BORDER (barva).

**Souřad. v prostoru textury**
$\langle 0.0 \,..\, m \rangle \times \langle 0.0 \,..\, n \rangle$




Souřadnice: $0.0 \,..\, velikost$
**GL_TEXTURE_RECTANGLE**
*g*sampler2DRect

- texelFetch+GL_TEXTURE_2D jen celočíselné souřadnice

# Textury - osnova

- Co jsou textury a proč se používají?
- Nanášení textur – mapování a transformace souřadnic
- Kroky při definici textur v OpenGL
  - definice texturovacího objektu
  - předání obrázku textury
  - nastavení parametrů (zvětšení, zmenšení a wrap)
- Nanášení textur ve fragment shaderu
  - předání čísla texturovací jednotky shaderu
  - kombinace textury a osvětlení
  - více textur přes sebe (multitexturing)
- Generování texturovacích souřadnic
  - mapování okolního prostředí

# Předání čísla texturovací jednotky do FS

```
// OpenGL
GLuint brickTex = pgr::createTexture("textures/brick.jpg");      // load textures by PGR framework
glActiveTexture(GL_TEXTURE0);                                    // select texture unit 0
glBindTexture(GL_TEXTURE_2D, brickTex);                          // and bind texture object to it

// get location of the uniform (fragment) shader attributes
GLint brickTexLoc = glGetUniformLocation(cubeShaderProgram, "brickTex");
glUseProgram(cubeShaderProgram);


glUniform1i(brickTexLoc, 0);        // info for GLSL – which texture unit
                                    // is brick texture bound to – 0

// draw the textured object
glDrawElements(…);
```

# Použití textury v shaderu (FS)

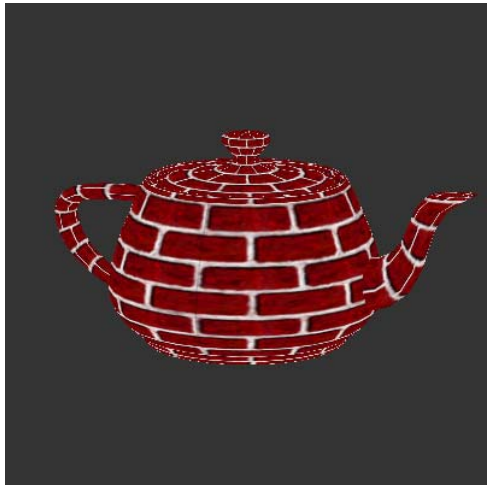
```
// fragment shader
#version 140
uniform sampler2D brickTex;  // brick texture sampler - texture unit 0

in vec2 texCoords_v;        // texture coordinates
out vec4 color_f;           // output fragment color
…
void main() {
  vec4 texColor = texture(brickTex, texCoords_v);   // sample textures

// compute lighting → color (vec3)
  …

  // compute final fragment color
  color_f = mix(texColor, vec4(color, 1.0), 0.65f);
}
```

# Kombinace textury a barvy povrchu

Textury s osvětlením simulují osvětlený materiál

Pro každý fragment se RGB barva fragmentu zkombinuje s texturou do výsledné barvy

**RGB textura**



**osvět-lený čajník (podklad)**

**Nějaká kombinující funkce**

# Kombinace textury a barvy povrchu

- Každý objekt má barvu (zadanou jako atribut vrcholů, nebo vypočítanou pomocí osvětlovacího modelu) a lze mu přiřadit textury
- Pro každý fragment se barva fragmentu kombinuje s barvou textury do výsledné barvy fragmentu
- 2 základní způsoby jak se kombinuje barva objektu s texturou:
  - textura nahradí barvu fragmentu (“decal” nebo “replace”),
  - textura moduluje (násobí) barvu fragmentu (“modulate”)
- Míchání se programuje ve fragment shaderu, lze tedy implementovat i další způsoby

# Odbočka: barvy – ALFA KANÁL

- Barva v aditivním modelu jsou vyjádřené pomocí 3 složek RGB
- Přidáváme čtvrtou složku, které říkáme parametr alfa
- Rozsah hodnot alfa je 0 až 1
- Alfa odpovídá neprůhlednosti fragmentu, který odpovídá popředí:
  - alfa = 1.0 ... Pixel je zcela neprůhledný
  - alfa = 0.0 ... Pixel je zcela transparentní (t.j. průhledný)
  - alfa = 0.3 … pixel je ze 70% transparentní a ze 30% neprůhledný
- Míchání barev pixelu pomocí alfa kanálu je dáno rovnící:

  Barva = alfa * barva(popředí) + (1-alfa) * barva(pozadí)
- Lze ji použít pouze v aditivním modelu barev RGB ! (pro CMYK by se musela redefinovat, pro HLS, HSV, YCrCb atd. nemá smysl)
- Viz ještě další přednáška a hesla “alpha blending, alpha compositing, RGBA color model”

## Smysluplné kombinace textury a barvy povrchu

| Složky textury | REPLACE | MODULATE | DECAL |
|---|---|---|---|
| GL_ALPHA | $C = C_F$<br>$A = A_T$ | $C = C_F$<br>$A = A_F A_T$ | nedefinováno |
| GL_RGB | $\mathbf{C = C_T}$<br>$A = \mathbf{A_F}$ | $\mathbf{C = C_F C_T}$<br>$A = A_F$ | $C = C_T$<br>$A = A_F$ |
| GL_RGBA | $\mathbf{C = C_T}$<br>$A = \mathbf{A_T}$ | $\mathbf{C = C_F C_T}$<br>$A = A_F A_T$ | $\mathbf{C = C_F(1-A_T)+C_T A_T}$<br>$A=A_F$ |

Index T označuje hodnotu textury (která se „lepí“ na podklad),

F hodnotu fragmentu (podklad),

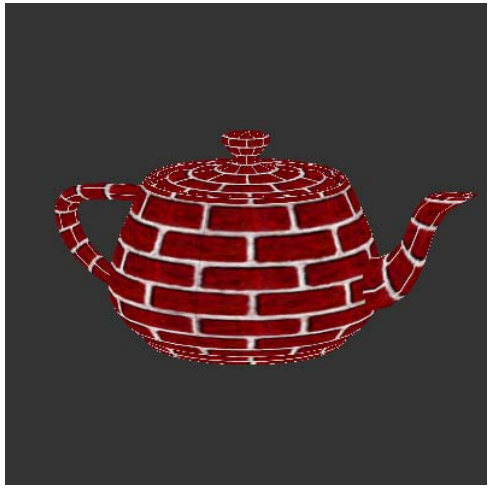
písmeno A, C bez indexu znamená výslednou vypočítanou hodnotu Alfa a Color.

