DCGI

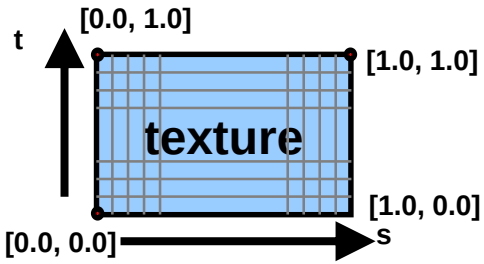
Cvičení IV

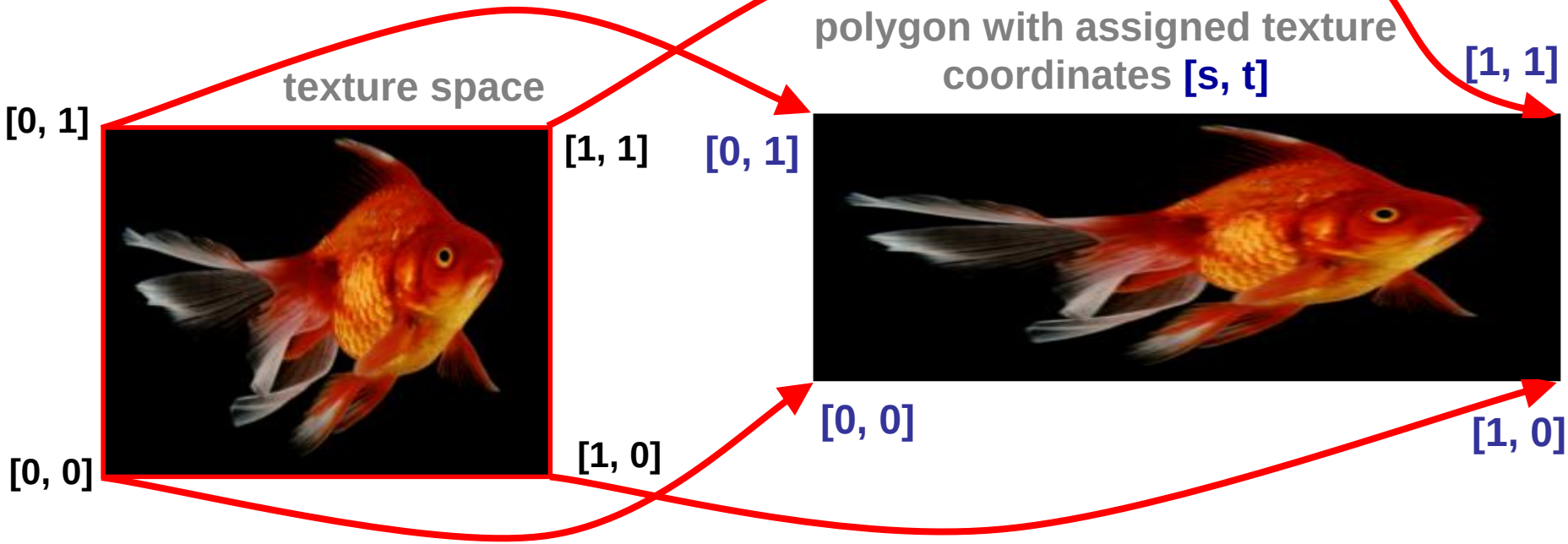
Mapování dynamických textur

Texture mapping

process of assigning texture coordinates to the polygon vertices

normalized texture coordinates
$\langle 0.0 \,..\, 1.0 \rangle^2$





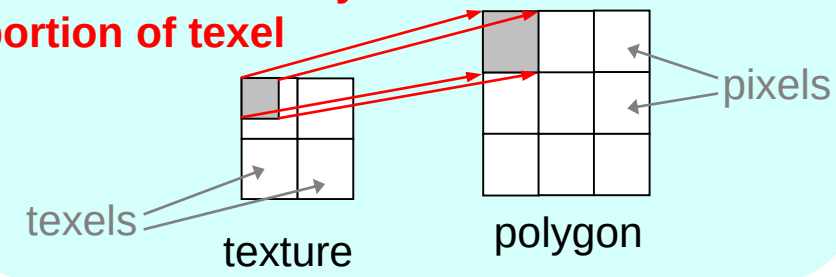
opakování

Texture filtering

DCGI

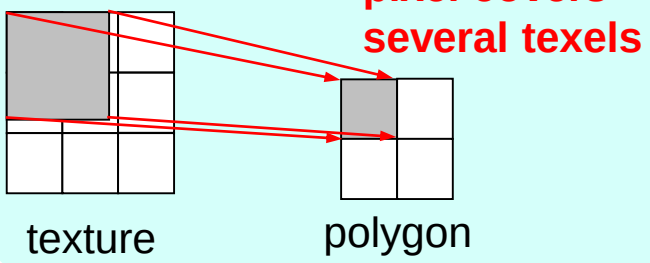
texture magnification

pixel is covered by portion of texel



texture minification

pixel covers several texels



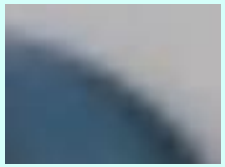
glTexParameteri(GL_TEXTURE_2D, GLenum **param,** GLenum **filter);**

