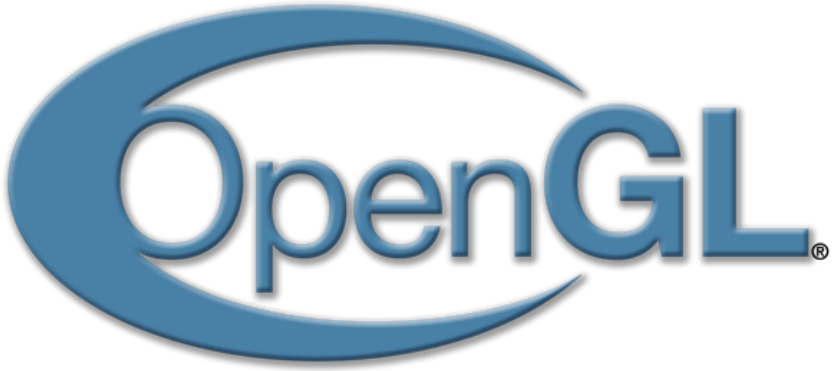
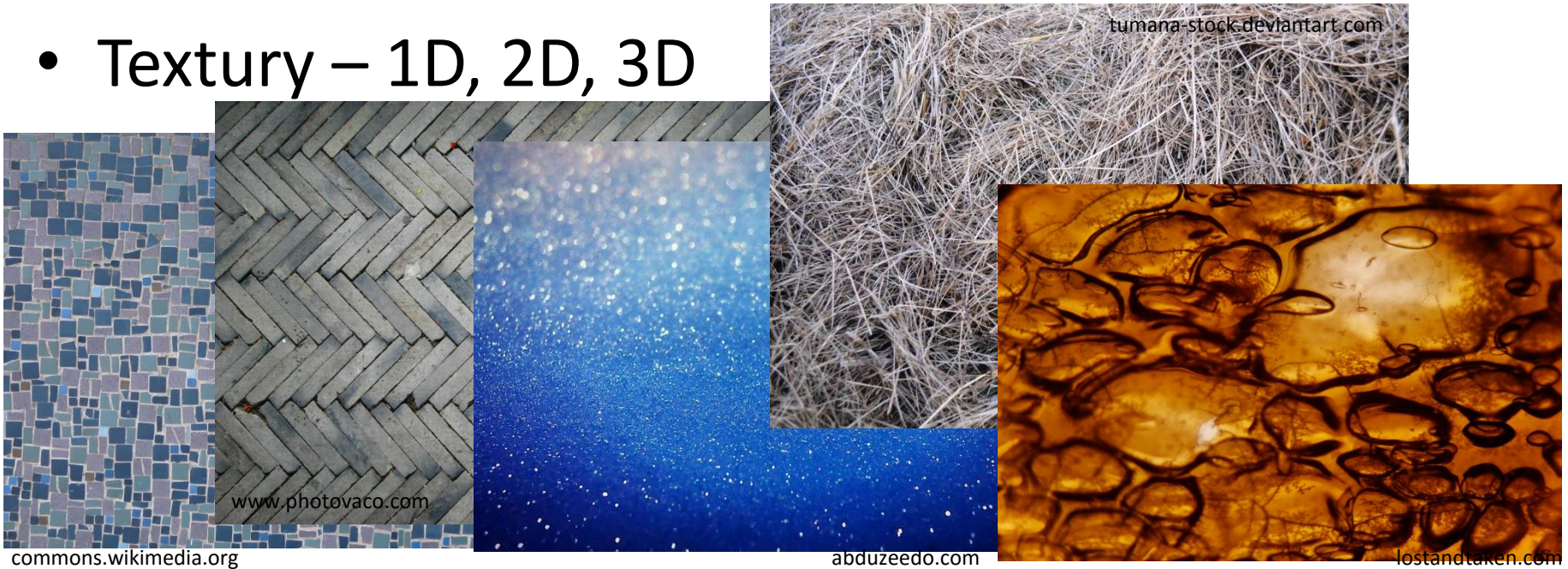


PV112 – Programování grafických aplikací

7. přednáška – Textury

Texturování

- Způsob obarvení povrchu zobrazovaných těles různými obrazci – **texturami**
- Neměníme geometrické vlastnosti těles, pouze jinak zobrazujeme jejich povrch
- Textury – 1D, 2D, 3D

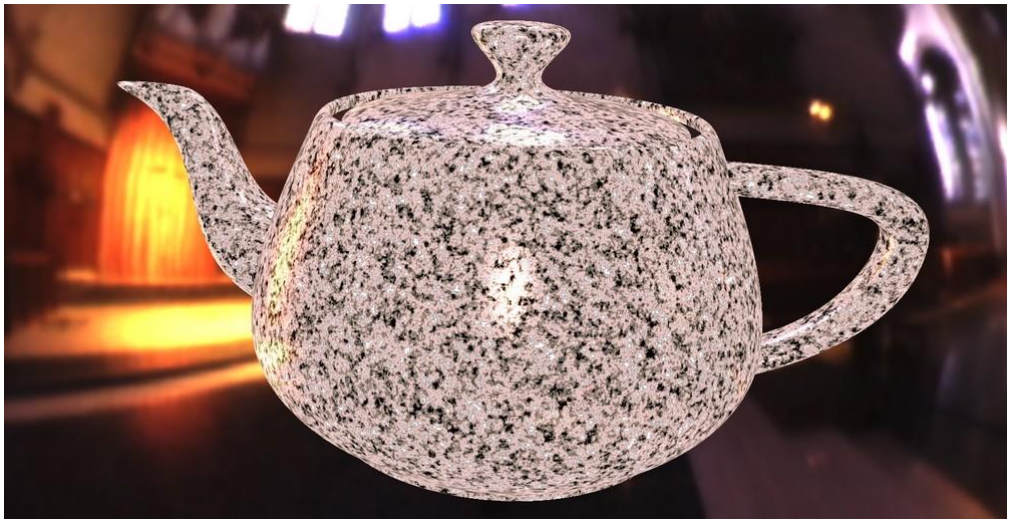


Textury

- Vytvoření více způsoby:
  - Klasické rastrové obrázky (malování, fotka, scan)
  - Generování pomocí algoritmů – většinou fraktální techniky – **procedurální textury**
- Procedurální textury se dají využít:
  - Pro výpočet rastrových obrázků před samotným vykreslováním (poté s ní zacházíme jako s běžným rastrovým obrázkem)
  - Pro výpočet textury v reálném čase až při vykreslování – OpenGL podporuje v shaderech