Zkratky: A = Alpha (0,0,0,A) a C = barva (R,G,B,1)

# Kombinace textury a barvy povrchu

- barva fragmentu se **vynásobí** příslušnou barvou v textuře
  $\Rightarrow$ modulace se používá při osvětlování

**RGB textura**



**osvět-lený čajník**

**modulace**

**textura modulovaná barvou**

# Kombinace textury a barvy povrchu

**NÁZEV**

Podklad pro nanášení textury na dalších stránkách

TEXTURA

**textura s alfa kanálem (alfa = 0 v okolí, jinak má nenulovou hodnotu)**

**osvětlený čajník**

**neosvětlený čajník (nastavena bílá barva)**

# Kombinace textury a barvy povrchu

**MODULATE**

- **barva fragmentu se násobí barvou textury**
- **alfa fragmentu se násobí alfou textury**

  ⇒ používá se nejčastěji, s osvětlením dodává textura difúzní složku barvy

**texEnvModes…**

**textura s alfa kanálem (alfa = 0 v okolí, jinak má nenulovou hodnotu)**

**osvětlený čajník**

**neosvětlený čajník (nastavena bílá barva)**

# Kombinace textury a barvy povrchu

**REPLACE** **- neprůhledný obtisk, nálepka**

- **barva fragmentu se nahradí texturou**
  ⇒ používá se k potažení objektu neprůhledným objektem (neprůhledná nálepka na konzervě)
- **osvětlení se ignoruje** (nic se nemíchá, osvětlení fragmentu překryto)
- **neleskne se** (nikde)

**textura s alfa kanálem (alfa = 0 v okolí, jinak má nenulovou hodnotu)**

**osvětlený čajník (osvětlení překryto)**

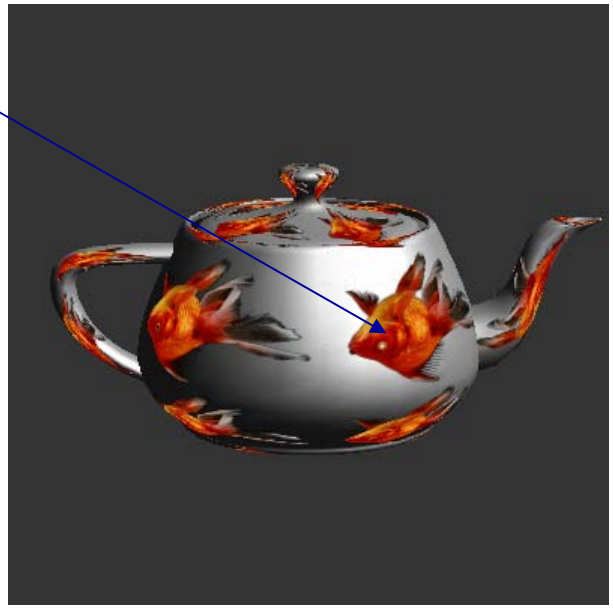
**neosvětlený čajník (barva překryta)**

# Kombinace textury a barvy povrchu

**DECAL** **- průhledný obtisk, nálepka**

- **barva fragmentu se smíchá s texturou** - poměr určuje **alfa textury**
  ⇒ používá se k potažení objektu texturou s kanálem alfa (např. znak na křídle letadla)
- osvětlení se projeví v místech, kde není textura (alpha = 0)
- **neleskne se**

**textura s alfa kanálem (alfa = 0 v okolí, jinak má nenulovou hodnotu)**



**osvětlený čajník**

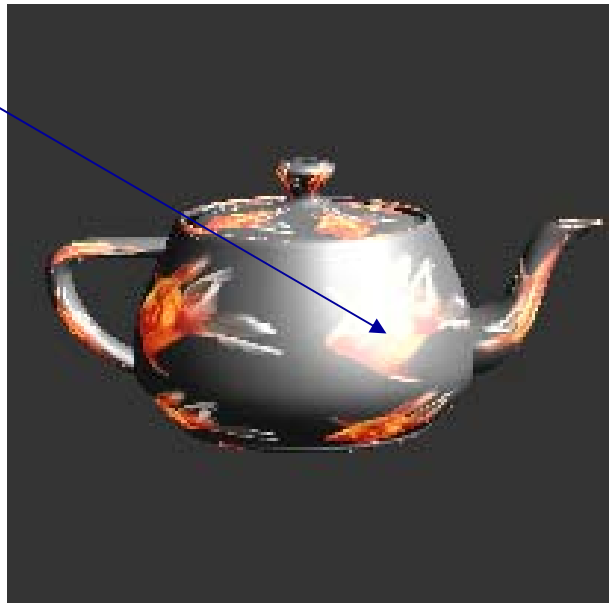
**neosvětlený čajník (nastavena bílá barva)**

# Kombinace textury a barvy povrchu

**ADD** - další možný způsob

- barva fragmentu se sečte s texturou
- Při přetečení jsou barvy „přepálené“

textura s alfa kanálem (alfa = 0 v okolí, jinak má nenulovou hodnotu)



osvětlený čajník

neosvětlený čajník (nastavena bílá barva)

# Příklad – strana aplikace (OpenGL)

```
GLuint texID;
glGenTextures(1, &texID);                    // generate texture object name

// bind texture object and set parameters related to the texture mapping & filtering
glBindTexture( GL_TEXTURE_2D, texID);
glActiveTexture(GL_TEXTURE0);   // texture unit
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
// set image to be used as a texture
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
     GL_UNSIGNED_BYTE, (const GLvoid *)texData);
glGenerateMipmap(GL_TEXTURE_2D);
// compile and link shaders
....
GLint samplerID = glGetUniformLocation(programID, "texSampler");

glUniform1i(samplerID, 0);   // texture unit number
```

# Příklad – vertex shader

```
#version 130

uniform mat4 VMmatrix;          // modelview transformation matrix
uniform mat4 Pmatrix;           // projection transformation matrix
in vec3 position;               // input vertex position
in vec4 color;                  // input vertex color
in vec2 texCoord;               // input vertex texture coordinates

smooth out vec4 color_v;                // vertex output color
smooth out vec2 texCoord_v;             // output vertex texture coordinates

void main()
{
   color_v = color;
   texCoord_v = texCoord;
   gl_Position = Pmatrix * VMmatrix * vec4(position, 1);
}
```

# Příklad – fragment shader verze A

**REPLACE**

```
#version 130

smooth in vec4 color_v;              // interpolated fragment color
smooth in vec2 texCoord_v;           // interpolated fragment texture coordinates
out vec4 outputColor_f;              // fragment final color

uniform sampler2D texSampler;        // texture sampler (texture unit)


void main()
{
  // REPLACE way to combine object color with texture
  outputColor_f = texture(texSampler, texCoord_v) ;
}
```

# Příklad – fragment shader verze B

**MODULATE**

```
#version 130

smooth in vec4 color_v;              // interpolated fragment color
smooth in vec2 texCoord_v;           // interpolated fragment texture coordinates
out vec4 outputColor_f;              // fragment final color

uniform sampler2D texSampler;        // texture sampler (texture unit)


void main()
{
  // MODULATE way to combine object color with texture
  outputColor_f  = color_v * texture(texSampler, texCoord_v) ;
}
```

