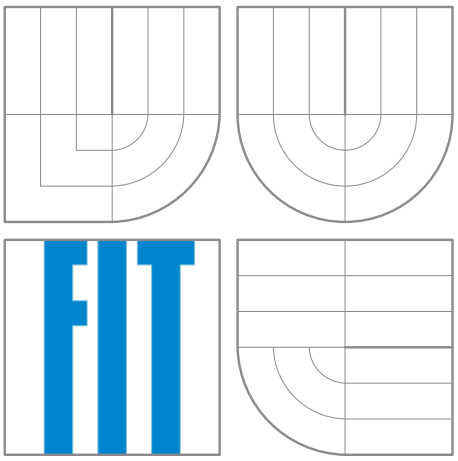


Projekt do předmětu GMU – Grafické a multimediální procesory

Výpočet siluet pomocí GPU

4. ledna 2015

Řešitelé: Zdeněk Biberle (xbiber00@stud.fit.vutbr.cz)
Vít Hodes (xhodes00@stud.fit.vutbr.cz)
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

1 Zadání

- Výpočet stínových těles
  - Implementace na GPU
  - Implementace na CPU pro porovnání rychlosti
- Vytvoření demonstrační aplikace
  - Vykreslení scény se stíny pomocí patřičně modifikovaného z-fail algoritmu
  - Měření rychlosti
  - Různé vizualizační pomůcky
- Volba či vytvoření vhodných modelů pro demonstraci finálního výsledku

2 Použité technologie

3 Technologie potřebné pro běh

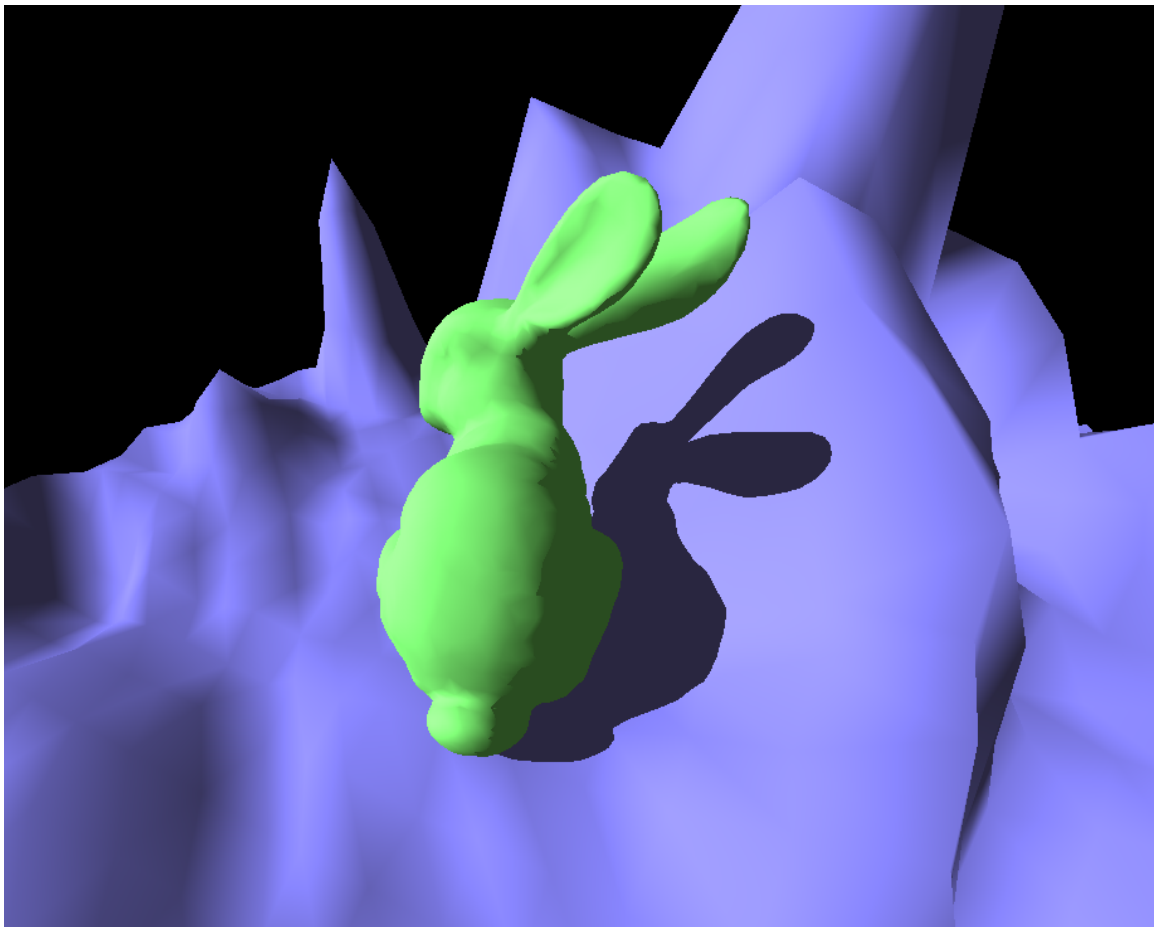
- Grafický akcelerátor s podporou OpenGL 4.3
  - SSBO
  - Image load/store
  - Compute shadery
- Knihovna SDL2
- Knihovna GLM
- Knihovna GLEW

