VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENIE TERÉNU NAD VULKAN API

TERRAIN RENDERING ON TOP OF VULKAN API

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JOZEF MÉRY**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN KÁČERIK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Méry Jozef**
Program: Informační technologie
Název: **Zobrazení terénu nad API Vulkan**
Terrain Rendering on Top of Vulkan API
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se se základy vykreslování pomocí API Vulkan.
2. Seznamte se s technikami vhodnými pro procedurální generování terénu a s knihovnami, které je implementují.
3. Prostudujte techniky umožňující efektivní zobrazování rozsáhlých scén.
4. Navrhněte renderer využívající API Vulkan, který umožní s pomocí nastudovaných technik zobrazovat rozsáhlý vygenerovaný terén.
5. Naimplementujte navrženou aplikaci.
6. S výslednou aplikací proveďte měření a zhodnoťte vizuální kvalitu jejího výstupu.
7. Vytvořte video s demonstrací odvedené práce.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- body 1 až 4 zadání
- prototyp aplikace

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Káčerik Martin, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Práca popisuje framework, ktorý je schopný vytvoriť a vykresliť jednoduchý procedurálny terén pomocou Vulkan API. Obsahuje ľahký úvod do teórie procedurálneho generovania a popis vytvorených systémov ako napríklad systém udalostí a entít. Cieľom tejto práce nie je skúmanie rôznych techník na tvorenie procedurálneho terénu. Terén slúži ako vizuálna demonštrácia funkčnosti frameworku.

Abstract

This thesis describes a framework, which has the ability to create and draw a procedural terrain using Vulkan API. It includes a simple introduction to the theory of procedural generation and description of systems for example entity or event system. The goal of this thesis isn't investigating various procedural terrain generation techniques. The purpose of the terrain is a visual demonstration of functionality of the framework.

Kľúčové slová

počítačová grafika, Perlinov šum, procedurálna generácia, procedurálny terén, Vulkan API, LunarG SDK, GLSL, CMake, SDL2, GLM

Keywords

computer graphics, Perlin noise, procedural generation, procedural terrain, Vulkan API, LunarG SDK, GLSL, CMake, SDL2, GLM

Citácia

MÉRY, Jozef. *Zobrazenie terénu nad Vulkan API.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Káčerik

Zobrazenie terénu nad Vulkan API

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Martina Káčerika. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Jozef Méry
16. mája 2019

Poďakovanie

Chcel by som sa veľmi pekne poďakovať vedúcemu Ing. Martinovi Káčerikovi za všetkú pomoc a odborné rady často nad rámec tejto práce. Ďalej by som sa chcel poďakovať svojej rodine za finančnú a morálnu podporu počas štúdia.

Obsah

Kapitola 1

Úvod

Túto tému som si vybral, pretože grafika je oblasť IT, ktorá ma najviac zaujíma a chcel by som sa s ňou zaoberať aj v budúcnosti. Táto práca pre mňa znamená prvý krok za týmto cieľom. V tejto práci sa zaoberám hlavne návrhom a implementáciou rôznych systémov a rozhraní, pričom sa funkcionalitou a rozhraním pokúšam priblížiť k existujúcim frameworkom ako napríklad `SFML`, `P5.js` alebo `Processing`. Môj osobný cieľ pre tento projekt je naučiť sa pracovať s `Vulkan API`, `CMake` a komplexnou programovou štruktúrou.

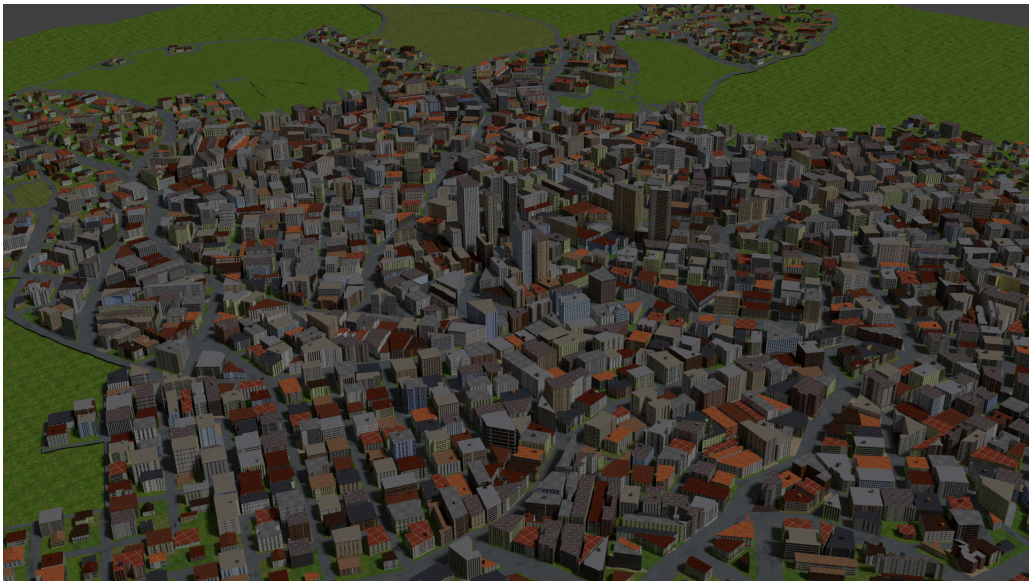
Hlavným cieľom tejto práce je vytvoriť tento framework tak, aby bol schopný procedurálne vytvoriť jednoduchý terén a následne vykresliť pomocou Vulkan API. V kapitole 2 sa zaoberám ľahkým úvodom do teórie procedurálneho generovania. Keďže prakticky všetky systémy som vytvoril zo svoje fantázie, nepopisujem v tejto kapitole teóriu návrhu systémov. V kapitole 3 prezentujem minimálny potrebný návrh, ktorého implementácia by mala byť schopná vyprodukovať požadovaný výstup. V kapitole 4 popisujem štruktúru projektu, vytvorené systémy, rozhrania a abstrakcie a ľahký úvod do práce s frameworkom. V kapitole 5 porovnávam návrh a implementáciu a hodnotím výstup projektu, ďalej popisujem výhody a nedostatky a možnosti na vylepšenie do budúcna.

