Animace na bázi vrcholů a texturování

Zadání projektu pro předmět Základy počítačové grafiky – IZG

Michal Španěl
spanel@fit.vutbr.cz

Michal Matýšek
imatysek@fit.vutbr.cz

29. 2. 2015

1 Kostra projektu

Základem projektu je funkční kostra programu, která realizuje softwarový rendering jednoduchého polygonálního modelu.

- Aplikace načítá zobrazovaný model z textových souborů s příponou `.tri`. Pro testování je na výběr několik připravených modelů.
- Základním zobrazovaným primitivem je trojúhelník.
- Aplikace řeší celý proces vykreslování včetně rasterizace trojúhelníku softwarově (bez grafické karty, bez OpenGL).
- Manipulaci se scénou zajišťuje primitivní manipulátor, který pouze otáčí a zvětšuje/zmenšuje zobrazovaný model. Pozice kamery je fixní.
- Renderer řeší veškeré transformace modelu včetně perspektivní projekce.

S naprostou většinou algoritmů, které jsou v rendereru použity (transformace, rasterizace, osvětlení, apod.) se postupně seznámíte na přednáškách a cvičeních.

(a) Kostra programu (b) Cíl projektu – animace modelu a perspektivně korektní texturování

Obrázek 1: Vykreslení modelu před (a) a po vypracování projektu (b).

2 Zdrojový kód

Kostra programu je obdobou frameworku, který používáte ve cvičení a je napsána v *čistém C*.

- Kreslíme do vlastního alokovaného frame bufferu.
- Pro vykreslení frame bufferu na obrazovku a reakce na události (myšování, apod.) se používá knihovna *SDL* (*Simple Directmedia Layer*).

`http://www.libsdl.org/`

Předkompilované balíčky knihovny SDL (testováno s verzí *1.2.15*) najdete na webu:

`http://www.libsdl.org/download-1.2.php`

2.1 Překlad ve zkratce

- *Windows/MinGW* – Na Windows lze použít překladač *MinGW* (GCC portované na windows) a přiložený *Makefile*. V učebnách fakulty mi funguje:
  - Otevřít okno s příkazovou řádkou.
  - Nastavit cesty k MinGW překladači spuštěním: `q:/mingw/set_mingw.bat` (alternativně: `set path=q:/mingw/mingw/bin;%path%`)
  - Spustit překlad: `mingw32-make`
  - V tomto případě se použije již instalovaná knihovna SDL z adresáře `q:/mingw/SDL`.
- *Windows/MSVC* – Otevřete dodaný solution v *MS Visual Studiu* a nechte proběhnout případnou konverzi solution a projektu. SDL knihovnu nainstalujete takto:
  - Stáhněte si balíček binárních knihoven pro MSVC: `http://www.libsdl.org/release/SDL-devel-1.2.15-VC.zip`
  - A rozbalte jej do adresáře *vedle* projektu, tzn.:
    ```
    .
    ..
    SDL-1.2.15
    izg_proj
    ```
  - F7 (Ctrl+F7) spustí překlad
  - Při problémech s překladem pod VS2015 vyzkoušejte řešení popsané v souboru `vs2015.txt`.
- *Linux/GCC* – Pro překlad na Linuxu je připravený *Makefile* stačí tedy: `make`. SDL knihovna bývá součástí většiny distribucí, pokud ne:
  - Stáhněte si zdrojové kódy z `http://www.libsdl.org/release/SDL-1.2.15.zip`
  - Zadejte typickou sekvenci `./configure; make; make install`

  - Na Ubuntu funguje také:
    `sudo apt-get install libsdl1.2-dev`

- *Mac OS X* – Pro překlad na macu je nutné doinstalovat knihovnu SDL. Nejjednodušší je instalace přes Rudix (kolekce předinstalovaných UNIX programů pro Mac OS X):

  - Instalace Rudix
    `curl -O https://raw.githubusercontent.com/rudix-mac/rpm/2014.10/rudix.py sudo python rudix.py install rudix`
  - Instalace SDL
    `sudo rudix install sdl`
  - Potom už stačí:
    `make`

3 Poznámky k řešení

Pro vaše modifikace je připravena oddělená varianta rendereru v souborech `student.h` a `student.c`. S projektem experimentujte dle libosti (měňte a modifikujte co chcete), ale pamatujte, že odevzdané soubory `student.h` a `student.c` musí být funkční s originálním frameworkem! Pro odevzdané řešení je povoleno modifikovat pouze tyto dva soubory.

