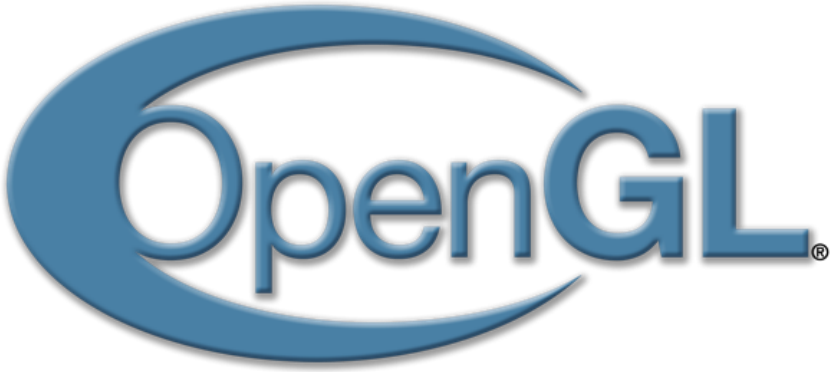


PV112 – Programování grafických aplikací

7. přednáška – Textury pokračování, framebuffer objekty

Přístup ze shaderu

- Speciální datový typ *sampler**

```
sampler1D
sampler2D
...
```

- Uniformní proměnné
- Je to „opaque type“, jde pouze předat funkcím

Přístup ze shaderu

- Příklad

```
uniform sampler1D my_colors;
uniform sampler2D my_tex;
```

Přístup ze shaderu

- OpenGL API k nim přistupuje jako k jakékoliv jiné uniformní proměnné
- Získáme její ID pomocí *glGetUniformLocation*

```
my_tex_loc = glGetUniformLocation(
   my_program, ”my_tex”);
```

Texturovací jednotky

- Textury nepředáváme do shaderů přímo, ale přes texturovací jednotky (*Texture Unit*)
- Minimálně 16 texturovacích jednotek
- Textury navážeme na texturovací jednotku
- Proměnné v shaderech vzorkují z texturovacích jednotek

Texturovací jednotky

```
void glActiveTexture(GLenum texture)
```

- texture = texturovací jednotka (*GL_TEXTURE0*, *GL_TEXTURE1*, …)
- nastavuje aktuální texturovací jednotku
- funkce *glBindTexture* navazuje texturu na právě aktivní texturovací jednotku
- funkce *glTexParameter**, *glTexImage** atd. pracuje s aktivní texturovací jednotkou

Texturovací jednotky

- Pomocí *glUniform1i* nastavím číslo texturovací jednotky

```
glUniform1i(my_tex_loc, 0);
```

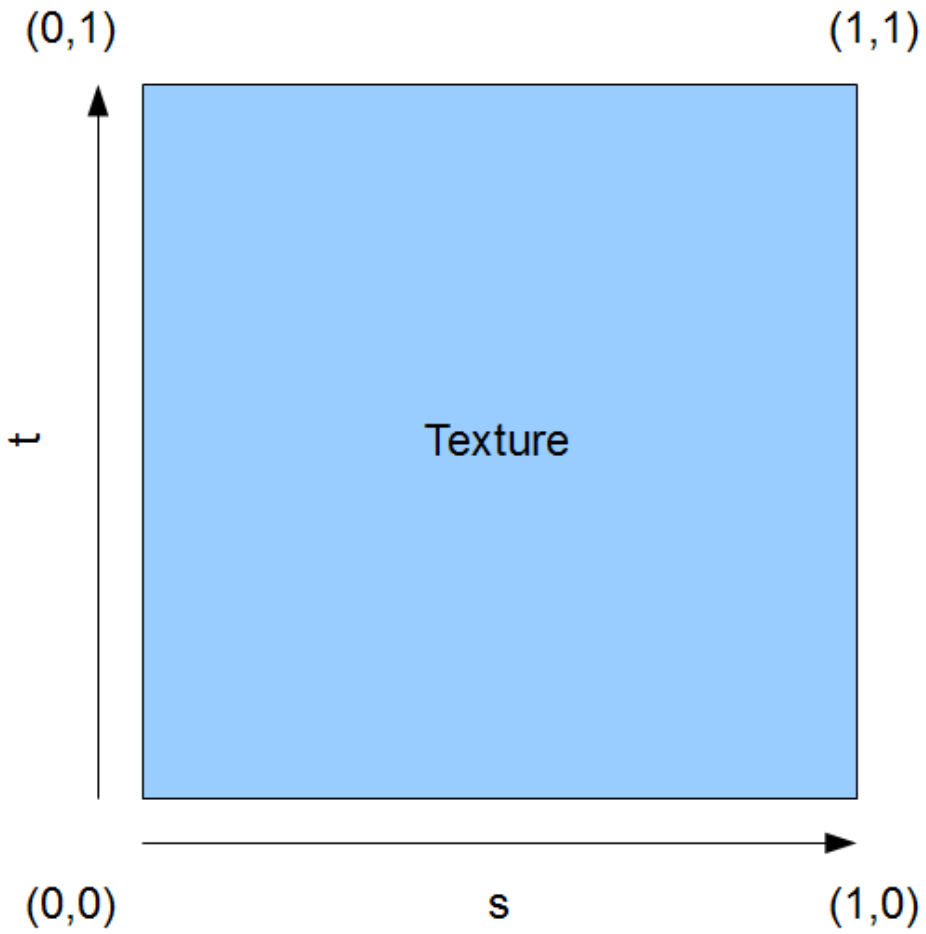
- Příklad:

```
uniform sampler2D my_tex1;
uniform sampler2D my_tex2;

...

my_tex1_loc = glGetUniformLocation(my_program, "my_tex1");
my_tex2_loc = glGetUniformLocation(my_program, "my_tex2");

...

glUniform1i(my_tex1_loc, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, wood_tex);
glUniform1i(my_tex2_loc, 1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, rocks_tex);
...
glDraw*(...);
```

```
uniform sampler2D my_tex1;
uniform sampler2D my_tex2;

...

my_tex1_loc = glGetUniformLocation(my_program, "my_tex1");
my_tex2_loc = glGetUniformLocation(my_program, "my_tex2");
glUniform1i(my_tex1_loc, 0);
glUniform1i(my_tex2_loc, 1);

...

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, wood_tex);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, rocks_tex);
...
glDraw*(...);
```

Texturovací souřadnice

- Určuje, na kterém místě máme texturu navzorkovat
- Normalizovaná do intervalu [0, 1]
  - nezávislá na rozlišení textury
- 1D – 4D
  - *s*, *t*, *r*, *q* v OpenGL API
  - *s*, *t*, *p*, *q* v GLSL
  - *u*, *v*, *w* v jiných knihovnách či softwarech


(0,1)
(1,1)
t
Texture
(0,0)
s
(1,0)

Vzorkování textury

```
vec4 texture(sampler* sampler, vec* coord)
```

- *sampler* = objekt reprezentující texturu
- *coord* = souřadnice, na které texturu vzorkovat
  - dimenze závisí na sampleru

- Navzorkuje texturu na dané texturovací souřadnici, vrací *vec4*, tj. vč. hodnoty alpha

Vzorkování textury

- Příklad:

```
in vec2 tex_coord;

uniform sampler2D my_tex;

out vec4 final_color;

void main()
{
    vec3 color = texture(my_tex, tex_coord).rgb;
    final_color = vec4(color, 1.0);
}
```

