Path Tracer na platformě CUDA

Tomáš Král

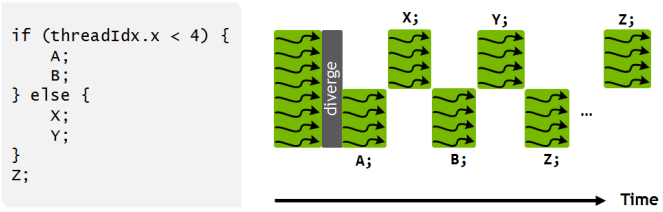
***Abstract*—Poznámky k projektu na PGRF3. Velice laické. Nemusí být zcela korektní. Work-In-Progress.**

I. Programovací model GPU

A. Grafický hardware

Základní vlastnost hardwaru od firmy Nvidia je, že výpočty jsou vždy prováděny ve skupině 32 threadů, která se nazývá *warp*. Všechny thready ve warpu běží paralelně. Problém může nastat ve chvíli, kdy některé thready potřebují vykonat jiný kód než ostatní thready. Taková situace může nastat například když jedna skupina threadů splňuje podmínku v konstruktu `if-else` a druhá nikoliv. Tato situace se ozančuje pojmem *divergence*.

Divergence je na hardwaru řešena tím, že některé thready jsou *zamaskovány*, tzn. po dobu vykonávání divergentní části kódu jsou vypnuty, viz. Obr. 1:



Obr. 1: Vizualizace maskování threadů při divergenci [1].

Divergence má pochopitelně negativní vliv na rychlost výpočtu - čím více budou jednotlivé thready divergovat, tím nižší bude úroveň paralelizace. Pro maximálně efektivní využití hardwaru je tedy nutné implementovat algoritmy tak, aby běh programu pokud možno co nejméně divergoval.

B. Paralelizace Path Traceru

Path-Tracing je možné paralelizovat několika způsoby. Asi nejjednodušší by bylo rozdělit výpočet na úrovni pixelů - co jeden pixel, to jeden thread. Jasnou nevýhodou takového postupu je vysoká divergence. Známou vlastností algoritmů path-tracingu totiž je, že množství výpočtů se pro různé pixely může masivně lišit. To je způsobené například odlišnou složitostí scény v různých segmentech obrazu.

Lepším způsobem by bylo paralelizovat výpočet jednotlivých vzorků jednoho pixelu. Pro typické použití Path-Tracingu je totiž běžné pro každý pixel počítat stovky až tisíce vzorků. Touto cestou je možné do jisté míry snížit množství divergence vycházející z rozdílných výpočetních nároků pro různé pixely. Nicméně zůstává divergence způsobená např. výpočtem různých BRDF, materiálů, průsečíků s různými geometrickými útvary atd.

Řešením problému paralelizace je tzv. *Wavefront* algoritmus [2], který vychází z tzv. *streaming path-tracing* postupu [3].

II. Implementace

A. Konfigurace systému a požadavky

Implementace byla provedena na platformě CUDA 12.2. Byl použit standard C++23 na kompilátoru GCC 13.2.1. Program byl testován na grafické kartě Nvidia GeForce MX 550M na OS Linux s verzí kernelu 6.5.4.

Implementace používá CUDA funkcionalitu jménem *Unified Memory* [4], která vyžaduje GPU s architekturou SM 3.0 nebo vyšší (řada Kepler a novější). Některé pokročilé funkce *Unified Memory* jsou dostupné pouze na OS Linux, ty ale **doufám** nebyly použity.

Bylo použito několik externích knihoven: fmt, GLM. Tyto knihovny jsou do projektu zakomponovány pomocí package manageru vcpkg a buildovacího systému CMake.

References

[1] "Inside Volta: The World's Most Advanced Data Center GPU," 2017. Accessed: Sep. 28, 2023. [Online]. Available: https://developer.nvidia.com/blog/inside-volta/

[2] S. Laine, T. Karras, and T. Aila, "Megakernels considered harmful: wavefront path tracing on gpus," in *Proc. 5th High-Performance Graph. Conf.* in Hpg '13, Anaheim, California, 2013, p. 137, doi: 10.1145/2492045.2492060. [Online]. Available: https://doi.org/10.1145/2492045.2492060

[3] D. G. Van Antwerpen, "Unbiased physically based rendering on the GPU," 2011. Accessed: Sep. 29, 2023. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A4a5be464-dc52-4bd0-9ede-faefdaff8be6

[4] "1. Introduction — CUDA C Programming Guide." Accessed: Sep. 29, 2023. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#unified-memory-introduction