Kapitola 2

Teória

2.1 Čo je procedurálne generovanie?

Procedurálne generovanie je schopnosť počítača vytvárať isté prvky virtuálneho sveta bez alebo s minimálnym zásahom človeka. Táto tvorba býva parametrizovateľná, vďaka čomu je možné dosiahnuť takmer nekonečné množstvo variácií tvoreného prvku. Typické procedurálne tvorené prvky sú mapy, levely, terén, úlohy, rastliny, poloha, postavy, budovy alebo dokonca celé planéty a galaxie. Nevyhnutnou vlastnosťou akéhokoľvek takto vytvoreného prvku je použiteľnosť a zmysluplnosť v danom kontexte alebo virtuálnom svete. Vytvorená úloha by mala byť splniteľná, postava by sa mala istými vlastnosťami dostatočne podobať na ostatné, ktoré nemuseli byť vytvorené procedurálne [12].



Obr. 2.1: Ukážka procedurálne vygenerovaného mesta, prevzaté z [10]

Obr. 2.2: Ukážka procedurálne vygenerovaných stromov, prevzaté z [6]

2.2 Prečo využiť procedurálne generovanie?

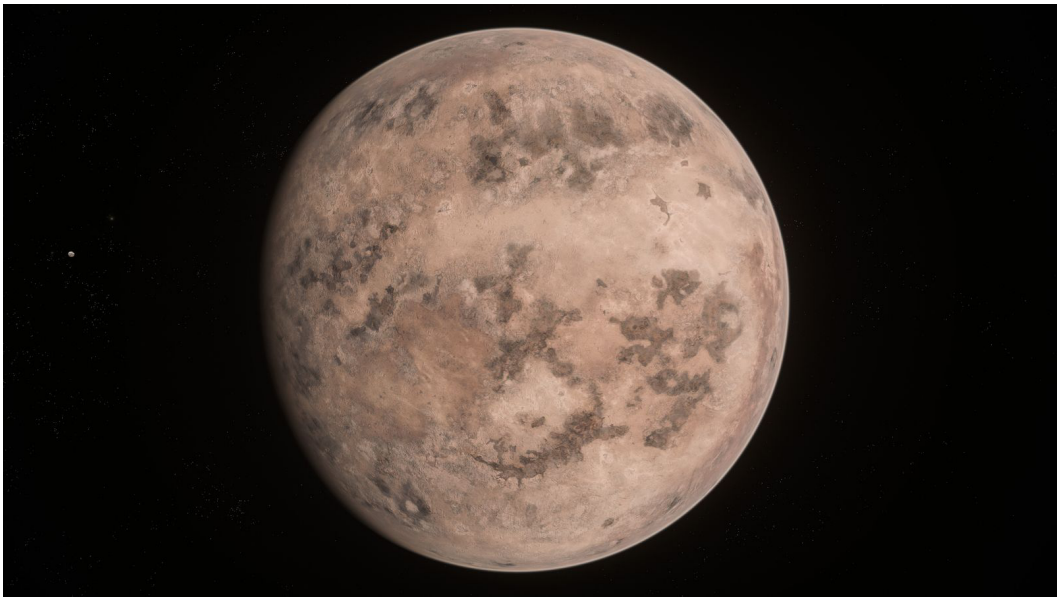
Veľmi jednoduchým príkladom, čo by sa mohol považovať za procedurálne generovanie je hra Solitaire. Jedná sa o veľmi jednoduché náhodné rozloženie kariet, avšak tak, aby to odpovedalo pravidlám hry. Vďaka tomuto procedurálnemu vytvoreniu každej novej hry existuje viac možných hier ako prešlo sekúnd od veľkého tresku. Je nepredstaviteľné, aby všetky tieto možnosti vytvorili vývojári hry manuálne. Z toho vyplýva, že procedurálne generovanie je vhodné využiť v prípadoch keď je prakticky nemožné manuálne vytvoriť všetky alebo dostatočné množstvo variácií daného prvku, či už z časového dôvodu alebo aj napríklad finančného.

2.3 Nevýhody procedurálneho generovania

Napriek rôznym vyššie uvedeným výhodám, ako snáď všetko, aj procedurálne generovanie má svoje nevýhody a prehnané použitie môže viesť ku katastrofálnym výsledkom. Keďže počítače a algoritmy nie sú schopné myslieť „mimo krabice“, nimi vytvorené prvky môžu byť neunikátne a nudné [11].