Vzorkování textury - problémy

- Při zavolání funkce *texture* OpenGL automaticky spočítá úroveň mipmapy

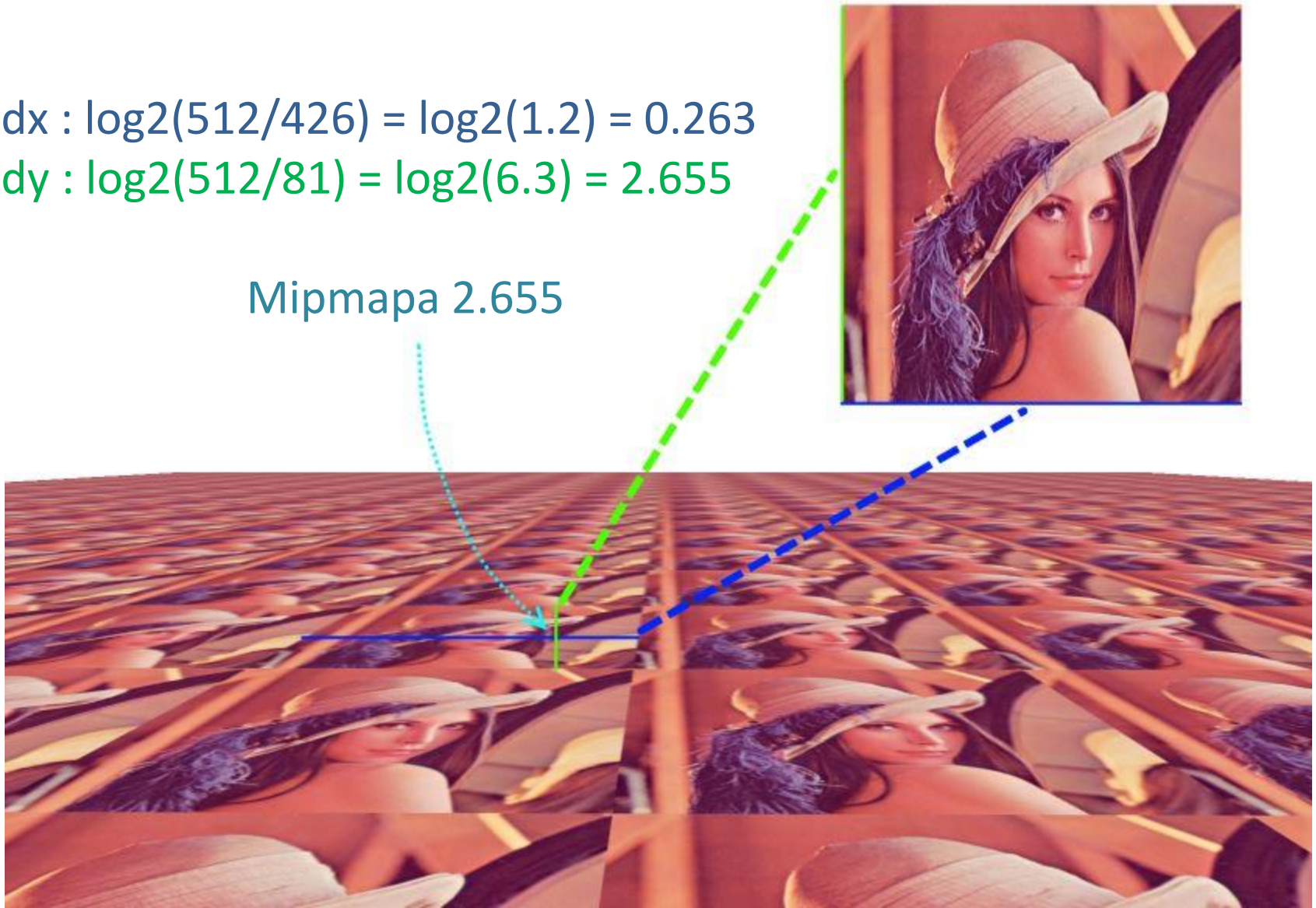
- Mipmapa se počítá na základě derivací v každém směru

Výpočet mipmapy

dx : log2(512/426) = log2(1.2) = 0.263

dy : log2(512/81) = log2(6.3) = 2.655

Mipmapa 2.655



Vzorkování textury - problémy

- Derivace se nepočítá analyticky, ale numericky na základě hodnot sousedních pixelů

- OpenGL vyžaduje, aby funkci *texture* volaly všechny fragmenty „stejně“, což ovšem ne vždy je možné zařídit
  - podmínky if, cykly, discard, return
  - vertex shader

Příklady

- Problém:

```
float my_alpha = texture(my_alpha_tex, tex_coord).a;
if (my_alpha < 0.5)
    discard;
vec3 my_color = texture(my_color_tex, tex_coord).rgb;
...
```

- Řešení:

```
float my_alpha = texture(my_alpha_tex, tex_coord).a;
vec3 my_color = texture(my_color_tex, tex_coord).rgb;
if (my_alpha < 0.5)
    discard;
...
```

Příklady

- Problém:

```
int count = 0;
while (texture(my_tex, coord).a < 0.5)
{
    coord += step;
    count++;
}
...
```

- Problém: vertex shader

Vzorkování textury - řešení

```
vec4 textureLod(sampler* sampler,
    vec* coord, float lod)
```

- *sampler, coord* = stejné jako u *texture*
- *lod* = úroveň mipmapy, kterou použít

- Explicitně stanovíme LOD

Další funkce pro texturování

```
vec4 texture(sampler* sampler,
    vec* coord [, float bias])
```

- *texture* má ve skutečnosti ještě nepovinný parametr, který ovlivňuje LOD

```
vec4 textureProj(sampler* sampler,
    vec* coord [, float bias])
```

- Dělí poslední souřadnicí před vzorkováním

```
vec4 textureProjLod(sampler* sampler,
    vec* coord, float lod)
```

- Kombinace *textureLod* a *textureProj*

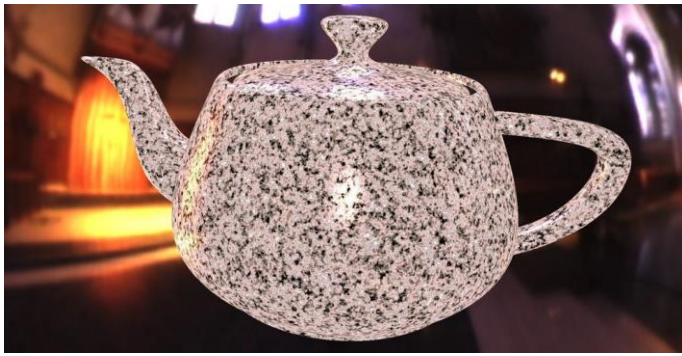
Další funkce pro texturování

- Existují další funkce
  - pro získání přímo daného texelu
  - pro získání velikosti textury
  - ...

Využití texturování

Textura definuje barvu objektu

ambientní a difusní složka



eraser85.wordpress.com

Textura definuje průhlednost

objektu, ovlivňuje alphu

pixelu

learnopengl.com

Využití texturování

- Ambientní a difusní složka

```
color =
  light_ambient * tex_color +
  light_diffuse * tex_color * Idiff +
  light_specular * mat_specular * Ispec;
```

- Alpha

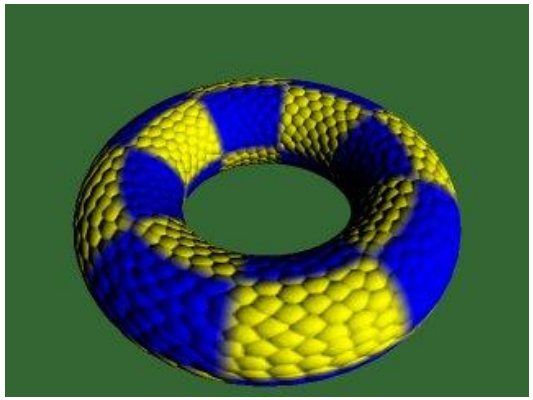
```
final_color = vec4(color, tex_alpha);
```

- Děravé objekty

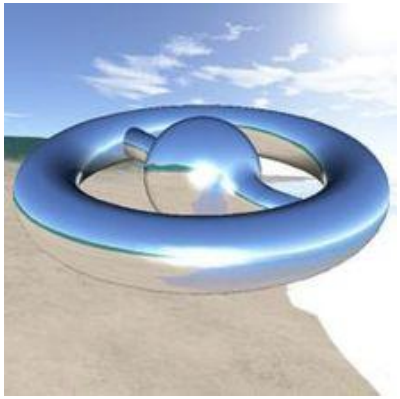
```
if (tex_alpha < 0.5)
    discard;
```

Využití texturování

- Dnes? Naprosto cokoliv:



paulsprojects.net



en.wikipedia.org

nvidia.com

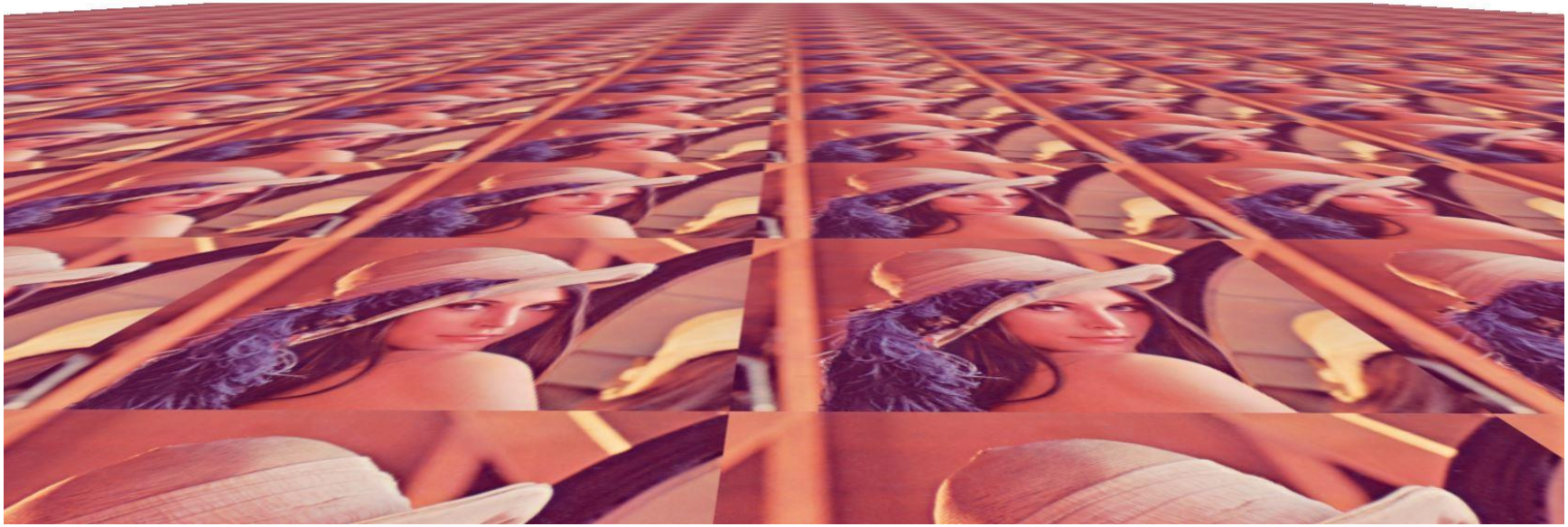
outerra.blogspot.cz

msdn.microsoft.com

Anisotropické vzorkování

- Rozšíření v OpenGL, ale je dostupné prakticky všude
- Řeší problém vzorkování textur, které jsou v jednom směru mnohem víc natažené než ve druhém směru
- Mipmapping vybere menší texturu, a tak jakoby „rozmazává“