param is **GL_TEXTURE_MAG_FILTER** magnification
or **GL_TEXTURE_MIN_FILTER** minification

filter

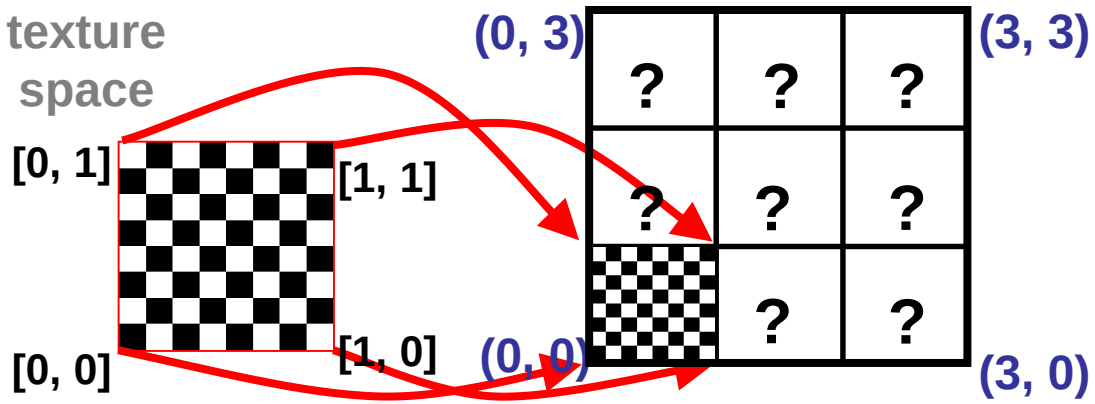
GL_NEAREST

GL_LINEAR

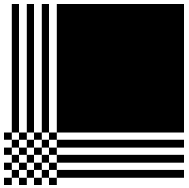


opakování

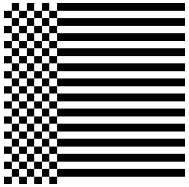
Repeating and clamping textures



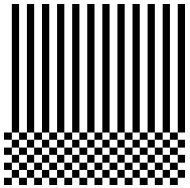
polygon with assigned texture coordinates



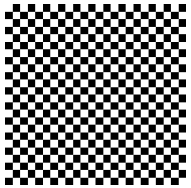
S: GL_CLAMP_TO_EDGE
T: GL_CLAMP_TO_EDGE



S: GL_CLAMP_TO_EDGE
T: GL_REPEAT



S: GL_REPEAT
T: GL_CLAMP_TO_EDGE



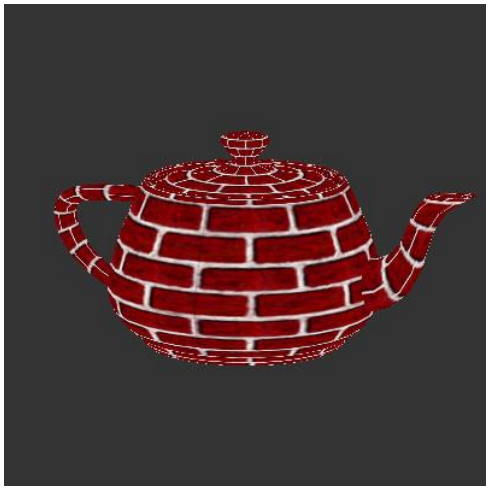
S: GL_REPEAT
T: GL_REPEAT

glTexParameter{if}(**GL_TEXTURE_2D**, GLenum ***pname***, TYPE ***param***);

- *pname* defines a texture coordinate to be wrapped
  GL_TEXTURE_WRAP_S, **_WRAP_T** or **_WRAP_R**
- *param* is **GL_CLAMP**, **GL_REPEAT, GL_CLAMP_TO_EDGE, or CLAMP_TO_BORDER**

Combining texture and surface color

RGB texture only



MODULATE

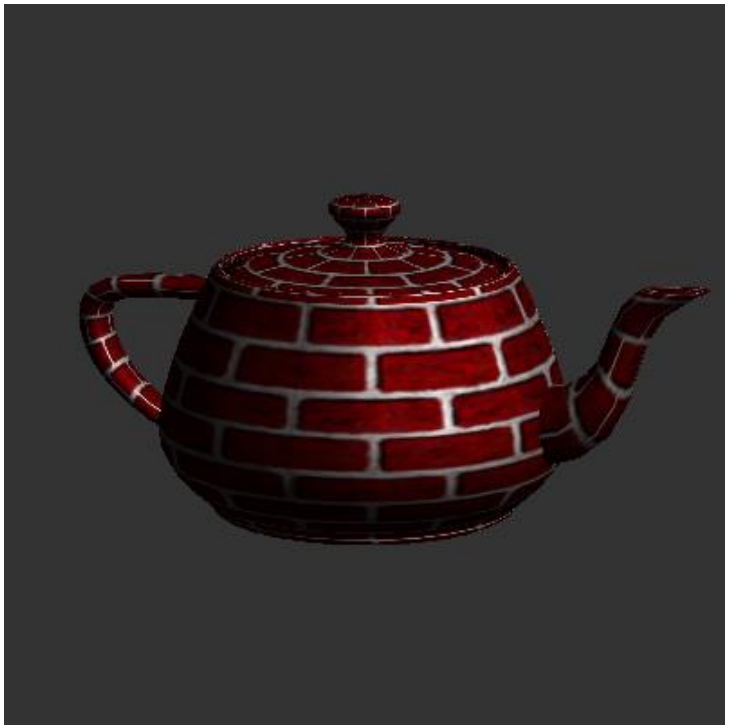
- object color is multiplied by a corresponding value in the texture