2.4 Zlatá stredná cesta procedurálneho generovania

Ideálny prípad je zdravý mix procedurálneho generovania a manuálny dotyk umelca. Procedurálne vytvoriť základ, kostru a manuálne pridať unikátne prvky. Týmto spôsobom je možné vyhnúť sa hlavnej nevýhode, avšak použitím tohto postupu je možné sa dopracovať opäť len k obmedzenému množstvu variácií, ktoré sa dajú považovať za unikátne a zaujímavé. Ďalšou výhodou tohto postupu je, že generovanie sa nemusí vykonať v reálnom čase.



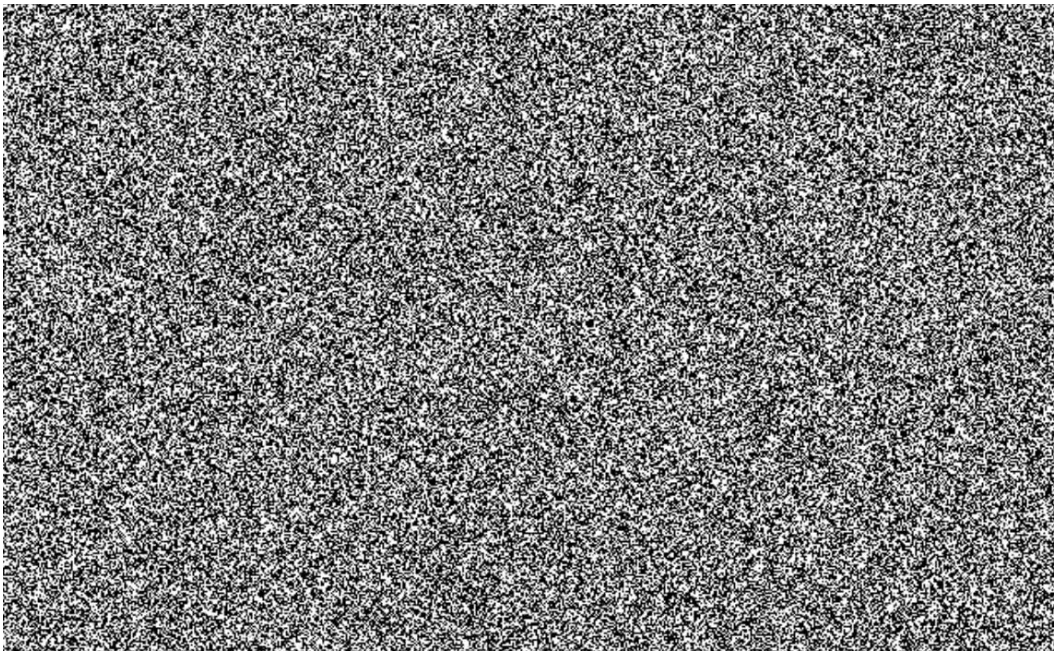
Obr. 2.3: Procedurálne vygenerovaný mesiac Daymar z hry Star Citizen doladený umelcami, prevzaté z [4]

2.5 Generovanie terénu

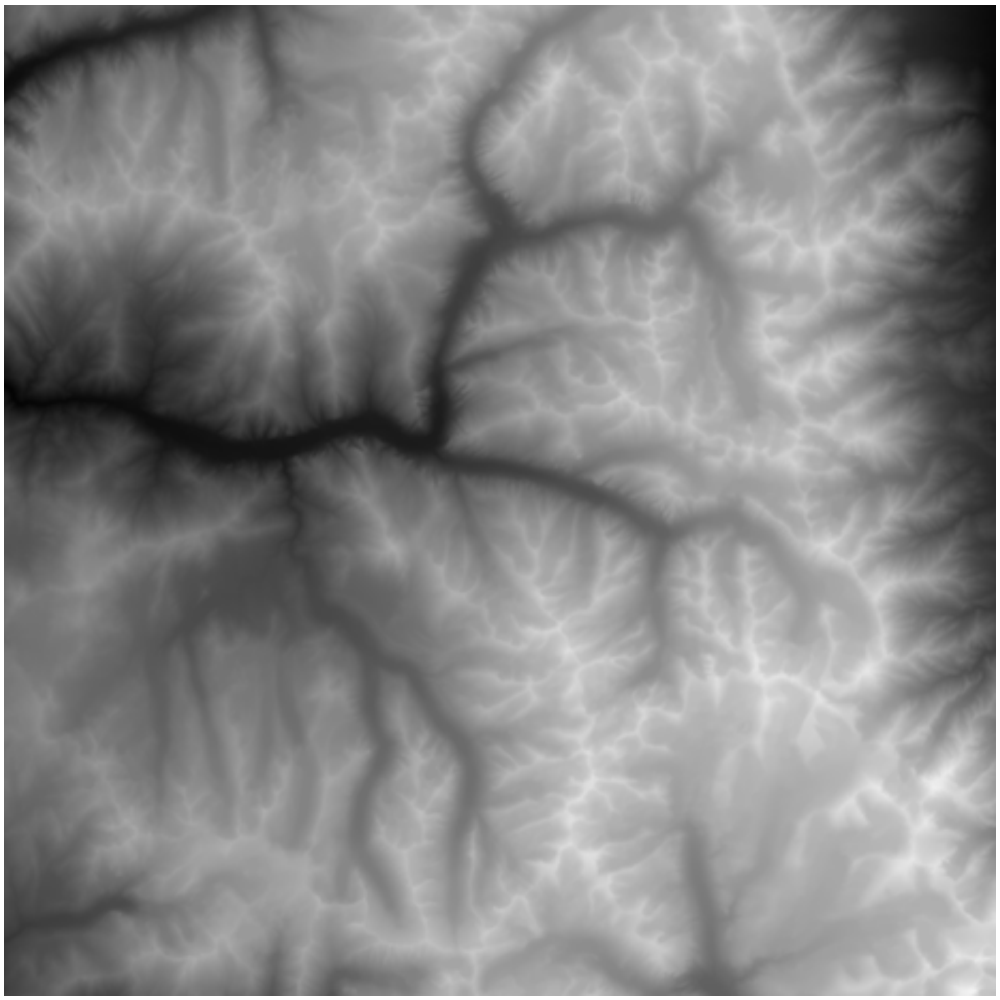
Terén je častým subjektom procedurálneho generovania, keďže manuálna tvorba je časovo veľmi náročná. Najčastejší postup pri tvorbe terénu je vytvorenie plochej trojuholníkovej siete a aplikácia výškovej mapy na túto sieť. Výšková mapa sa dá chápať ako 2D pole, ktoré obsahuje náhodné čísla, ktoré sú aplikované na sieť terénu. Ďalšia možná reprezentácia je čiernobiely obrázok a odtieň odzrkadľuje výšku.

