

Střední průmyslová škola strojní   
a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3

Paralelizace 3D renderování

Maturitní práce

Autor **Oliver Řezníček**

Obor **Informační technologie**

Vedoucí práce **Ing. Marek Pospíchal**

Školní rok **2022/23**

Anotace

Práce se zabývá paralelizací procesů, vykreslením stínů a efekty následného zpracování v open-source 3D herním enginu Dotrix. Bude primárně sloužit jako výzkum a reference před implementací do samotného enginu.

Summary

This work deals with process parallelization, shadow rendering and post-processing effects in the open-source 3D game engine Dotrix. It will primarily serve as research and reference prior to implementation into the engine itself.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne

Oliver Řezníček

Obsah

[Úvod 1](#_Toc118834279)

[1 První kapitola 2](#_Toc118834280)

[Závěr 3](#_Toc118834281)

[Seznam zkratek a odborných výrazů 4](#_Toc118834282)

[Seznam obrázků 5](#_Toc118834283)

[Použité zdroje 6](#_Toc118834284)

[A. Seznam přiložených souborů I](#_Toc118834285)

Úvod

Téma renderování a matematiky, která stojí za vykreslením počítačové grafiky mě vždycky zajímalo, ale nikdy jsem se mu nevěnoval tak do hloubky, jak bych chtěl. A o letních prázdninách jsem poměrně náhodou narazil na firmu Löwenware, kde jsem s odbornou pomocí vyhotovil demo *GPU driven renderování* (vykreslování řízené grafickou kartou) a toto téma mě zaujalo natolik, že jsem se rozhodl ho zpracovat jako maturitní práci. Postupně však k z počátku jednoduché scéně přibydou složitější způsoby vykreslování a náročnější efekty.

Tato práce tedy bude moci posloužit všem těm, kteří by chtěli více do hloubky pochopit, jak funguje renderování 2D nebo 3D grafiky. A mohu vám dopředu říct, že se znalostí WGPU budete mít velice dobré chápání toho, jak fungují dnešní moderní grafické API[[1]](#footnote-1).

# Použité technologie

## Rust

Rust je open-source, multiparadigmatický, víceúčelový programovací jazyk. Oproti jiným, jemu podobným programovacím jazykům jako je např. C nebo C++ nabízí bezpečnost paměti, která umožňuje vytvářet spolehlivé a bezpečné aplikace.

Unikátní na Rustu je hlavně to, jakým způsobem řeší bezpečný přístup do paměti. Má totiž tzv. systém vlastnictví a půjčování (ownership and borrowing), díky kterému dokáže řešit chyby, na které byste narazili v jazyce C až při běhu programu, už během kompilace. Takže se vyhnete častým chybám paměti, a to i bez použití garbage collectoru.[[2]](#footnote-2)

## WGPU

WGPU je grafické API napsané v Rustu, které umožňuje kompilaci pro více platforem díky tomu, že beží na nativních backendech jako je Vulkan, Metal, DirectX a OpenGL.

To znamená, že můžeme naši grafickou aplikaci přepnout na jiný backend a tak sestavení směřovat na úplně jinou platformu. Jinými slovy řečeno bude naše aplikace spustitelná na všech platformách. Někdo by mohl argumentovat, že aplikace napsaná ve Vulkanu nebo OpenGL bude taky multiplatformní a asi by měl pravdu, ale výhodou WGPU je že aplikace může běžet na takovém backendu, který je pro nás nebo danou platformu nejlepší.

Vývojáři by samozřejmě mohli integrovat víc grafických API do jejich herního enginu, ale to zabere spoustu času a vše se musí dopředu promyslet. Použití WGPU by jim v takovém případě práci dost usnadnilo, protože vytváří abstrakci nad všemi známými backendy.

### WebGPU

Pokud vás už podle nadpisu napadlo, že by mohla existovat určitá souvislost mezi WGPU a WebGPU, nejste na omylu (WGPU je pouze rust implementací WebGPU).

WebGPU je nadcházející moderní grafické API, které rozšíří možnosti grafiky na webu např. o compute shadery. Mluvím v budoucím čase, protože v době, kdy tuto práci píšu ještě WebGPU není plně integrované do prohlížečů. Zatím se objevilo pouze v testovacích verzích prohlížečů jako je Firefox Nightly nebo Chrome Canary.