  ⇒ modulation is a good texture function for use with lighting

modulation

lit teapot

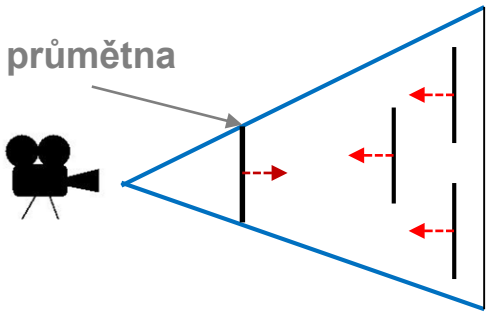
texture modulated by color

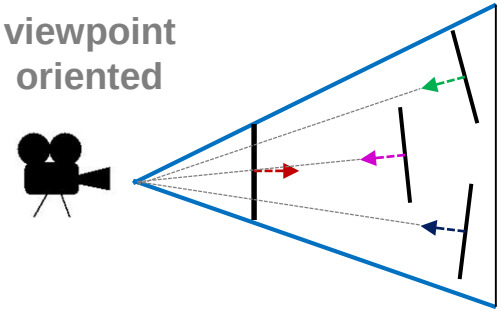


Billboard

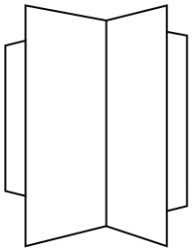
- billboard – otexturovaný polygon, který je vždy přední stranou natočen ke kameře
  - view plane aligned – billboard je rovnoběžný s průmětnou → stačí odstranit rotaci kamery
  - viewpoint oriented – každý billboard jinak natočený → změna bází

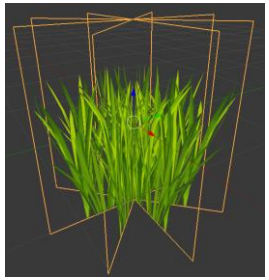
[Mark Harris]





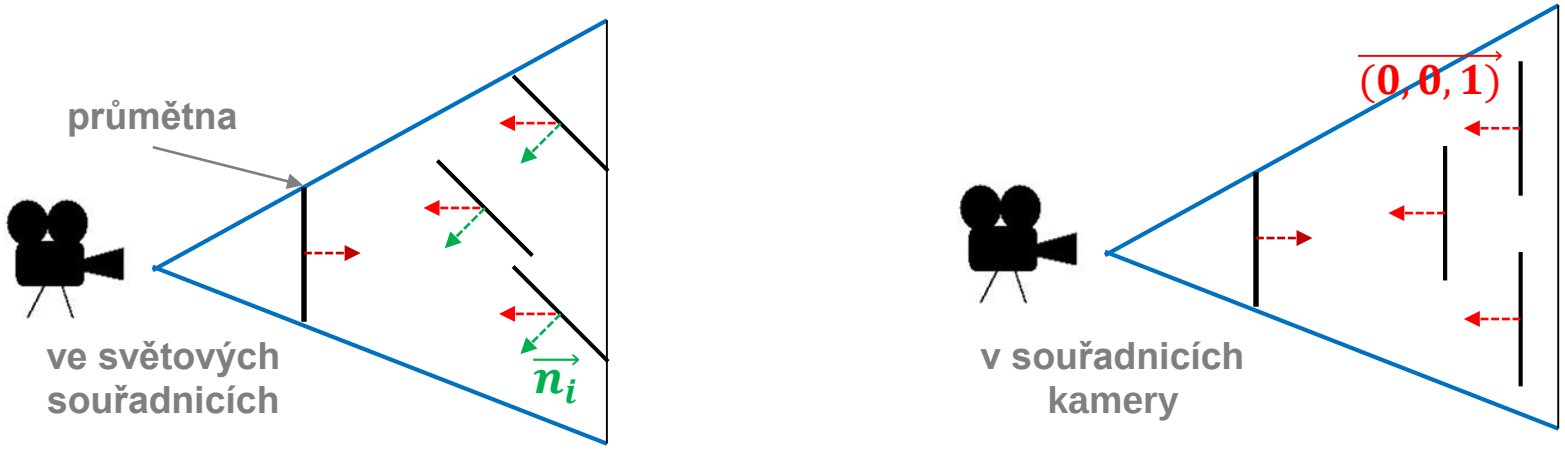
- pro zvýšení realismu lze využít více polygonů





Billboard – ostranění rotace

- billboard po transformaci do prostoru kamery musí mít normálu (0,0,1) → nutné billboard natočit ve světových souřadnicích tak, aby se eliminovala rotace kamery (→ směr $\overrightarrow{n_i}$)

