PRÁCE S TEXTURAMI

Kurz: **Moderní počítačová grafika**

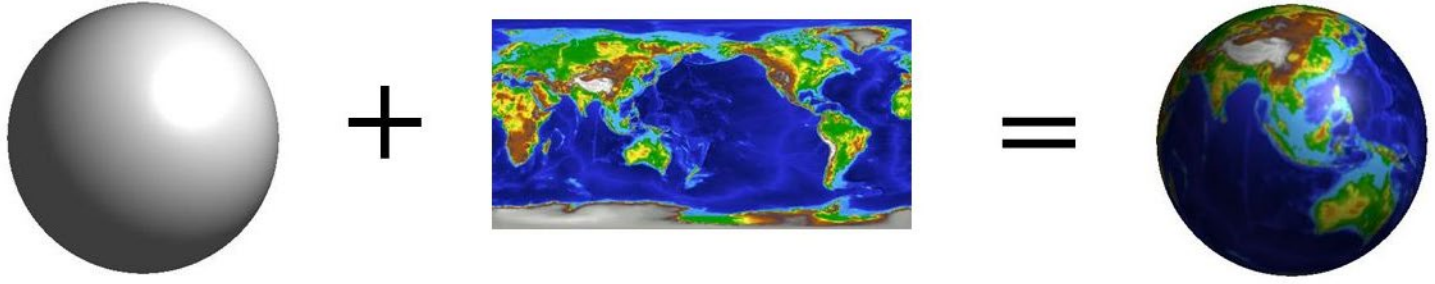
Lektor: Ing. Michal Švento

Náplň cvičení

1. Texturování
2. Blending

Texturování

- obarvení povrchu zobrazovaných těles rastrovou informací nebo na základě matematické funkce
- grafický hardware současnosti je přímo optimalizovaný na vykreslování rastrových textur, které si ukládá do své vlastní paměti



Inicializace textury

- načtení externího rastrového obrázku a uložení do paměti grafického akcelerátoru
- pro rozměry vstupních dat z důvodu přizpůsobení hardwaru platí:
  - šířka a výška musí být (celočíselné) mocniny dvojky
  - musíme respektovat minimální a maximální velikost textury u daného hardwaru
    - typicky od 1×1 tx do 8192×8192 tx
    - tx = texel (texture element), něco jako pixel

- `setTexture(char* obr, uint* texture, bool mipmap)`
  - první parametr ukazuje na soubor s texturou
  - funkce alokuje volnou texturu a její název (číslo) uloží do druhého parametru
  - třetí bude vysvětlen dále

Inicializace textury

- příklad inicializace
  - `// pole pro nazvy textur`
  - `GLuint textury[1]`
  - `setTexture("textura.bmp", &textury[0], false)`
- vložení do paměti grafického akcelerátoru
  - `glGenTextures(int n, int* textures)`
    - vygeneruje n názvů (číselných identifikátorů) textur, které jsou k dispozici v grafickém akcelerátoru
  - `glBindTexture(GL_TEXTURE_2D, int name)`
    - nastaví texturu s číslem `name` jako aktuální
    - první parametr udává, zda se jedná o 1D, 2D nebo 3D texturu
- nutné zapnout jednotku texturování
  - `glEnable(GL_TEXTURE_2D)`

Inicializace textury

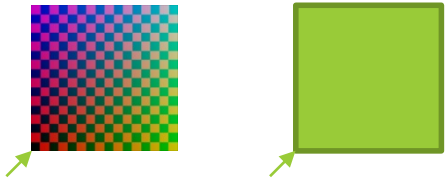
- funkce, která aktuálně nastavené textuře přiřadí obrázek z paměti:
- `glTexImage2D(GL_TEXTURE_2D, int level, format, width, height, border, enum format, enum type, void* pixels)`
  - `level` – číslo mipmap detailu (viz dále)
  - `format` – počet složek v barevné reprezentaci textury
    – využijeme `GL_RGBA` nebo `GL_RGB`
  - `width` – šířka obrázku textury, musí odpovídat $2^n + 2 \times$ `border`
  - `height` – výška obrázku textury, stejná podmínka jako u šířky
  - `border` – šířka okraje, 0 nebo 1
  - `format` – datový typ položek ve zdrojové paměti
    – využijeme `GL_UNSIGNED_BYTE`
  - `pixels` – ukazatel na paměť s daty

