Zadání projektu – Vykreslování s průhledností

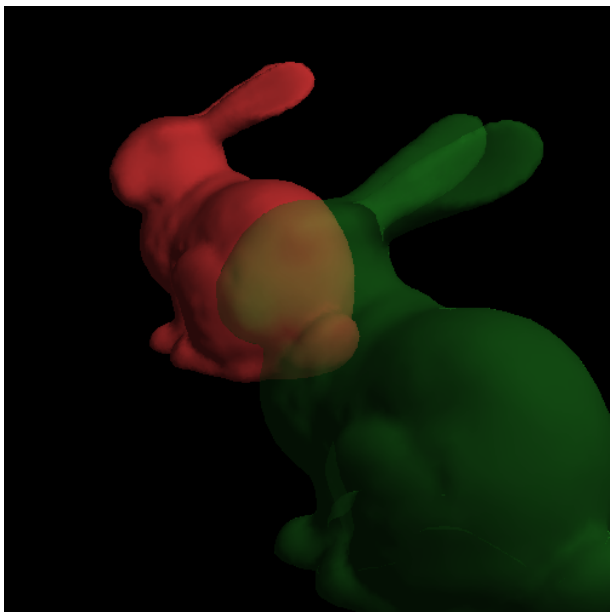
Základy počítačové grafiky (IZG)
ak. rok 2014/2015

Michal Španěl, spanel@fit.vutbr.cz

7.3.2015

1 První seznámení

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



Obrázek 1: Ukázka zobrazení modelu *bunny.tri* po vypracování projektu. Doplněno *vykreslování s průhledností* a *vícenásobné vykreslení modelu*.

- Aplikace načítá zobrazovaný model z textových souborů s příponou .TRI. Pro testování máte na výběr několik modelů.
- Standardně zobrazovaný model (BUNNY.TRI) lze změnit zadáním jiného názvu na příkazové řádce:

  Příklad: IZG_PROJ SKULL.TRI

- Základním zobrazovaným primitivem je trojúhelník.
- Aplikace řeší celý proces vykreslování včetně rasterizace trojúhelníku softwarově (bez grafické karty, bez OpenGL).
- Pro manipulaci se scénou je implementován 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.

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 spustí překlad

- *Linux/GCC* – Pro překlad na linuxu je připravený *Makefile* stačí tedy: MAKE. SDL knihovna bývá součástí většiny Linuxových 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

2.2 V případě potíží s překladem (v uvedeném pořadí)...

1. hledejte chybu na vašem konci klávesnice,
2. zeptejte se kamaráda,
3. nahlédněte do diskusního fóra IZG a případně pošlete dotaz,
4. zatancujte kolem svého počítače, proneste několik magických manter a pak zkuste překlad znovu,
5. napiště mi email.

3 Bodovaný úkol – Vykreslování s průhledností

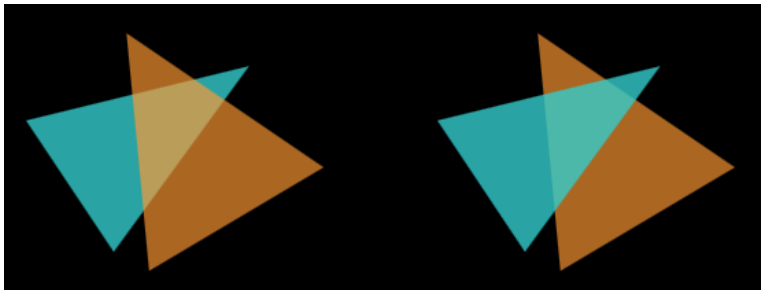
Cílem je doplnit do připraveného *„studentského“* rendereru následující funkce:

- *podporu vykreslovani s průhledností* (tzv. Order Independent Transparency) s využitím řazení vykreslovaných fragmentů samostatně pro každý pixel obrazovky.
- *vícenásobné vykreslení modelu* s různým nastavením materiálu a průhlednosti.