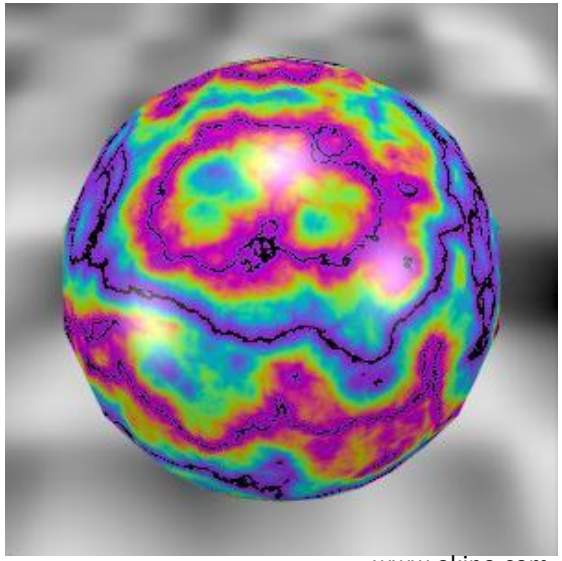
Příklady procedurálních textur

www.cg.tuwien.ac.at

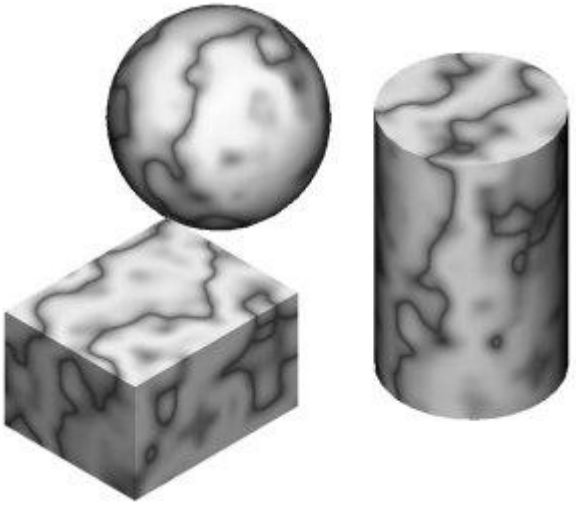


eraser85.wordpress.com

en.wikipedia.org



www.okino.com



docs.bentley.com

Texel

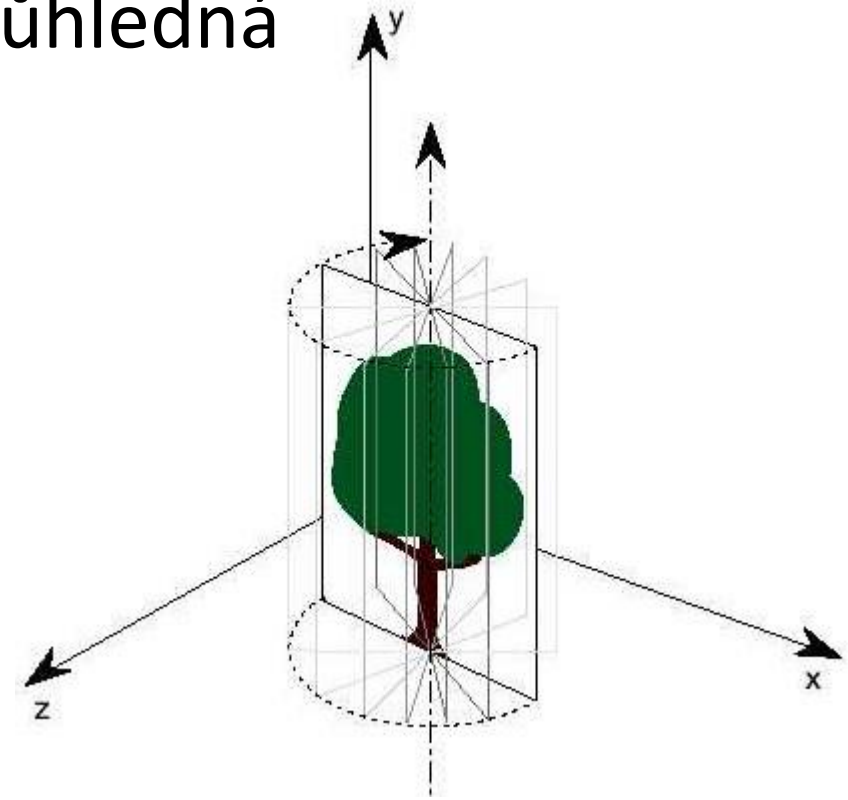
- Textura je složena z **texelů**
  - Rastrový element většinou dvourozměrné textury
- Pixely a texely mají stejné vlastnosti i velmi podobný (většinou ekvivalentní) způsob uložení v paměti
- Proces texturování = nanášení texelů na vykreslovaný povrch

Vhodnost použití textur

- Textury používáme v případě, kdy vykreslujeme tělesa se složitě strukturovanými povrchy, které nevykazují velké změny v geometrii povrchu
  - např. cihlová zeď, koberec, …
- Objekt reprezentujeme zjednodušeně a naneseme rastrový obrázek

Billboarding

- Technika mapování textury na sadu vzájemně se protínajících ploch – textura musí být v některých místech průhledná
- Např. stromy



Sprite

- Otexturované objekty (většinou obdélníky) natočeny k pozorovateli vždy stejnou stranou – **sprites**
- V OpenGL vykreslujeme jako vhodně natočené obdélníky s texturou

spriters-resource.com

http://www.freewebs.com/spritedemonx/Sargesheet.gif

Výhody použití rastrových textur

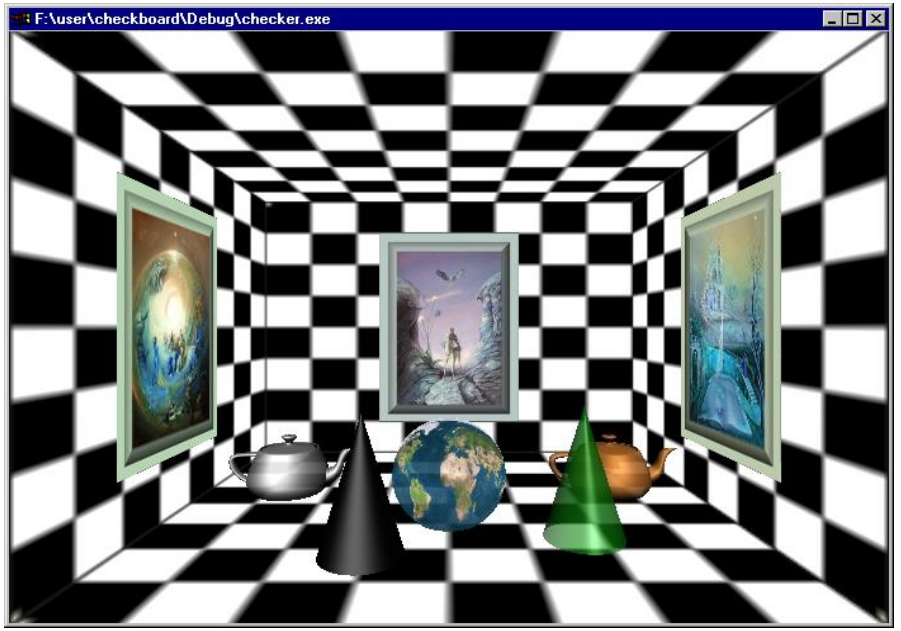
- Zjednodušení složitých povrchů těles – závislé na vhodné volbě textury, velikosti objektu a nasvícení scény
- Jednoduchá implementace ve vykreslovacím řetězci
- Při použití průhlednosti dokážeme vizuálně měnit geometrii objektu (např. díry)

Nevýhody použití rastrových textur

- Předem dané rozlišení textur
- Zvětšení/zmenšení počtu zobrazovaných pixelů textury – dochází k aliasu. Jeho odstranění zpomaluje vykreslování.
- Textury zabírají poměrně velké množství paměti. Pokud je dostatečně velká paměť grafického akcelerátoru, textury do ní nahrajeme.

Textury v OpenGL

- OpenGL podporuje pouze rastrové textury:
  - 1D – pás pixelů, realizace barevných přechodů
  - 2D – bitmapy a pixmapy, nejpoužívanější
  - 3D – objemové textury, pro specializované aplikace
  - Cubemap textury – pro odlesky
  - Pole textur
  - …



www.student.seas.gwu.edu

Postup při texturování

1. Vytvoření rastrové předlohy textury nebo její načtení ze souboru
2. Vytvoření nového texturovacího objektu, přiřazení textury tomuto objektu a nastavení formátu textury
3. Vykreslení scény se zadanými texturovacími souřadnicemi pro každý vrchol

1. Vytvoření/načtení rastrové předlohy textury

- OpenGL přímo nepodporuje výpočet rastrových textur ani načtení ze souboru
- Možnosti:
  - Využití knihovny pro práci s grafickými formáty (libtiff, libjpg, libpng)
  - Výpočet procedurální textury přímo v aplikaci
  - Přečtení vykresleného obrázku přímo z framebufferu (`glCopyTexImage2D()`)
  - Použití mipmappingu (viz dále)

2. Vytvoření nového texturovacího objektu, přiřazení textury, nastavení formátu textury

```
void glTexImage1D(GLenum target,
GLint level, GLint internalFormat, GLsizei
width, GLint border, GLenum format, GLenum
type, const GLvoid *pixels)
```

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

Význam parametrů

- *target* = dimenze textury (GL_TEXTURE_1D, GL_TEXTURE_2D, …)
- *level* = význam při mipmappingu. Pro běžnou texturu nastavíme na 0.
- *internalFormat* = vnitřní formát dat textury, specifikuje počet barevných komponent textury. Nabývá konstant GL_RGB, GL_RGBA, ...
  - formátem se zadává počet komponent, jejich typ a bitová hloubka

Význam parametrů

- *width, height* = rozlišení textury. Pro 1D texturu se height nezadává.