Jak přepnout na zobrazování pomocí studentského rendereru?
Konkrétní renderer (studentský i výchozí) se vytváří ve funkci `main()` v souboru `main.c`. Počáteční volbu lze ovlivnit definováním makra `USE_STUDENT_RENDERER` na začátku `main.c` a opětovným překladem. Renderer lze také přepnout za běhu pomocí kláves **O** a **P**.

Jak přepnout standardně zobrazovaný model a texturu?
Standardně zobrazovaný model (`yoshi.tri`) a textura (`yoshi.bmp`) lze nastavit přímo ve zdrojovém kódu (soubor `main.c`) nebo zadáním cest k modelu a k textuře na příkazové řádce (modely a textury jsou umístěny v adresáři `models`):

```
izg_proj models/marvin.tri models/marvin.bmp
```

4 Bodované úkoly

Cílem je doplnit do připraveného *„studentského“* rendereru následující funkce:

- podporu vykreslování modelů s animacemi na bázi vrcholů,
- perspektivně korektní bilineární vzorkování textury.

Studenti pracují na řešení projektu samostatně a každý odevzdá své vlastní řešení. Poraďte si, ale řešení vypracujte samostatně!

4.1 Animace na bázi vrcholů – morfing

Metoda *morfing* (alternativně *per-vertex animace*, *vertex blending*, *morph target animace*, ...) pracuje na jednoduchém principu *míchání* dvou nebo více variant určitého modelu na bázi jednotlivých vrcholů. Tato technika tedy funguje pouze v případě, mají-li všechny varianty modelu (klíčové snímky) stejný počet vrcholů a jsou-li polygony, respektive trojúhelníky, strukturovány stejným způsobem. Samotný pohyb je definován prostřednictvím sekvence klíčových snímků. Tyto snímky představují významné pózy, mezi kterými se provádí lineární interpolace odpovídajících si vrcholů. Každý snímek obsahuje polohy všech vrcholů modelu v dané póze, což v podstatě znamená uložit jeden model v desítkách až stovkách póz, viz obrázek 2. Metoda proto obecně není vhodná pro animaci detailních složitých modelů.

Obrázek 2: Sekvence pohybově významných póz (tzv. klíčových snímků) definují pohyb modelu v čase.

V projektu budete metodu morfing implementovat v souvislosti s animací jednoduchých nízkopolygonálních modelů.

4.2 Krok č. 1 – zprovoznění studentského rendereru

První krok řešení projektu zahrnuje přípravu výchozího studentského rendereru v souborech `student.h` a `student.c`. Doporučuji prostudovat princip základního rendereru (soubory `render.h` a `render.c`), velká část projektu je o modifikacích funkcí základního rendereru. Zprovoznění vašeho rendereru vyžaduje:

- Upravit funkci `studrenDrawTriangle()` – zkopírovat obsah funkce `renDrawTriangle()`.
- Upravit funkci `studrenProjectTriangle()` – zkopírovat obsah funkce `renProjectTriangle()` a ve funkci `studrenProjectTriangle()` nastavit volání vlastní funkce `studrenDrawTriangle()` namísto `renDrawTriangle()`.
- Upravit funkci `renderStudentScene()` – zkopírovat obsah funkce `renderDefaultScene()`.
- Upravit funkci `studrenCreate()` – nastavit ukazatele na vaše funkce (`studrenRelease()`, `studrenProjectTriangle()`, viz komentáře ve zdrojovém kódu).

Korektně připravený studentský renderer bude po provedení výše uvedených kroků vykreslovat výsledek identický s výchozím rendererem. Výchozí a studentský renderer lze přepnout za běhu pomocí kláves **O** a **P**.