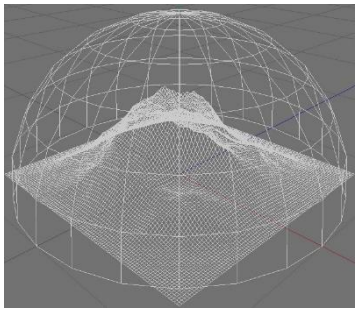


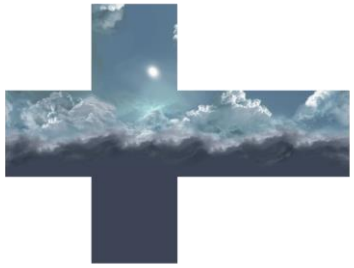

```
glm::mat4 billboardRotationMatrix = glm::mat4( // just take 3x3 rotation part of the view transform
    viewMatrix[0],
    viewMatrix[1],
    viewMatrix[2],
    glm::vec4(0.0f, 0.0f, 0.0f, 1.0f)
);
billboardRotationMatrix = glm::transpose(billboardRotationMatrix);      // inverse view rotation
glm::mat4 matrix = modelMatrix*billboardRotationMatrix;      // make billboard to face the camera
```

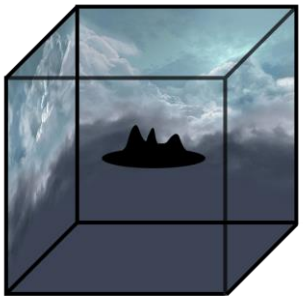
Skybox / skydome

- otexturovaný objekt nahrazující vzdálené okolí scény (mraky, hory, atd.)
- možnosti implementace
  - skydome – polokoule okolo scény
  - skybox – krychle okolo scény
  - obdélník přes celou obrazovku + cube mapa (varianta implementace skyboxu)

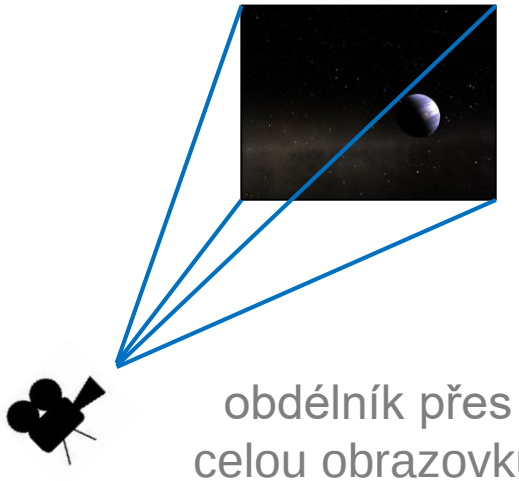
skydome







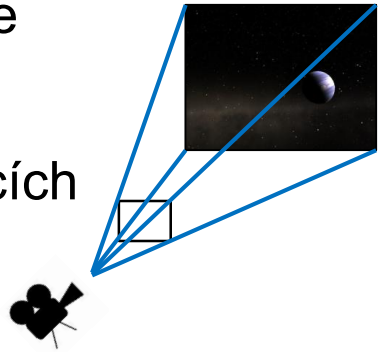
skybox



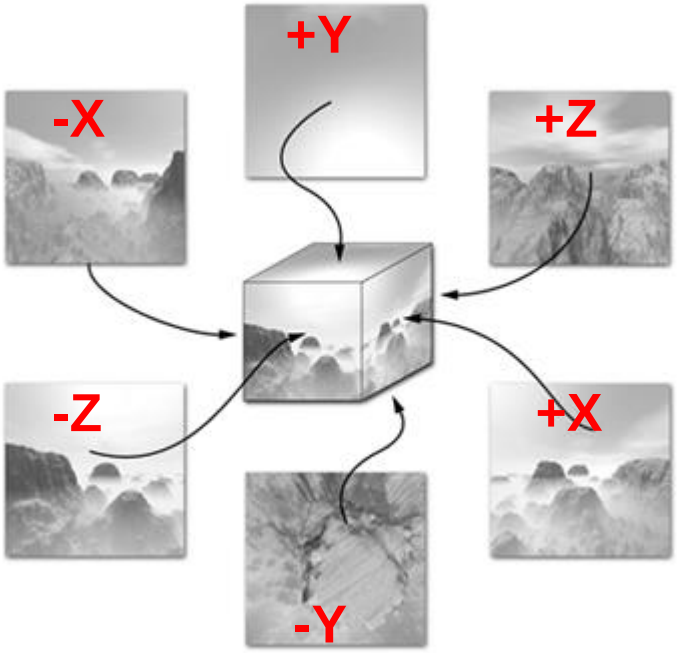
obdélník přes celou obrazovku

Skybox v asteroidech – úloha 1

- vykreslení obdélníku přes celou obrazovku na far plane
- textura uložena v cube mapě
- texturovací souřadnice vrcholů ve světových souřadnicích určeny inverzí transformačního řetezce