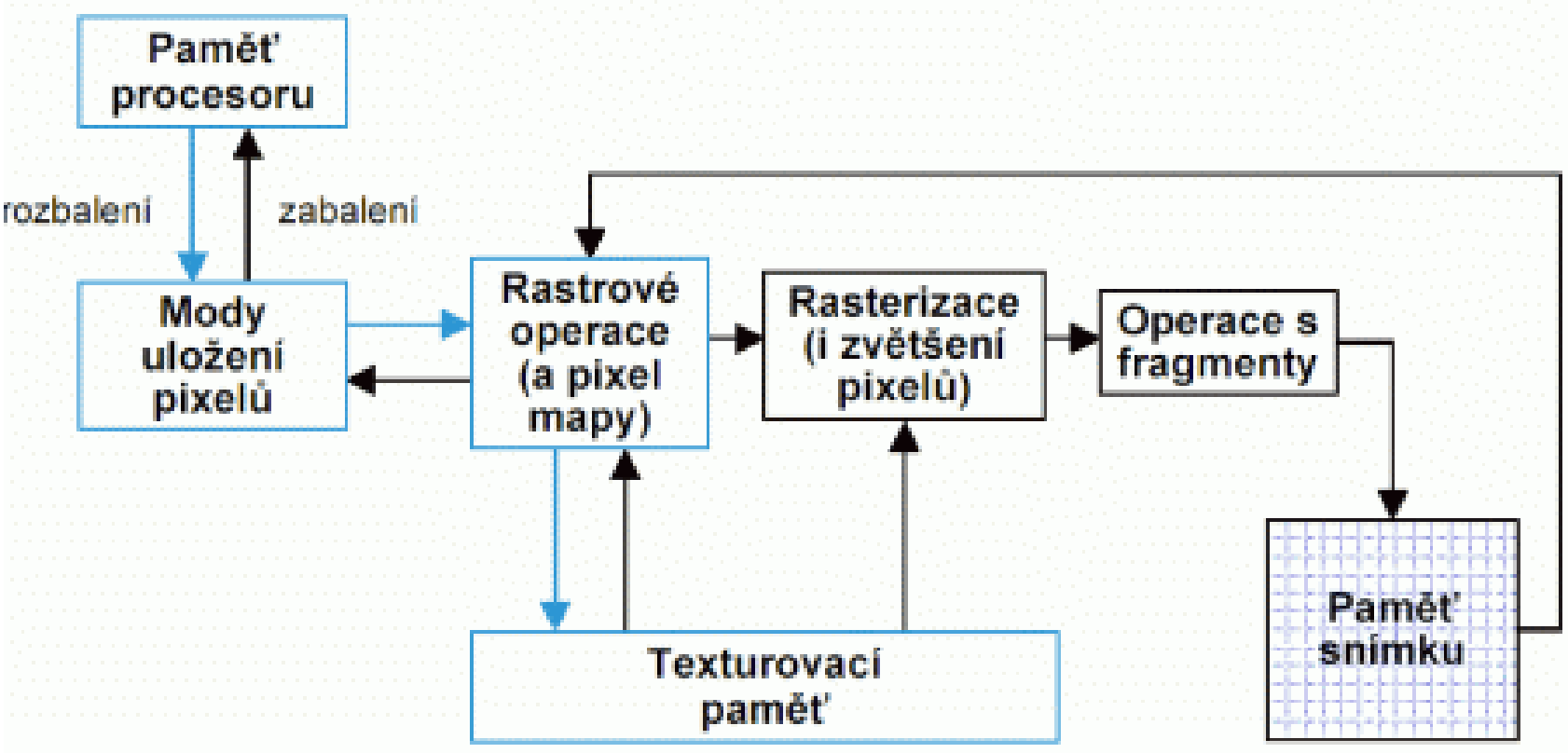
- *border* = šířka okraje textury v pixelech. Musí být nastavena na 0.

Význam parametrů

- *format* = formát použitý pro jednotlivé pixely textury – GL_RGB, GL_RGBA, …
- *type* = typ dat každé barevné složky pixelu – GL_BYTE, GL_INT, …
- *pixels* = ukazatel na rastrová data textury (často statické nebo dynamické pole)

Zápis do texturovací paměti

- Po zadání příkazu glTexImage*() jsou data textury odeslána do texturovací paměti

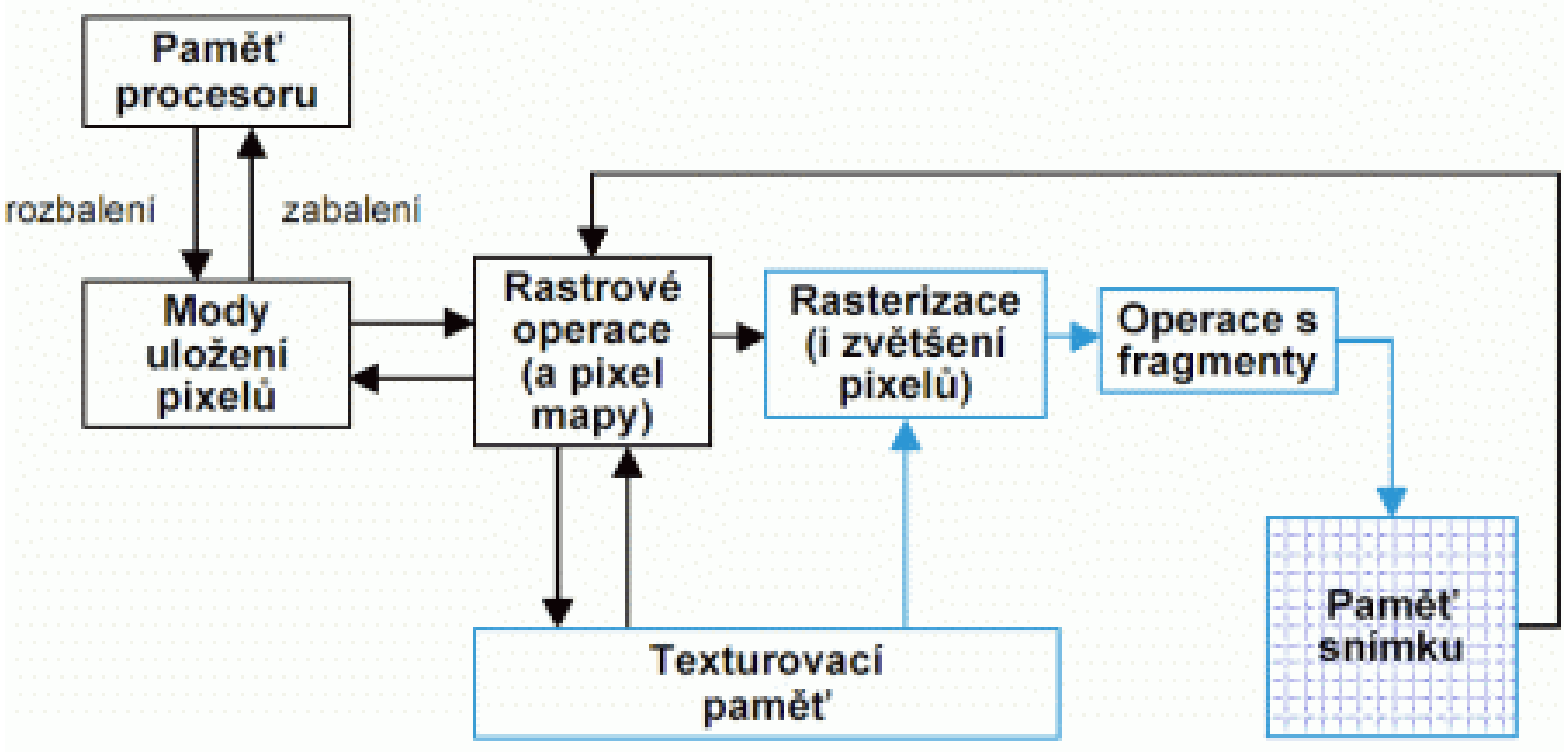


3. Vykreslení scény s texturami

- Podobně jako u barevných nebo osvětlených plošek, musíme navíc každému vrcholu specifikovat souřadnice v textuře
- Souřadnice v textuře zadáváme jako parametry vrcholu

Použití texturovací paměti

- Použití texturovací paměti při vykreslování



Transformace textury

- Texturové souřadnice mohou být před samotným mapováním textury vynásobeny hodnotami v texturové matici 4x4
- Po přiřazení textury objektu může být tato textura transformována stejným způsobem jako samotné objekty (rotace, scale, …)

Nastavení parametrů textur

```
void glTexParameter{fi}(GLenum target,
GLenum pname, TYPE value)
```

```
void glTexParameter{fi}v(GLenum target,
GLenum pname, const TYPE *values)
```

- *target* = GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D, …
- *pname* a *value* mohou nabývat hodnot:

Parametry *pname* a *value*

TEXTURE_WRAP_S

- Určuje, zda se má při překročení rozsahu texturovací souřadnice ve směru osy *s* provést opakování motivu na textuře (*value* = REPEAT, MIRRORED_REPEAT) nebo protažení první či poslední hodnoty (*value* = CLAMP_TO_EDGE, CLAMP_TO_BORDER, MIRROR_CLAMP_TO_EDGE)

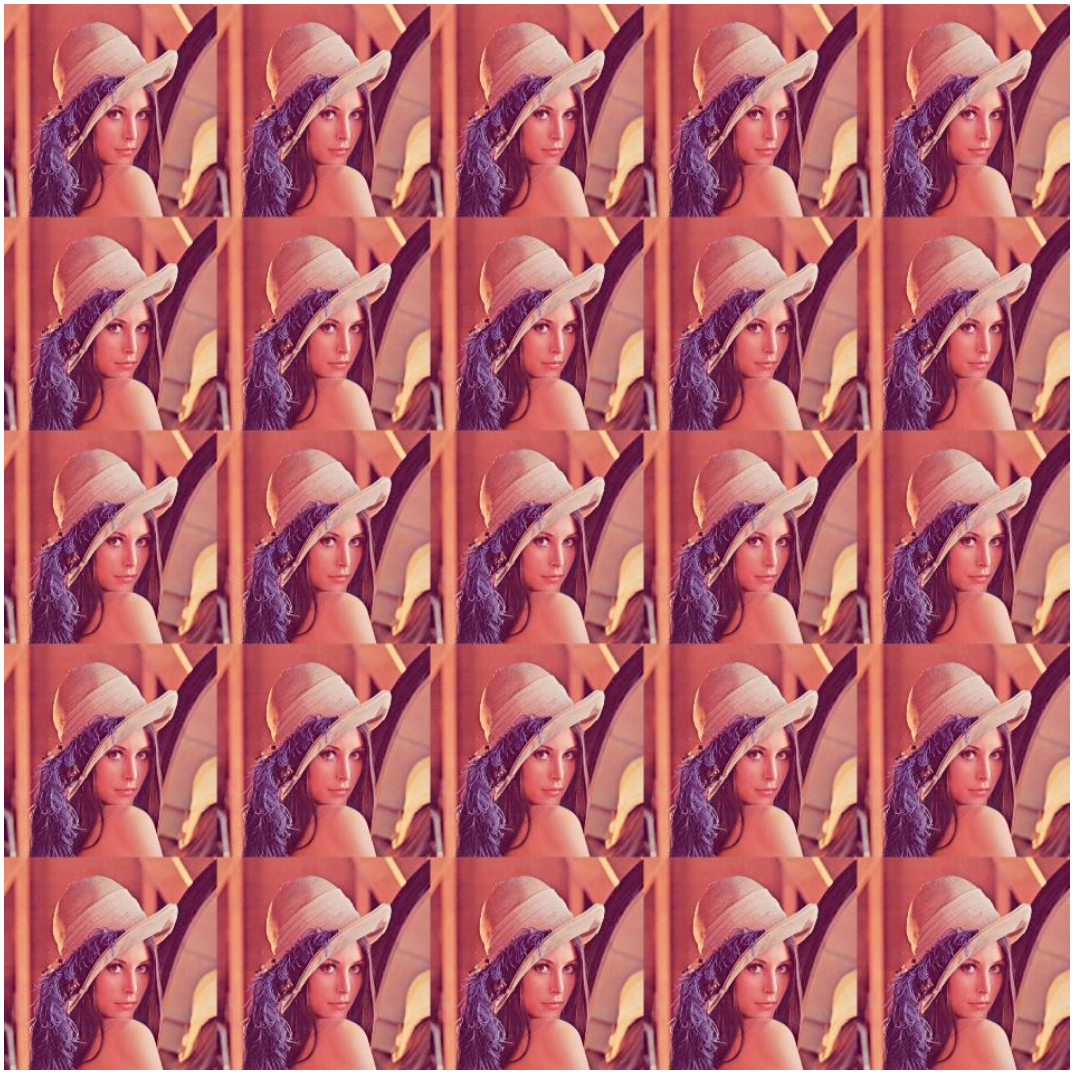
TEXTURE_WRAP_T

- Podobný význam jako předchozí parametr, pouze se vztahuje na souřadnici ve směru osy *t*

TEXTURE_WRAP_R

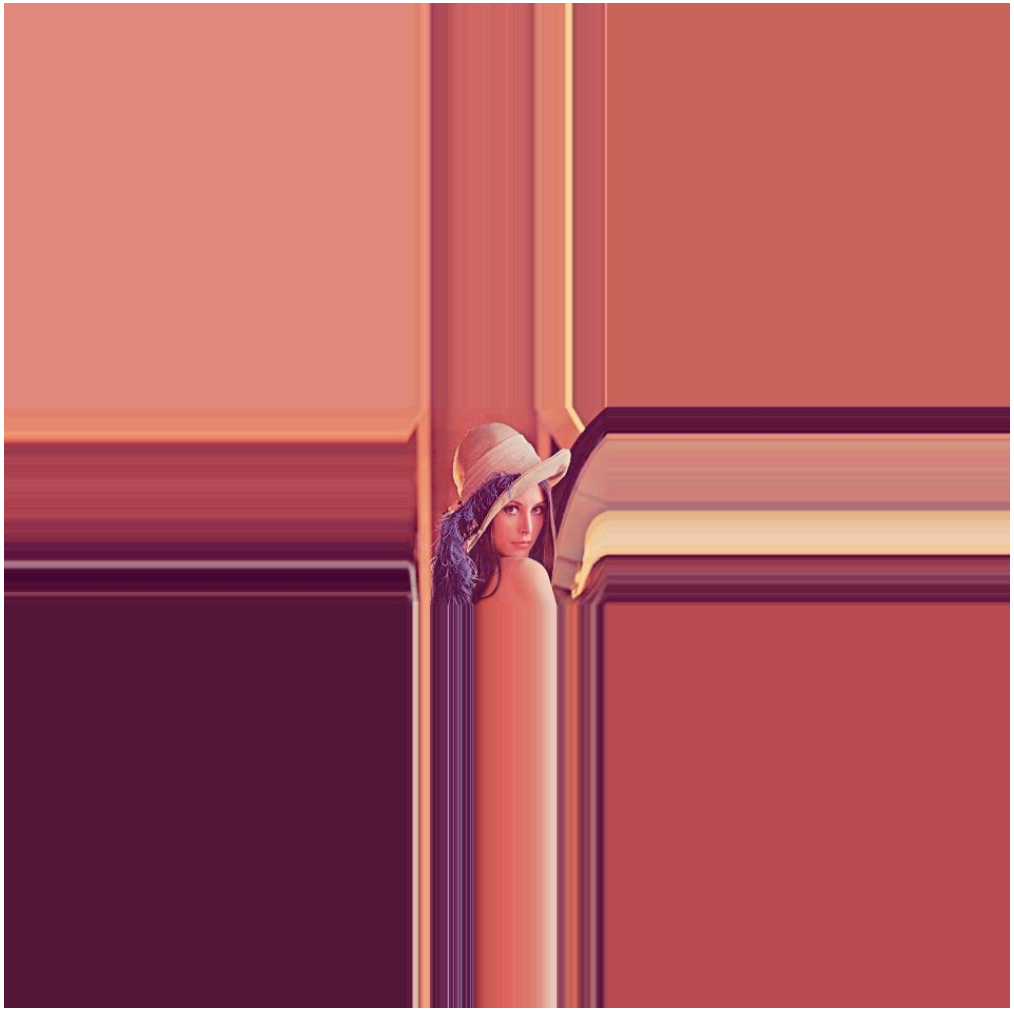
- Opět podobný význam, pouze pro souřadnici *r* (použita u objemových textur)