Anisotropické vzorkování

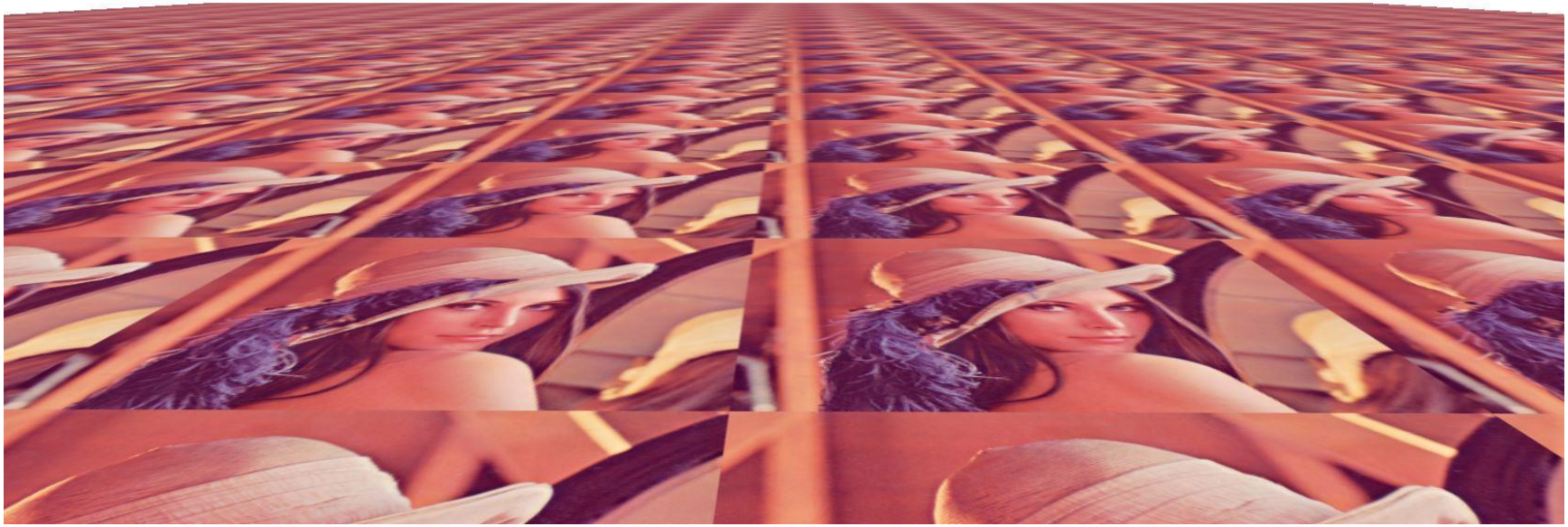


Anisotropické vzorkování

- Anisotropické vzorkování nastavujeme pomocí funkce *glTexParameterf* a parametru *GL_TEXTURE_MAX_ANISOTROPY_EXT*
- Hodnoty nabývají hodnot v intervalu [1, 16], kde 1 je „žádné anisotropické vzorkování“ a 16 je maximální hodnota, kterou většina grafických karet podporuje.

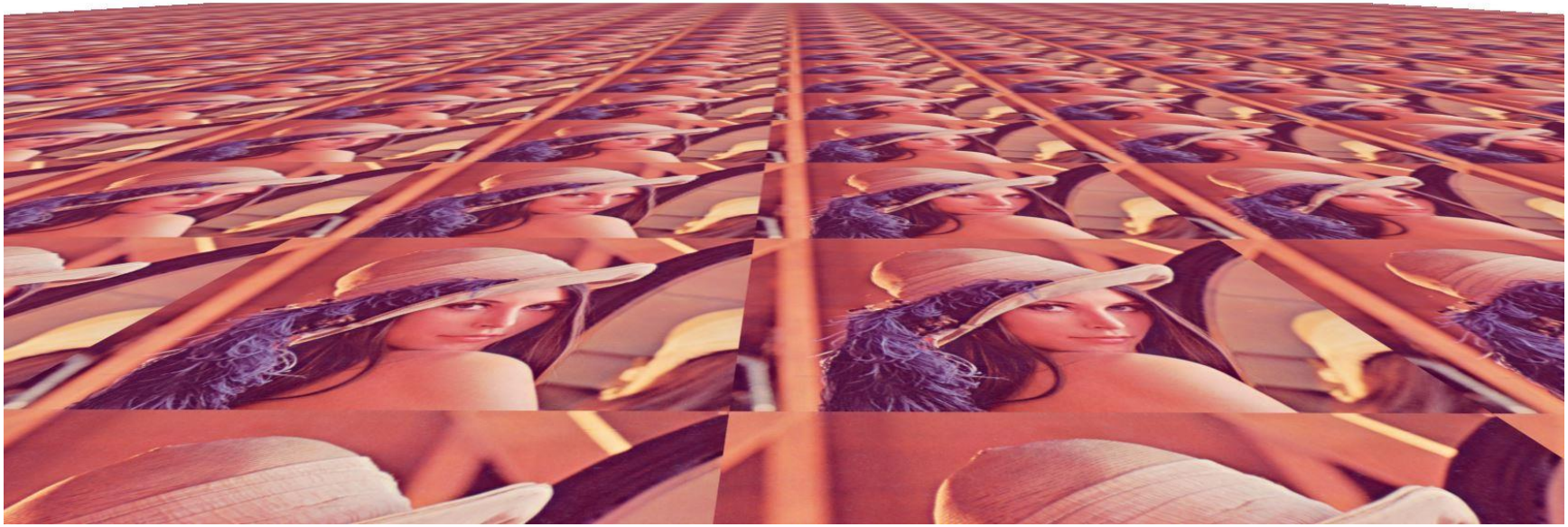
```
glTexParameterf(GL_TEXTURE_2D,
    GL_TEXTURE_MAX_ANISOTROPY_EXT, 8.0f);
```