4.3 Krok č. 2 – zobrazení sekvencí klíčových snímků

Pro zobrazování sekvence klíčových snímků je nezbytné získat aktuální čas pomocí callback funkce časovače `onTimer()`. Funkce `renProjectTriangle()`, respektive `studrenProjectTriangle()`, je implementována tak, aby byla schopna zobrazit jednotlivé trojúhelníky indexované parametrem `i` pro samostatné snímky dle parametru `n`. Celá část parametru `n` určuje index zobrazovaného snímku, desetinná část bude později použita pro interpolaci mezi snímky. Zobrazení sekvence klíčových snímku tedy vyžaduje:

- Vhodně doplnit callback funkci `onTimer()` – parametr `ticks` (počet milisekund od inicializace) využít pro získání aktuálního času. Funkce `onTimer()` je ve výchozím stavu volána periodicky každých 33ms (∼30 krát za sekundu, viz `main.c`).
- Upravit volání funkce `renderModel()` ve funkci `renderStudentScene()` – použít vhodně upravený čas získaný ve funkci `onTimer()` pro aktualizaci indexu `n` aktuálně kresleného snímku modelu. Mějte na paměti, že pro následnou interpolaci je nezbytné, aby byl parametr `n` reálné číslo. Ve výchozím stavu je funkce `renderModel()` volána s parametrem `n=0`. Funkce `studrenProjectTriangle()` tak kreslí trojúhelníky vždy pouze pro první klíčový snímek.

Po doplnění výše uvedených kroků bude studentský renderer postupně zobrazovat jednotlivé klíčové snímky modelu. Jen některé připravené modely (`yoshi.tri`, `ogro.tri`, `doomguy.tri`, `marvin.tri`) však obsahují více klíčových snímků (více variant jednoho modelu). Při implementaci animace tak dejte pozor, že využíváte animované modely uvedené výše. Ostatní modely jsou statické (pouze jedna varianta, tzn. jeden klíčový snímek).

4.4 Krok č. 3 – lineární interpolace mezi snímky

Protože jsou jednotlivé pózy (klíčové snímky) modelu *daleko* od sebe, dochází při jejich sekvenčním vykreslování k výrazným skokům v pohybu a animace tak není plynulá. Pro dosažení plynulé animace s nízkým počtem klíčových snímků se proto využívá lineární interpolace mezi dvojící sousedních snímků. Pro správnou funkci a korektní zobrazování animace je nezbytné interpolovat nejen mezi dvojicemi odpovídajících si vrcholů, ale také mezi normálovými vektory. Interpolace normálových vektorů zajistí korektní výpočet osvětlení mezi snímky a korektní funkci odstraňování odvrácených trojúhelníků (*backface culling*). Je tedy nutné interpolovat mezi dvojící normálových vektorů aktuálního trojúhelníku i mezi dvojicemi normálových vektorů v jednotlivých vrcholech.

Funkce `studrenProjectTriangle()` je implementována tak, aby byla schopna zpracovat trojúhelník s indexem `i` pouze jednoho snímku dle indexu `n`. Cílem tohoto kroku je dosažení plynulé animace prostřednictvím následujících úprav:

- Stejným způsobem, jako jsou získávány offsety na vrcholy a normálové vektory v `n`-tém snímku, získejte offsety pro vrcholy a normálové vektory ve snímku `n+1`. V případě, že je `n` poslední snímek zajistí zbytek po dělení interpolaci mezi posledním snímkem `n` a prvním snímkem `n+1`.
- Stejným způsobem, jako jsou získávány indexy na vrcholy a normálové vektory v `n`-tém snímku, získejte indexy pro vrcholy a normálové vektory ve snímku `n+1`.
- Před transformacemi je nezbytné provést interpolaci mezi odpovídajícími si dvojicemi vrcholů a normálových vektorů na základě desetinné části parametru `n`.

Interpolace odpovídajících si vrcholů, normálových vektorů trojúhelníku a normálových vektorů v jednotlivých vrcholech zajistí plynulý přechod mezi snímky, korektní výpočet osvětlení a korektní funkci odstraňování odvrácených trojúhelníků (*backface culling*).