REPEAT

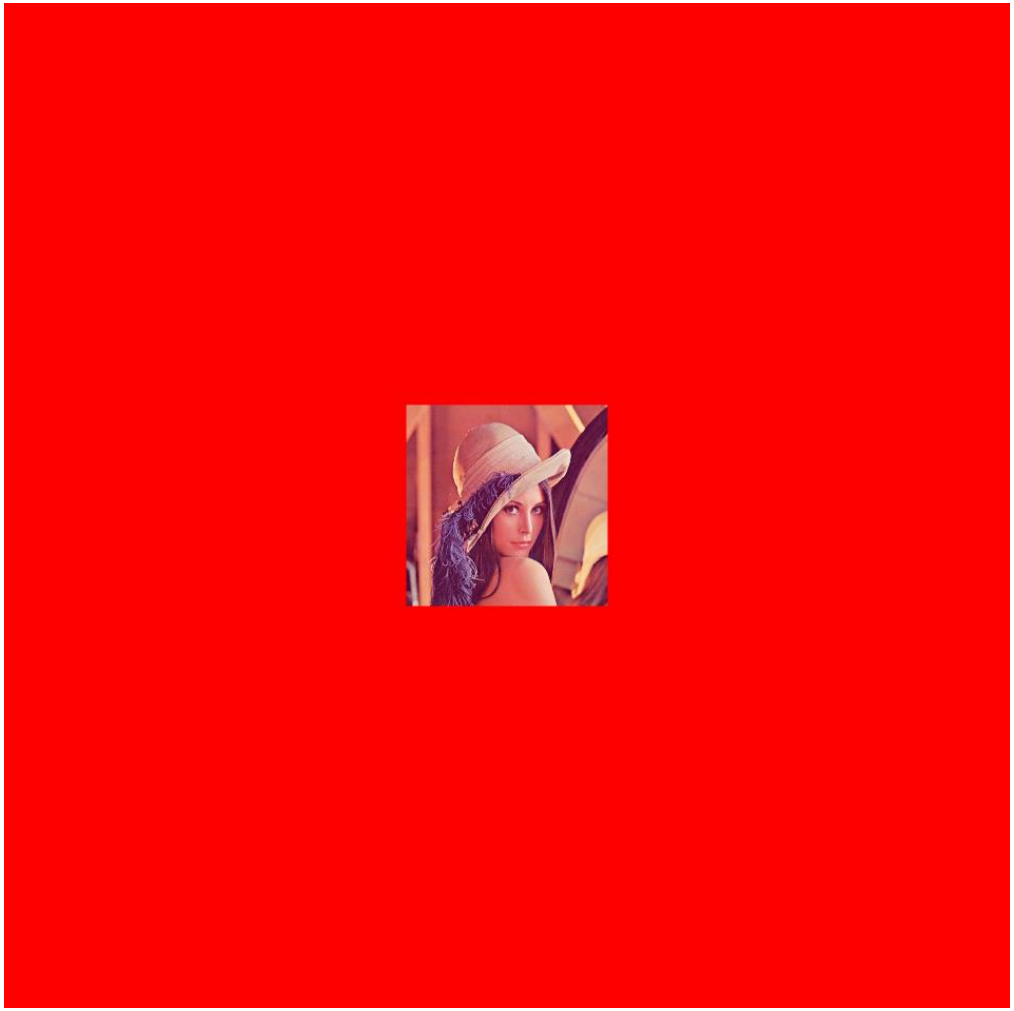


MIRRORED_REPEAT

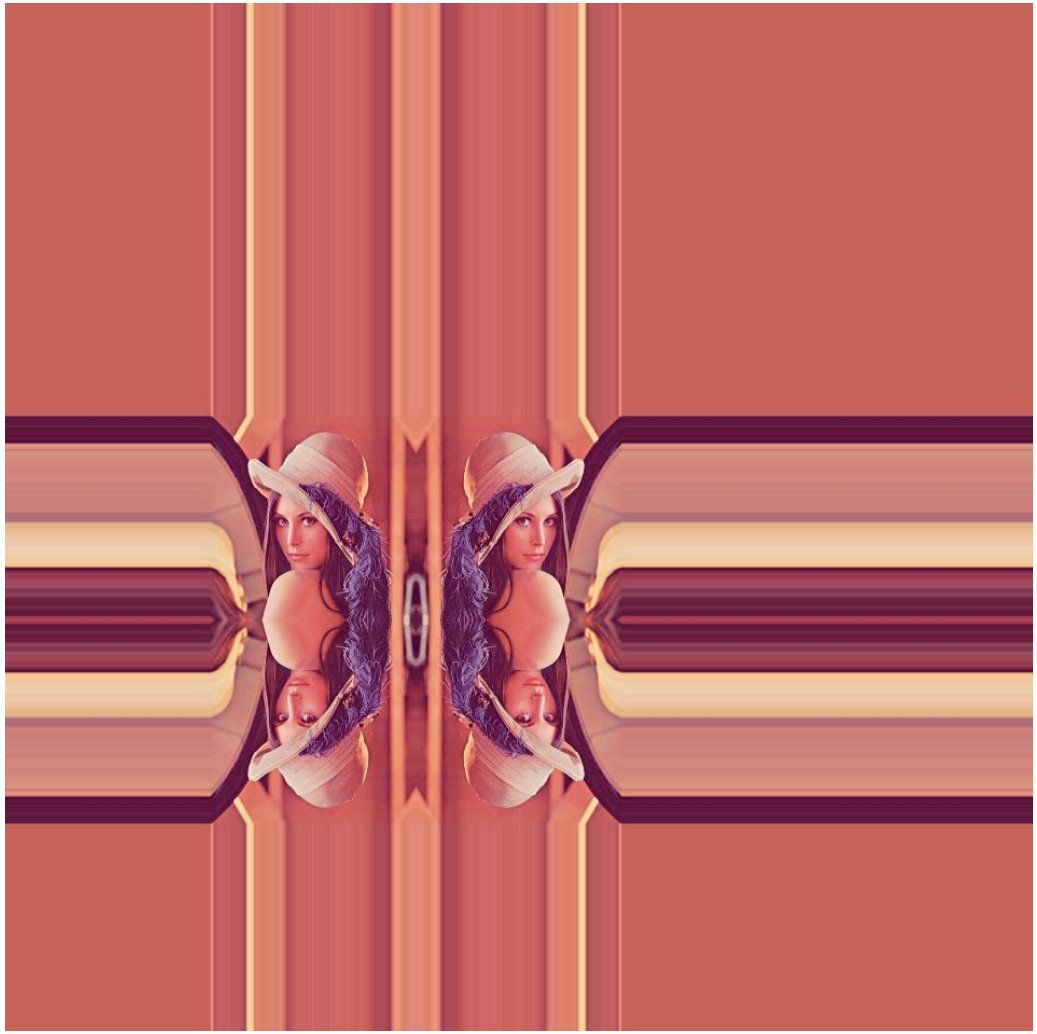
CLAMP_TO_EDGE



CLAMP_TO_BORDER



MIRROR_CLAMP_TO_EDGE



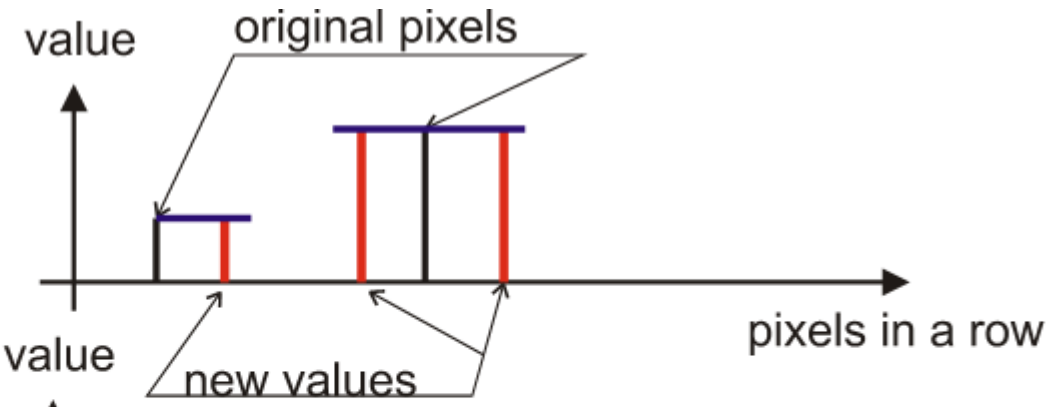
Parametry *pname* a *value*

TEXTURE_MIN_FILTER

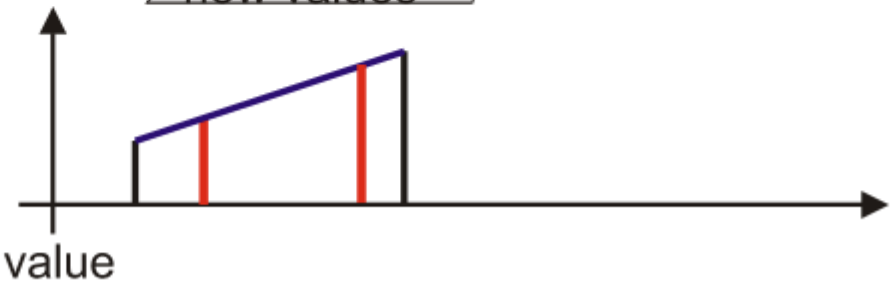
- Filtr použitý při zmenšování textury
- *value* =
  - NEAREST – barva pixelu se určí z barvy texelu, jehož souřadnice nejpřesněji odpovídají souřadnicím zadaným do textury
  - LINEAR – barva pixelu se spočítá pomocí bilineární interpolace z barev sousedních texelů
  - NEAREST_MIPMAP_NEAREST, NEAREST_MIPMAP_LINEAR, LINEAR_MIPMAP_NEAREST a LINEAR_MIPMAP_LINEAR – využívají mipmapy