Anisotropické vzorkování



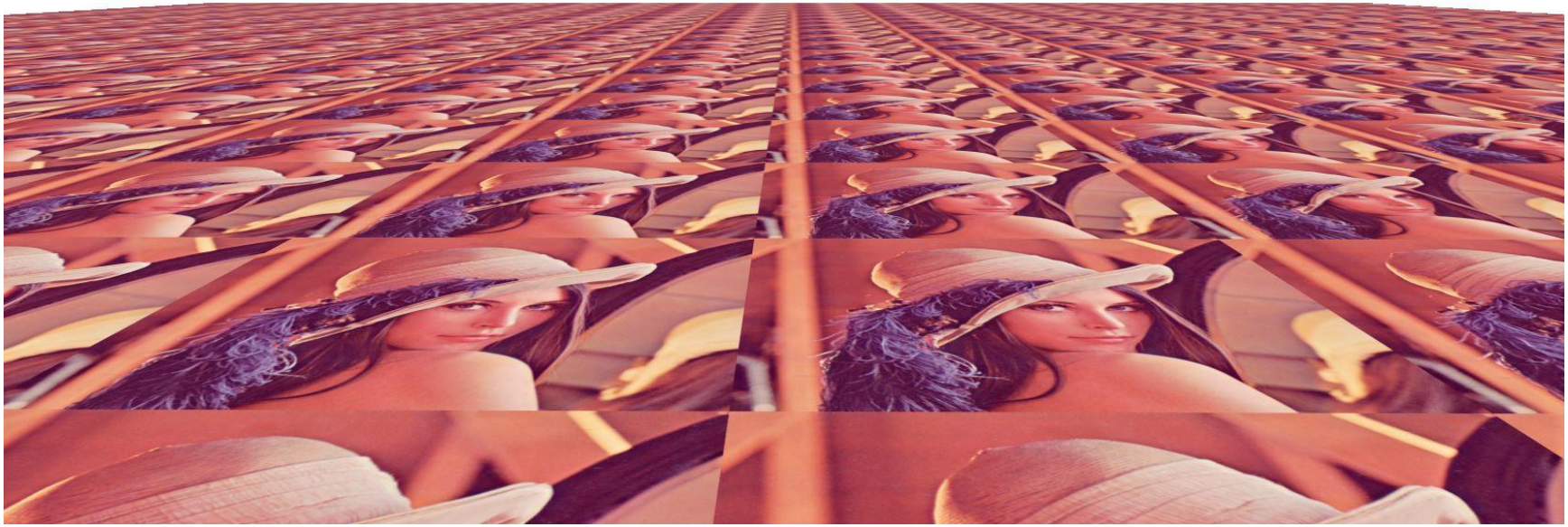
Faktor 1

Anisotropické vzorkování



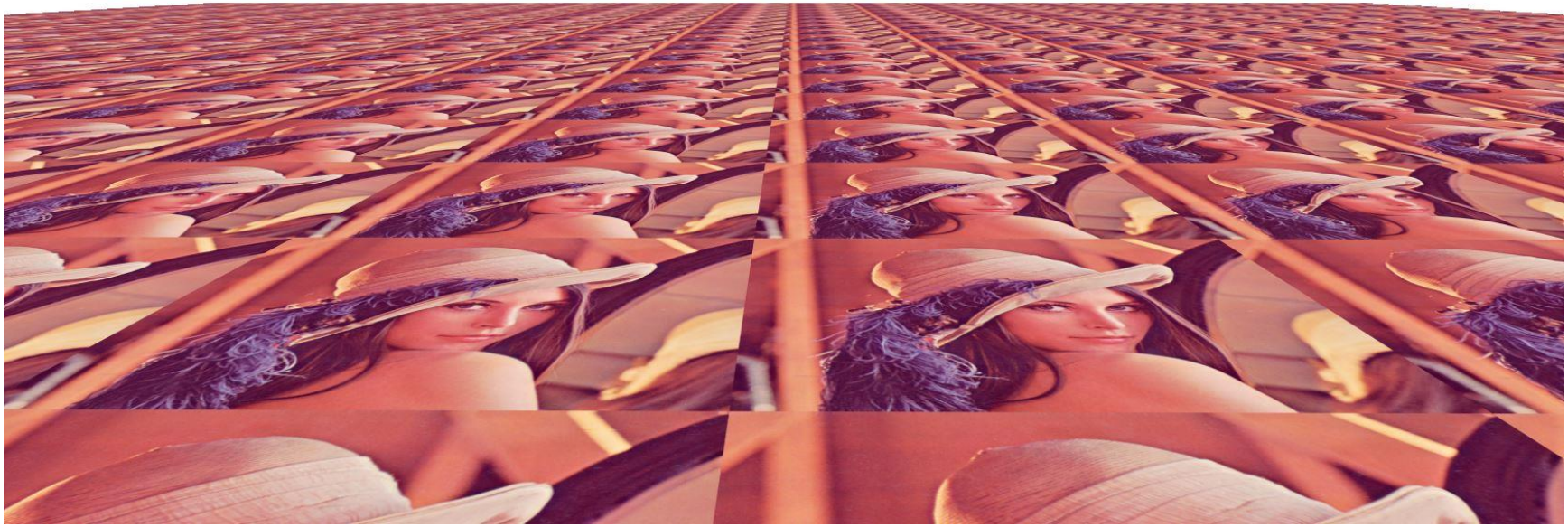
Faktor 2

Anisotropické vzorkování



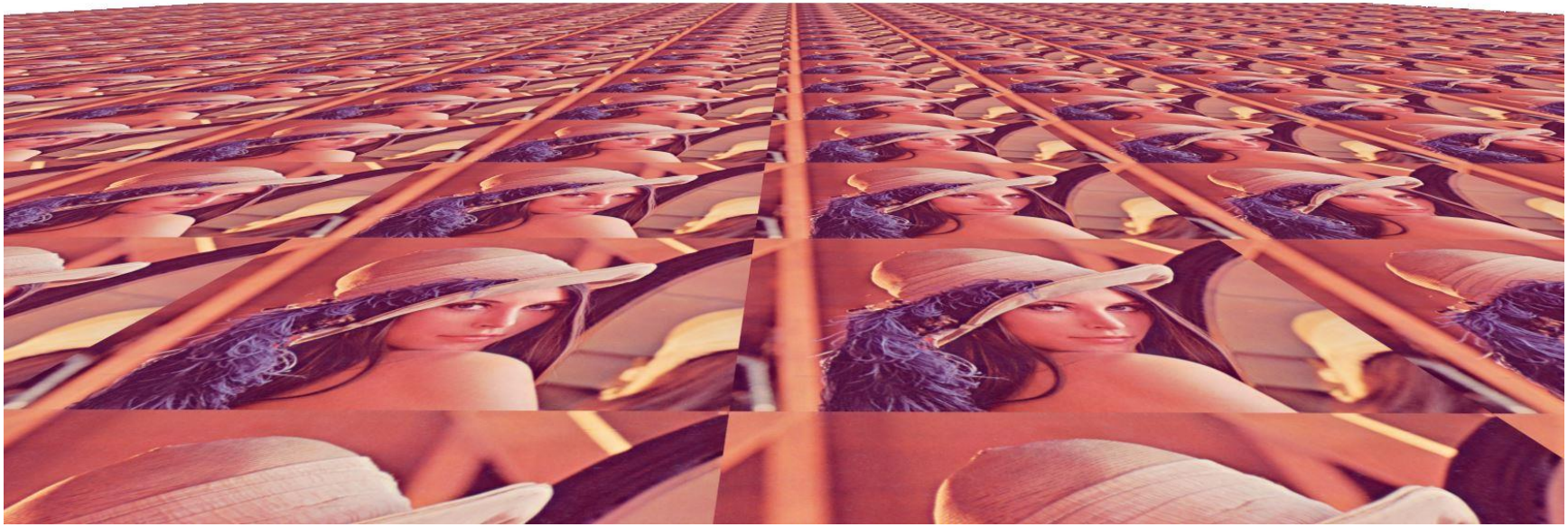
Faktor 4

Anisotropické vzorkování



Faktor 8

Anisotropické vzorkování



Faktor 16

Rozšíření OpenGL

- Umožňují, abychom mohli používat věci specifické pro nějaké grafické karty
- NV, AMD, INTEL / EXT / ARB
- Popis lze najít na internetu
- Počet dostupných rozšíření

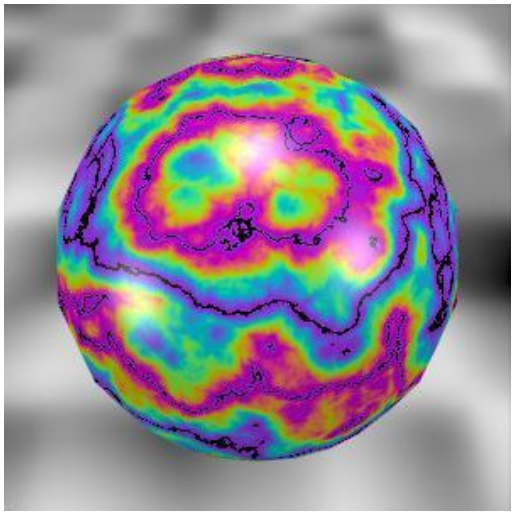
```
glGetIntegerv(GL_NUM_EXTENSIONS, &out);
```

- Jména jednotlivých rozšíření

```
glGetStringi(GL_EXTENSIONS, idx);
```

Generování tex. souřadnic

- Výpočet texturovací souřadnice na základě ostatních informací, nejčastěji pozice, normály a nějakých matic