Kým vyššie uvedený obrázok sa dá považovať za výškovú mapu, terén vytvorený takouto mapou by mal veľmi ďaleko od reálneho terénu a bol by prakticky nevyužiteľný v akejkoľvek aplikácií. Z toho dôvodu sa prakticky často používa Perlinov šum. Jedná sa o náhodné čísla, ktoré sú si navzájom podobné. Ak uvažujeme, že čierny pixel vyššie uvedeného obrázku má hodnotu 0 a biely 1, potom maximálny rozdiel hodnoty akéhokoľvek pixelu a jeho „suseda“ je 1. Pomocou Perlinovho šumu je možné vytvoriť výškovú mapu, kde sú tieto rozdiely oveľa menšie, napríklad 0,01. Vďaka tomu sú prechody medzi hodnotami oveľa jemnejšie a terén vytvorený takouto mapou vyzerá oveľa dôveryhodnejšie [1].

Obr. 2.4: Manuálne umiestnené budovy na povrchu vyššie uvedeného mesiaca Daymar, prevzaté z [3]



Obr. 2.5: Ukážka výškovej mapy, ktorá obsahuje nezávislé náhodné hodnoty, prevzaté z [5]



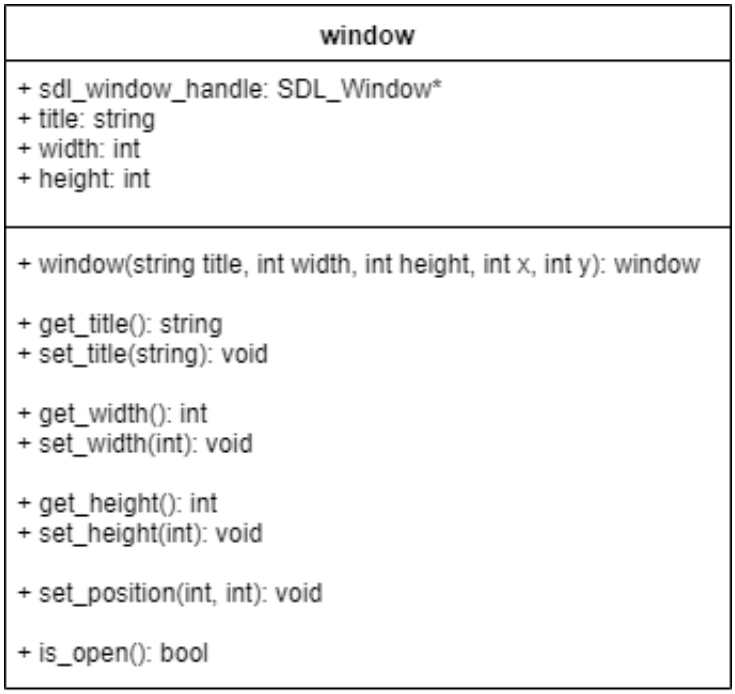
Obr. 2.6: Ukážka výškovej mapy, ktorá obsahuje závislé náhodné hodnoty, prevzaté z [13]