Typy filtrů

Nearest neighbor



Linear interpolation



Parametry *pname* a *value*

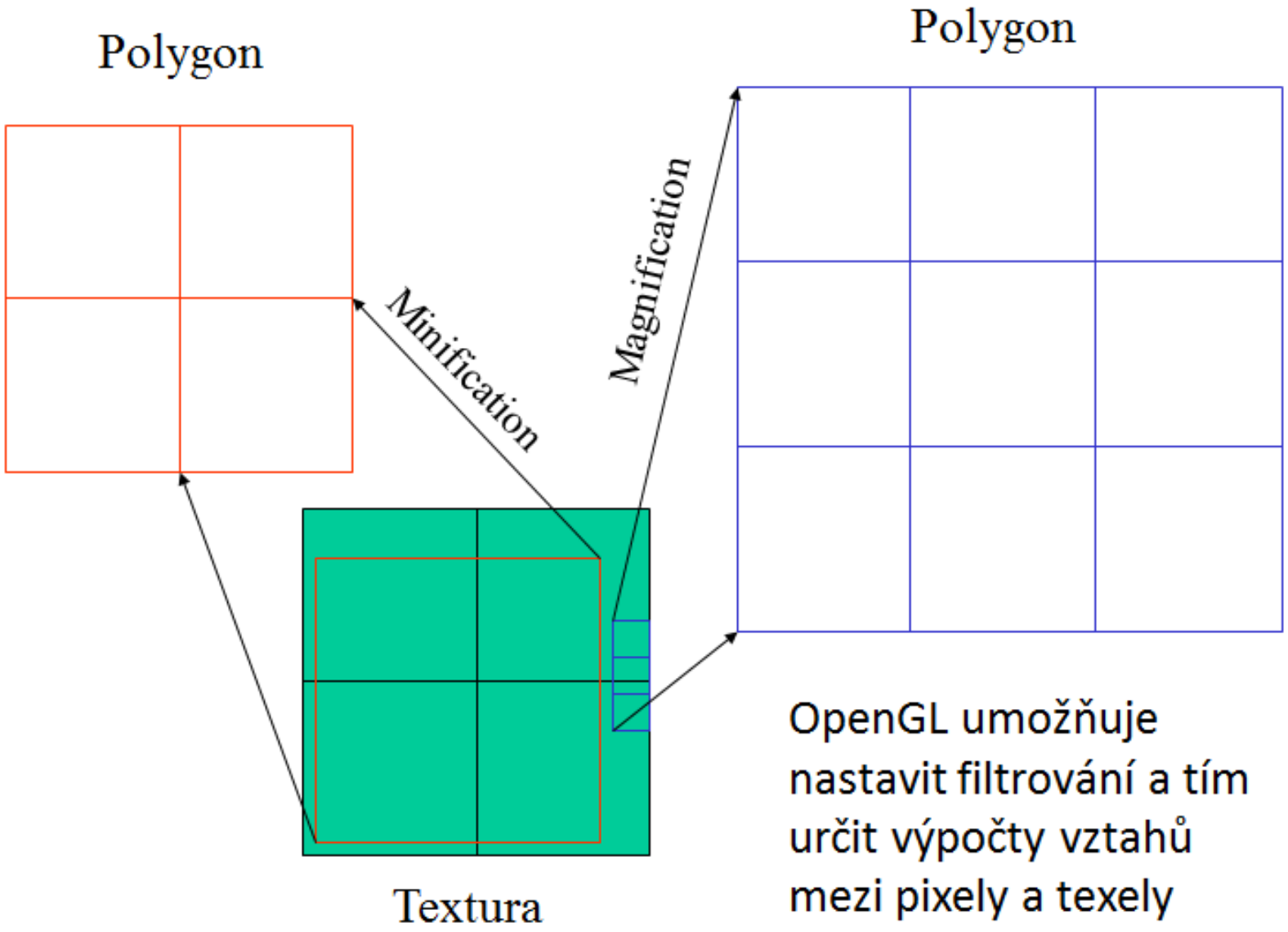
TEXTURE_MAG_FILTER

- Filtr použitý při zvětšování textury
- *value* =
  - NEAREST – použije se nejbližší texel
  - LINEAR – lineární interpolace mezi barvami sousedních texelů

TEXTURE_BORDER_COLOR

- barva rámečku okolo textury (pokud je rámeček použit). Implicitní barva je (0, 0, 0, 0). Je to jediný způsob, jak zadat barvu rámečku.

Zvětšení/zmenšení textury



OpenGL umožňuje nastavit filtrování a tím určit výpočty vztahů mezi pixely a texely

Zvětšení/zmenšení textury

Implicitní nastavení pro **TEXTURE_MIN_FILTER** je **NEAREST_MIPMAP_LINEAR.** V tomto okamžiku filtr požaduje pro výpočet správného výsledku plnou a konzistentní množinu mipmap. Pokud však není k dispozici, je návratová hodnota (0,0,0,1).

Pokud chceme použít glTexImage2D() bez mipmap, tak musíme nastavit buď: **TEXTURE_MIN_FILTER, LINEAR** nebo **TEXTURE_MIN_FILTER, NEAREST**

Rozšířená podpora formátů textur

- Od verze 1.2
- Výhody:
  - Použitím vhodných formátů s malou nebo dostatečnou bitovou hloubkou můžeme ušetřit paměť alokovanou pro textury
  - Možnost použít textury načtené z různých externích souborů majících různé formáty (např. BGR)
- Tabulka rozšířených formátů textur:

Korespondence mezi rozšířenými a základními formáty	
Rozšířený formát	Základní formát
GL_R3_G3_B2	GL_RGB
GL_RGB4	GL_RGB
GL_RGB5	GL_RGB
GL_RGB8	GL_RGB
GL_RGB10	GL_RGB
GL_RGB12	GL_RGB
GL_RGB16	GL_RGB
GL_RGBA2	GL_RGBA
GL_RGBA4	GL_RGBA
GL_RGB5_A1	GL_RGBA
GL_RGBA8	GL_RGBA
GL_RGB10_A2	GL_RGBA
GL_RGBA12	GL_RGBA
GL_RGBA16	GL_RGBA

Rozšířený formát	Počet bitů na bar. složku					
	R	G	B	A	L	I
GL_R3_G3_B2	3	3	2			
GL_RGB4	4	4	4			
GL_RGB5	5	5	5			
GL_RGB8	8	8	8			
GL_RGB10	10	10	10			
GL_RGB12	12	12	12			
GL_RGB16	16	16	16			
GL_RGBA2	2	2	2	2		
GL_RGBA4	4	4	4	4		
GL_RGB5_A1	5	5	5	1		
GL_RGBA8	8	8	8	8		
GL_RGB10_A2	10	10	10	2		
GL_RGBA12	12	12	12	12		
GL_RGBA16	16	16	16	16		