4 Technologie použité pro tvorbu

- Blender
- Textový editor/vývojové prostředí vlastní volby

5 Použité zdroje

- Stanford bunny
- Utah teapot
- Byungmoon Kim, Kihwan Kim, Greg Turk - Real Time Shadow of Transparent Casters Using Shadow Volume, 2007
- Dokumentace OpenGL 4.3



Obrázek 1: Králíček vrhající stín na terén

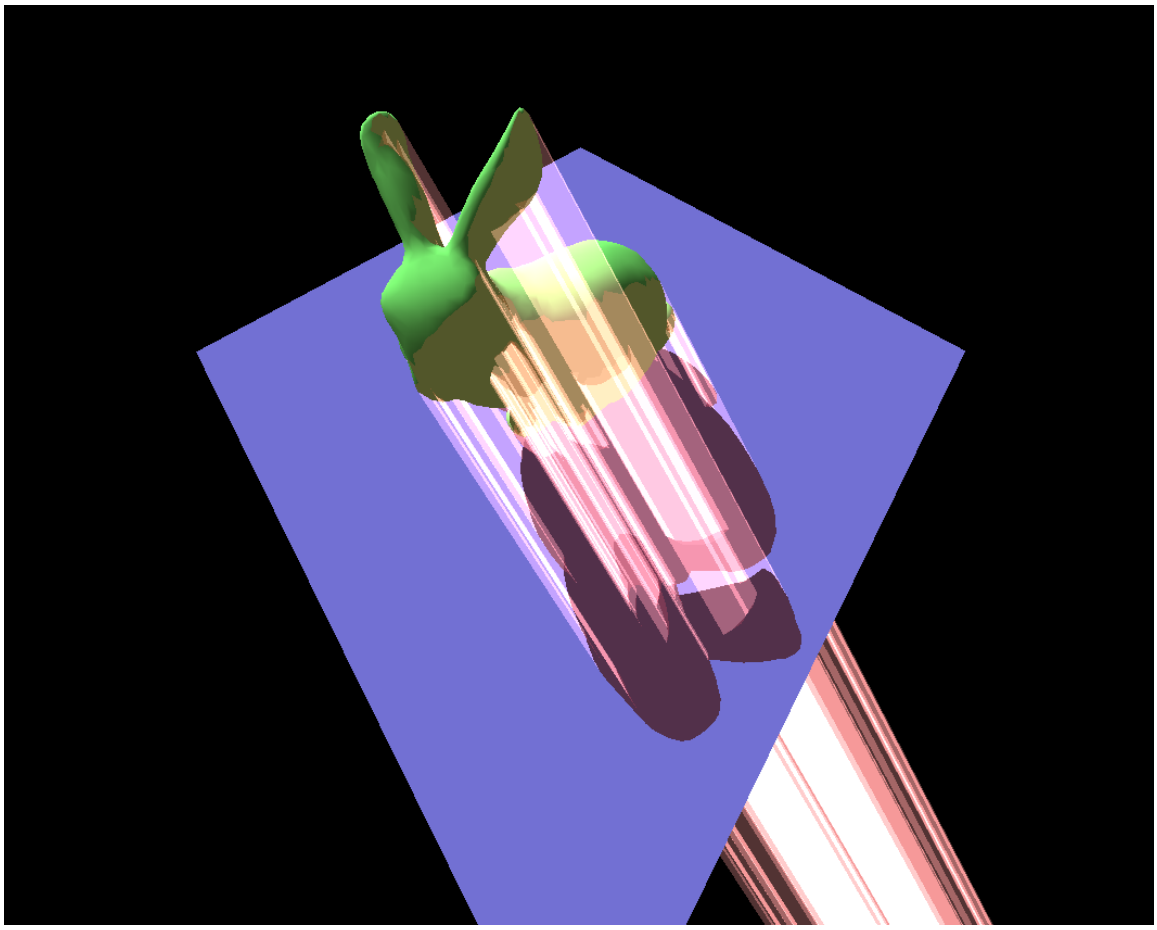
6 Nejdůležitější dosažené výsledky

6.1 Obecná funkčnost a funkcionalita aplikace

Aplikace provádí generování stínového tělesa pro libovolný model na CPU i na GPU a scénu poté zobrazuje s jeho použitím. Dodatečně umožňuje i zobrazení samotného stínového tělesa. Ovšem dle našich zkušeností nefunguje generování stínového tělesa na grafických kartách od společnosti AMD.

6.2 Generování stínových těles i pro non-manifold geometrii

Zadaný algoritmus umožňuje generování stínových těles i pro non-manifold geometrii. Na obrázku 3 vidíme několik čtverců vrhajících korektní stín. Každý tento čtverec je složen pouze ze dvou trojúhelníků a není tedy možné na ně aplikovat běžný algoritmus generování stínových těles (tj. protažení hran mezi dvěma trojúhelníky, kde jeden je natočen ke světlu a druhý je od světla odvrácen) tak, aby dosáhl správného výsledku.