Kapitola 3

Návrh

3.1 Práca s oknom

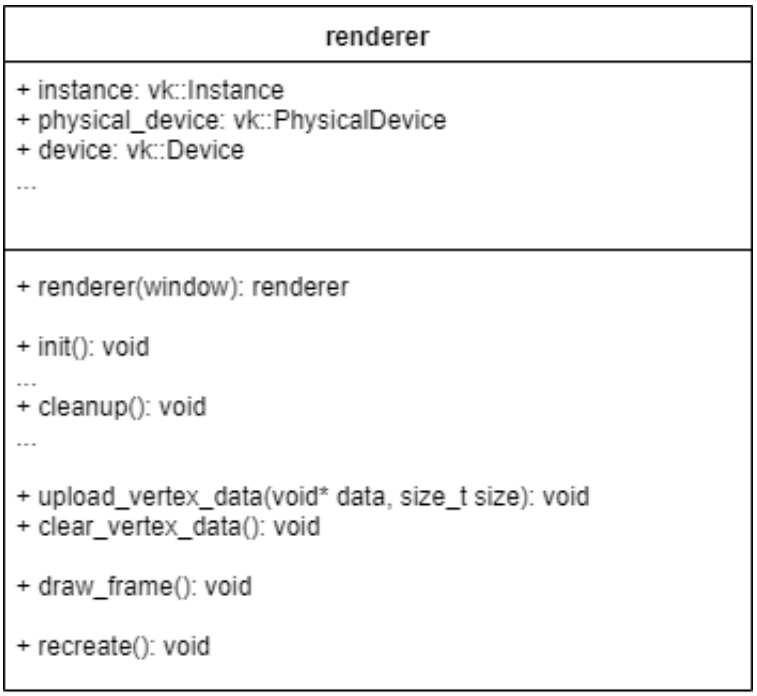
Pre prácu s oknom plánujem použiť externú knižnicu SDL, aby aplikácia fungovala na rôznych platformách. Táto trieda má byť zodpovedná za spustenie a zastavenie knižnice SDL a vytvorenie okna tak, aby jeho povrch bol použiteľný s Vulkan API. Ďalej by mala ponúkať rozhranie na základnú prácu s oknom ako nastavenie a získanie názvu.



Obr. 3.1: Základné rozhranie pre prácu s oknom

3.2 Renderer

Renderer by mal byť schopný kresliť na povrch aspoň jedného okna, pričom ponúka možnosť nahrať vertex dáta a ich mazanie a reagovanie na zmenu veľkosti okna. Tento objekt by mal zapuzdrovať všetky objekty, ktoré sú potrebné pre vykresľovanie.



Obr. 3.2: Základné rozhranie rendereru

3.3 Demonštračná aplikácia

Táto aplikácia má byť zodpovedná za spustenie a zastavenie všetkých systémov, vytvorenie potrebných objektov ako renderer a okno a obsahovať hlavný cyklus, ktorý sa ukončí po zatvorení okna.

```
main() {

    main_window = window("demo", 500, 500, 0, 0)

    vulkan_renderer = renderer(main_window)

    demo_terrain = terrain()

    renderer.upload_vertex_data(demo_terrain.data(), demo_terrain.size())

    while(main_window.is_open()) {

        renderer.draw_frame()
    }

    renderer.clear_vertex_data()
}
```

Obr. 3.3: Pseudokód reprezentujúci hlavnú činnosť aplikácie

Kapitola 4

Implementácia

4.1 Použité nástroje

Ako implementačný jazyk som zvolil `C++`, štandard 17 kvôli výkonu a pestrej štandardnej knižnici. Na popis štruktúry projektu, závislostí a automatizáciu istých úkonov som použil `CMake`, čo mi výrazne uľahčilo zostavenie a inštaláciu. Na vývoj Vulkan rendereru som použil `LunarG SDK` [8].

