
**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**


# GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

**GRAPHICS INTRO 64KB USING OPENGL**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **ANDREJ BAÁŠ**
**AUTHOR**

**VEDOUCÍ PRÁCE** **prof. Ing. ADAM HEROUT, Ph.D.**
**SUPERVISOR**



**BRNO 2017**

*Zadání bakalářské práce/19606/2016/xbaasa00*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

# Zadání bakalářské práce

Řešitel: **Baáš Andrej**
Obor: Informační technologie
Téma: **Grafické intro 64kB s použitím OpenGL**
**Graphics Intro 64kB Using OpenGL**
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s fenoménem grafického intra s omezenou velikostí.
2. Prostudujte knihovnu OpenGL a její nadstavby.
3. Popište vybrané techniky použitelné v grafickém intru s omezenou velikostí.
4. Implementujte grafické intro s použitím OpenGL, aby velikost spustitelné verze nepřesáhla 64kB.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, experimenty směřující k vyřešení bodu 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
*vedoucí ústavu*

# Abstrakt

Bakalárska práca sa zaoberá tvorbou grafického intra s obmedzenou veľkosťou. Práca popisuje metódy a techniky pre generovanie grafického obsahu. Zaoberá sa procedurálnym generovaním textúr a objektov, tvorbou časticových systémov, osvetlením a animáciou vybraných objektov. Tiež popisuje metódy pre obmedzenie veľkosti výslednej aplikácie. Výsledkom je grafické intro zobrazujúce podmorské prostredie s korálovým útesom a plávajúcimi rybami, ktorého veľkosť je menšia než 64kB.



# Abstract

This thesis deals with the creation of a graphic intro with limited size. This work describes methods and techniques for generating the graphic content. It deals with procedural generation of textures and objects, creation of particle systems, lighting and animation of selected objects. It describes methods for reducing the size of the final application. The result is a graphical intro showing an underwater environment with a coral reef and floating fish whose size is less than 64kB.



# Kľúčové slová

OpenGL, GLSL, 64kB, intro, obmedzená veľkosť, procedurálne generovanie, Perlinov šum, Voroného diagram, výšková mapa, L-systém, skybox, časticový systém, billboarding, Phongov osvetľovací model, hudba, exe packer



# Keywords

OpenGL, GLSL, 64kB, intro, limited size, procedural generation, Perlin Noise, Voronoi diagram, heightmap, L-system, skybox, particle system, billboarding, Phong lighting model, music, exe packer



# Citácia

BAÁŠ, Andrej. *Grafické intro 64kB s použitím OpenGL*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Herout Adam.

# Grafické intro 64kB s použitím OpenGL

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostane pod vedením pána prof. Ing. Adama Herouta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Andrej Baáš
17. mája 2017


## Poďakovanie

Rád by som poďakoval prof. Ing. Adamovi Heroutovi, Ph.D. za jeho odborné vedenie a cenné rady v priebehu práce a svojej rodine za gramatickú korekciu tejto práce a všestranú podporu.

# Obsah

# Kapitola 1

# Úvod

Cieľom tejto bakalárskej práce je vytvorenie grafického intra s obmedzenou veľkosťou. Toto zadanie poskytuje riešiteľovi voľnosť pri výbere použitých metód tvorby grafického obsahu a v rozsahu jeho tvorivej činnosti. Súčasne ho však obmedzuje veľkostným limitom. Rozhodol som sa v grafickom intre demonštrovať svoju kreativitu a vybrané postupy prostredníctvom podmorského prostredia s korálovým útesom. Na tomto útese sa nachádzajú jemne vlniace sa podmorské riasy a rôzne druhy korálov. Nad nimi plávajú nie len pestrofarebné ryby, ale aj žralok. Celé intro je doprevádzané hudbou.