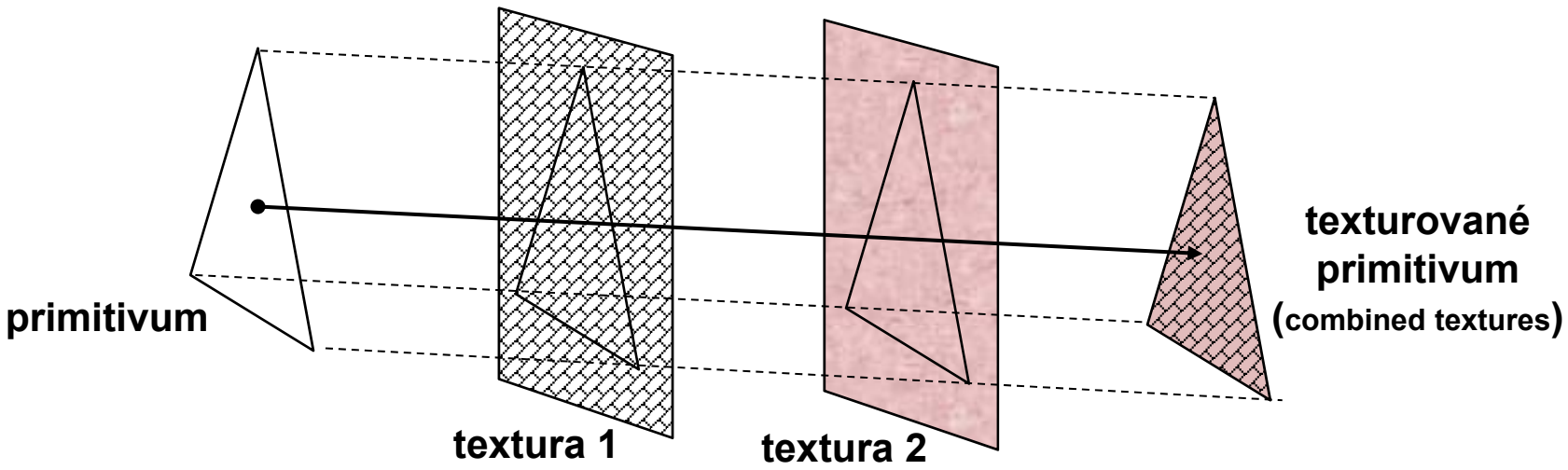
# Multitexturing

**Multitexturing** = upatnění dvou či více textur na jediný fragment během jednoho vykreslení

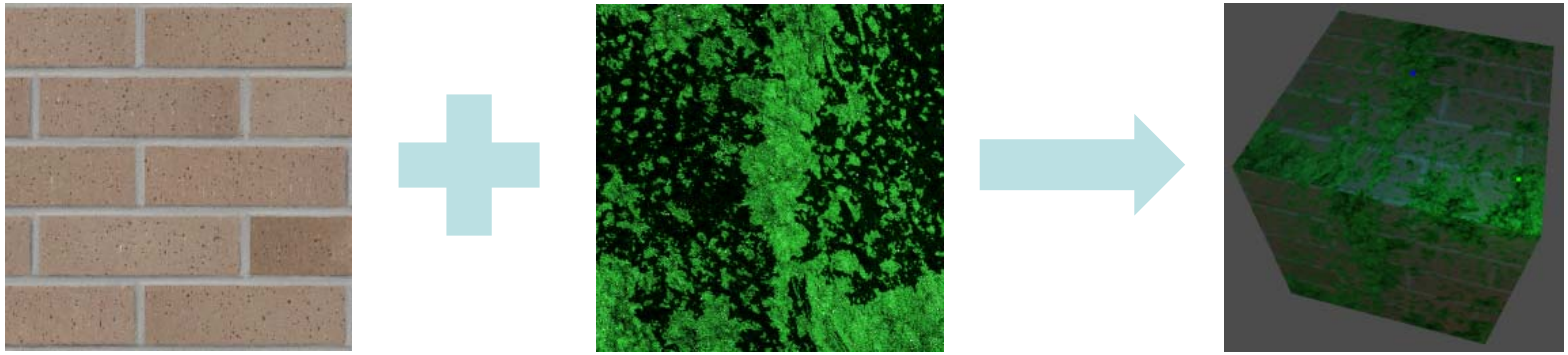
- sada **texturovacích jednotek** ⇒ každá se stará o jednu texturu

**glGetIntegerv(GL_MAX_TEXTURE_IMAGE_UNITS,** GLint ***numOfUnits);**

- Vrátí **numOfUnits** – maximální počet dostupných texturovacích jednotek (musí být aspoň 16)



# Multitexturing – příklad – OpenGL



```
GLuint brickTex = pgr::createTexture("textures/brick.jpg");          // load textures by PGR framework
GLuint mossTex = pgr::createTexture("textures/moss.png");

glActiveTexture(GL_TEXTURE0);                    // select texture unit 0
glBindTexture(GL_TEXTURE_2D, brickTex);
glActiveTexture(GL_TEXTURE1);                    // select texture unit 1
glBindTexture(GL_TEXTURE_2D, mossTex);
// get locations of the uniform fragment shader attributes
GLint mossTexLoc = glGetUniformLocation(cubeShaderProgram, "mossTex");
GLint brickTexLoc = glGetUniformLocation(cubeShaderProgram, "brickTex");

glUseProgram(cubeShaderProgram);

glUniform1i(brickTexLoc, 0);        // brick texture is bound to texture unit 0
glUniform1i(mossTexLoc, 1);         // moss texture is bound to texture unit 1
```

# Multitexturing – příklad – FS

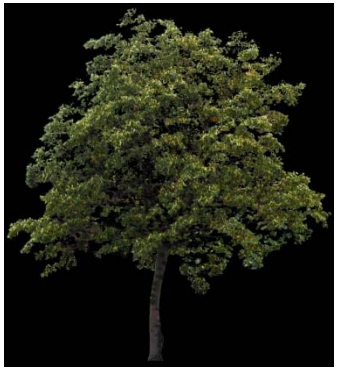
Computer Graphics Group

texture-multi

```
// fragment shader
#version 140
uniform sampler2D brickTex;  // brick texture sampler - texture unit 0
uniform sampler2D mossTex; // moss texture sampler – texture unit 1
in vec2 texCoords_v;        // texture coordinates
out vec4 color_f;           // output fragment color
…
void main() {
  vec4 brickTexColor = texture(brickTex, texCoords_v);   // sample textures
  vec4 mossTexColor = texture(mossTex, texCoords_v);
  // mix textures together – mixing ratio is given by moss texture alpha component
  vec4 texColor = mix(brickTexColor, mossTexColor, mossTexColor.a);
  // compute lighting → color (vec3)
  …
  // compute final fragment color
  color_f = mix(texColor, vec4(color, 1.0) * texColor, 0.65f);
}
```

# Zahazování fragmentů – alpha test

Computer Graphics Group



Černé pixely zahozeny (discard)

Složka alpha je v okolí stromu nulová (černé pixely)

```
#version 140
uniform sampler2D treeTex;          // texture sampler - texture unit 0
in vec2 texCoords_v;                // texture coordinates
…
void main() {
  // sample texture at a given texture coordinates
  vec4 texColor = texture(treeTex, texCoords_v);
  // discard the fragment from rendering if the alpha component is less than 0.5f
  if(texColor.a < 0.5f) discard;
  …
}
```


Texture-alpha

# Textury - osnova

Computer Graphics Group

- Co jsou textury a proč se používají?
- Nanášení textur – mapování a transformace souřadnic
- Kroky při definici textur v OpenGL
  - definice texturovacího objektu
  - předání obrázku textury
  - nastavení parametrů (zvětšení, zmenšení a wrap)
- Nanášení textur ve fragment shaderu
  - předání čísla texturovací jednotky shaderu
  - kombinace textury a osvětlení
  - více textur přes sebe (multitexturing)
- Generování texturovacích souřadnic
  - mapování okolního prostředí