www.okino.com

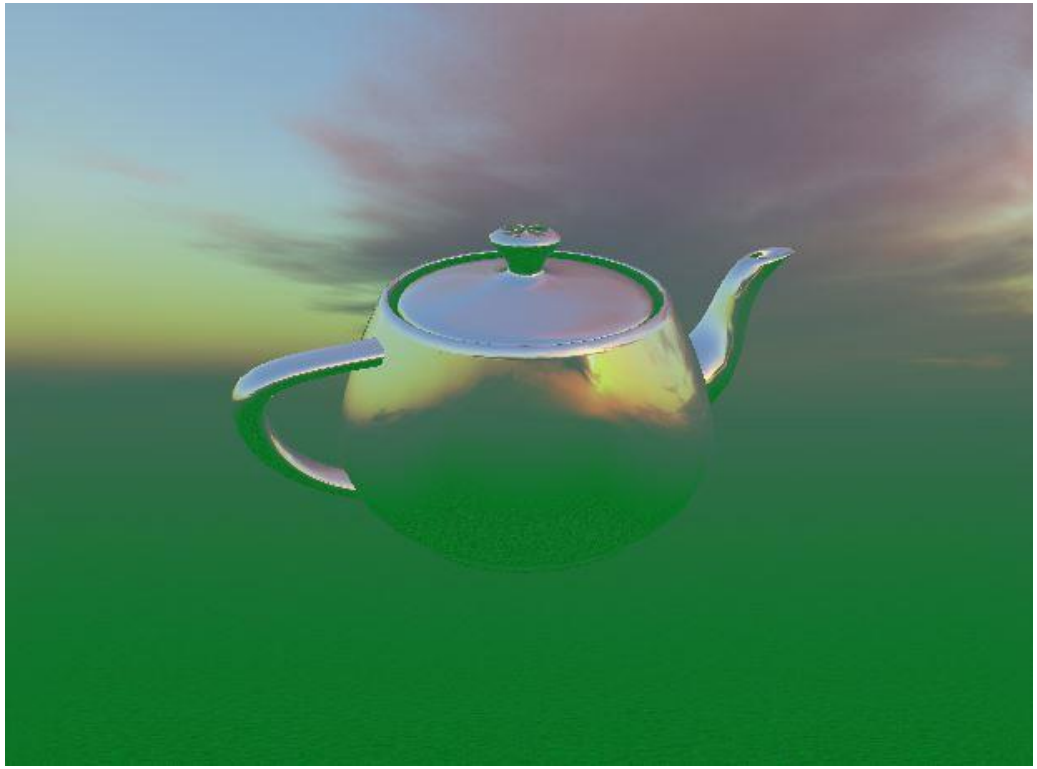
Lokální souřadnice

batman.wikia.com

Světové souřadnice

Environment mapping

- Jednoduchá technika pro zrcadlení okolních objektů



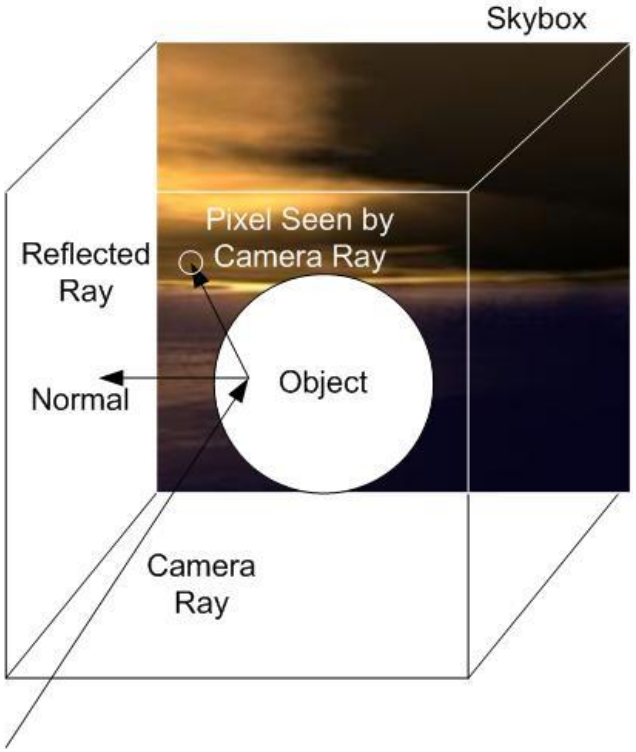
Environment mapping

- Odraz směru, ve kterém se pozorovatel dívá

$$I_{out} = I_{in} - 2(N \cdot I_{in})N$$

```
I_out = reflect(I_in, N)
```

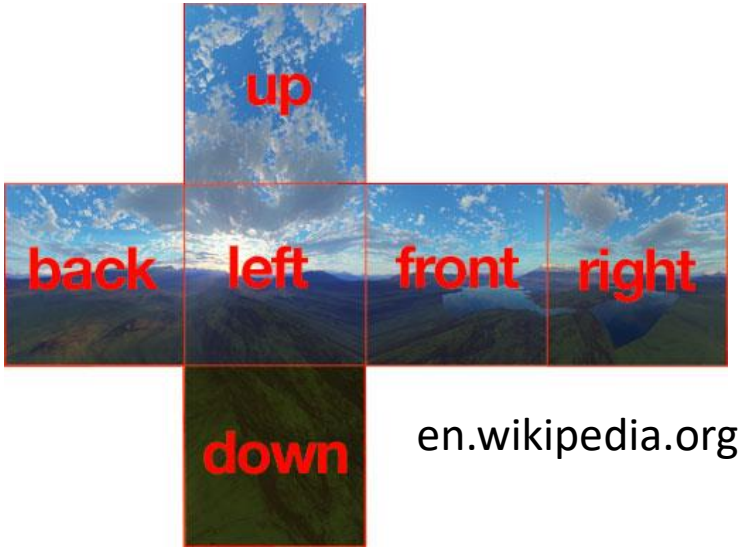
- Odražený směr se použije pro vzorkování cube textury



en.wikipedia.org

Cube textura

- Šest 2D textur v jednom objektu textury
- OpenGL používá typ *GL_TEXTURE_CUBE_MAP*
- Načítání pomocí *glTexImage2D*, první parametr je *GL_TEXTURE_CUBE_MAP_**



en.wikipedia.org

* jsou stěny krychle:

POSITIVE_X

…

NEGATIVE_Z

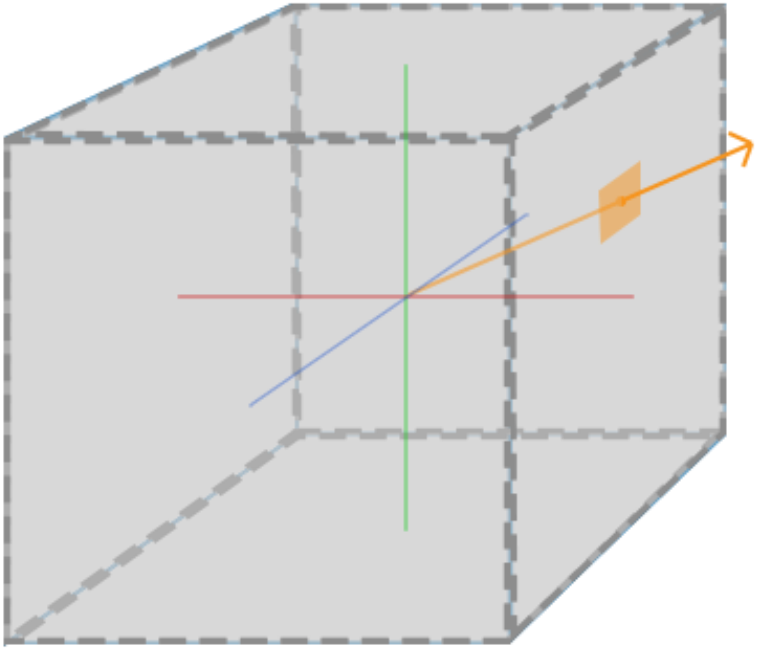
Cube textura

```
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGB,
    512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, data_px);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0, GL_RGB,
    512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, data_py);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0, GL_RGB,
    512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, data_pz);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0, GL_RGB,
    512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, data_nx);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, GL_RGB,
    512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, data_ny);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, GL_RGB,
    512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, data_nz);
```

- *width* = *height*, *internal_format* u všech stejný

Cube textura

- V GLSL se používá typ *samplerCube*
- Vzorkuje se použitím 3 souřadnic reprezentujících směr, kterým se dívám



learnopengl.com

Environment mapping

```
uniform samplerCube my_env;
...
vec3 reflected = reflect(incoming, normal);
vec3 color = texture(my_env, reflected);
...
```

```
...
glUniform1i(my_env_loc, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_CUBE_MAP, skybox);
glDraw*
...
```

Environment mapping

- Výhody:
  - rychlé
  - jednoduché
  - hezké
- Nevýhody:
  - okolí je „v nekonečnu“
  - objekt neodráží sebe sama

Kompletnost textury

- Zejména u mipmap a cube textury
  - Cube textura musí mít definovány obrázky všech stěn, jinak je nekompletní
  - Textura, u níž se používá pro filtrování mipmaping, musí mít definovány všechny levely do velikosti 1x1x1, jinak je nekompletní
- Pokud budeme v shaderu vzorkovat nekompletní texturu, získáme (0,0,0,1)

Kompletnost textury

- Pro mipmapping lze určit, které úrovně jsou definovány, a to pomocí parametrů *GL_TEXTURE_BASE_LEVEL* a *GL_TEXTURE_MAX_LEVEL*

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BASE_LEVEL, 1);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_LEVEL, 3);
```

- Lze tak použít mipmapping a nemít přitom definovány všechny obrázky

Komprimované textury

- Snaží se ušetřit množství paměti, kterou textura zabírá
- Dekomprimace probíhá až při vzorkování

Komprimační algoritmy

- Obrázek dělí do bloků (např. 4x4)
- Jednotlivé bloky jsou zkomprimovány a zabírají konstantní velikost, většinou 64 bitů, 128 bitů, apod.
- Liší se kvalitou a efektivitou komprese
- Pro RGB textury, alpha textury, normal mapy, HDR textury atd.

Komprimované textury

- V OpenGL pomocí speciálních konstant pro *internal_format*
- Lze načíst pomocí funkce *glTexImage**, která provede kompresi textury v okamžiku načtení obrázku

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGB, ...
```

- Nebo je možné načíst přímo zkomprimovaná data

Komprimované textury

```
void glCompressedTexImage2D(GLenum target,
  GLint level, GLenum internal_format,
  GLsizei width, GLsizei height,
  GLint border, GLsizei imageSize,
  const GLvoid *data)
```

- Slouží k načtení zkomprimovaných dat
- Parametry podobné funkci *glTexImage2D*

Formáty komprimovaných textur

- *S3TC*, dostupné přes rozšíření
  - patent, velmi používané, DXT1, DXT3, DXT5
- *RGTC*, od OpenGL 3.0
  - pro RED a RG textury
- *BPTC*, od OpenGL 4.2
- *ETC*, od OpenGL 4.3
- *ASTC*, nové rozšíření
  - „Adaptive Scalable Tex. Comp.“, snaží se nahradit ostatní