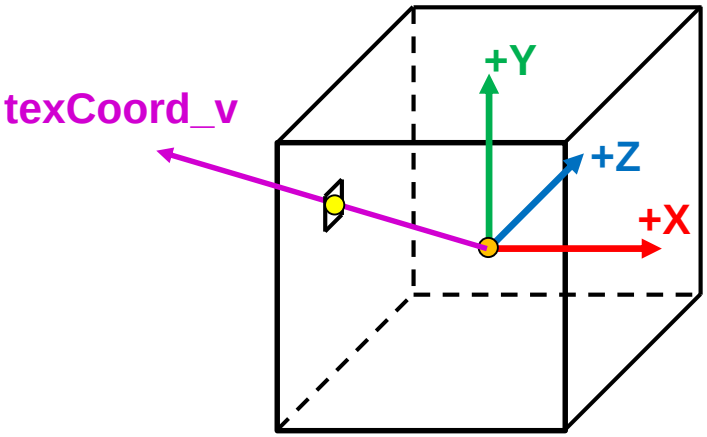
cube map = 6x 2D textura



uložení textur pro jednotlivé strany cube mapy

Cube map a její vzorkování

- cube map vzorkována pomocí směru (vektor) → **texCoord_v**
- vyžaduje speciální sampler → samplerCube
- místo kde směr **texCoord_v** protne cube mapu určuje barvu

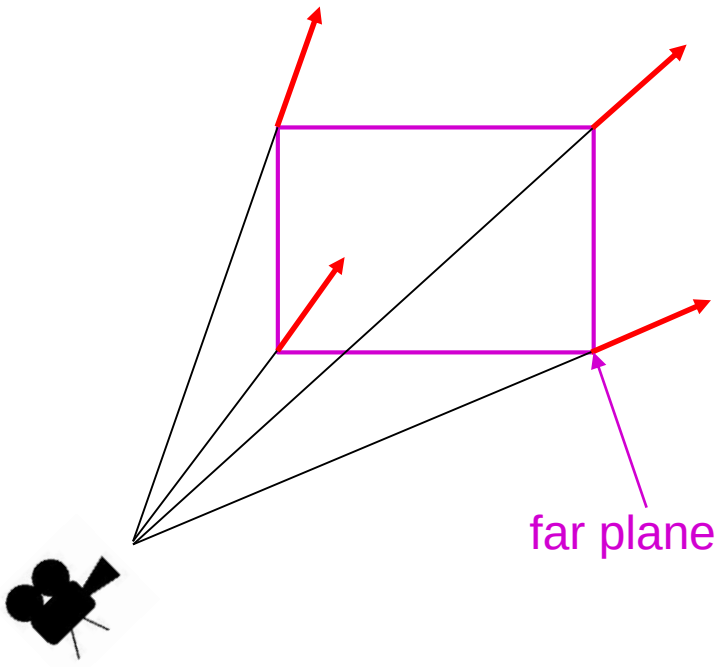


vzorkování cube map

```
#version 140                                    fragment shader

uniform samplerCube skyboxSampler;

// texturovací souřadnice fragmentu interpolovány z vrcholů
in vec3 texCoord_v;

// výsledná barva fragmentu
out vec4 color_f;

void main() {
  color_f = texture(skyboxSampler, texCoord_v);
}
```

Jak určit texturovací souřadnice rohů?

- vykreslení obdélníku přes celou obrazovku na **far plane**
- rohům přiřazeny **pozice v NDC** (normalizované souřadnice)
- **směry v rozích** ve světových souřadnicích pro vzorkování cube mapy jsou vypočítány inverzí části transformačního řetězce NDC souřadnic



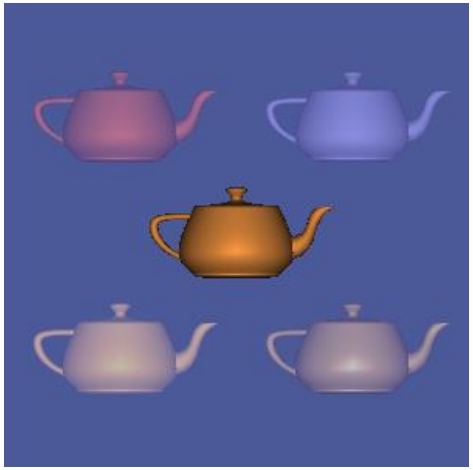

```
static const float screenCoords[] = { // NDC souřadnice rohů
    -1.0f, -1.0f,
     1.0f, -1.0f,
    -1.0f,  1.0f,
     1.0f,  1.0f
};
```

vertex shader

```
#version 140

uniform mat4 inversePVmatrix;
in vec2 screenCoord;
out vec3 texCoord_v;

void main() {
  vec4 farplaneCoord = vec4(screenCoord, 0.9999, 1.0);
  vec4 worldViewCoord = inversePVmatrix * farplaneCoord;
  texCoord_v = worldViewCoord.xyz / worldViewCoord.w;
  gl_Position = farplaneCoord;
}
```