# Mapování prostředí (*environment mapping*)

- cílem mapování prostředí (okolí objektu) je nakreslit objekt tak, jako by byl perfektně lesklý a odrážel barvy ze svého okolí. Směr odrazu paprsku jednoznačně určuje barvu pixelu
- okolí se zobrazí na povrch pomocného tělesa (koule či kostka) a uloží se do jedné či šesti textur – uloží se, co vidíme daným směrem
- zobrazovaný objekt se umístí do středu tohoto pomocného tělesa, odražený paprsek pak protne povrch tělesa v místě E, kde je uložen obraz okolí v daném směru (barva odrazu v bodě O)

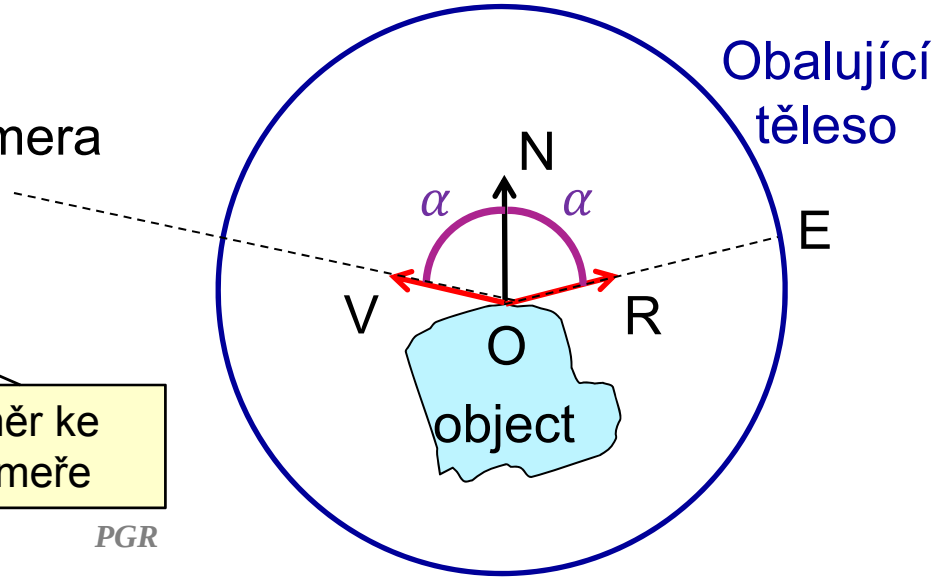
Směr odraženého paprsku

normálový vektor v bodě odrazu

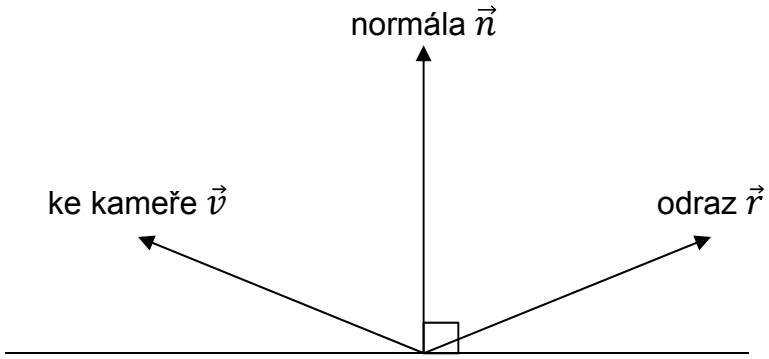
$$\vec{r} = 2\cos(\alpha)\,\vec{n} - \vec{v}$$

Úhel mezi normálou povrchu a směrem ke kameře, odkud dopadá pozorovací paprsek

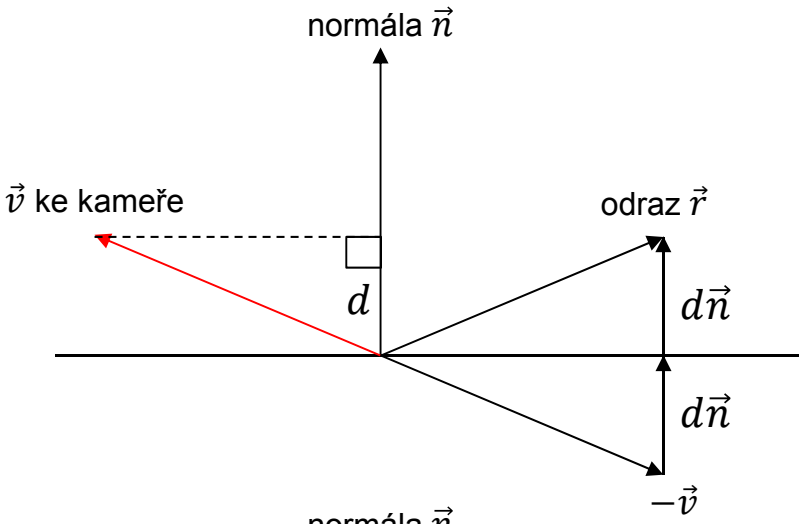
Směr ke kameře

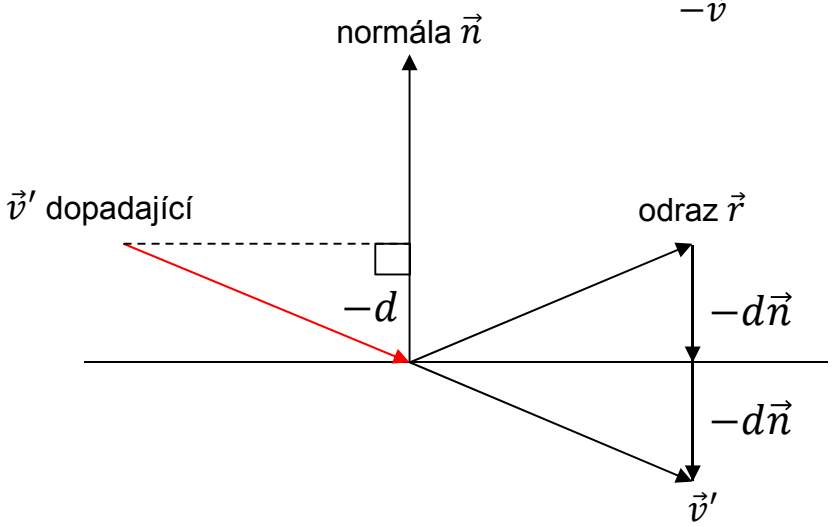


# Vzorec pro odražený vektor - odvození




$$\vec{r} = -\vec{v} + 2d\vec{n}$$
$$\vec{r} = -\vec{v} + 2\cos(\alpha)\,\vec{n}$$
$$\vec{r} = -\vec{v} + 2\,(\vec{v}.\vec{n}\,)\,\vec{n}$$
$$\vec{r} = -\vec{v} + 2\,\mathrm{dot}\,(\vec{v},\vec{n})\,\vec{n}$$