Pojem grafické intro je popísaný v druhej kapitole. Okrem neho je tu uvedená história a súčasnosť demoscény. V práci je pre vykreslenie všetkej grafiky použitá knižnica OpenGL, ktorá je stručne popísaná v tretej kapitole. Keďže sa intro musí zmestiť do 64kB, je potrebné pri jeho tvorbe použiť vhodné postupy. Všetky dôležité techniky a metódy, ktoré sú použité pre generovanie grafického obsahu sú popísané v štvrtej kapitole. Jedná sa o Perlinov šum, Voroného diagram, skybox, výškovú mapu, časticové systémy, billboarding, L-systémy a Phongov osvetľovací model. Popis implementácie týchto techník a metód je uvedený v piatej kapitole. V tejto kapitole je tiež popísaná problematika pridávania hudby do grafického intra. Šiesta kapitola sa venuje nie len nastaveniu prekladača pre optimalizáciu veľkosti výsledného programu, ale aj exe packeru, ktorého použitie je dôležité pre nepresiahnutie veľkostného limitu. Výsledok finálneho programu a zhodnotenie rýchlosti zobrazovania je prezentované v siedmej kapitole. Záver obsahuje zosumarizovanie postupov pre vytvorenie grafického intra a návrhy na jeho další možný rozvoj.

# Kapitola 2

# Grafické intro a demoscéna

Grafické intro predstavuje určitú formu umenia, ktoré má demonštrovať schopnosti autora prípadne autorov, využiť možnosti počítačového výkonu. Tento pojem je v podstate synonymom slova demo, ktoré vychádza zo skratky slova demonštrácia. V minulosti sa tieto termíny rozlišovali. O intro sa jednalo v prípade veľkostného obmedzenia, inak šlo o demo. V súčastnosti sa tieto pojmy často zamieňajú. Viac o grafickom intre je možné nájsť v podkapitole 2.2.

Nielen tvorcovia intier, ale aj nadšenci tohto umenia tvoria celosvetovú komunitu takzvanú demoscénu. Členovia tejto komunity sa stretávajú na demopárty, kde sa premietajú nové, ale aj staré intrá. Väčšinou je táto udalosť spojená s hlasovaním o najlepšie intro v danej kategórii. Viac informácií nielen o vzniku demoscény obsahuje nasledujúca podkapitola.

## 2.1 Demoscéna

Počiatky demoscény popísané Reunanenom [17] siahajú do 80. rokov 20. storočia. Jej vzniku dopomohla zvyšujúca sa dostupnosť počítačovej technológie bežným užívateľom. Keďže nie všetci boli ochotní zaplatiť za drahé komerčné programy, začali vznikať pirátske skupiny, ktoré sa pokúšali prelomiť ochranu proti kopírovaniu daného softwaru (väčšinou hier).

K takýmto upravením hrám začali pridávať krátke grafické, neskôr hudobno-grafické intrá, ktoré slúžili ako podpis danej skupiny. Tieto intrá sa stali veľmi rýchlo populárne a tlak na ich kvalitu stúpal závratnou rýchlosťou. Preto ľudia, ktorí stáli za ich výrobou už jednoducho nemali čas na prelamovanie ochrán a založili si demoskupiny, ktoré sa špecializujú na výrobu grafických intier. Postupom času sa demoscéna úplne oddelila od pirátskej scény.

V demoskupine okrem programátorov pôsobia aj grafici a hudobníci, zriedka aj scenáristi. Jednotlivé skupiny sa stretávajú na demopárty, kde môžu prezentovať svoju tvorbu a súťažiť s ňou v daných kategóriách. Spočiatku sa jednalo o menšiu udalosť, kde sa stretlo pár desiatok nadšencov. V súčasnosti sú demopárty veľmi dobre organizované medzinárodné akcie, ktorých súčasťou bývajú aj prednášky o tvorbe grafických intier. Medzi najprestížnejšie patrí fínska demopárty s názvom Assembly a nemecký Breakpoint. Okrem demopárty sa veľká časť demoscény a jej tvorby sústreduje na rôznych webových portáloch. Najznámejšie sú *www.pouet.net* a *www.scene.org*.