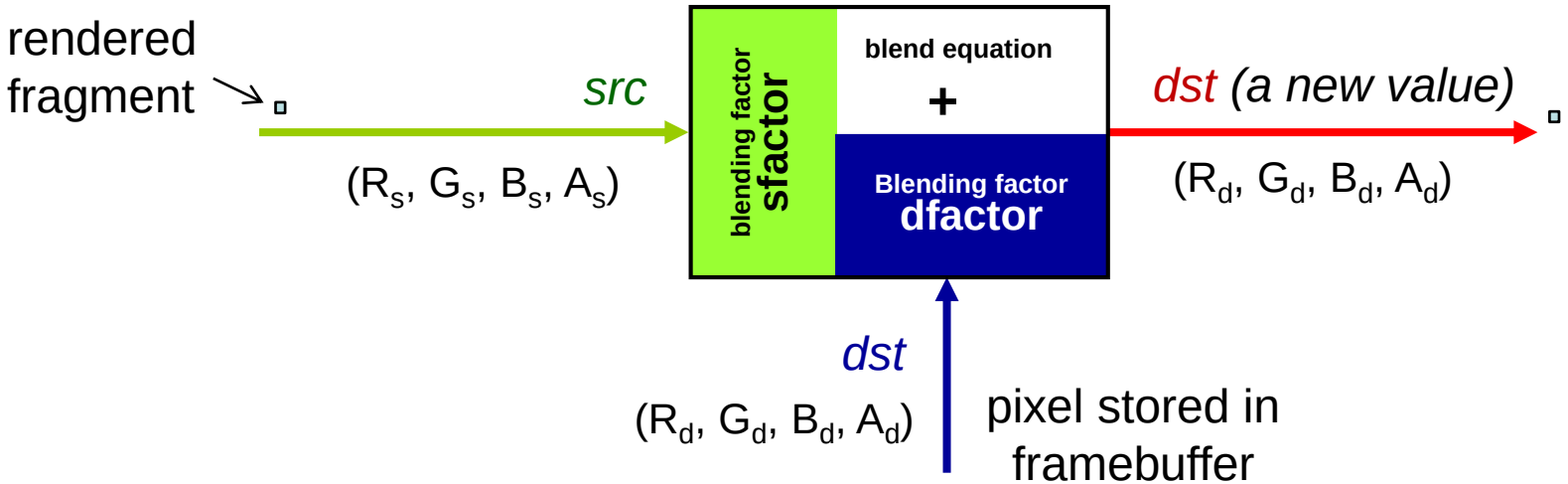
Blending (*míchání barev, α-míchání*)

Blending = mixing color of incoming (*src*) fragment with pixel color in the frame buffer (*dst*)



Enable blending `glEnable(GL_BLEND);`

Disable blending `glDisable(GL_BLEND);`



Setup of blending factors

DCGI

glBlendFunc(GLenum **sfactor,** GLenum **dfactor);**

Use **symbolic constants** to set up blend factors **sfactor** (S_R,S_G,S_B,S_A) and **dfactor** (D_R,D_G,D_B,D_A) for the fragments.

blend function parameter	computation of blending factors (f_R, f_G, f_B, f_A)
GL_ZERO	(0, 0, 0, 0)
GL_ONE	(1, 1, 1, 1)
GL_DST_COLOR	(R_d, G_d , B_d, A_d)
GL_SRC_COLOR	(R_S, G_S, B_S, A_S)
GL_ONE_MINUS_DST_COLOR	(1, 1, 1, 1) – (R_d, G_d, B_d, A_d)
GL_ONE_MINUS_SRC_COLOR	(1, 1, 1, 1) – (R_S, B_S, B_S, A_S)
GL_SRC_ALPHA	(A_S, A_S, A_S, A_S)
GL_ONE_MINUS_SRC_ALPHA	(1, 1, 1, 1) – (A_S, A_S, A_S, A_S)
GL_DST_ALPHA	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	(1, 1, 1, 1) – (A_d, A_d, A_d, A_d)
GL_SRC_ALPHA_SATURATE	(f, f, f, 1); f = min(A_s, 1 – A_d)

Úlohy

- mapování dynamické textury exploze na billboard → klávesa “e”
- mapování textury s textem “GAME OVER” tak, aby se posouvala zleva doprava → klávesa “g”

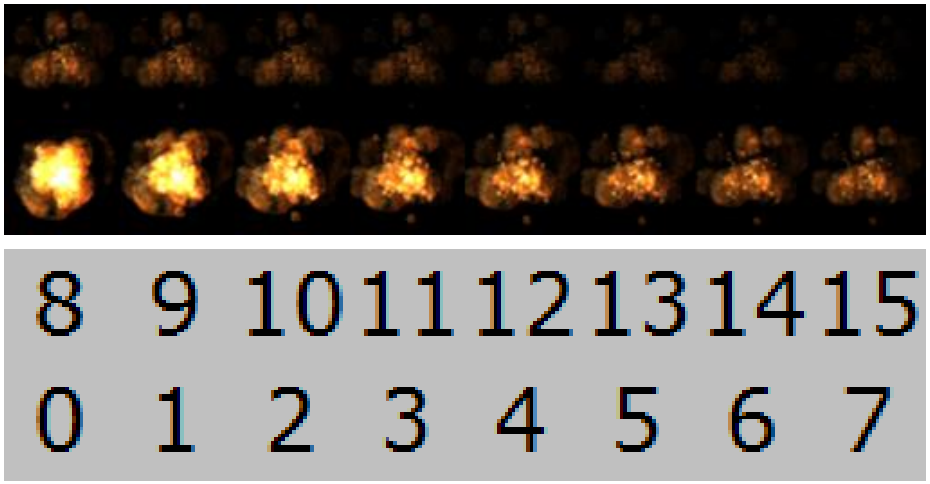
3 kroky společné pro obě úlohy

- počáteční mapování textury na geometrii – soubor data.h
- přemapování souřadnic ve vertex či fragment shaderu
- vykreslení se zapnutým mícháním barev