Obrázek 2: Králíček a jeho stínové těleso

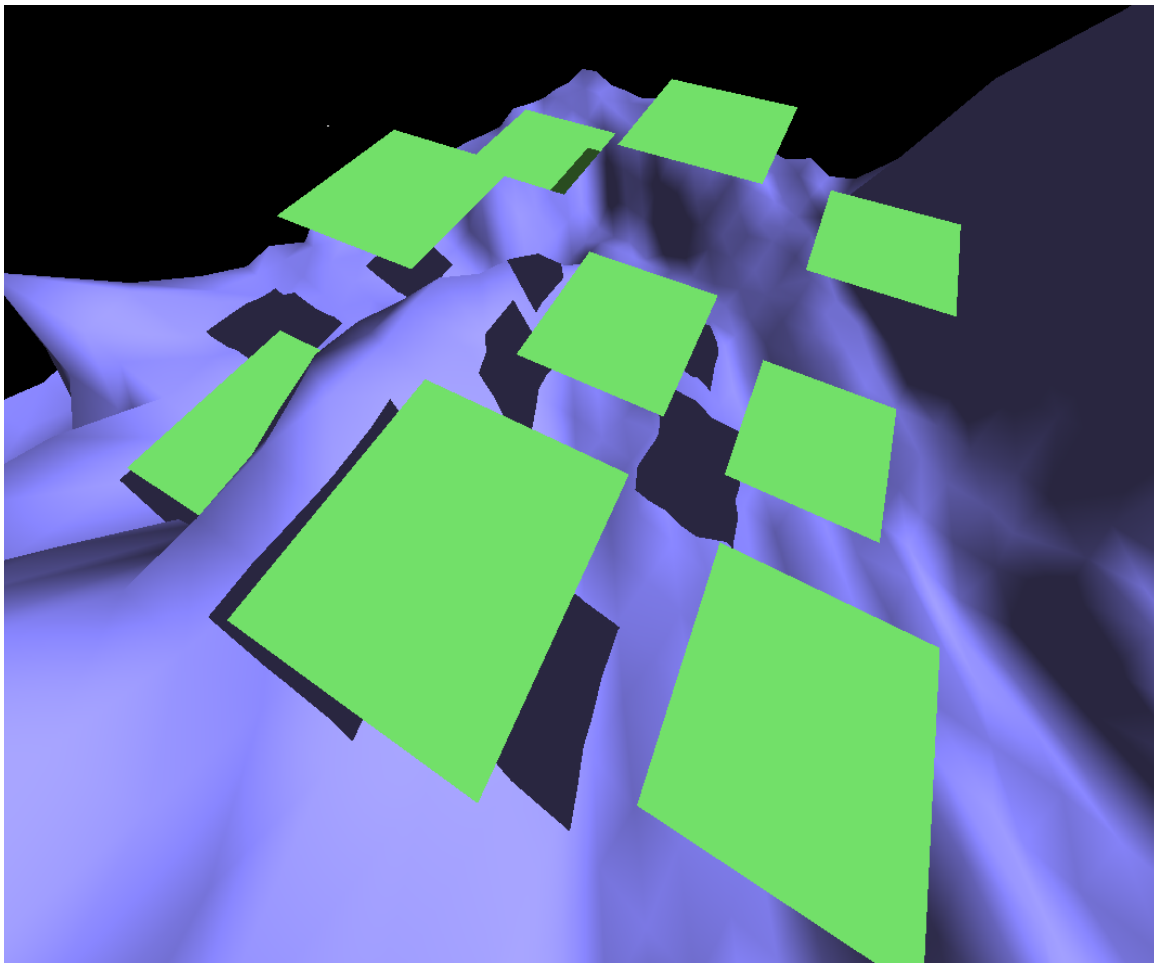
6.3 Zrychlení GPU implementace

Měřením času provádění CPU i GPU implementace při použití různých modelů jsme získali následující data:

Název modelu	Počet trojúhelníků	CPU - čas	GPU - čas	Zrychlení GPU oproti CPU
cube.obj	12	0,931 ms	0,195 ms	0,210
terrain.obj	1458	4,973 ms	1,302 ms	3,820
bunny.obj	4968	15,015 ms	2,095 ms	7,167
teapot.obj	36096	0,107 s	7,135 ms	15,023
teapot-big.obj	324864	0,949 s	50,974 ms	18,620

Měření bylo provedeno na jediném počítači s grafickou kartou od společnosti nVidia, který jsme měli k dispozici. Jeho HW a SW vybavení bylo následující:

- CPU: Intel Core i5-3230M, 2,60 GHz
- GPU: nVidia 730M
- RAM: 4 GiB DDR3 1.6 GHz
- Operační systém: Arch Linux, jádro linux-ck 3.18.1-8



Obrázek 3: Geometrie složená z několika non-manifold částí vrhající stíny

- Ovladač GPU: nvidia-ck, 343.36-3

Ze získaných výsledků vidíme očekávanou neefektivitu GPU implementace pro modely s opravdu nízkým počtem trojúhelníků, která je pravděpodobně způsobena dodatečnou režií. V případě většího počtu trojúhelníků již ovšem vidíme znatelné urychlení činnosti, zjevně stoupající s počtem trojúhelníků.

7 Obecný popis implementace algoritmu tvorby stínového tělesa

Algoritmus pro každý trojúhelník stínícího objektu provede následující:

- Zkontroluje, zda je trojúhelník natočen ke světlu. Pokud ano, tak jej přenese do výstupního bufferu a zároveň přidá i patřičně upravenou verzi tohoto trojúhelníku na konci stínového tělesa.
- Pro každou hranu tohoto trojúhelníku provede následující:
  - Binárním vyhledáváním nalezne v pomocném poli pro vyhledávání trojúhelníků všechny ostatní trojúhelníky, které sdílí tuto hranu.
  - Zkontroluje, zda je tento trojúhelník vlastníkem této hrany (tj. neexistuje trojúhelník, který sdílí tuto hranu a má nižší index). Pokud tomu tak není, tak přejde na další hranu.