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

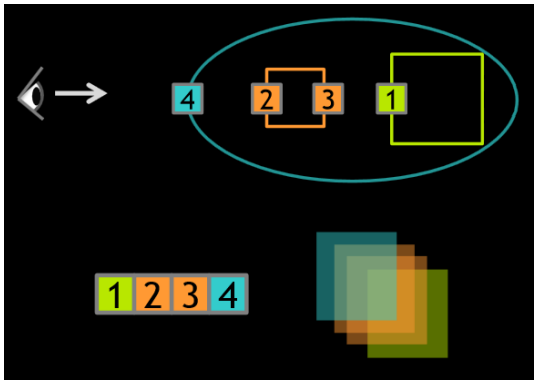
3.1 Order Independent Transparency

Při vykreslování transparentních objektů nestačí klasický způsob řešení viditelnosti (např. z–buffer). V daném pixelu obrazovky není viditelný pouze polygon nejbližší ke kameře, ale výsledná barva pixelu vzniká kombinací všech poloprůhledných polygonů, které jsou v daném pixelu zobrazeny. Zobrazení navíc musí proběhnout ve správném pořadí.



Obrázek 2: Rozdílné výsledky při vykreslení částečně průhledných polygonů v různém pořadí.

Naivní *objektové algoritmy* provádějí extrémně náročné řazení polygonů podle vzdálenosti od kamery s vykreslováním „od zadu“ a mícháním barev polygonů na základě průhlednosti.



Obrázek 3: Obrazové algoritmy řeší správné pořadí fragmentů na úrovni pixelů.

Efektivnější a jednodušší je vykreslování s využitím tzv. per-pixel seznamů fragmentů (Obrázek 3):

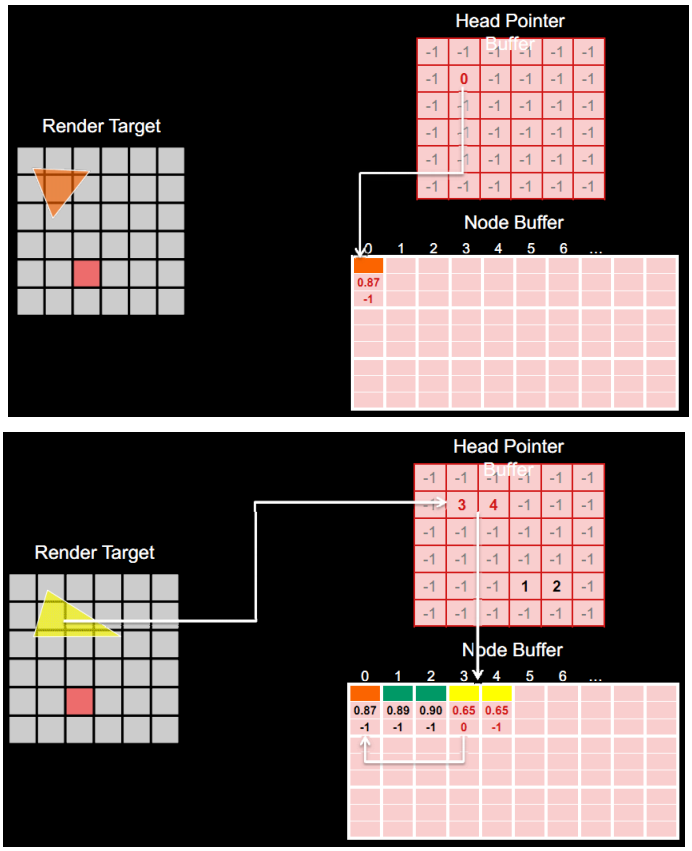
- V průběhu rasterizace polygonů budujeme seznam fragmentů (barva, průhlednost, hloubka ve scéně) pro každý pixel obrazovky.
- Pro každý pixel obrazovky seřadíme vytvořený seznam fragmentů na základě vzdálenosti od kamery v pořadí od nejbližšího po nejvzdálenější (tzv. front-to-back order).
- Výslednou barvu pixelu určíme smícháním barev jednotlivých fragmentů na základě jejich průhlednosti (tzv. blending) a získanou barvu zapíše do frame bufferu.