Úloha 2 - exploze

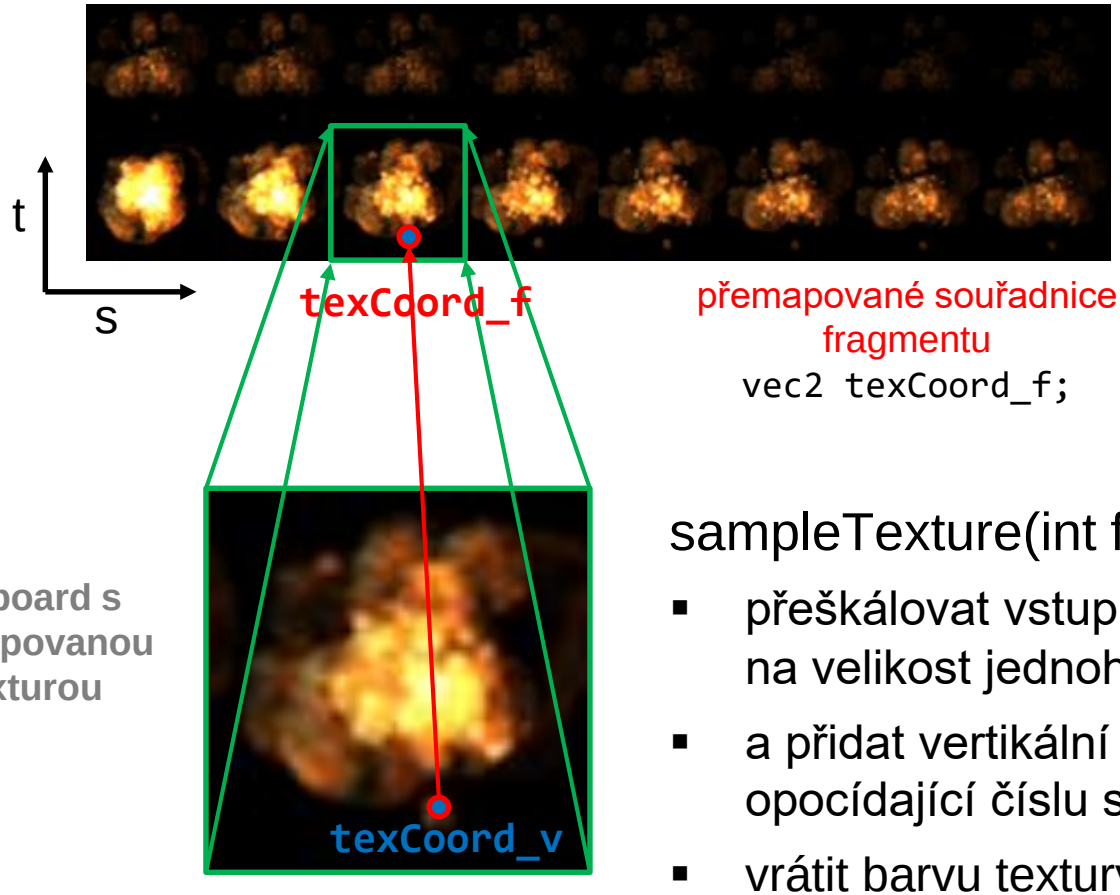
mapování dynamické textury exploze na billboard → klávesa “e”

- počáteční mapování – textura roztažena na celý billboard
  → doplnit texturovací souřadnice v poli `explosionVertexData`
- layout textury definován uniformem ivec2 pattern = ivec2(8, 2);
- vyříznutí správného snímku a roztažení na celý billboard
  → sampleTexture(int frame) ve fragment shaderu explosion.frag



- vykreslení billboardu se zapnutým mícháním barev – míchání v poměru 1:1 (černá barva v texture bude nahrazena barvou pozadí)

Vyříznutí správného snímku ze sekvence



realizace ve fragment shaderu explosion.frag

přemapované souřadnice fragmentu

```
vec2 texCoord_f;
```

billboard s namapovanou texturou

vstupní souřadnice fragmentu

```
smooth in vec2 texCoord_v;
```

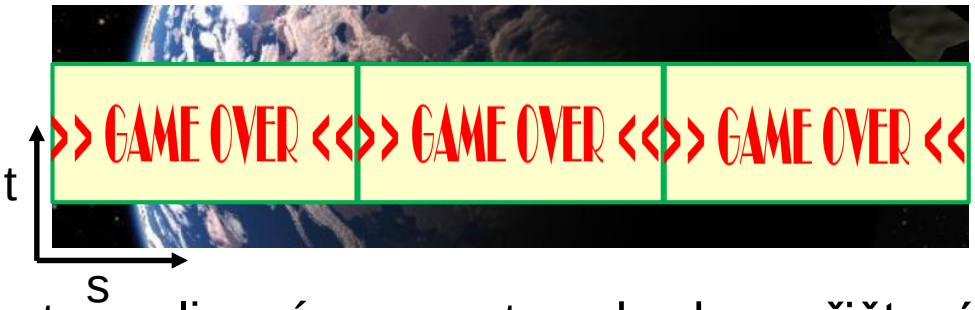
sampleTexture(int frame)

- přeškálovat vstupní souřadnice **texCoord_v** na velikost jednoho políčka
- a přidat vertikální a horizontální posun opocídající číslu snímku
- vrátit barvu textury odpovídající vypočítaným souřadnicím **texCoord_f**
  → volání funkce **texture**(…)