4.5 Krok č. 4 – použití texturovacích souřadnic

Aplikace textury na model vyžaduje použití texturovacích souřadnic. Následující kroky umožní nanesení šachovnicové (procedurální) textury:

- Texturovací souřadnice pro vrcholy aktuálně zpracovávaného trojúhelníku jsou uloženy ve struktuře definující trojúhelník (proměnná `triangle` ve funkci `studrenProjectTriangle()`). Použijte texturovací souřadnice při volání funkce `studrenDrawTriangle()` – upravte (přidejte) parametry na vstup funkce, které budou obsahovat texturovací souřadnice (nezapomeňte upravit hlavičku také v souboru `student.h`).
- Stejným způsobem, jako je ve funkci `studrenDrawTriangle()` interpolována hloubka (souřadnice) `z`, interpolujte pomocí barycentrických souřadnic texturovací souřadnice aktuálně kresleného pixelu.
- Pro získání barvy z textury použijte funkci `studrenTextureValue()`. Tato funkce ve výchozím stavu implementuje generování *procedurální* šachovnice dle texturovacích souřadnic. Barvu z textury zkombinujte s barvou získanou z výpočtu osvětlovacího modelu pomocí *modulace*. Výsledná RGB barva po modulaci, jež bude zapsána do frame bufferu, je obecně získána jako

$$\begin{aligned} R &= R_l * R_t, \\ G &= G_l * G_t, \\ B &= B_l * B_t, \\ R_l, G_l, B_l, R_t, G_t, B_t &\in < 0, 1 >, \end{aligned} \tag{1}$$

  kde $R_l G_l B_l$ je barva určená osvětlovacím modelem a $R_t G_t B_t$ je hodnota z textury.
- Definujte si nové vlastní *bílé* materiály `MAT_WHITE_*` a nastavte je namísto `MAT_RED_*` ve funkci `renderStudentScene()`. Bílé materiály umožní dle vzorce 1 rovnoměrně zkombinovat barevné složky vypočítaného osvětlení s barvou nanášené textury.

Aplikací výše uvedených kroků vykreslíte model s nanesenou texturou (šachovnicí) s osvětlením na základě *Lambertova odrazu* implementovaného ve funkci `renLambertianReflectance()`, viz obrázek 3b.

(a) Lambertův odraz (b) Procedurální textura (c) Textura ze souboru

Obrázek 3: Vykreslení modelu s výpočtem osvětlení na základě Lambertova odrazu (a) zkombinovaného s procedurální šachovnicovou texturou (b) a s texturou ze souboru (c).

4.6 Krok č. 5 – použití textur ze souboru

Namísto šachovnicové textury použijeme v tomto kroku texturu načtenou ze souboru, viz obrázek 3c. Nanesení *obrázkové* textury na model vyžaduje:

- Doplnit vlastní reprezentaci textury s využitím typu `S_RGBA` do struktury `S_StudentRenderer` – tzv. aktivní kreslící textura. (Vyžaduje mimo pole `S_RGBA` také parametry šířky a výšky textury.)
- Upravit funkce `studrenCreate()` a `studrenRelease()` – přidat vytvoření a nakonec uvolnění textury. Pro vytvoření (načtení) textury ze souboru využijte funkci `loadBitmap()` (`bmp.c`), která vrací pole `S_RGBA` barev texelů a rozměry textury skrze parametry funkce. Jako název načítané textury pro funkci `loadBitmap()` uveďte proměnnou `TEXTURE_FILENAME` nastavovanou ve funkci `main()`.
- Upravte funkci `studrenTextureValue()` pro získání hodnoty z textury s využitím bilineární interpolace. Interpolujte čtyři nejbližší pixely dle souřadnic `u` a `v` v intervalu $< 0, 1 >$.