# Teoretická část

## Paralelizace

Paralelizace je proces, při kterém je náš program nebo výpočet rozdělen na více úloh, které běží na více procesorech. Díky rozdělení zátěže mezi více procesorů můžeme ušetřit energii a znásobit výpočetní výkon (Výkon jednoho procesoru × počet procesorů[[3]](#footnote-3)).

V případě této práce se bude jednat o přesun grafických výpočtů ze strany procesoru na grafickou kartu. Grafické výpočty jsou opakované často za sebou a dají se snadno rozložit mezi více jader (např. násobení matic). Grafická karta je tedy ideální výpočetní jednotkou pro provádění těchto operací. Má v sobě spoustu jader, které ani zdaleka nedosahují výkonu hlavního procesoru, ale jejich síla spočívá právě v počtu.

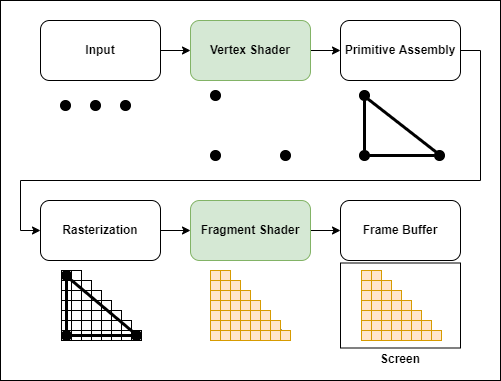
Paralelizaci chceme tedy používat hlavně v případě výpočtů, které se dají rozložit na menší, jednodušší a využijeme u nich spíš velké množství procesorů než procesor jeden s velkou frekvencí.

U paralelních výpočtů je však důležitá ještě jedna věc, kterou jsem nezdůraznil, a to je souběh. Někdy potřebujeme, aby jednotlivé dílčí úlohy byly vykonávány synchronně. Předávání informací mezi úlohami může být v takovém případě dost problematické.

## Vykreslování (Rendering)

Rendering neboli vykreslování je proces, kdy měníme naše modely objektů (jejich matematickou reprezentaci) na viditelný 2D obraz, který si můžeme zobrazit na monitoru nebo jiném zobrazovacím zařízení.

## Rendering pipeline

Rendering pipeline je sekvence kroků, kterými musí naše objekty projít, než se vykreslí na obrazovku. Typicky se dělí na tyto uvedené na obrázku:

Zeleně vybarvené části jsou programovatelné. Kód, kterým můžeme ovlivnit chování v těchto krocích se píše do shaderu.

### Vertex stage

Vertex stage je zodpovědná za zpracování dat jednotlivých vertexů. To většinou zahrnuje transformaci souřadnic z world space do view space[[4]](#footnote-4). Typicky zde provádíme maticové operace jako např. násobení.

### Geometry stage

Po vertex stage následuje geometry stage nebo také tesselation stage. Ta je zodpovědná za sestavování primitiv (např. trojúhelníků) z vertexů.

### Rasterization stage

Rasterization stage určuje pixely, které překrývají daná primitiva, abychom mohli vypočítat barvu všech překrytých pixelů.

### Fragment stage

Ve fragment stage spočítáme pro každý pixel překrytý primitivem očekávanou barvu. Barva se většinou předává rovnou s vertexy nebo ji můžeme namapovat z textury. Také zde můžeme provádět složitější výpočty jako světlo

## Herní engine

Herní engine je vývojové prostředí speciálně navržené tak, aby pomáhalo vývojářům vytvářet hry efektivněji. Poskytuje vývojářům sadu nástrojů, které umožňují tvorbu počítačových her bez nutnosti stavět vše od začátku. Typicky herní engine poskytuje nástroje pro vykreslování 2D a 3D grafiky, simulaci fyziky, skriptovací jazyk a různé další funkce.

Zjednodušeně se tak vývojáři mohou více soustředit na vytváření herních mechanismů a ostatní věci nechat na enginu.

### Vykreslovací engine

Vykreslovací engine je zodpovědný za vykreslení jednotlivých objektů, stínů a dalších věcí v naší scéně. Všechno, co v počítačové hře vidíte na obrazovce je tedy (mimo spousty práce, kterou museli hře věnovat herní vývojáři a designéři) zásluhou vykreslovacího enginu.

Důležitý je pro nás tento termín hlavně proto, že vykreslovací engine bude jedinou částí herního enginu, kterou se tato práce bude zabývat.