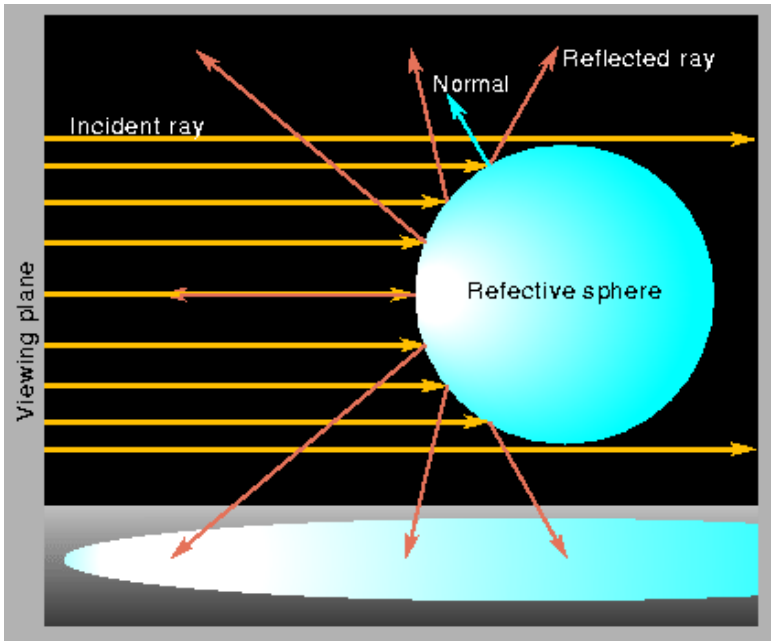




V GLSL je funkce *reflect*()
počítá s dopadajícím vektorem $\vec{v}'$

$$\vec{r} = \vec{v}' - 2\,\mathrm{dot}\,(\vec{v}',\vec{n})\,\vec{n}$$

# Sférické mapování prostředí

- uložení barvy pro všechny směry odrazu $\text{color} = f(\vec{r})$ v rovinné textuře odpovídá fotografii přesné vyleštěné koule umístěné do středu prostředí a vyfotografované z velké dálky fotoaparátem s velkou ohniskovou vzdáleností



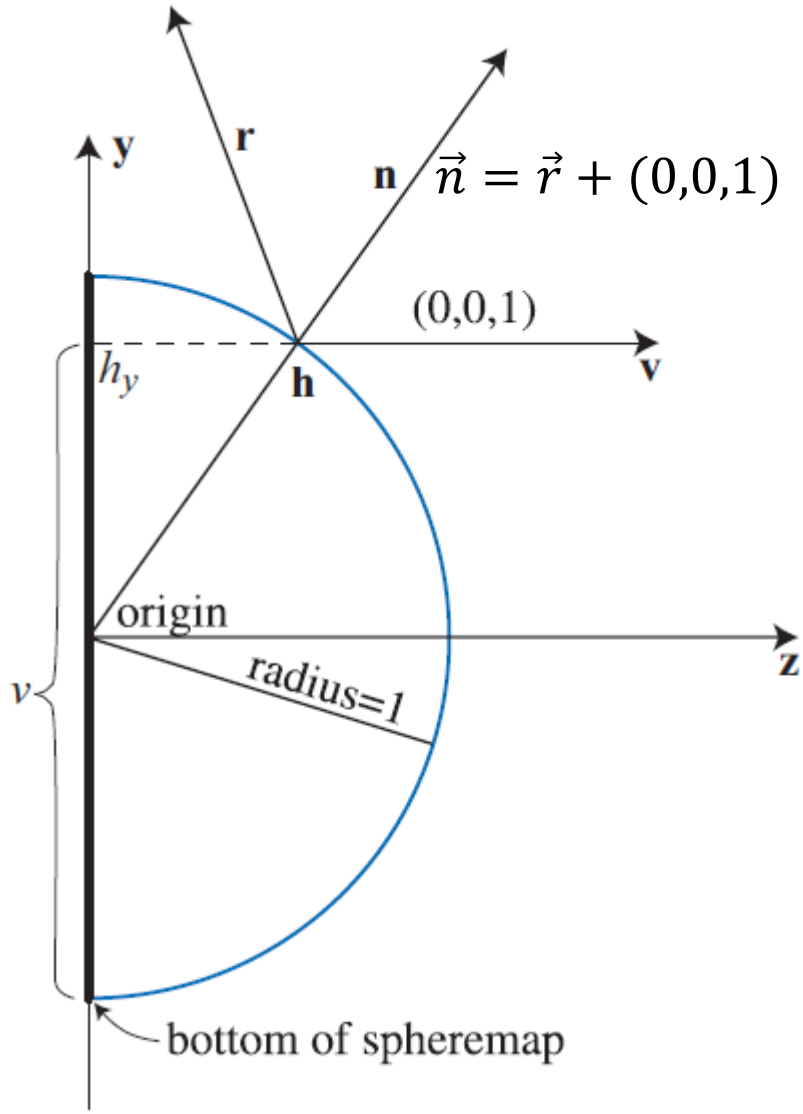

**textura pro environment mapping**

$$\vec{r} = 2\,\mathrm{dot}\,(\vec{v}, \vec{n})\,\vec{n} - \vec{v}$$ // odražený vektor

$$m = \sqrt{\vec{r}_x^{\,2} + \vec{r}_y^{\,2} + (\vec{r}_z + 1)^2}$$ // délka vektoru $\vec{n}$

souřadnice v textuře: $s = \frac{\vec{r}_x}{2m} + 0.5, \quad t = \frac{\vec{r}_y}{2m} + 0.5$ // převod do rozsahu 0..1

# Sférické mapování prostředí




- Textura střed do počátku, rozměr -1..1
- Pozorovatel: $z = +\infty \Rightarrow \mathbf{v} = [0 \quad 0 \quad 1]^t$
- Jednotková polokoule
- Odraz v bodě $\mathbf{h}$
  - Odražený paprsek $\mathbf{r} = 2\,\mathrm{dot}\,(\mathbf{v}, \mathbf{n})\,\mathbf{n} - \mathbf{v}$
  - Barva okolí ve směru $\mathbf{r}$ uložena do textury
  - Normála $\vec{n}$ na povrchu odpovídá radiusvektoru bodu $\mathbf{h}$
  - Projekce $\vec{n}$ resp. $\mathbf{h}$ do textury dá pozici, v níž je uložena hodnota ve směru $\mathbf{r}$
  - $\vec{n}$ je v ose $\vec{r}$ a $\vec{v}$, $\vec{n} = \frac{\vec{r} + \vec{v}}{\|\vec{r} + \vec{v}\|} = \frac{(r_x \quad r_y \quad (r_z+1))^t}{\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}}$
  - Pozici převedeme z $\langle -1 \,..\, 1 \rangle$ do $\langle 0 \,..\, 1 \rangle$