Komprimované textury

khronos.org

ASTC, 8 bpp

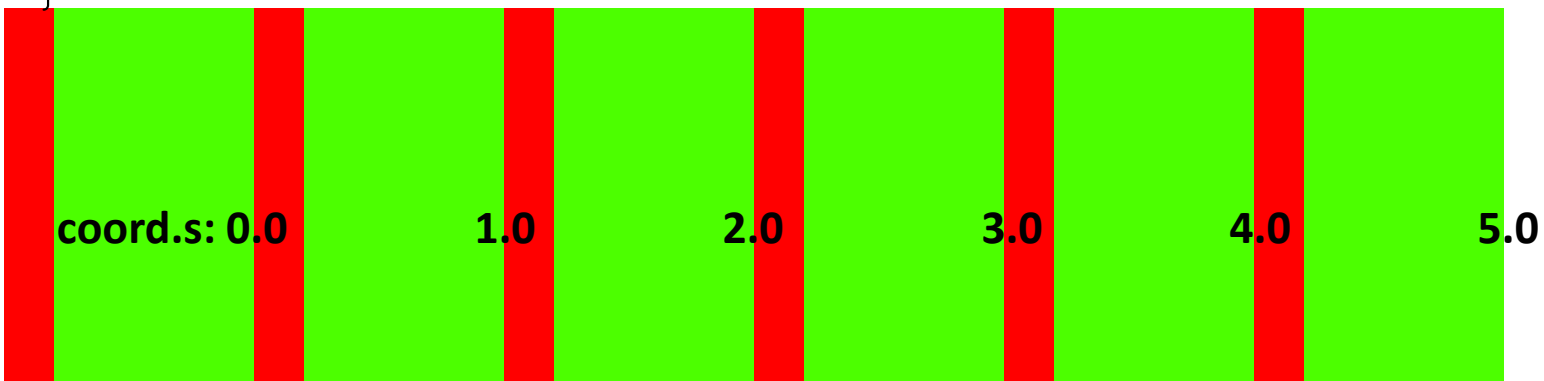
ASTC, 3.56 bpp

ASTC, 2 bpp

Procedurální textury

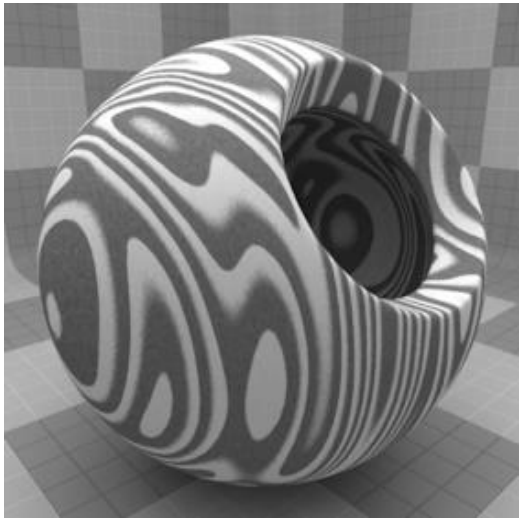
- Barva je spočítána procedurou
- Příklad: textura pruhů

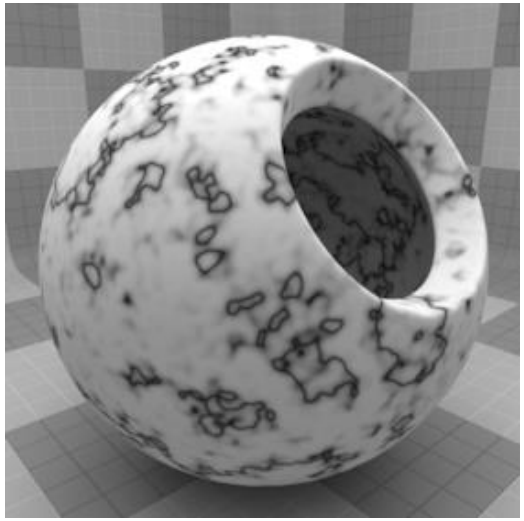
```
vec3 textureStripes(vec2 coord)
{
    if (fract(coord.s) < 0.2)
        return vec3(1.0, 0.0, 0.0);
    else
        return vec3(0.0, 1.0, 0.0);
}
```



Procedurální textury

- Funkce pro dřevo, mramor apod.
- Využívají funkce šumu a generátory náhodných čísel





zdroj: MODO

Procedurální textury

- Výhody:
  - nezabírají místo v paměti
  - jsou spojité (v 1D, 2D i 3D)
  - lze je libovolně zvětšovat
- Nevýhody:
  - Je nutné je spočítat

Sampler objekty

- Něco jiného než datový typ v GLSL!
- Od OpenGL 3.3
- Obsahuje některé parametry filtrování textur
  - *GL_TEXTURE_WRAP_** módy
  - Border color
  - *GL_TEXTURE_MIN/MAG_FILTER* pro filtrování
  - Hodnotu anisotropického filtrování

Sampler objekty

- Jsou to objekty, podobně jako buffery nebo textury

```
void glGenSamplers(GLsizei n,
    GLuint *samplers)
```

  - Vytvoří *n* sampler objektů

```
void glDeleteSamplers(GLsizei n,
   const GLuint * samplers)
```

  - Zruší sampler objekty

```
GLboolean glIsSampler(GLuint id)
```

  - Vrátí *true,* jestli daný sampler objekt existuje

Sampler objekty

```
void glSamplerParameter[if][v](
  GLuint sampler, GLenum pname, T param)
```

- *sampler* = sampler objekt
- *pname* = parametr, jako u *glTexParameter*
- *param* = hodnota, jako u *glTexParameter*

- Nastavuje parametry sampler objektu

Sampler objekty

```
void glBindSampler(
  GLuint unit, GLuint sampler)
```

- Naváže *sampler* na texturovací jednotku *unit*
- Odpovídající parametry u textur se přestanou používat a začnou se používat parametry sampleru
- Nastavením *sampler* na 0 se opět začnou používat parametry u textury

Sampler objekty

```
GLuint my_sampler;
glGenSamplers(1, &my_sampler);
glSamplerParameteri(my_sampler,
    GL_TEXTURE_WRAP_S, GL_REPEAT);
glSamplerParameteri(my_sampler, set other parameters);

...

glUniform1i(my_tex_loc, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, my_tex);
glBindSampler(0, my_sampler);
glDraw*
```

glBindAttribLocation

- Zapomněli jsme zmínit :-)

```
void glBindAttribLocation(GLuint program,
  GLuint index, const GLchar * name)
```

- Oznámí shader programu *program*, aby navázal vstupní atribut *name* na index *index*
- Je nutné zavolat před *glLinkProgram*
- Můžeme si zajistit, že všechny programy budou mít atributy na indexech, na kterých chceme

Framebuffer objekty

- Umožňují offscreen rendering a kreslení do textur
- Použití
  - postprocessing (bloom, HDR, …)
  - render to texture (shadows)
  - HD rendering
  - …

en.wikipedia.org

msdn.microsoft.com

Framebuffer objekty

- Opět to jsou to objekty, jako jiné

```
void glGenFramebuffers(GLsizei n,
    GLuint *framebuffers)
```

  - Vytvoří *n* framebuffer objektů

```
void glDeleteFramebuffers(GLsizei n,
   const GLuint *framebuffers s)
```

  - Zruší framebuffer objekty

```
GLboolean glIsFramebuffer(GLuint id)
```

  - Vrátí *true* jestli daný framebuffer objekt existuje

Framebuffer objekty

```
void glBindFramebuffer(GLenum target,
    GLuint framebuffer)
```

- *target* :
  - *GL_READ_FRAMEBUFFER* – čtu z něj data
  - *GL_DRAW_FRAMEBUFFER* – kreslím do něj
  - *GL_FRAMEBUFFER* – i čtu i kreslím
- *framebuffer* = objekt framebufferu, 0 je okno aplikace

Připojení textur

- Framebuffer objekt neobsahuje žádná data, je to kontejner, ke kterému se připojují textury
- 8 míst pro barvu: *GL_COLOR_ATTACHMENT0 – GL_COLOR_ATTACHMENT7*
- 1 místo pro hloubku: *GL_DEPTH_ATTACHMENT*
- 1 místo pro stencil: *GL_STENCIL_ATTACHMENT*
- Textury nemusí mít stejnou velikost, kreslí se do velikosti odpovídající průniku textur

Připojení textur

```
void glFramebufferTexture2D(GLenum target,
  GLenum attachment, GLenum textarget,
  GLuint texture, GLint level)
```

- Připne texturu k framebufferu
- Jedna textura může být připojena k více FBO
- *target* :
  - *GL_READ_FRAMEBUFFER* – k FBO, ze kterého čtu
  - *GL_DRAW_FRAMEBUFFER* – k FBO, do něhož kreslím
  - *GL_FRAMEBUFFER* = *GL_READ_FRAMEBUFFER*