- Projde nalezené přilehlé trojúhelníky a na základě jejich pozice v prostoru vůči světlu spočítá multiplicitu hrany.
- Pokud výsledná multiplicita není nulová, tak hranu protáhne a přenese do výstupního bufferu.

8 Ovládání vytvořeného programu

- Levé tlačítko myši - Společně s pohybem myši umožňuje změnu úhlu pohledu na scénu.
- Kolečko myši - Přiblížení a oddálení zobrazené scény.
- Klávesa R - Přepíná autonomní otáčení zobrazené scény.
- Klávesa T - Přepíná zobrazení vypočteného stínového tělesa.
- Klávesa C - Přepíná mezi CPU a GPU implementací výpočtu stínového tělesa.

9 Zvláštní použité znalosti

Použití OpenGL compute shaderů se různě liší od použití OpenCL či CUDA. To vyžadovalo získání dodatečných informací o jejich využívání a o komunikaci s nimi. Dále také využití image load/store v OpenGL shaderech pro vytvoření vlastního ”stencil bufferu”.

10 Rozdělení práce v týmu

Zdeněk Biberle Základ aplikace, GPU výpočet stínového tělesa, CPU výpočet stínového tělesa, dokumentace.

Vít Hodes Vykreslení scény za použití stínového tělesa, nahrávání modelů, CPU výpočet stínového tělesa.

11 Co bylo nejpracnější

Zdeněk Biberle Značnou část času jsem strávil nad komunikací mezi aplikací a compute shaderů. Později jsem zjistil, že implementace SSBO na grafických kartách od AMD se v současnosti vyznačuje různými problémy a tudíž jsem řešil nevyřešitelné.

Náročné bylo také vymyslet, jak vygenerovaná data použít pro z-fail algoritmus. To se nám ovšem ve finále úspěšně nepodařilo a ve výsledné aplikaci je tak použit algoritmus z-pass.

Vít Hodes Přijít na to, jak implementovat celočíselný ”stencil buffer”s obecným přístupem ze shaderů. Po prozkoumání několika slepých cest se dospělo k řešení pomocí iimage2D a atomického sčítání. Taktéž implementovat stíny nad shadow volume, který se mi na AMD kartě negeneruje správně se moc nedá.

12 Zkušenosti získané řešením projektu

Zdeněk Biberle

- Compute shadery
- SSBO
- Image load/store

Vít Hodes

- Framebuffer object, color/depth attachment
- V GLSL existují i shadow samplery a image samplery
- Image load/store, atomické operace obecně v shaderech

13 Autoevaluace

Technický návrh (75%): Technický návrh je slušný, problém byl vhodně rozložen na podproblémy, snad jen samotný výpočet stínového tělesa mohl být dodatečně rozdělen na dvě části (jedna zpracovávající trojúhelníky a druhá zpracovávající hrany).

Programování (65%): Vzhledem ke kratšímu rozsahu se moc neuplatňuje zapouzdření ve vykreslovací části. Znovupoužitelnost spíše znalostí než konkrétního kódu. Mimo vykreslovací část je kód poměrně dobře čitelný.

Vzhled vytvořeného řešení (40%): Estetická kvalita nebyla cílem řešení.

Využití zdrojů (80%): Byl využit pouze jeden literární zdroj, což je ovšem pro tento projekt dostatečné. Veškerý kód je původní a při jeho tvorbě byla značně využívána dokumentace OpenGL.

Hospodaření s časem (70%): Od počátku prací na projektu byl postup přiměřený a stabilní. Mohlo ovšem být věnováno více času implementaci algoritmu z-fail.

Spolupráce v týmu (80%): Komunikace a spolupráce byla z obou stran dobrá.

Celkový dojem (70%): Projekt byl zajímavý a potenciálně velmi užitečný. Škoda problémů s různými platformami (AMD x nVidia).

14 Doporučení pro budoucí zadávání projektů

Témata projektů byla dostatečně pestrá, ale příště by mohla být zadána dříve. Ovšem na druhou stranu se řešení tohoto projektu příliš nepřekrývalo s jinými projekty v semestru.