Souřadnice textur

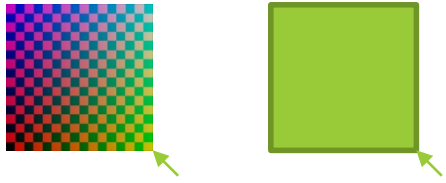
- používá se tzv. **UV mapování textury**
- každému vertexu přiřazena texturová souřadnice [*u, v*]
- pro body, které leží mezi vertexy na povrchu objektu, se hodnota textury dopočítává interpolací
- souřadnice [*u, v*] jsou relativní vzhledem k rozměrům textury
  - rozsah 0 až 1
  - levý dolní roh textury má souřadnice [0, 0]
- před definicí vertexu se zavolá `glTexCoord2f(float u, v)`

Korektní vs. nekorektní texturování

1. `glTexCoord2f(0.0f, 0.0f);`
   `glVertex2i(100, 100);`

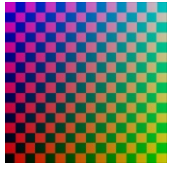


2. `glTexCoord2f(1.0f, 0.0f);`
   `glVertex2i(300, 100);`

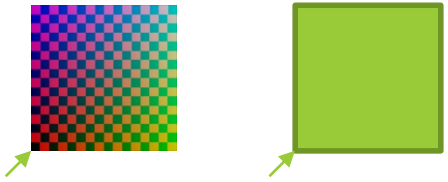


3., 4. analogicky

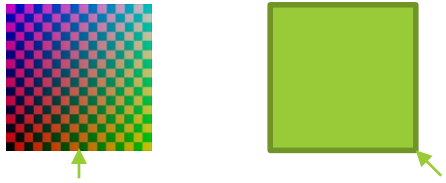
výsledek:



1. `glTexCoord2f(0.0f, 0.0f);`
   `glVertex2i(100, 100);`



2. `glTexCoord2f(0.5f, 0.0f);`
   `glVertex2i(300, 100);`



3., 4. jako v korektním případě

výsledek:

Úkol 1

1. Vyzkoušejte si načíst externí texturu a korektně ji zobrazit na čtverec tak, aby byla textura na čtverec nanesena celá a proporčně nezkreslená.
2. Soubor `textura.bmp` musí být v adresáři projektu.
3. Čtverec má stranu 10 a z = 0

Nápověda:

- čtverec vytvoříte primitivou `GL_QUADS`
- mezi `glBegin(…)` a `glEnd()` vždy zadejte relativní souřadnici textury a následně souřadnici vertexu

Úkol 2

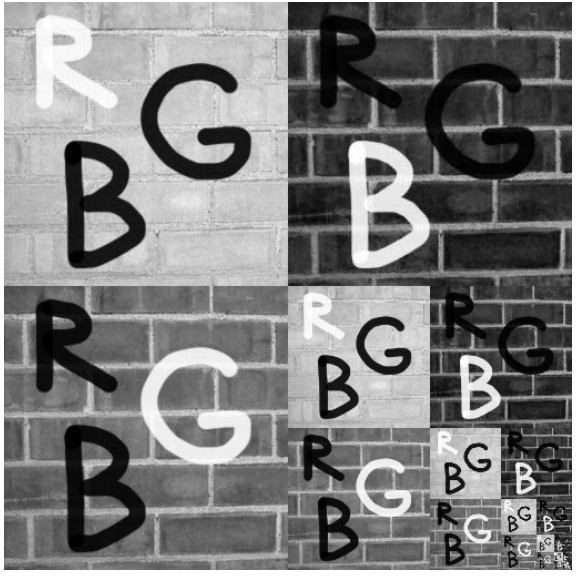
1. Vyzkoušejte si načíst externí texturu a zobrazit ji nekorektně na čtverec.

Mipmapping

- MIP = multum in parvo = mnohé v malém
- způsob uložení textury pro několik měřítek současně, kdy nejbližší menší měřítko je polovina předchozího, postupně až do velikosti jediného pixelu
- v OpenGL si můžeme mipmapy generovat sami
  - pomocí parametru `level` ve funkci `glTexImage2d`
  - pomocí funkce GLU `gluBuild2dMipmaps(enum target, int format, width, height, enum format, type, void* data)`

Mipmapping

- Typický způsob uložení RGB bitmapové textury v paměti technikou mipmap. Vlevo nahoře červený kanál v plném rozlišení, vpravo od něj modrý, dole zelený. Ve zbytku prostoru totéž zmenšeno na poloviční velikost.