# Examples

Celý projekt je rozdělen na několik příkladů, které popisují, jak se projekt postupně vyvíjel. Příklady nám tak pomáhají lépe pochopit jednotlivé problémy a jejich konkrétní implementaci.

## *Framework.rs*

Celý návrh tohoto projektu vychází z WGPU githubu a stěžejní pro nás bude soubor *framework.rs* nacházející se ve složce examples.

*Obsah obrázku text

Popis byl vytvořen automaticky*Na začátku souboru se nachází *trait[[5]](#footnote-5)* s názvem *Example*, který představuje náš konkrétní příklad. Ten budeme pro ulehčení používat vždy při vytváření nového examplu.

Následují tři podprogramy *setup*, *start* a *run.*

## Triangle

Úplně prvním příkladem je jednoduchý trojúhelník, na kterém si vysvětlíme proces vytváření render pipeline,

Závěr

Tak jsem se dostal až na konec.

Seznam zkratek a odborných výrazů

Engine

Softwarový engine je software, který poskytuje základní funkce nebo mechanismy pro ostatní software. Je to jako "motorka" nebo "stroj", který umožňuje ostatnímu software fungovat nebo vykonávat určité úkoly. Softwarové enginy se často používají k vytváření her, 3D grafiky, databází, vyhledávačů nebo dalších typů aplikací, a poskytují společné funkce nebo technologie pro tyto aplikace. Představte si softwarový engine jako "podpůrný systém", který umožňuje ostatnímu software být vyvinut rychleji nebo snadněji, protože mnoho základních funkcí je již implementováno v rámci engine.

Vertex

Vertex je bod v prostoru

Seznam obrázků

Použité zdroje

1. **sotrh.** Textures and bind groups. *Learn Wgpu.* [Online] https://sotrh.github.io/learn-wgpu/beginner/tutorial5-textures/.

2. **WebGPU Shading Language. *W3C.* [Online] https://www.w3.org/TR/WGSL/.**

**3. WGPU documentation. *Docs.rs.* [Online] https://docs.rs/wgpu/latest/wgpu/.**

**4. *The Rust Programming Language - book.* [Online] https://doc.rust-lang.org/book/.**

**5. Paralelní výpočty. *Wikipedie.* [Online] https://cs.wikipedia.org/wiki/Paraleln%C3%AD\_v%C3%BDpo%C4%8Dty.**

**6. AZAT. History of Rust Programming Language. *TechnoSuggest.* [Online] 28. Červenec 2022. https://technosuggest.com/history-of-rust-programming-language/.**

**7. Rust (programovací jazyk). *Wikipedie.* [Online] https://cs.wikipedia.org/wiki/Rust\_(programovac%C3%AD\_jazyk).**

1. Seznam přiložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

* **MP2010-Novák-Jan-L4-Tepelné\_čerpadlo.docx** – editovatelná verze dokumentace maturitní práce
* **MP2010-Novák-Jan-L4-Tepelné\_čerpadlo.pdf** – tisknutelná verze dokumentace maturitní práce
* **Výkresy** – kompletní výkresová dokumentace
* **Aplikace** – zdrojové kódy

1. Příkladem takového moderního grafického API může být Vulkan. [↑](#footnote-ref-1)
2. Jednotlivými koncepty a syntaxem jazyka Rust se ve své práci nebudu dopodrobna zabývat, takže pokud byste se chtěli o Rustu dozvědět víc a třeba si v něm i zkusit napsat vlastní program, určitě bych pro začátek doporučil elektronickou knihu Rust book, kterou mám uvedenou ve zdrojích. [↑](#footnote-ref-2)
3. Procesor možná není v tomto případě úplně vhodný termín, protože se může jednat pouze o vlákno nebo jádro našeho procesoru v PC. Vhodnější by byl asi termín výpočetní jednotka, ale pro jednoduchost necháme procesor. [↑](#footnote-ref-3)
4. World space je náš 3D svět do kterého jsme si umístili nějaké objekty. Oproti tomu view space je prostor naší 2D obrazovky, kam chceme naše objekty přetransformovat. [↑](#footnote-ref-4)
5. Trait v Rustu je něco jako abstraktní třída nebo interface. Umožňuje přidat chování do třídy bez nutnosti použití dědičnosti. V našem případě nám definuje nějaké metody a mohl by i vytvářet jejich defaultní implementaci. [↑](#footnote-ref-5)