Připojení textur

- *attachment* : *GL_COLOR_ATTACHMENTi*,
  GL_DEPTH_ATTACHMENT,
  GL_STENCIL_ATTACHMENT,
  GL_DEPTH_STENCIL_ATTACHMENT
- *target* : *GL_TEXTURE_2D*, nebo jedna ze stěn cube mapy *GL_TEXTURE_CUBE_MAP_**
- *texture* : objekt textury, kterou připínám
- *level* : level mipmapy, který chci připnout

Připojení textur

- Můžu připnout i 1D texturu i 3D texturu, i když framebuffer je vždy 2D

```
void glFramebufferTexture1D(GLenum target,
  GLenum attachment, GLenum textarget,
  GLuint texture, GLint level)
```

```
void glFramebufferTexture3D(GLenum target,
  GLenum attachment, GLenum textarget,
  GLuint texture, GLint level, GLint layer)
```

- *layer* = „plátek“ 3D textury, který chci připojit

Příklad

```
GLuint my_tex[3];
glGenTextures(3, my_tex);

glBindTexture(GL_TEXTURE_2D, my_tex[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8,
  2048, 2048, 0, GL_RGBA, GL_UNSIGNED_BYTE, nullptr);

glBindTexture(GL_TEXTURE_2D, my_tex[1]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_R32F,
  2048, 2048, 0, GL_RED, GL_FLOAT, nullptr);

glBindTexture(GL_TEXTURE_2D, my_tex[2]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH24_STENCIL8,
  2048, 2048, 0, GL_DEPTH_STENCIL, GL_UNSIGNED_INT_24_8,
  nullptr);
```

Příklad

```
GLuint my_fbo;
glGenFramebuffers(1, &my_fbo);

glBindFramebuffer(GL_FRAMEBUFFER, my_fbo);
glFramebufferTexture2D(GL_FRAMEBUFFER,
  GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, my_tex[0], 0);
glFramebufferTexture2D(GL_FRAMEBUFFER,
  GL_COLOR_ATTACHMENT1, GL_TEXTURE_2D, my_tex[1], 0);
glFramebufferTexture2D(GL_FRAMEBUFFER,
  GL_DEPTH_STENCIL_ATTACHMENT, GL_TEXTURE_2D,
  my_tex[2], 0);
```

Nastavení výstupu ze shaderů

```
GLint glGetFragDataLocation(GLuint program,
    const char *name)
```

- Podobná jako *glGetAttribLocation* nebo *glGetUniformLocation*
- Vrací index, na který je přiřazena daná výstupní proměnná
  - Pozor, *index* != *GL_COLOR_ATTACHMENTi*

Nastavení výstupu ze shaderů

```
void glBindFragDataLocation(GLuint program,
  GLuint index, const char *name)
```

- Vynutí index, jako *glBindAttribLocation*
- Musí se zavolat před *glLinkProgram*

Nastavení výstupu ze shaderů

```
void glDrawBuffers(GLsizei n,
    const GLenum *bufs)
```

- Nastaví framebuffer aktuálně navázanému na *GL_DRAW_FRAMEBUFFER*, který color attachment se má použít pro který index.
- *bufs* = pole s hodnotami *GL_COLOR_ATTACHMENTi* nebo *GL_NONE*
- *glDrawBuffer* nastavuje pouze první index

Příklad

```
out vec4 final_color;
out float final_brightness;

...

glBindFragDataLocation(my_program, 0, ”final_color”);
glBindFragDataLocation(my_program, 1, ” final_brightness”);

...

glBindFramebuffer(GL_FRAMEBUFFER, my_fbo);
GLenum bufs[] = {GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1};
glDrawBuffers(2, bufs);

...

glDraw*

...
// use textures my_tex for postprocessing
```

Parametry framebufferů

```
void glGetFramebufferAttachmentParameteriv(
  GLenum target, GLenum attachment,
  GLenum pname, GLint *params)
```

- Vrátí parametr framebufferu (příp. hl. okna)
- *target* = jako u *glFramebufferTexture2D*
- *attachment* :
  - u FBO *GL_COLOR/DEPTH/STENCIL_ATTACHMENTi*
  - u hl. okna *GL_FRONT/BACK/LEFT/RIGHT/DEPTH/STENCIL*

Parametry framebufferů

- *pname* = co chceme vědět, *GL_FRAMEBUFFER_ATTACHMENT_** :
  - **_RED/GREEN/BLUE/ALPHA/DEPTH/STENCIL_SIZE*
  - **_OBJECT_NAME* : id objektu navázané textury
  - **_TEXTURE_LEVEL* : navázaný level
  - **_CUBE_MAP_FACE* : navázaná stěna cube textury
- *params* : výstup

Renderbuffer objekty

- Podobně jako textury je lze připojit k framebuffer objektu
- Starší verze OpenGL nedovolují kreslit do textur některých formátů, ale do renderbufferů ano
  - Multisample textury od verze 3.2, čistě stencil textury od OpenGL 4.4
- Dnes už to jsou ale výjimky

Feedback loops

- Nesmíme číst z textury, do které zapisujeme
  - Textura nesmí být navázaná na nějakou texturovací jednotku
  - Nesmím pomocí funkce *glCopyTexImage** kopírovat data textury do sebe sama
- Výsledkem je „undefined behaviour“

Kompletnost framebufferu

- Pokud něco špatně nastavíme, framebuffer bude „nekompletní“
- Správně formáty na správných attachmentech
  - Barevné formáty u color attachmentů
  - Hloubka u depth attachmentu
  - Stencil u stencil attachmentu
- Některé kombinace vnitřních formátů u color attachmentů nemusí být podporované

Kompletnost framebufferu

```
GLenum glCheckFramebufferStatus(
    GLenum target)
```

- Zkontroluje stav framebufferu napojeného na *target*
- Vrací status:
  - *GL_FRAMEBUFFER_COMPLETE*
  - *GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT*
  - *GL_FRAMEBUFFER_UNSUPPORTED*
  - …

Kopírovaní mezi framebuffery

```
void glBlitFramebuffer(
 GLint srcX0, GLint srcY0, GLint srcX1, GLint srcY1,
 GLint dstX0, GLint dstY0, GLint dstX1, GLint dstY1,
 GLbitfield mask, GLenum filter)
```

- Kopíruje data z *GL_READ_FRAMEBUFFER* do *GL_DRAW_FRAMEBUFFER*
- *srcX0* … *dstY1* určují kopírovanou oblast
- *mask* určuje, které buffery se kopírují (jako u *glClear*
- *filter* = *GL_NEAREST*, *GL_LINEAR*

glReadBuffer

```
void glReadBuffer(GLenum mode)
```

- Určuje, ze kterého attachmentu číst data
- *mode* :
  - u FBO *GL_COLOR_ATTACHMENTi*
  - u hl. okna *GL_FRONT/BACK/LEFT/RIGHT*
- Ovlivňuje funkce *glCopyTexImage**, *glReadPixels*, *glBlitFramebuffer*, …

Separátní operace

- Některé operace je možné nastavit pro každý draw buffer index jinak
  - Jedná se o draw buffer index (ten, kam navazujeme výstup ze shaderů) a nikoliv color attachment!
  - Maskování zápisu do color bufferu
  - Blending

Separátní operace

```
void glColorMaski(GLuint buf,
  GLboolean red, GLboolean green,
  GLboolean blue, GLboolean alpha)
```

- Maska určující, jestli do daného draw buffer indexu kreslit nebo ne
- *buf* = index draw bufferu
- *red, green, blue, alpha = GL_TRUE / GL_FALSE*