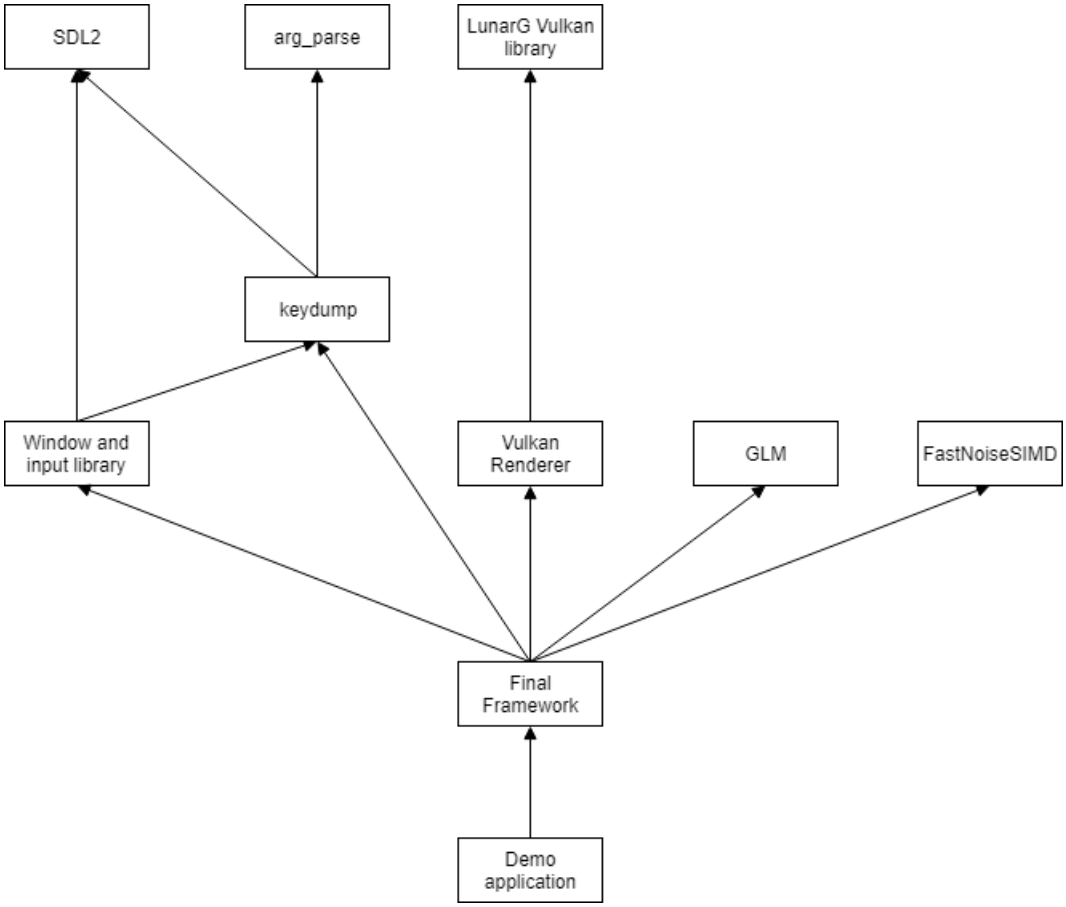
4.2 Štruktúra projektu

Projekt som rozdelil do štyroch hlavných častí, ktoré sú reprezentované nasledujúcimi knižnicami alebo spustiteľnými súbormi:

- Knižnica na prácu s oknami a užívateľského vstupu
- Knižnica na vykresľovanie pomocou `Vulkan API`
- Knižnica, ktorá spája a zapúzdruje vyššie uvedené časti a pridáva množstvo iných systémov ako udalosti a entity
- Demonštračná aplikácia

4.3 Knižnica pre prácu s oknami

Ako prvú hlavnú časť som implementoval knižnicu na prácu s oknami a užívateľským vstupom. Aby som si prácu mnohokrát uľahčil, použil som opensource knižnicu `SDL` (Simple DirectMedia Layer), verzia 2.0.9. Hlavnou myšlienkou tejto knižnice je zapuzdrenie základných časti `SDL` knižnice do `C++` objektov a to hlavne inicializáciu samotnej knižnice, základná práca s oknom a vstupom, konkrétne myši a klávesnice. Zaujímavou funkcionalitou okien je registrácia obslužných funkcií pre rôzne udalosti, ktoré ponúka knižnica `SDL` a to pre každé okno samostatne [7].



Obr. 4.1: Graf závislostí projektu

4.3.1 Pomocný program keydump

Keďže som chcel vytvoriť knižnicu bez závislosti na `SDL`, vytvoril som pomocný program, ktorý som nazval keydump. Jeho jediná úloha je vytvorenie hlavičkového súboru, ktorý obsahuje všetky dostupné klávesy, ktoré ponúka `SDL` reprezentované pomocou enum triedy, pričom hodnoty odpovedajú hodnotám v `SDL`. Týmto spôsobom je možné priame mapovanie takto vytvorenej enum hodnoty na hodnoty v `SDL`. Program pracuje s jedným hlavným argumentom a to `„-namespace=VALUE"`, pomocou ktorého možno vytvorenú enum triedu vložiť do zadaného namespace. Spracovanie argumentov z príkazového riadku som riešil pomocnou knižnicou `arg_parse` a automatizáciu vytvárania hlavičkového súboru pomocou `CMake`.

4.4 Vulkan API renderer

Ďalšou podstatnou implementovanou časťou je `Vulkan API renderer`. Knižnica je rozdelená do nasledovných podstatných objektov:

- Jadro reprezentujúce inštanciu, fyzické a logické zariadenie a ďalšie objekty, ktoré sú potrebné len raz ako napríklad command pool. Na jeho spustenie je potrebný referenčný povrch, ktorý ponúka vyššie popísaná knižnica. Vďaka tomuto je jadro pripravené pred vytvorením prvého okna alebo vykresľovacieho kontextu. Ďalšími úlohami jadra sú nastavenie ladiaceho výpisu a nastavenie overovacích vrstiev v prípade, že aplikácia je prekladaná v ladiacom režime.
- Swapchain a pipeline reprezentujúce samotné Vulkan objekty swapchain a pipeline a súvislé objekty ako napríklad obrázky. Podporujú znovu vytvorenie pre prípad, že sa zmení veľkosť okna.
- Renderovací kontext, ktorý zapuzdruje všetky objekty špecifické pre dané okno, napríklad vyššie uvedené swapchain a pipeline, frame buffery a command buffery. Je zodpovedný za vytváranie command bufferov na základe nahratých dát a vytvárania vykresľovacích požiadavkov.

Aktuálna implementácia pracuje len s dvoma fixnými shadermi a to vertex a fragment, ktoré sú prispôsobené pre potreby tejto práce. Ich kompilácia a inštalácia je automatizovaná pomocou `CMake`.