Princip implementace per-pixel seznamů fragmentů je dobře popsaný zde:

`http://developer.amd.com/wordpress/media/2013/06/2041_final.pdf`

Metoda využívá dvou pomocných bufferů (Obrázek 4):

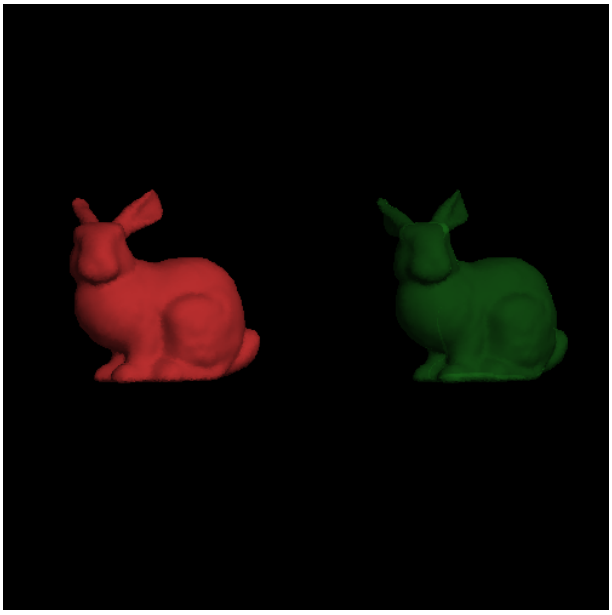
- HEAD_POINTER_BUFFER – buffer o stejné velikosti jako frame buffer (tj. velikost renderovaného obrazu), který obsahuje index prvního fragmentu na daném pixelu (odkaz do NODE_BUFFER). Pokud je daný pixel prázdný (žádný polygon nepadl do tohoto místa obrazovky), bude hodnota indexu -1.
- NODE_BUFFER – pomocné pole všech fragmentů. Kromě barvy, průhlednosti a hloubky ve scéně obsahuje fragment také index následující fragmentu v seznamu (nebo -1).



Obrázek 4: Princip vytváření seznamů fragmentů pro jednotlivé pixely obrazovky.

3.2 Krok č.1 – Vícenásobné vykreslení modelu

Upravte funkci pro vykreslení scény vašeho studentského rendereru tak, aby načtený model byl vykreslen dvakrát, ale pokaždé s jiným nastavením materiálu a na jiné pozici ve scéně.



Obrázek 5: Vícenásobné vykreslení modelu.

První objekt bude mít nastavený neprůhledný materiál MAT_RED_ a bude posunutý mírně vlevo od středu scény. Druhý objekt bude mít poloprůhledný materiál MAT_STUD_ a bude posunutý vpravo od středu. Oba objekty spolu nesmějí kolidovat.

Co bude třeba udělat:

- Upravit fci RENDERSTUDENTSCENE() - zkopírovat obsah funkce RENDERDEFAULTSCENE() z MAIN.C a doplnit vícenásobné kreslení.

Pro práci s transformacemi využijte fce: TRTRANSLATE(), TRGETMATRIX() a TRSETMATRIX().

3.3 Krok č.2 – Rasterizace trojúhelníku

Při vykreslování s průhledností nebudete v projektu pracovat se z–bufferem. Nejprve bude nutné nachystat potřebné datové struktury, doplnit je do vašeho rendereru a upravit rasterizaci trojúhelníku:

- Do vašeho rendereru S_STUDENTRENDERER doplňte datové struktury pro oba pomocné buffery. Doporučuji použít strukturu S_VECTOR dostupnou ve VECTOR.H a strukturu S_FRAGMENT dostupnou ve FRAGMENT.H.
- Doplnit fce pro vytvoření rendereru STUDRENCREATE(), zrušení rendereru STUDRENRELEASE(), inicializaci vykreslovacích a pomocných bufferů STUDRENCREATEBUFFERS() a vymazání bufferů před renderováním nového snímku STUDRENCLEARBUFFERS().

- Upravit fci STUDRENDRAWTRIANGLE() pro rasterizaci trojúhelníku tak, aby místo zápisu do frame bufferu a z–bufferu vybudovala seznam fragmentů pro každý pixel obrazovky.
- Upravit STUDRENPROJECTTRIANGLE() - musí volat vaši novou funkci STUDRENDRAWTRIANGLE().