- Funkce *glColorMask* ovlivňuje všechny buffery

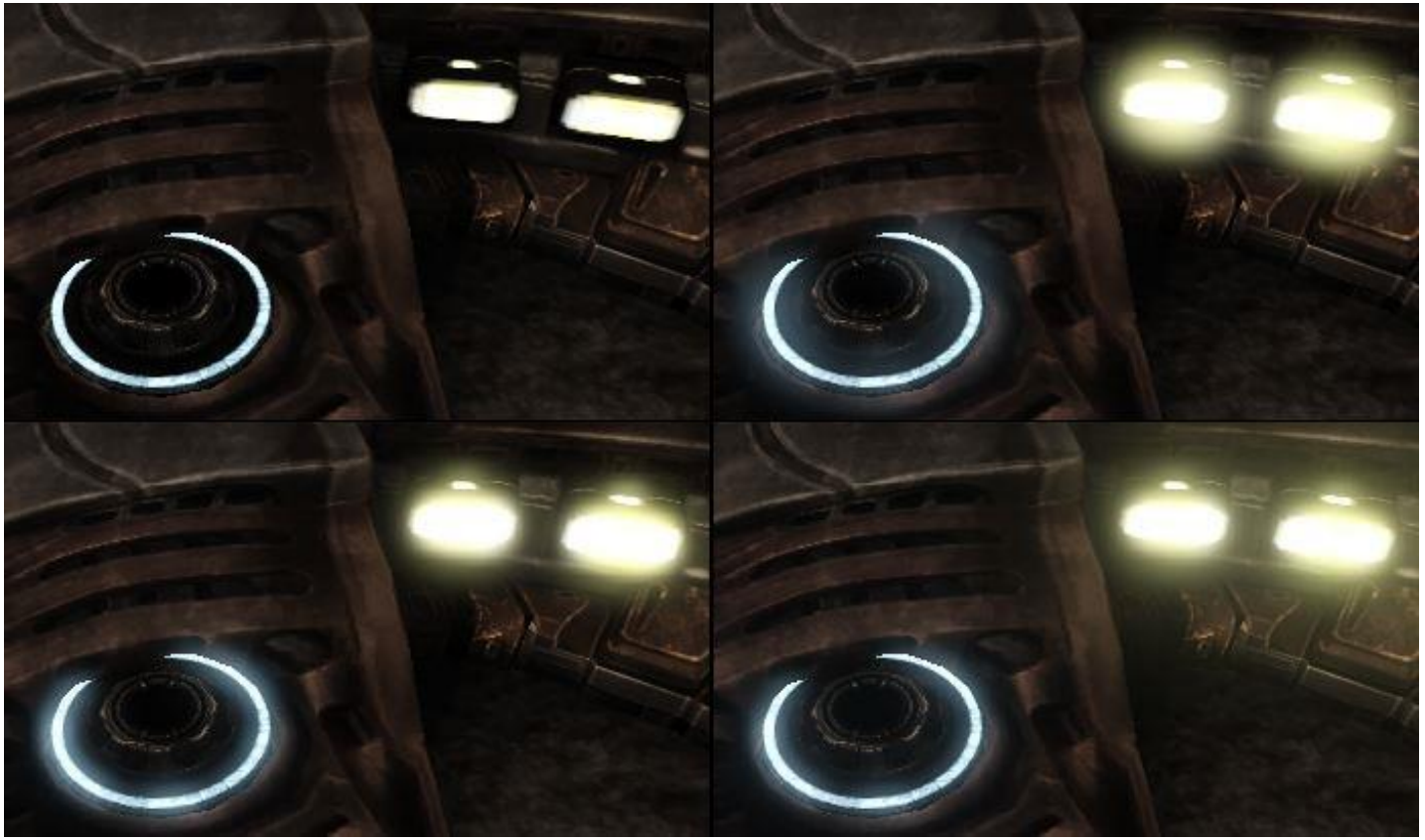
Separátní operace

```
void glEnablei(GLenum target, GLuint index)
void glDisablei(GLenum target, GLuint index)
```

- Pokud je *target GL_BLEND*, funkce povolí / zakáže míchání barev pro daný index
- *glEnable/glDisable(GL_BLEND)* ovlivňuje všechny buffery
- Od OpenGL 4.0 lze nastavit pro každý index nastavit i rovnice míchání a další parametry

Použití framebufferů

- Postprocessing (bloom efekt)



epicgames.com

Použití framebufferů

- Postprocessing (Screen Space Amb. Occlusion)

unity3d.com

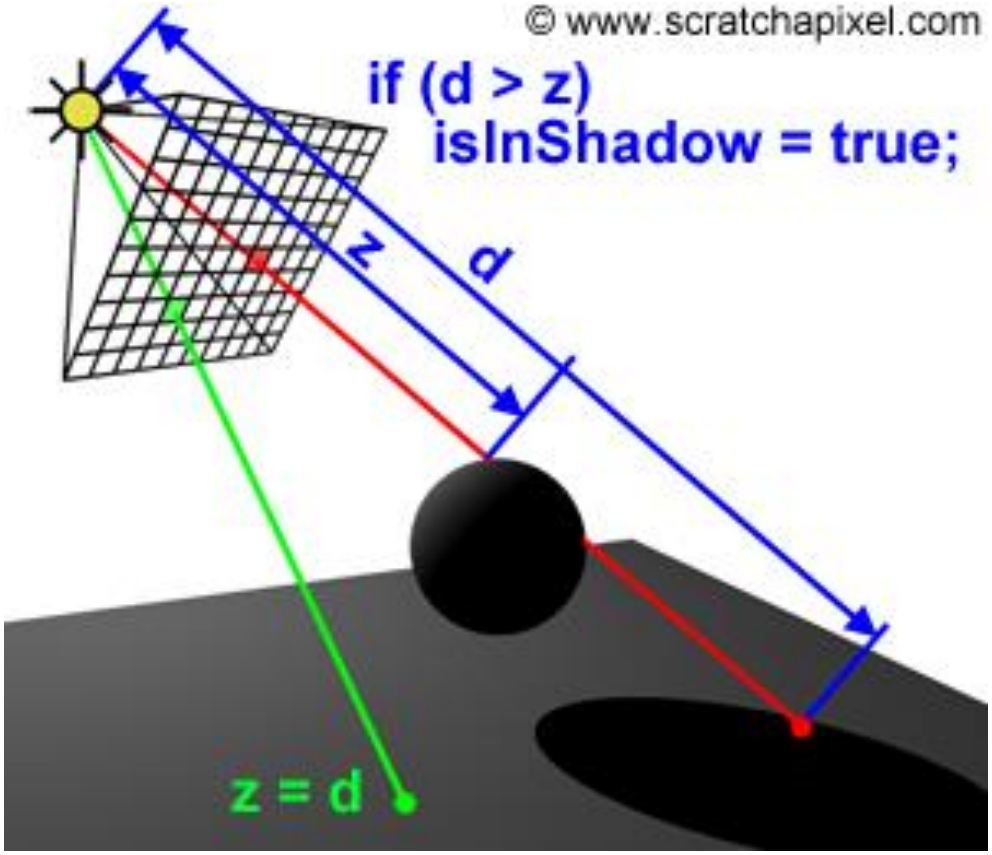
Použití framebufferů

- Render to texture

cryengine.com

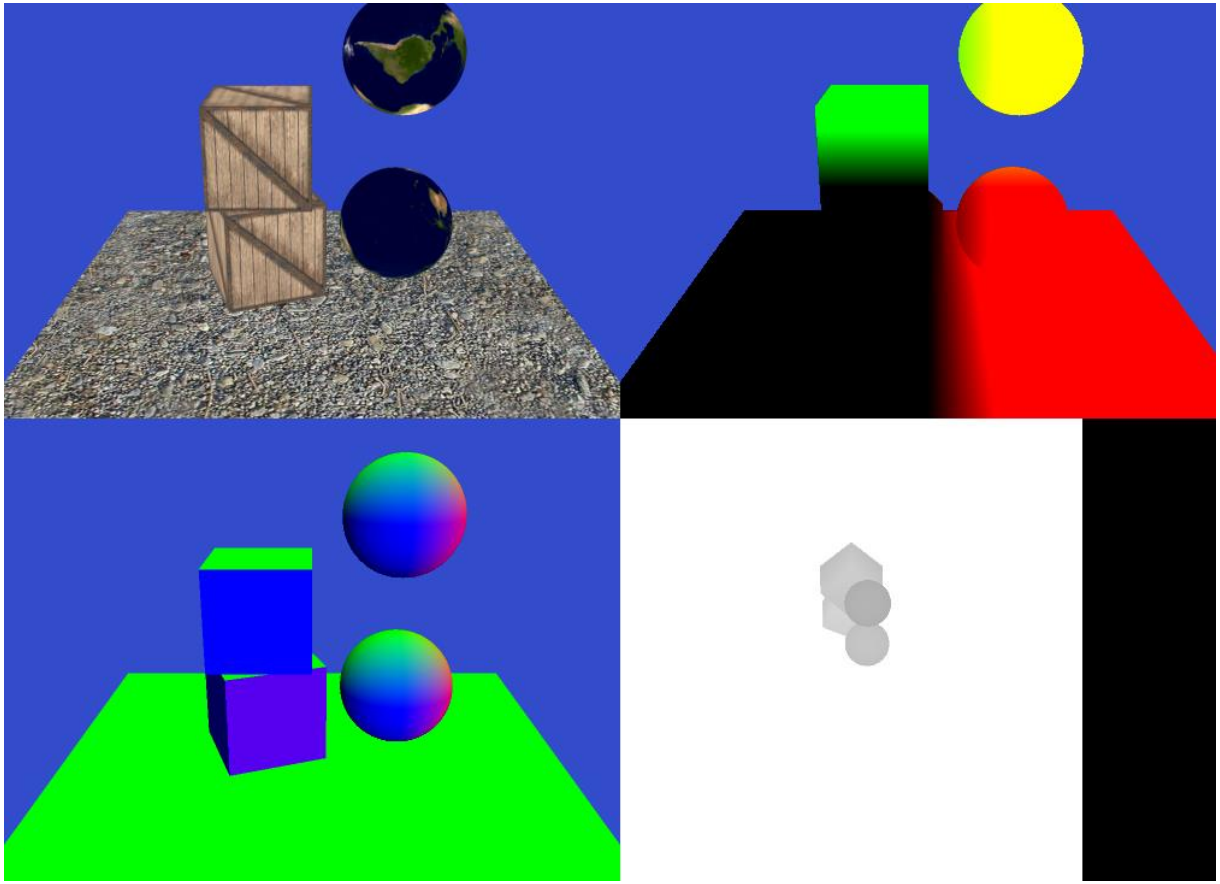
Použití framebufferů

- Stíny



Použití framebufferů

- Deferred shading



codinglabs.net