Kvalita zobrazení

- kvalita zobrazení a další parametry textur se nastavují pomocí několika různých funkcí
  - `glTexParameteri(GL_TEXTURE_2D, enum pname, int value)`
  - `glTexParameterf(GL_TEXTURE_2D, enum pname, float value)`
  - parametr `pname` určuje, jaký typ filtru se použije při zmenšování textury pod základní rozměr (hodnota `GL_TEXTURE_MIN_FILTER`) nebo naopak pro zvětšování textury (hodnota `GL_TEXTURE_MAG_FILTER`)
  - parametr `value` je potom následujících hodnot:
    - `GL_NEAREST` – pouze nalezení bodu metodou nearest neighbor
    - `GL_LINEAR` – lineární interpolace
  - s využitím mipmappingu existují následující 4 volby:
    - `GL_NEAREST_MIPMAP_NEAREST` – vybere mipmapu, použije nearest vyhlazení
    - `GL_LINEAR_MIPMAP_NEAREST` – vybere mipmapu, použije lineární vyhlazení
    - `GL_NEAREST_MIPMAP_LINEAR` – vybere průměr dvou mipmap, použije nearest vyhlazení
    - `GL_LINEAR_MIPMAP_LINEAR` – vybere průměr dvou mipmap, použije lineární vyhlazení

Kvalita zobrazení

- kvalita zobrazení a další parametry textur se nastavují pomocí několika různých funkcí
  - `glTexParameteri(GL_TEXTURE_2D, enum pname, int value)`
  - `glTexParameterf(GL_TEXTURE_2D, enum pname, float value)`
  - parametr `pname` alternativně určuje, jaké chování požadujeme pro opakování textury ve směru *u* (hodnota `GL_TEXTURE_WRAP_S`) a ve směru *v* (hodnota `GL_TEXTURE_WRAP_T`)
  - parametr `value` je potom následujících hodnot:
    - `GL_CLAMP` – hodnota větší než 1 znamená protáhnutí okrajové barvy
    - `GL_MIRRORED_REPEAT` – hodnota větší než 1 znamená zrcadlové opakování textury
    - `GL_REPEAT` – hodnota větší než 1 znamená periodické opakování textury
- obecně
  - parametr `pname` udává, co chci nastavit
  - parametr `value` udává, jak to chci nastavit
  - https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexParameter.xhtml

Kvalita zobrazení

- je vhodné pomocí již zmiňovaného `glHint()` nastavit vlastnost `GL_PERSPECTIVE_CORRECTION_HINT` na hodnotu `GL_NICEST`
  - `glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)`

Úkol 3

1. Vypnutí/zapnutí mipmapping pomocí klávesy `m`
2. Sledujte změny ve vykreslování textury při vzdalování objektu a srovnejte s výsledkem při oddalování s vypnutým mipmappingem.
   - Ovládání přiblížení – stisknutím pravého tlačítka a pohybem myši.
3. Ve funkci `glTexCoord2f` změňte rozsah pro mapování tak, aby byl patrný opakovací efekt textury.
   - Stačí nastavit rozsah *u*, *v* (ve funkci `glTexCoord2f`) přesahující hodnoty 0 až 1.
4. Opakování nahraďte protáhnutím okrajové barvy.

- Všechna makra nejsou dostupná, lze vyhledat HEX kód
- https://javagl.github.io/GLConstantsTranslator/GLConstantsTranslator.html

Blending

- blending (míchání) je proces, při kterém je vykreslovaný (zdrojový, source) fragment ovlivněn barvou pixelu, který již ve framebufferu existuje na stejném místě (cílový, destination)
- realizace pomocí čtyř barevných složek (RGBA)
- míchání probíhá pomocí koeficientů
  - zdroj: $\boldsymbol{s} = [\, s_R,\ s_G,\ s_B,\ s_A \,]$
  - cíl: $\boldsymbol{d} = [\, d_R,\ d_G,\ d_B,\ d_A \,]$
  - jednotlivé koeficienty jsou v rozsahu 0 až 1, každá barevná složka má svoje vlastní koeficienty obecně nezávislé na ostatních
- mějme dva fragmenty
  - zdroj s barvou $[\, R_S,\ G_S,\ B_S,\ A_S \,]$
  - cíl s barvou $[\, R_D,\ G_D,\ B_D,\ A_D \,]$
- výsledná zobrazená barva se vypočítá pomocí barev a koeficientů následovně
  - $R = R_S \cdot s_R + R_D \cdot d_R, \quad G = G_S \cdot s_G + G_D \cdot d_G, \quad B = B_S \cdot s_B + B_D \cdot d_B, \quad A = A_S \cdot s_A + A_D \cdot d_A$