    $u = \frac{r_x}{2m} + \frac{1}{2} \qquad v = \frac{r_y}{2m} + \frac{1}{2}$

    $m = \sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$

# Sférické mapování prostředí

Mapa okolí

čajník s mapou okolí,
která se na něm zrcadlí


texture-sheremap

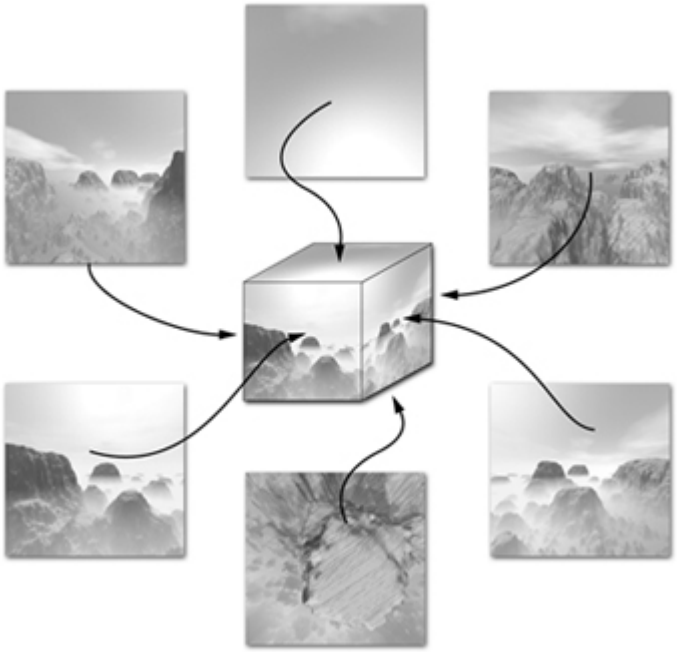
# Sférické mapování prostředí

- Výhody
  - jen 1 textura na celé okolí
  - lze použít i na starém HW
  - získá se fotografováním lesklé koule či dalších speciálních tvarů pomocí teleobjektivu

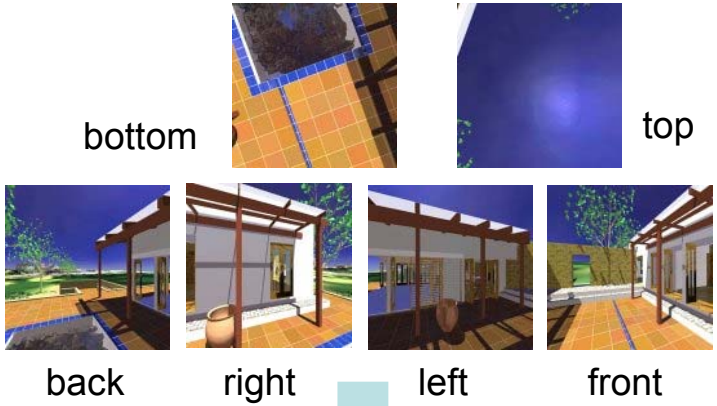
- Nevýhody
  - Pouze pro jeden směr pohledu pozorovatele a 1 pozici objektu ve scéně
  - není lineární - lineární interpolace zkresluje – více u okrajů
  - singularita na obvodové hraně (bod za koulí [0,0,-1])
  - Velká část textury nevyužita (rohy)

# Mapování prostředí na krychli (*cube map*)

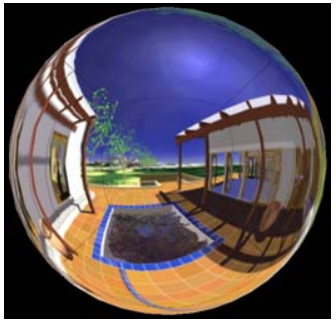
Texturovací objekt *cube map* tvoří **šest čtvercových 2D textur** - šest stěn krychle – každá stěna zobrazuje, co vidíme v jejím směru

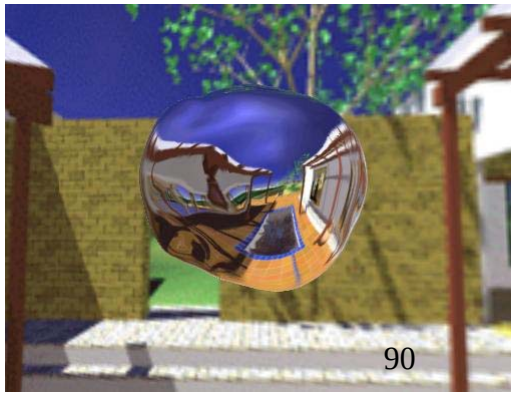


Textury pro *cube map*

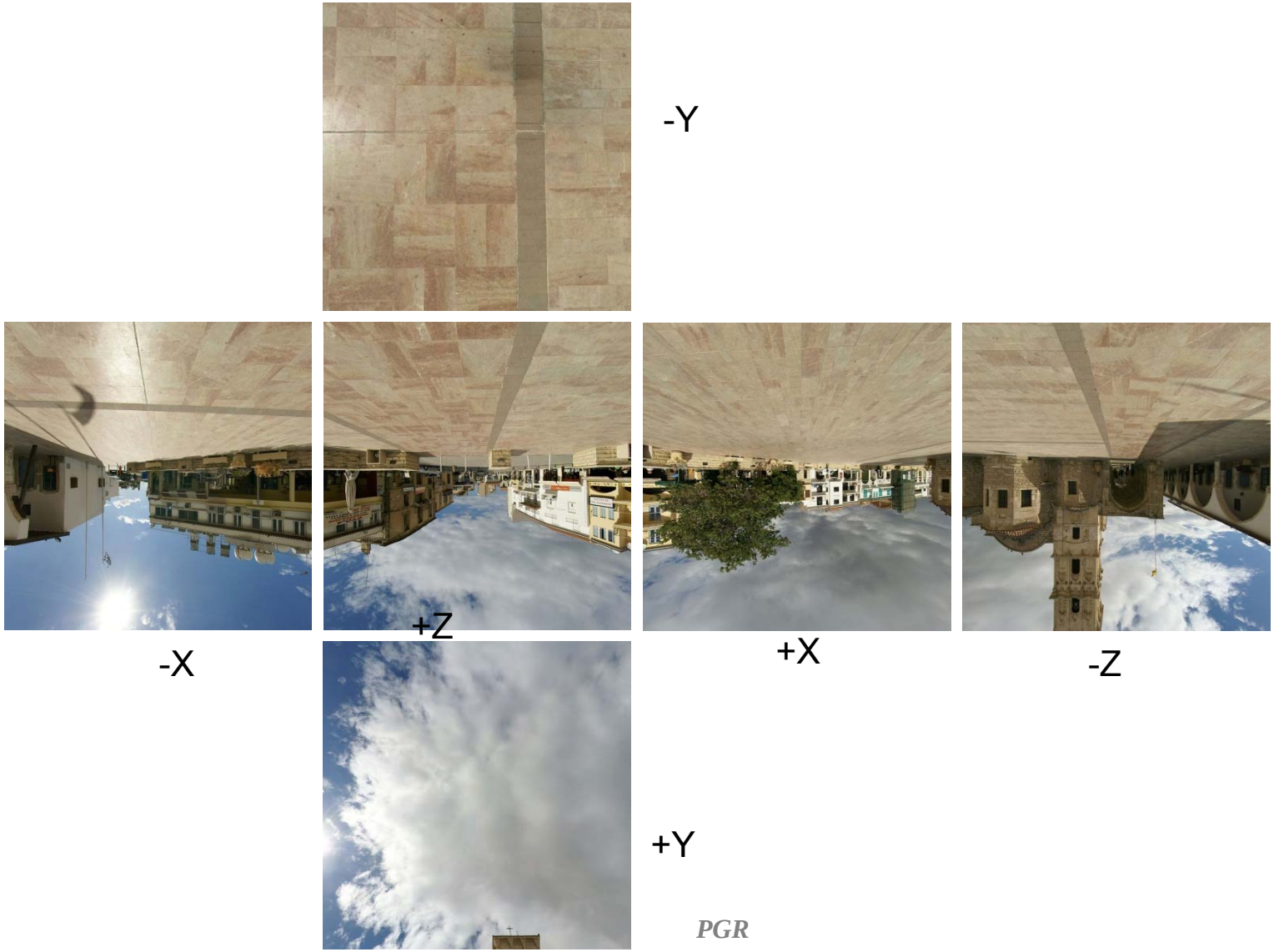



[nVidia]