4.5 Finálny framework

Poslednou významnou časťou implementácie je `framework`, ktorý spája a zapuzdruje všetky ostatné diely. Ponúka nasledujúce významné abstrakcie, systémy a rozhrania:

- Rozhranie aplikácie `application_base`, pomocou ktorého aplikácia, ktorá využíva tento framework definuje body vykonávania aplikácie a to:
  - `setup` – príprava, vykoná sa raz na začiatku behu programu
  - `update` – slučka, cyklicky sa vykonáva počas behu aplikácie
  - `cleanup` – vyčistenie, vykoná sa raz na konci teda po zatvorení posledného okna

```cpp
// 3rd party includes
#include <pulsar/pulsar.hpp> // include all headers

using namespace pulsar;

class my_app final : application_base {

    void setup() override {

        // setup code goes here
        // is called once at the beginning of the application
    }

    void update() override {

        // update code goes here
        // is called every frame
    }

    void cleanup() override {

        // cleanup code goes here
        // is called once at the end of the application
    }
};

REGISTER_APPLICATION(my_app)
```

Obr. 4.2: Demonštrácia vytvorenia aplikácie pomocou frameworku

Vytvorenie vstupného bodu pre aplikáciu, teda funkciu main je riešené pomocným makrom `REGISTER_APPLICATION`, ktoré prijíma jeden parameter a to triedu aplikácie.

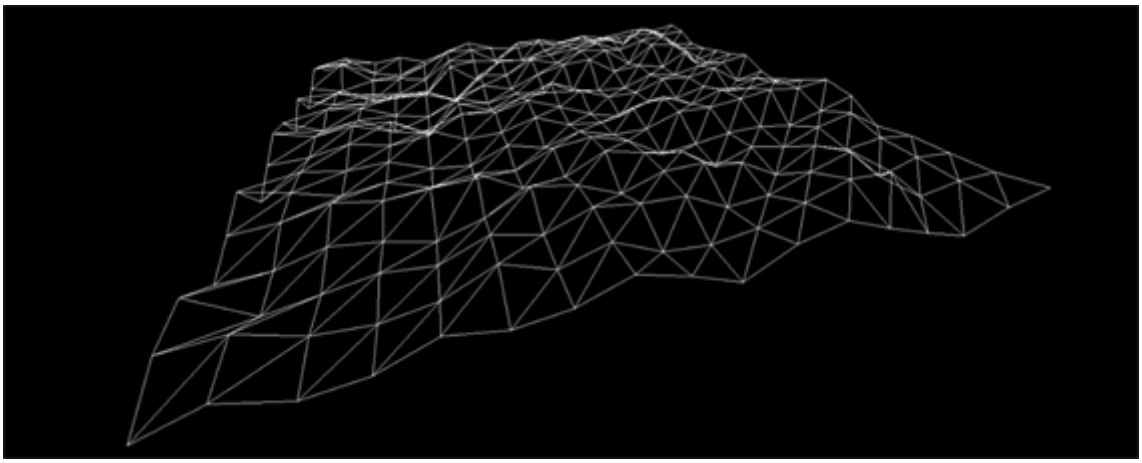
- Vstupný bod aplikácie, ktorý je volaný vyššie uvedeným makrom `REGISTER_APPLICATION`. Obsahuje spustenie a zastavenie frameworku, predanie aplikácie jadru a ošetrenie výnimiek.
- Jadro zodpovedné za spustenie a zastavenie všetkých ostatných systémov, hlavný cyklus, cyklické volanie aplikačnej metódy update.
- Entitný systém, ktorého podstatnou časťou je registrácia všetkých inštancií a metóda update, prostredníctvom ktorej je možné definovať činnosť jednotlivých entít. Táto metóda je jadrom automaticky volaná v rámci hlavného cyklu. Ďalšou možnosťou je registrácia obslužných funkcií na rôzne udalosti ako napríklad zmena stavu nejakej klávesy. Obslužná funkcia prijíma jeden parameter a to štruktúru, ktorá obsahuje ID volajúcej entity a prípadne ďalšie atribúty špecifické pre danú udalosť.
- Systém udalostí, ktorý ponúka entitám možnosť registrovať obslužné funkcie na jednotlivé udalosti. Jediné objekty tohto systému, ktoré pracujú s dynamicky alokovanou pamäťou sú tie zo štandardnej knižnice a tie považujem za maximálne optimálne. Vďaka tomu tento systém nepracuje na báze polymorfizmu, ale na základe indexácie do asociatívneho poľa na základe typu vytváranej udalosti, ktorú následne vloží

do príslušnej fronty. Udalosti sa potom naraz odošlú všetkým obslužným funkciám v rámci hlavného cyklu, čo zabezpečuje jadro.

- Systém užívateľského vstupu, ktorý podporuje dve formy získavania informácií a to:
  - `forma udalosti` – vhodná na definíciu logiky, ktorá vyžaduje informáciu o zmene stavu nejakej klávesy, napríklad stlačenie klávesy escape na zatvorenie okna
  - `forma dotazu` – vhodná na definíciu logiky, ktorá vyžaduje informáciu o držaní istej klávesy v čase ako napríklad pohyb kamery
- Abstrakcia modelu ponúka možnosť pridávania nezávislých trojuholníkových sietí. Tieto siete podporujú nezávislé vykresľovanie a pridávanie a mazanie z vykresľovacieho kontextu nejakého okna. Pomocou tejto triedy je definovaný aj terén, ktorého funkcionalita je prispôsobená pre potreby tohto projektu.
- Abstrakcia kamery, vďaka ktorej je možné sa voľne pohybovať v scéne pomocou kláves WASD a myši.
- Rozhranie lineárnej algebry, ktoré ponúka knižnica `GLM` [2] a rozhranie šumu, ktoré ponúka knižnica `FastNoiseSIMD` [9].