Úloha 3 – posouvající se textura

mapování textury s textem "GAME OVER" tak, aby se posouvala zleva doprava → klávesa "g"

- počáteční mapování – textura roztažena na celý billboard
  → doplnit texturovací souřadnice v poli `bannerVertexData`

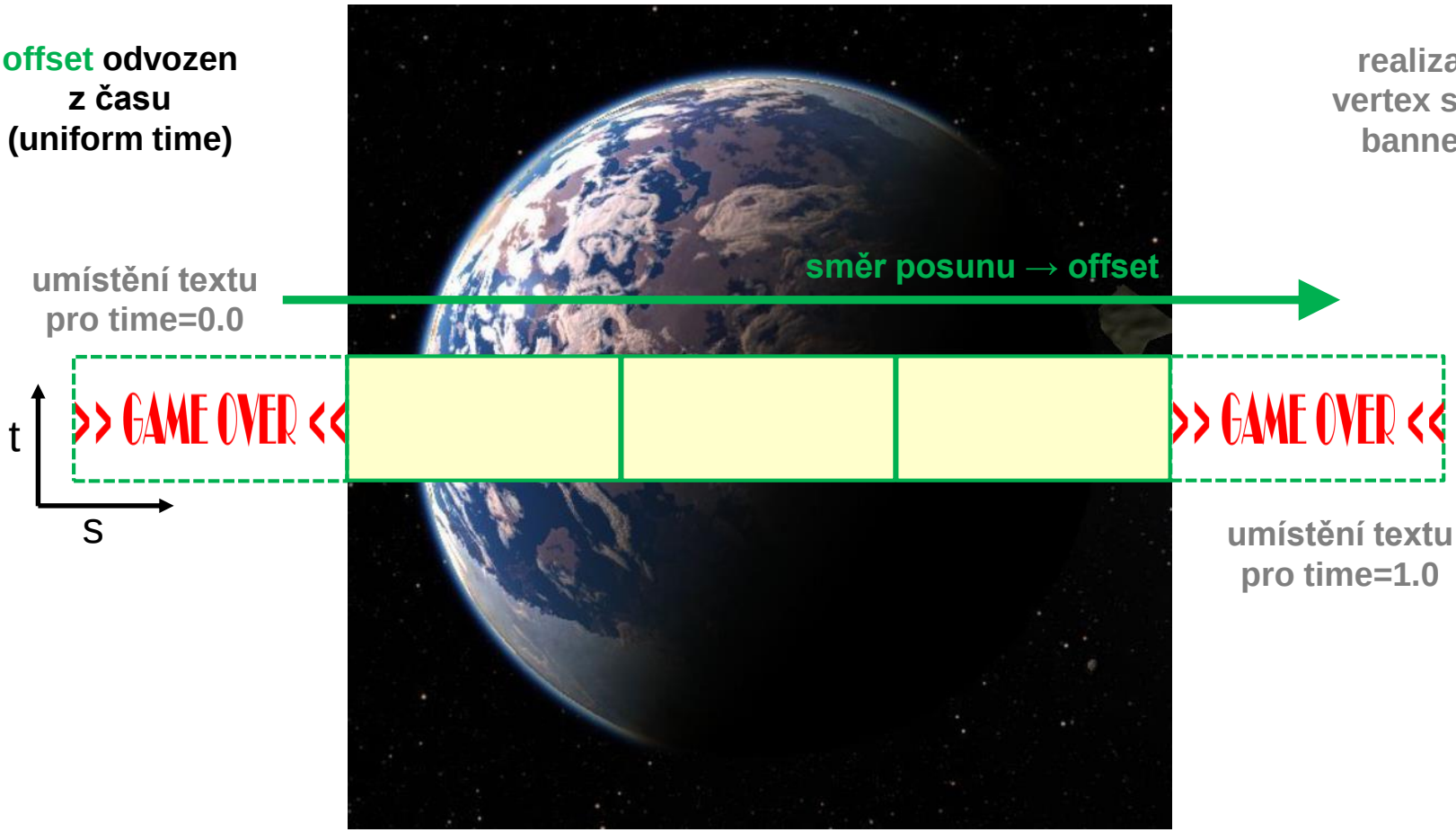


- posun textu realizován ve vertex shaderu přičtením offsetu měnícího se v závislosti na čase

- textura obsahuje alfa kanál
  → míchání dle alfa složky kresleného fragmentu v poměru

  SRC_ALPHA : (1-SRC_ALPHA)

Jak posouvat texturu?



- pro výsledné zpomalení času lze využít proměnnou localTime (až po implementaci správného posunu)
- **Pozor**: nutno zakázat opakování textury v horizontálním směru (texturovací souřadnice s)

Užitečné rady

- funkce GLSL floor() → vrací celočíselnou část reálného čísla
- při mapování textury výbuchu lze pro lepší kontrolu správnosti mapování využít texturu s čísly snímků → stačí odkomentovat řádek se správným jmeném souboru textury

```
// const char* EXPLOSION_TEXTURE_NAME = "data/explode.png";
const char* EXPLOSION_TEXTURE_NAME = "data/digits.png";
```

- soubor README.txt (*součást zipu projektu*) obsahuje popis, kam vkládat řešení jednotlivých úloh