# Mapování prostředí na krychli



# Mapování prostředí na krychli (příklad.)

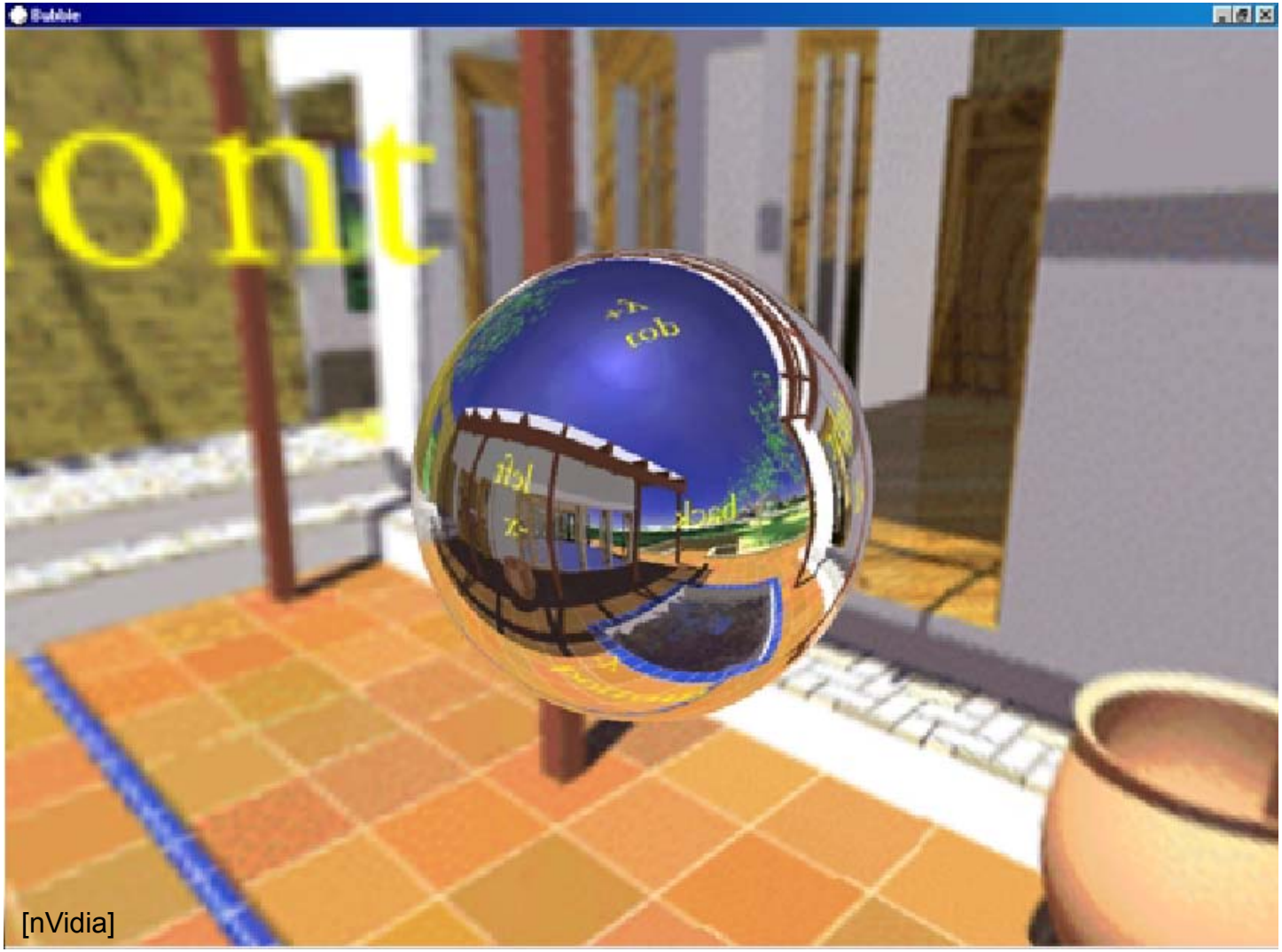
```
// cube-map faces definition
GLenum targets[6] = {
  GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z };

GLuint cubeMap;
glGenTextures(1, &cubeMap);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubeMap);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
unsigned char *texels_p  =  new char[cubeMapSize*cubeMapSize*4];
for (int i = 0; i < 6; i++)     {
  // naplň texels_p obrázkem stěny i
  glTexImage2D(
    targets[i],
    0,  GL_RGBA8,                 // internal format
    cubeMapSize,                  // width
    cubeMapSize,                  // height
    0,  GL_RGBA,                  // format
    GL_UNSIGNED_BYTE, texels_p);
}
```

# Mapování prostředí na krychli (příklad)

```
// fragment shader
noperspective in vec3 reflectDir;
uniform samplerCube cubeMapTex;
uniform float reflectFactor;
void main() {
 …
 // reflection direction reflectDir is used to sample the cube map
 // sampler determines which face has to be sampled and where (texture coordinates)
  vec4 cubeMapColor = texture(cubeMapTex, reflectDir);
  …
  color_f = mix( vec4(color, 1.0), cubeMapColor, reflectFactor);
}
```


textue-cubemap

# Mapování na krychli – Cube Mapping




[nVidia]

# Mapování prostředí na krychli

**Určení stěny a souřadnice texelu v ní**

Souřadnice $(s, t, r)$ chápeme jako směrový vektor ze středu krychle

1. Největší složka (hlavní osa) vektoru určí stěnu krychle (např. pro vektor $(s, t, r) = (-1, \quad -8, \quad 5)$ to je stěna ve směru záporné osy T, proto -T)
2. Zbylé dvě souřadnice určí texel uvnitř stěny (vydělíme je |–T|, zde tedy číslem 8 )