4.6 Demonštračná aplikácia

Táto aplikácia pracuje hlavne s funkciou setup, v ktorej pripraví jedno okno a terén, ktorý následne vloží do vykresľovacieho kontextu tohto okna. Keďže som neimplementoval osvetlenie, terén som vykreslil v režime `wireframe`, aby boli zmeny výšky viditeľné.



Obr. 4.3: Ukážka výstupu aplikácie

Kapitola 5

Zhodnotenie

Kým istými časťami implementácia značne presahuje návrh a potreby tohto projektu, veľa z nich potrebuje ešte mnoho iterácií a to z viacerých hľadísk, ako napríklad čistota kódu, komentáre alebo vytvorenie testov. Z časových dôvodov som bol nútený niektoré časti priamo prispôsobiť pre potreby tohto projektu. Pri ďalších iteráciách frameworku sa oplatí zamerať na nasledovné časti:

- renderer:
  - dynamický stav pre pipeline, aby ju nebolo nutné znova vytvárať pri zmene veľkosti okna
  - viacero pipeline pre rôzne vykresľovacie režimy
  - rozhranie pre shadery a textúry
  - optimálnejšia alokácia pamäte pre buffery
  - rôzne nastavenia ako clear color, ladiaci výpis, a ďalšie
  - transformačné matice pre jednotlivé modely
- okno:
  - definícia copy a move operácií
  - práca s kurzorom
  - ďalšie možnosti vstupu
  - testy
- framework:
  - robustnejší systém udalostí
  - zaistenie správneho mazania globálnych objektov
  - testy
  - vláknovanie
  - pridanie ďalších systémov ako napríklad zvuk

Keďže vykresľovaný terén je pomerne jednoduchý, nie je prekvapivé, že aplikácia sa obnový aspoň 1000 krát za sekundu na grafickom čipe Nvidia GTX 1050. Vďaka vykresľovaciemu režimu wireframe a voľnej kamere sú zmeny vo výške terénu dostatočne viditeľné.

Kapitola 6

Záver

Cieľom práce bolo vytvoriť framework, ktorý by mal schopný vykresliť jednoduchý terén. Kým terén nie je taký pôsobivý ako som dúfal na začiatku, vzhľadom na komplexitu Vulkan API som spokojný aj s jednoduhším výstupom a prácu považujem za úspešnú. Mala pre mňa veľký prínos z viacero hľadísk a nadobudnuté znalosti isto zúžitkujem v budúcnosti.

Literatúra

[1] Biagioli, A.: *Understanding Perlin Noise*. [Online; navštíveno 20.04.2019]. URL http://flafla2.github.io/2014/08/09/perlinnoise.html

[2] Creation, G.-T.: *OpenGL Mathematics*. [Online; navštíveno 8.2.2019]. URL https://glm.g-truc.net/0.9.9/index.html

[3] Games, C. I.: *Citizen Spotlight Fan Art*. [Online; navštíveno 12.04.2019]. URL https://www.robertsspaceindustries.com/community/citizen-spotlight/10064-ArcCorp-Outpost-Daymar

[4] Games, C. I.: *Daymar zo Star Citizen*. [Online; navštíveno 12.04.2019]. URL https://starcitizen.tools/Daymar

[5] Gutelle, S.: *A Video Featuring Nothing But White Noise Has Received Five Content ID Claims Since 2015*. [Online; navštíveno 12.04.2019]. URL https://www.tubefilter.com/2018/01/05/white-noise-youtube-content-id/

[6] Hewitt, C.: *Procedural Generation of Tree Models in Blender*. [Online; navštíveno 12.04.2019]. URL https://80.lv/articles/procedural-generation-of-tree-models-in-blender/

[7] Lantinga, S.: *Simple DirectMedia Layer*. [Online; navštíveno 20.09.2018]. URL https://www.libsdl.org/

[8] LunarG: *Vulkan SDK*. [Online; navštíveno 18.10.2018]. URL https://www.lunarg.com/vulkan-sdk/

[9] Peck, J.: *FastNoiseSIMD*. [Online; navštíveno 5.5.2019]. URL https://github.com/Auburns/FastNoiseSIMD

[10] Sauder, J.: *Procedural city generation in Python*. [Online; navštíveno 12.04.2019]. URL https://josauder.github.io/procedural_city_generation/

[11] Schier, G.: *Pros and Cons of Procedural Level Generation*. [Online; navštíveno 14.04.2019]. URL https://schier.co/blog/2015/10/23/pros-and-cons-of-procedural-level-generation.html

[12] Shaker, N.; Togelius, J.; Nelson, M. J.: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[13] Unity: *Infinite mountain height map*. [Online; navštíveno 12.04.2019].
URL https://answers.unity.com/questions/1375363/infinite-mountain-height-map.html

Príloha A

DVD

Obsah priloženého DVD:

- zdrojové súbory projektu
- spustiteľný program na platforme Microsoft Windows x64
- zdroj písomnej správy
- video