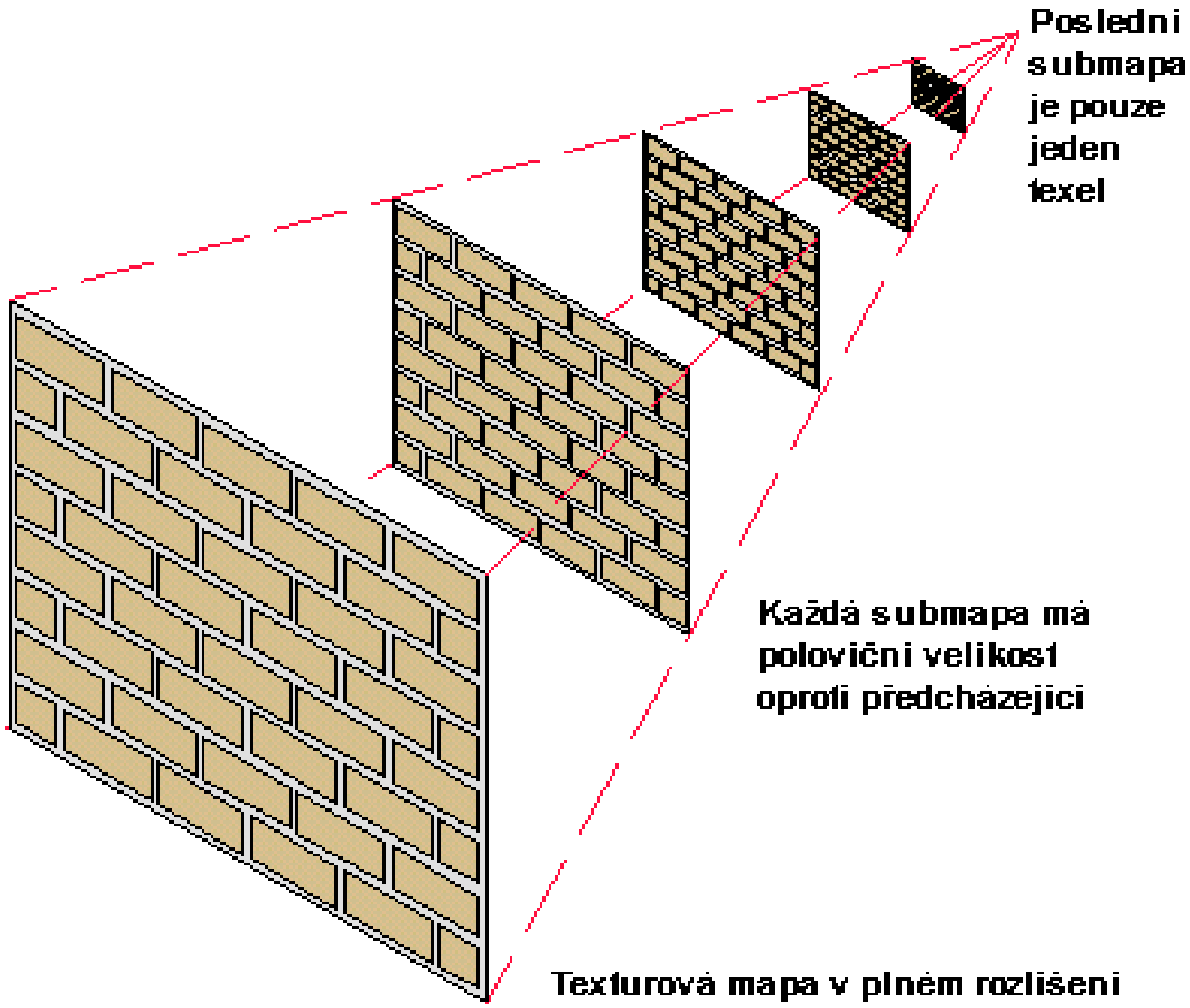
Mipmapping

- Mip Map = Multum in Parvo (mnoho v málu)
- Odstranění/zmírnění některých vizuálních chyb vzniklých při pohybu objektů s nanesenou texturou v 3D scéně nebo při pohybu celé scény – objekty se na obrazovce zmenšují, zvětšují či jinak deformují → nutné deformovat i texturu při nanášení texelů na pixely

Mipmapping

- Nejdříve se zjistí relativní velikost povrchu vůči celé textuře a poté se vybere vhodné rozlišení textury, která se nanese
- Pro zamezení problikávání při přechodu mezi úrovněmi se zavádí interpolace – vybere se nejbližší větší a nejbližší menší textura a barva pixelů se vypočte interpolací mezi těmito dvěma texturami

Mipmapping


Posledni
submapa
je pouze
jeden
texel
Každá submapa má
poloviční velikost
oproti předcházejici
Texturová mapa v plném rozlišeni

Originální textura

Mipmapa 1/4

1/16

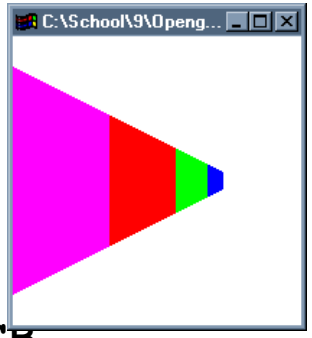
1/64

Mipmapping v OpenGL

```
void glTexImage2D(GLenum target,
GLint level, GLint components, GLsizei width,
GLsizei height, GLint border, GLenum format,
GLenum type, const GLvoid *pixels)
```

- parametr *level* zadává úroveň textury v hierarchii
- nejvyšší rozlišení = 0, poloviční rozlišení = 1, …
- Mipmapa rozdělena podle jednotlivých RGB složek

Příklad




```
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,32,32,0,GL_RGB,
             GL_UNSIGNED_BYTE,&image32[0][0][0]);//32x32
glTexImage2D(GL_TEXTURE_2D,1,GL_RGB,16,16,0,GL_RGB,
             GL_UNSIGNED_BYTE,&image16[0][0][0]);//16x16
glTexImage2D(GL_TEXTURE_2D,2,GL_RGB,8,8,0,GL_RGB,
             GL_UNSIGNED_BYTE,image8[0][0][0]);  //8x8
glTexImage2D(GL_TEXTURE_2D,3,GL_RGB,4,4,0,GL_RGB,
             GL_UNSIGNED_BYTE,&image4[0][0][0]); //4x4
glTexImage2D(GL_TEXTURE_2D,4,GL_RGB,2,2,0,GL_RGB,
             GL_UNSIGNED_BYTE,&image2[0][0][0]); //2x2
glTexImage2D(GL_TEXTURE_2D,5,GL_RGB,1,1,0,GL_RGB,
             GL_UNSIGNED_BYTE,&image1[0][0][0]); //1pixel
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
                GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                GL_NEAREST_MIPMAP_NEAREST);
```

Zmenšování/zvětšování textur

- Pro mipmapping další typy filtrů (kromě `NEAREST, LINEAR`):
  - NEAREST_MIPMAP_NEAREST – pro obarvení pixelu je vybrán nejbližší texel z nejbližší větší nebo menší textury
  - NEAREST_MIPMAP_LINEAR – dva nejbližší texely z obou textur, mezi nimi lineární interpolace
  - LINEAR_MIPMAP_NEAREST – bilineární interpolace nejbližších texelů v jedné textuře
  - LINEAR_MIPMAP_LINEAR – interpolace pro výpočet barev texelů v obou texturách, poté další interpolace mezi dvěma předešlými

Automatické generování mipmap

```
void glGenerateMipmap( GLenum target );
```

- Generování mipmapy bez nutnosti pracovat s texturou na CPU
- Parametr *target* může být `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, `GL_TEXTURE_3D`, `GL_TEXTURE_1D_ARRAY`, `GL_TEXTURE_2D_ARRAY`, `GL_TEXTURE_CUBE_MAP`, `GL_TEXTURE_CUBE_MAP_ARRAY`

Texturové objekty

- Ukládají texturová data, ta jsou pak velmi rychle dostupná
- Jediný způsob, jak aplikovat textury – opětovné použití existující textury je rychlejší než opětovné načítání obrázku textury použitím glTexImage*()

Používání texturových objektů

Tři kroky:

1. Generování texturových jmen.
2. Počáteční vytvoření a navázání texturových objektů na texturová data.
3. Opakované navázání texturových objektů pro jejich dostupnost pro aktuálně renderované texturové objekty.

Pojmenování texturových objektů

- Nenulová unsigned integer hodnota může být použita jako jméno texturového objektu

```
void glGenTextures(Glsizei n, Gluint *textureNames);
```

- Funkce vrátí *n* aktuálně nepoužívaných jmen pro texturové objekty v poli *textureNames*

```
GLboolean glIsTexture(Gluint textureName);
```

- Určí, zda je dané texturové jméno aktuálně používáno. GL_TRUE, pokud byla textura s daným jménem již navázána.