3.4 Krok č.3 – Alpha blending fragmentů

Alpha blending řeší smíchání barev jednotlivých průhledných fragmentů ve výslednou barvu, kterou v daném pixelu zobrazíte. Rovnice pro alpha blending fragmentů v pořadí front-to-back vypadají takto:

$$\begin{aligned} C_{dst} &= A_{dst}(A_{src}C_{src}) + C_{dst}, \\ A_{dst} &= (1 - A_{src})A_{dst}, \end{aligned}$$

kde C_{src} a A_{src} představují barvu přimíchávaného fragmentu (průhlednost v intervalu $< 0, 1 >$). C_{dst} představuje výslednou barvu a A_{dst} výslednou průhlednost pixelu. Počáteční inicializace by měla být:

$$\begin{aligned} C_{dst} &= (0, 0, 0) \\ A_{dst} &= 1 \end{aligned}$$

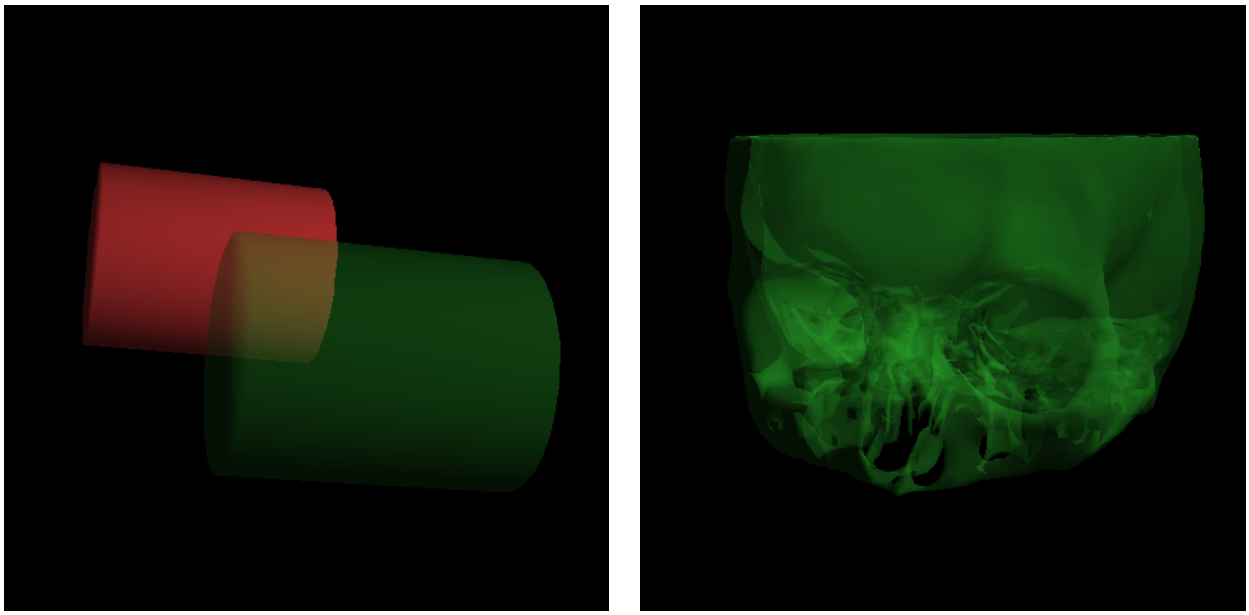
Na závěr spočítanou hodnotu spojíme s barvou pozadí (většinou počáteční inicializace pixelů ve frame bufferu):

$$\begin{aligned} C_{dst} &= A_{dst}C_{bg} + C_{dst} \\ A_{dst} &= 1 \end{aligned}$$

Detailnější vysvětlení rovnic najdete například zde (str. 6):
`http://www.jzy3d.org/documentation/DualDepthPeeling.pdf`

Co bude potřeba:

- Upravit fci RENDERSTUDENTSCENE() – po vykreslení scény přidat dodatečné zpracování. Pro každý pixel obrazovky:
  - Získat seznam fragmentů z pomocných bufferů a seřadit podle vzdálenosti od kamery (např. insert sort).
  - Projít seřazený seznam fragmentů v pořadí front-to-back a spočítat výslednou barvu pixelu pomocí rovnic pro alpha blending.
  - Barvu zkombinovat s barvou pozadí a uložit do frame bufferu.



Obrázek 6: Ukázky dalších modelů...

4 Poznámky k řešení

Pro vaše modifikace je připravená 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.

Doporučuji prostudovat princip základního rendereru (soubory RENDER.H a RENDER.C), velká část projektu je o modifikacích již existujících funkcí.

Jak přepnout na zobrazování pomocí studentského rendereru?
Konkrétní renderer se vytváří ve fci 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**.

5 Odevzdání

viz. Wiki stránka

6 Závěrem

Ať se dílo daří a ať vás grafika alespoň trochu baví!