4.7 Krok č. 6 – perspektivní korekce textur

Při vzorkování textur prostřednictvím texturovacích souřadnic získaných lineární interpolací (viz krok č. 4 v sekci 4.5) dochází v určitých situacích ke zkreslení (deformaci) textury. Perspektivně korektní nanášení textur proto vyžaduje úpravu interpolace texturovacích souřadnic dle vzorce

$$u = \frac{w_1 \cdot \frac{u_1}{h_1} + w_2 \cdot \frac{u_2}{h_2} + w_3 \cdot \frac{u_3}{h_3}}{w_1 \cdot \frac{1}{h_1} + w_2 \cdot \frac{1}{h_2} + w_3 \cdot \frac{1}{h_3}}, v = \frac{w_1 \cdot \frac{v_1}{h_1} + w_2 \cdot \frac{v_2}{h_2} + w_3 \cdot \frac{v_3}{h_3}}{w_1 \cdot \frac{1}{h_1} + w_2 \cdot \frac{1}{h_2} + w_3 \cdot \frac{1}{h_3}}, \tag{2}$$

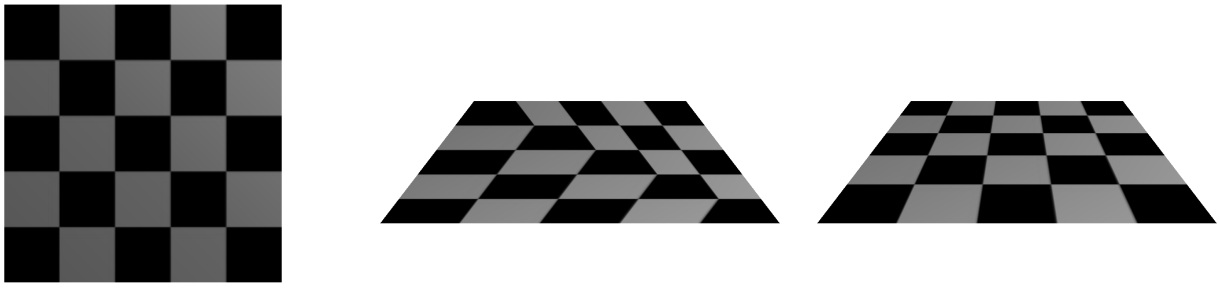
kde u a v jsou výsledné texturovací souřadnice; w_1, w_2, w_3 jsou barycentrické souřadnice; u_i a v_i pro $i \in \{1, 2, 3\}$ jsou texturovací souřadnice vrcholu i v daném trojúhelníku a h_i je homogenní souřadnice vrcholu i.

Interpolace texturovacích souřadnic z kroku č. 4 neuvažuje zkreslení závislé na hloubce a perspektivní projekci. Dochází tak k nepřirozené deformaci nanášené textury, viz obrázek 4. Pro demonstraci tohoto jevu spusťte renderer se zobrazením modelu `plane.tri` a textury `plane.bmp`:

```
izg_proj models/plane.tri models/plane.bmp
```

Závěrečná úprava rendereru zahrnuje:

- Získat homogenní souřadnice jednotlivých vrcholů trojúhelníku – čtvrtou homogenní souřadnici vrcholu vrací funkce `trProjectVertex()` jako návratovou hodnotu. Projekce probíhá ve funkci `studrenProjectTriangle()`. Homogenní souřadnice pro všechny tři vrcholy použijte při volání funkce `studrenDrawTriangle()` – upravte (přidejte) parametry na vstup funkce, které budou pro každý vrchol obsahovat čtvrtou homogenní souřadnici (nezapomeňte upravit hlavičku také v souboru `student.h`).
- Upravit funkci `studrenDrawTriangle()` – s využitím homogenních souřadnic jednotlivých vrcholů interpolujte texturovací souřadnice dle vzorce 2. Výsledkem bude perspektivně korektní nanesení textury, viz obrázek 4c.



(a) Přímý pohled (b) Bez perspektivní korekce (c) S perspektivní korekcí

Obrázek 4: Eliminace deformací textury s využitím perspektivní korekce.

5 Odevzdání

viz. Wiki stránka

6 Závěrem

Ať se dílo daří a ať vás grafika alespoň trochu baví! V případě potřeby se nebojte zeptat (na fóru nebo napište přímo vedoucímu projektu).