## 2.2 Grafické intro

Grafické intro je možné definovať ako neinteraktívnu multimediálnu prezentáciu vo forme počítačového programu, ktorej hlavnou snahou je poskytnúť divákovi čo najpôsobivejší audiovizuálny zážitok za pomoci maximálneho využitia hardwaru, na ktorom je prezentovaný. Nejedná sa teda o dielo, ktoré by prinášalo nejaký priamy úžitok, ale ide skôr o ukážku autorových schopností a tiež umeleckého citu.

Ako už bolo spomenuté v predchádzajúcej časti, na tvorbe grafického intra sa podieľa skupina ľudí, ale aj jednotlivci. Intrá medzi sebou súťažia v rôznych veľkostných kategóriách. Najtypickejšie sú 128kB, 64kB a 4kB, ale existujú aj veľkostne menšie. Tieto kategórie určujú veľkostný limit, do ktorého sa musí dané grafické intro zmestiť.

Dôvodom prečo stále existujú tieto obmedzenia aj napriek rýchlemu technologickému pokroku je súťaživosť členov demoscény. Tí sa snažia prezentovať svoju šikovnosť, kreativitu a programátorské schopnosti prostredníctvom vytvoreného grafického intra a ukázať tak, čo všetko je možné po grafickej a hudobnej stránke vtesna do určitého veľkostného limitu.

Kvôli týmto obmedzeniam je potrebné vytvárať grafické intrá určitými postupmi. Nie je možné používať obrázky vo vysokom rozlíšení, a preto sa grafický obsah musí procedurálne generovať. Táto technika je popísaná v podkapitole 4.1. Na prehrávanie hudby sa používajú takzvané syntetizátory. Viac o problematike pridávania zvukovej zložky do intra nájdete v podkapitole 5.11. Nakoniec sa intro skompiluje pomocou takzvaného exe packeru. Viac sa o ňom dočítate v podkapitole 6.2. Samozrejme existujú aj kategórie bez veľkostných limitov, ale tie už nie sú po technologickej stránke až tak zaujímavé.

## 2.3 Téma intra

Tvorcovia grafických intier sa nesnažia diváka zaujať len čo najkomplexnejšou a najprepracovanejšou grafickou stránkou intra, ale aj pôsobivou a originálnou témou. Obsah intier býva rôzny, od prezentácie nejakej autorovej myšlienky, až po krátky príbeh. Existujú aj intrá, ktorých dej je inšpirovaný určitým dobre známym dielom či už filmom alebo knihou.

Grafické intrá s veľmi malou veľkosťou (4kB a menej) majú väčšinou abstraktnú tematiku. Zobrazujú nerealistické scenérie, ktoré sú síce bohaté na rôzne grafické efekty, ale postupom času a rýchlosti technologického pokroku zostarnú a neposkytnú už taký plnohodnotný grafický zážitok. Preto je veľmi dôležité do intra zakomponovať originálny obsah s nezabudnuteľným príbehom alebo myšlienkou.

V mojom intre som sa rozhodol zobraziť podmorskú prírodnú scenériu. Jedná sa konkrétne o korálový útes s piesočnou lagúnou. Na tomto útese sa nachádzajú morské riasy a rôzne druhy korálov. Nad útesom plávajú rôznofarebné ryby a žralok. Celé intro je doprevádzané hudbou.

# Kapitola 3

# Knižnica OpenGL a jej nadstavby

Zadanie tejto bakalárskej práce určuje použiť pre vykreslenie všetkej grafiky knižnicu OpenGL. Táto knižnica je stručne popísaná v nasledujúcej podkapitole. Podkapitola 3.2 sa venuje programovaciemu jazyku GLSL, ktorý sa používa na vytváranie programov pre programovateľný grafický reťazec grafickej karty. Tieto programy sa nazývajú shadery a sú popísané v podkapitole 3.3.