   S’ = [ S / |-T| + 1] / 2

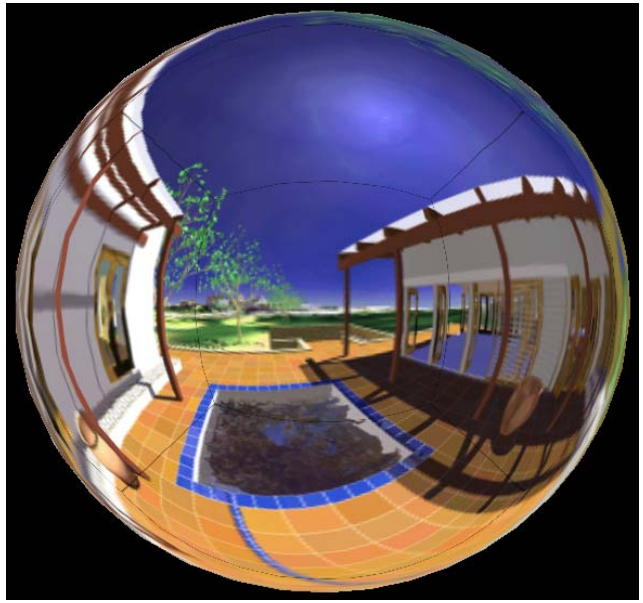
   R’ = [ R / |-T| + 1] / 2

   Tedy pro vektor $(-1, \quad -8, \quad 5)$

   Bude souřadnice v rámci stěny

   S´ = [-1 / 8 + 1] / 2 = 7/16

   R´ = [-5 / 8 + 1] / 2 = 3/16 …. –rz see next slide



# Determine cube side and texel coordinates

Cubemap texture coordinate $(s, t, r)$ is taken as direction vector $(rx,\ ry,\ rz)$ emanating from the center of the cube

1. Determine the major axis direction (largest magnitude coordinate)

   Ex: $(s, t, r) = (rx,\ ry,\ rz) = (-1,\ -8,\ 5) \Rightarrow$ direction is -ry

| direction | target | sc | tc | ma |
|---|---|---|---|---|
| +rx | GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT | -rz | -ry | rx |
| -rx | GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT | +rz | -ry | rx |
| +ry | GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT | +rx | +rz | ry |
| -ry | GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT | +rx | -rz | ry |
| +rz | GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT | +rx | -ry | rz |
| -rz | GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT | -rx | -ry | rz |

2. Calculate updated (s,t)

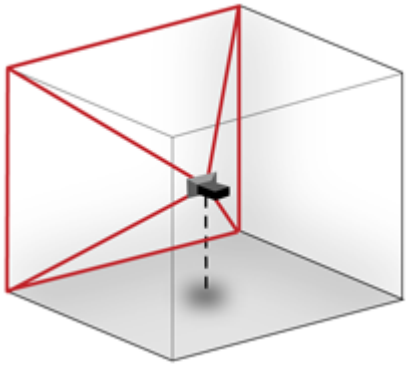
$$s = \left(\frac{sc}{|ma|} + 1\right)/2 \qquad s = \left(\frac{-1}{8} + 1\right)/2 = \frac{7}{16}$$

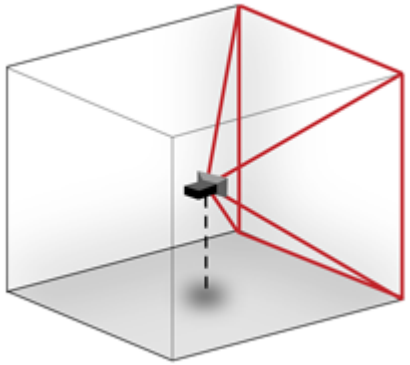
$$t = \left(\frac{tc}{|ma|} + 1\right)/2 \qquad t = \left(\frac{-5}{8} + 1\right)/2 = \frac{3}{16}$$

# Postup vytvoření cube maps ve scéně

- Nastavit kameru na úhel FOV 90°
- umístit kameru do místa objektu

- zamířit do směru +X a zobrazit scénu
- zamířit do směru –X a zobrazit scénu
- zamířit do směru +Y a zobrazit scénu
- zamířit do směru –Y a zobrazit scénu
- zamířit do směru +Z a zobrazit scénu
- zamířit do směru –Z a zobrazit scénu





(scénu lze přitom generovat s nižší přesností)

Osy X a Y musí otočit obrázek o 180º (flip s, t)

[Steven Wittens, http://acko.net/files/making-worlds/planet-2-cubemap-rendering.png]

# Mapování na krychli - shrnutí

- Výhody
  - je pohledově nezávislá
  - získá se vyfocením šesti snímků či zobrazením šesti pohledů

- Nevýhody
  - Pouze pro jednu polohu pozorovatele (lze ale generovat )
  - 6x větší spotřeba paměti
  - nutná podpora HW
    - současný přístup do 6 textur, interpolace na hranách

# Odkazy

[MPG2004]

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

[AM2007]

T. Akenine Moeller, E. Haines: Realtime Rendering (3rd ed), A. K. Peters, 2007, ISBN 1-56881-182-9

[TT97] R.Wolfe*Teaching Texture Mapping Visually,* SiggraphCourse, Computer Graphics, 1997.

http://public.beuth-hochschule.de/~godberse/wiese/cga1/07_3Dlocal/sw/texturen/texture.html

Odkazy na textury a mipmapy

- http://arcsynthesis.org/gltut/Texturing/Tutorial%2014.html
- http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-2.2:-Shaders.html
- http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-2.1:-Buffers-and-Textures.html
- http://www.opengl.org/wiki/Common_Mistakes#Automatic_mipmap_generation

# Typy textur - targets

- GL_TEXTURE_1D: 1-rozměrná. Má pouze šířku width.
- **GL_TEXTURE_2D**: 2-rozměrná Má width a height
- GL_TEXTURE_3D: 3-rozměrná Má width, height a depth.
- **GL_TEXTURE_RECTANGLE**: 2-rozměrný obrázek, bez mip-mappingu
  Souřadnice nenormalizovány – 0.. size
- GL_TEXTURE_BUFFER: 1-rozměrné pole, bez mip-mappingu.
  Obrázek uložený v buffer objektu.
- **GL_TEXTURE_CUBE_MAP**: 6 stejně velkých2D obrázků, 6 stěn krychle.
- GL_TEXTURE_1D_ARRAY: pole 1-rozměrných obrázků
- GL_TEXTURE_2D_ARRAY: pole 2-rozměrných obrázků.
- GL_TEXTURE_CUBE_MAP_ARRAY: pole šestic obrázků – cube map.
- GL_TEXTURE_2D_MULTISAMPLE: 2-rozměrný obrázek, bez mip-mappingu.
  V každém pixelu je uloženo několik vzorků
- GL_TEXTURE_2D_MULTISAMPLE_ARRAY: Pole 2-rozměrných obrázků
  typu multisample. Bez mipmappingu.
  (Generují se rederováním do textur)

# Seznam dvojic target - sampler

- gsampler1D: GL_TEXTURE_1D
- **gsampler2D**: **GL_TEXTURE_2D**
- gsampler3D: GL_TEXTURE_3D
- **gsamplerCube**: **GL_TEXTURE_CUBE_MAP**
- **gsampler2DRect:** **GL_TEXTURE_RECTANGLE !!!**
- gsampler1DArray: GL_TEXTURE_1D_ARRAY
- gsampler2DArray: GL_TEXTURE_2D_ARRAY
- gsamplerCubeArray: GL_TEXTURE_CUBE_MAP_ARRAY (vyžaduje GL 4.0 nebo ARB_texture_cube_map_array)
- gsamplerBuffer: GL_TEXTURE_BUFFER
- gsampler2DMS: GL_TEXTURE_2D_MULTISAMPLE
- gsampler2DMSArray: GL_TEXTURE_2D_MULTISAMPLE_ARRAY

g označuje (nic - float, i - signed integer, a u - unsigned integer).