Vytvoření a použití texturových objektů

```
void glBindTexture(Glenum target, Gluint textureName);
```

- Vytvoří a naváže texturové objekty
- Následné volání glTexImage*(), glTexSubImage*(), glCopyTexImage*()glCopyTexSubImage*(), glTexParameter*() a glPrioritizeTextures() ukládá data do texturových objektů
- Pokud je daný texturový objekt opět navázán, jeho data se stanou aktivní (aktuální stav textury)

glBindTexture() – funkce

1. Pokud navazujeme již dříve vytvořený texturový objekt, stane se aktivním
2. Pokud navazujeme na texturový objekt s *textureName* 0, OpenGL při dalším volání funkcí používajících tento texturový objekt začne hlásit chyby

Odstranění texturových objektů

- Uvolnění texturových zdrojů

```
void glDeleteTextures(Glsizei n,
const Gluint *textureNames);
```

- Smaže *n* texturových objektů se jmény uloženými v *textureNames*
- Jména mohou být použita znovu
- Smazání aktuálně používané textury – návrat k defaultní textuře
- Pokus o smazání textury s neexistujícím jménem nebo s *textureName* 0 je ignorován

Texturové objekty – příklad

```
glGenTextures(2, texName);
glBindTexture(GL_TEXTURE_2D, texName[0]);
glTexParameteri( ...
...
glTexImage2D(...
glBindTexture(GL_TEXTURE_2D, texName[1]);
...
void display(void)
{
  glBindTexture(GL_TEXTURE_2D, texName[0]);
  ...
}
```

Další funkce

```
void glTexSubImage2D(GLenum target,
GLint level, GLint xoffset, GLint yoffset,
GLsizei width, GLsizei height, GLenum format,
GLenum type, const GLvoid *texels);
```

- Definuje 2D texturový obrázek, který nahrazuje celou oblast nebo spojitou podoblast aktuálního 2D texturového obrázku
- *target* = GL_TEXTURE_2D, …
- *level*, *format*, *type* – podobně jako u glTexImage2D()

Další funkce

```
void glCopyTexImage2D(GLenum target,
GLint level, GLint internalFormat, GLint x,
GLint y, GLsizei width, GLsizei height,
GLint border);
```

- Vytvoří 2D texturu, přičemž pro definici texelů využívá data z framebufferu
- *target* = GL_TEXTURE_2D, …
- *level*, *internalFormat*, *border* – stejné jako ve funkci glTexImage2D()

Další funkce

```
void glCopyTexSubImage2D(GLenum target,
GLint level, GLint xoffset, GLint yoffset,
GLint x, GLint y, GLsizei width,
GLsizei height);
```

- Použije obrazová data z framebufferu pro nahrazení celé oblasti nebo spojitého podregionu aktuálního 2D texturového obrázku
- *target* = GL_TEXTURE_2D, …

Obrázky

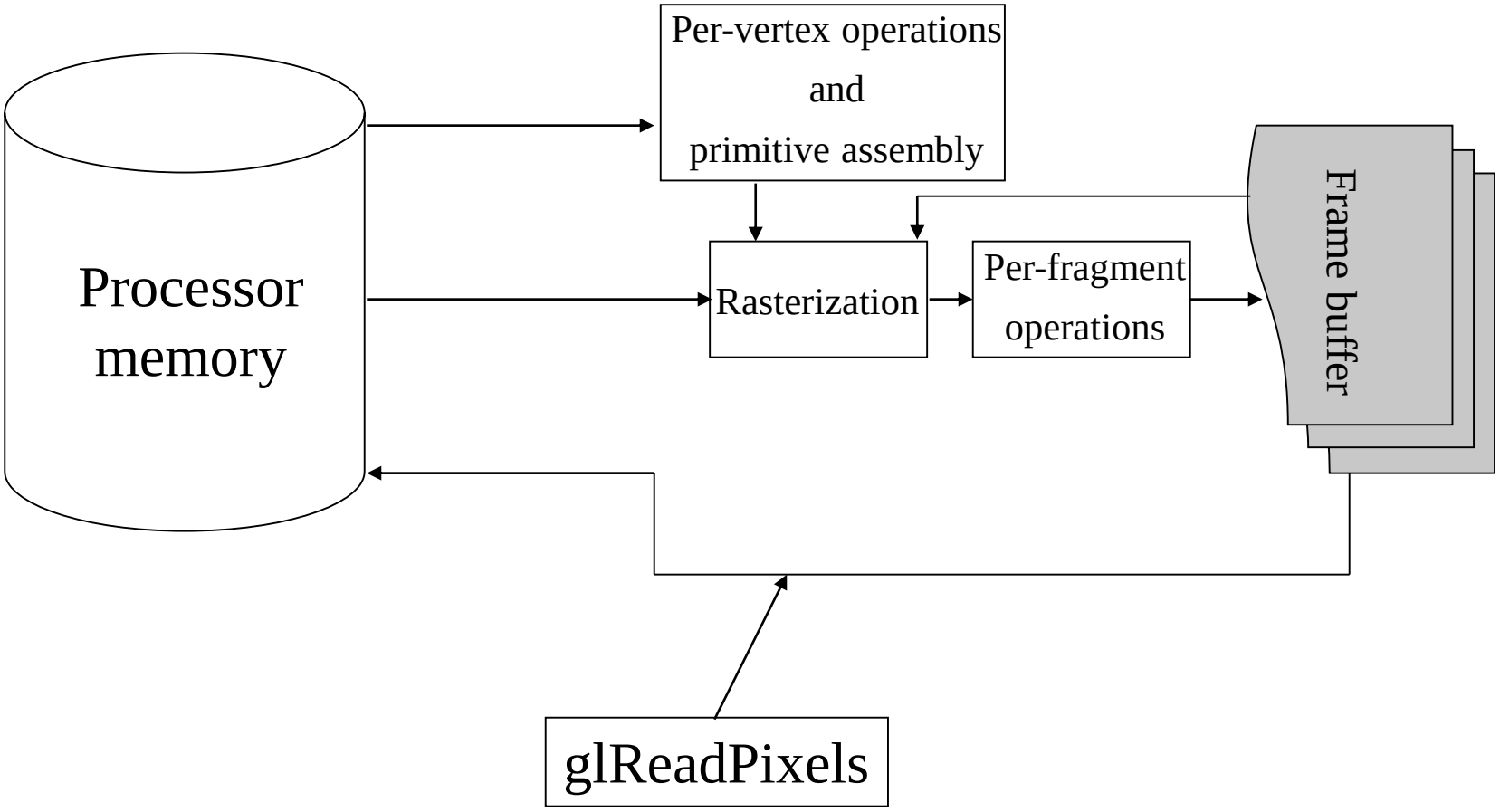
- Matice sestávající z pixelů
- Každý pixel může obsahovat (R, G, B, A) barvu

```
void glReadPixels(GLint x,GLint y,
GLsizei w, GLsizei h, GLenum format,
GLenum type, GLvoid *dta)
```

- Funkce načte obdélníkové pole pixelů z framebufferu

Parametry

- *format* může nabývat hodnot
  `GL_STENCIL_INDEX`, `GL_DEPTH_COMPONENT`, `GL_DEPTH_STENCIL`,`GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_RGB`, `GL_BGR`, `GL_RGBA`, `GL_BGRA`
- *type* může nabývat hodnot
  `GL_UNSIGNED_BYTE`, `GL_BYTE`, `GL_UNSIGNED_SHORT`, `GL_SHORT`, `GL_UNSIGNED_INT`, `GL_INT`, `GL_HALF_FLOAT`, `GL_FLOAT`
  atd.


Per-vertex operations
and
primitive assembly
Processor
memory
Rasterization
Per-fragment
operations
Frame buffer
glReadPixels

Módy uložení obrázku

- Pro každý obrázek je možné uložit jeden až čtyři elementy – podle formátu pixelových dat

```
void glPixelStore{if}(GLenum pname,TYPE param)
```

  - Nastavení módu pro uložení pixelů, které ovlivňuje operace glReadPixels(), glTexImage*D(), …

Parametry glPixelStore()

Parameter Name	Type	Initial Value	Valid Range
GL_UNPACK_SWAP_BYTES, GL_PACK_SWAP_BYTES	GLboolean	FALSE	TRUE/FALSE
GL_UNPACK_LSB_FIRST, GL_PACK_LSB_FIRST	GLboolean	FALSE	TRUE/FALSE
GL_UNPACK_ROW_LENGTH, GL_PACK_ROW_LENGTH	GLint	0	any nonnegative integer
GL_UNPACK_SKIP_ROWS, GL_PACK_SKIP_ROWS	GLint	0	any nonnegative integer
GL_UNPACK_SKIP_PIXELS, GL_PACK_SKIP_PIXELS	GLint	0	any nonnegative integer
GL_UNPACK_ALIGNMENT, GL_PACK_ALIGNMENT	GLint	4	1, 2, 4, 8