## 3.1 Open Graphics Library

Open Graphics Library (ďalej len OpenGL) popísané v dokumentácii [19] je aplikačné programové rozhranie k akceleračním grafickým kartám. Toto rozhranie pozostáva z niekoľkých stoviek funkcií a procedúr, ktoré programátorovi umožňujú špecifikáciu a manipuláciu s grafickými dátami za účelom tvorby vysokokvalitného grafického obsahu. V roku 1992 bola knižnica OpenGL uvedená spoločnosťou Sillicon Graphics, Inc. V súčasnosti je OpenGL vo viacerých verziách, ktoré sú spätne kompatibilné. Od verzie 2.0 je k dispozícii jazyk GLSL, ktorý slúži k programovaniu kresliaceho grafického reťazca.

OpenGL funguje ako stavový automat, čo znamená, že príkazy pre vykresľovanie môžu priebežne meniť vlastnosti vykresľovaných primitív (napr. farba, priehľadnosť). Vykreslenie scény sa vykonáva procedurálne, volaním funkcií OpenGL. Dáta reprezentujúce informácie o jednotlivých pixeloch (napr. farba, hĺbka, priehľadnosť) sú uložené vo framebuffery, z ktorého sú vyberané pre zobrazenie.

Táto knižnica je nezávislá na použitej platforme, preto neobsahuje funkcie pre vytvorenie okna, zachytávanie udalostí a iné systémovo závislé funkcie. V tejto práci je použitá knižnica GLFW vo verzii 3.2.1, ktorá zaisťuje prácu s oknami. Pre vykresľovanie všetkej grafiky sa používa OpenGL vo verzií 4.5. Spočiatku bola použitá knižnica GLEW 2.0.0 pre prácu s rozšíreniami, ktoré poskytuje OpenGL od verzie 3.0. Keďže použitie tejto knižnice zvyšuje výslednú veľkosť programu, bolo nutné všetky nové funkcie vytiahnuť z hlavičkového súboru glext.h. To je realizované pomocou funkcie `glfwGetProcAddress()`.

## 3.2 OpenGL Shading Language

OpenGL Shading Language (ďalej len GLSL) popísaný v dokumentácii [8] je programovací jazyk, ktorý je súčasťou knižnice OpenGL od verzie 2.0. Používa sa na vytváranie programov (shaderov) pre programovateľný grafický reťazec grafickej karty. Tento programovací jazyk je založený na syntaxi jazyka C.

Jazyk GLSL je navrhnutý tak, aby rýchlo zvládal operácie s vektormi a maticami. Podporuje teda dátový typ pre vektor (`vec1` – `vec4`) a dátový typ pre maticu (`mat2` – `mat4`). Pre rýchly prístup k prvkom vektora slúži operátor `swizzle`. Ďalej tento programovací jazyk obsahuje dátový typ sampler, ktorý umožňuje prístup k textúram. Jazyk GLSL poskytuje veľké množstvo matematických funkcií a funkcií pre prácu s farbami a textúrami, nepodporuje však ukazatele.

## 3.3 Shadery

Shader je počítačový program určený pre riadenie jednotlivých častí programovateľného grafického reťazca grafickej karty. Tieto programy sú nahrávané a paralelne spracovávané na grafickej karte ako kolekcia zreťazených programov. Existuje niekoľko typov shaderov. Rozdeľujú sa na základe toho, ktorú časť grafického reťazca ovplyvňujú. V tejto práci sú použité tieto typy:

- **Vertex Shader** — je určený na spracovávanie vrcholov, ktoré sú čítané z globálnej pamäti na grafickej karte. Vrcholy tu môžu byť transformované pomocou modelovej, pohľadovej, alebo projekčnej matice.
- **Geometry shader** — slúži hlavne k pridávaniu a odoberaniu vrcholov, čím sa ovplyvňuje tvar výsledného objektu. V tejto práci sa používa na vytváranie textúr pre jednotlivé druhy korálov, ktoré je uvedené v podpodkapitole 5.2.2. Ďalej sa využíva pre generovanie vegetácie popísanej v podkapitole 5.3.
- **Fragment shader** — postupne spracováva jednotlivé fragmenty objektov a scény. Výstupom sú hodnoty hĺbky a farby daného fragmentu, ktoré sú zapísané do framebufferu. V tejto práci je fragment shader použitý najmä pre Phongov osvetľovací model, ktorým sa zaoberá podkapitola 4.2.

# Kapitola 4

# Použité techniky a metódy pre generovanie grafického obsahu

V tejto kapitole sú popísané teoretické časti bakalárskej práce pre zoznámenie sa s problematikou tvorby grafického intra s obmedzenou veľkosťou. Sú tu vysvetlené metódy procedurálneho generovania grafického obsahu, ktoré sú použité v tejto práci. Ďalej sú tu popísané techniky počítačovej grafiky, ktoré boli využité pri tvorbe grafického intra. Sú to skybox, výšková mapa, časticové systémy, billboarding, L-systémy a Phongov osvetľovací model.

## 4.1 Procedurálne generovanie

Kvôli veľkostnému limitu intra nie je možné mať textúry alebo modely uložené ako statické dáta. Preto sa musí celý grafický obsah generovať procedurálne, teda pomocou algoritmov. Výsledok takéhoto generovania je možné veľmi ľahko upraviť zmenou parametrov. Veľmi dôležité je, aby výsledok procedurálneho generovania bol vždy rovnaký. Preto sa používajú pseudonáhodné čísla. Tieto čísla sú vytvárané pomocou postupnosti, ktorá sa javí ako náhodná, ale v skutočnosti je generovaná deterministickým algoritmom. Jeden z najpoužívanejších algoritmov na generovanie pseudonáhodných číslic je lineárny kongruenčný generátor popísaný Hörmannom a Derflingerom [7]. Generuje hodnoty na základe vzťahu:

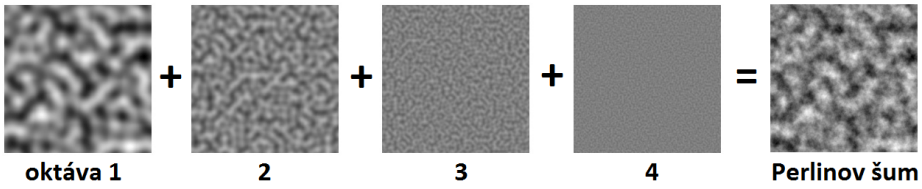
$$X_{n+1} = (aX_n + c) \bmod \ m \tag{4.1}$$

kde $a$, $c$ a $m$ sú vhodne zvolené kladné čísla, $X_n$ predstavuje vstupnú hodnotu a $X_{n+1}$ výstupnú. Operácia mod je zvyšok po celočíselnom delení a to znamená, že konštanta $m$ určuje horné obmedzenie rozsahu generovaných hodnôt.

### 4.1.1 Perlinov šum

Perlinov šum popísaný Perlinom [13] [14] je metóda, ktorá sa používa na generovanie grafického šumu. Predstavil ju v roku 1985 profesor informatiky Ken Perlin. Jedná sa v podstate o deterministický generátor, ktorý generuje pre rovnaké súradnice rovnakú hodnotu v rozsahu $\langle -1, 1 \rangle$.

Princíp tejto metódy spočíva vo vytvorení adekvátneho počtu vrstiev (oktáv), ktoré sa nakoniec sčítajú do jednej výslednej vrstvy, ktorú môžeme ďalej použiť, ako napríklad textúru pre rôzne druhy materiálov. Toto je možne vidieť na obrázku 4.1. Počet vygenerovaných vrstiev záleží od jedného vstupného parametra. Každá vrstva znázorňuje určitú




Obr. 4.1: Princíp Perlinovho šumu spočíva vo vytvorení adekvátneho počtu vrstiev (oktáv), podľa zadaných parametrov, ktoré sa sčítajú do jednej výslednej (prevzaté z [22]).