Blending

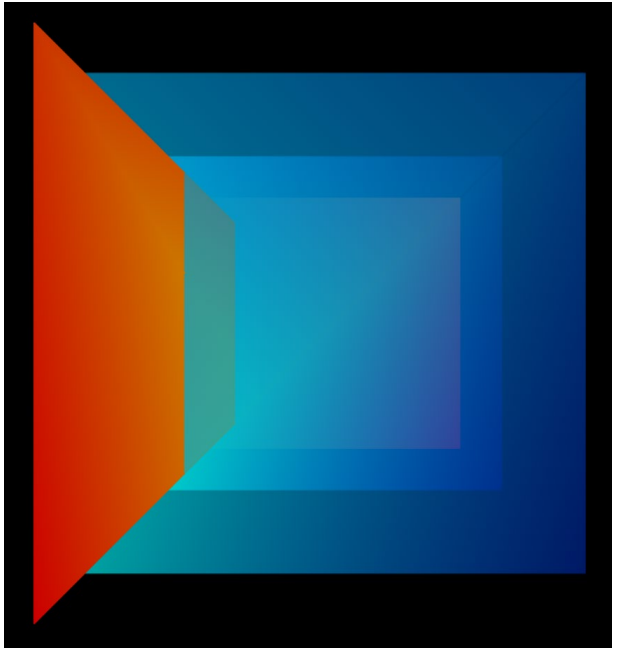
- zapnutí blendingu: `glEnable(GL_BLEND)`
- definice rovnice míchání: `glBlendFunc(enum sfactor, enum dfactor)`
- zvlášť barvy a alfa: `glBlendFuncSeparate(enum srcRGB, enum dstRGB, enum srcAlpha, enum dstAlpha)`
- jednotlivé parametry nastavují hodnoty vektorů $\boldsymbol{s}$ a $\boldsymbol{d}$ a vybírají se z následujících variant:
  - `GL_ONE` $[\,1,\ 1,\ 1,\ 1\,]$
  - `GL_ZERO` $[\,0,\ 0,\ 0,\ 0\,]$
  - `GL_SRC_COLOR` $[\,R_S,\ G_S,\ B_S,\ A_S\,]$
  - `GL_ONE_MINUS_SRC_COLOR` $[\,1-R_S,\ 1-G_S,\ 1-B_S,\ 1-A_S\,]$
  - `GL_DST_COLOR` $[\,R_D,\ G_D,\ B_D,\ A_D\,]$
  - `GL_ONE_MINUS_DST_COLOR` $[1-R_D,\ 1-G_D,\ 1-B_D,\ 1-A_D\,]$
  - `GL_SRC_ALPHA` $[\,A_S,\ A_S,\ A_S,\ A_S\,]$
  - `GL_DST_ALPHA` $[\,A_D,\ A_D,\ A_D,\ A_D\,]$
  - `GL_ONE_MINUS_SRC_ALPHA` $[\,1-A_S,\ 1-A_S,\ 1-A_S,\ 1-A_S\,]$
  - `GL_ONE_MINUS_DST_ALPHA` $[1-A_D,\ 1-A_D,\ 1-A_D,\ 1-A_D\,]$

Průhlednost

- využití parametrů GL_SRC_ALPHA a GL_ONE_MINUS_SRC_ALPHA
- nekompatibilní se z-bufferem!
  - blending se aplikuje až na úrovni vykreslovacího bufferu
  - test hloubky v z-bufferu ovšem z vykreslovacího řetězce vyřazuje to, co není vidět (fragmenty zakryté jinými fragmenty)
- nutné při nastavení průhlednosti test hloubky vypínat
  - příkaz `glDepthMask(GL_FALSE)`

Úkol 4

1. Zobrazte si scénu definovanou v kódu `ukol4.cpp`.
2. Vhodným použitím funkce `glDepthMask` a implementací průhlednosti dosáhněte výsledku, který je na obrázku.



Nápověda:

- z-buffer je třeba před definicí průhledné stěny vypnout a potom zase zapnout
- naopak blendování nejdříve zapnete a po definici průhledné stěny opět vypnete

Kombinace zobrazení s texturami

- pomocí nastavení prostředí textur (texture environment) můžeme specifikovat, jak se bude výsledný fragment texturovat s ohledem na ostatní vlastnosti (osvětlení, blending)
  - `glTexEnvi(enum target, enum pname, int value)`
  - `glTexEnvfv(enum target, enum pname, float* value)`

- `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, int value)`
  - `GL_DECAL` u RGB textur překreslí celý fragment barvou textury, u RGBA textur bere v potaz průhlednost, textura se míchá se zbytkem fragmentu
  - `GL_REPLACE` přepisuje vše vlastnostmi textury
  - `GL_MODULATE` násobí barevné složky textury s barevnými složkami fragmentu, tímto způsobem lze vytvářet nasvícené textury
  - `GL_BLEND` barevné složky použije jako koeficienty pro splynutí s barvou prostředí
- `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, float* value)`
  - parametr `value` je barva prostředí jako pole 4 hodnot RGBA

https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glTexEnv.xml

Úkol 5

1. Nastudujte modulaci textur s osvětlením v kódu `ukol5.cpp`, kde je použita jednoduchá textura šachovnice.