úroveň detailu výslednej vrstvy. Úroveň detailu danej vrstvy záleží od použitých parametrov, konkrétne frekvencie, amplitúdy a perzistencie. Frekvencia každej oktávy je vyjadrená ako

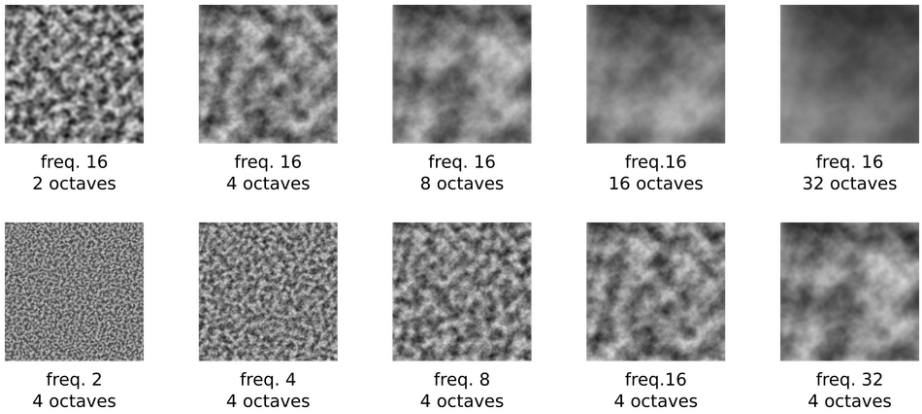
$$f = 2^i \tag{4.2}$$

a amplitúda je daná vzťahom:

$$a = p^i \tag{4.3}$$

kde $p$ je perzistencia s konštantnou hodnotou.

Perlinov šum počíta hodnotu každého pixelu zvlášť, je preto výpočtovo náročný. Jeho časová zložitosť je $O(2^n)$. Početnosť – koľko krát sa má algoritmus vykonať je daný počtom oktáv. Teda čím väčší je počet oktáv, tím viac narastá zložitosť. Preto je vhodné určiť optimálny počet oktáv. V hornej polovici obrázku 4.2 si je možné všimnúť, že od vyššieho počtu oktáv sa výsledok Perlinovho šumu výrazne nemení. Spodná polovica tohoto obrázku zobrazuje vplyv zvyšujúcej sa frekvencie na výsledok pri rovnakom počte oktáv. Je teda možné dosiahnuť požadovaný výsledok rýchlejšie, adekvátnym nastavením frekvencie pri nižšom počte oktáv.



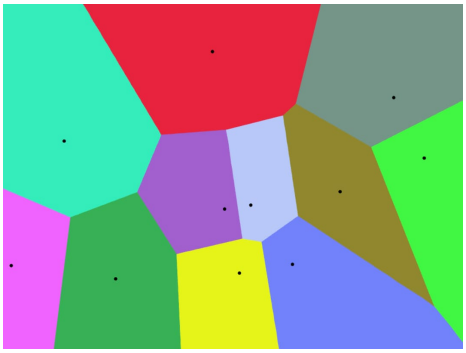

Obr. 4.2: Vplyv frekvencie a počtu oktáv na Perlinov šum. Zložitosť Perlinovho šumu narastá s počtom oktáv. Od vyššieho počtu oktáv sa výsledok Perlinovho šumu výrazne nemení, čo si je možné všimnúť v hornej polovici obrázku. Preto je vhodné určiť optimálny počet oktáv a adekvátnu frekvenciu pre získanie požadovaného výsledku (prevzaté z [1]).

Perlinov šum sa v počítačovej grafike najčastejšie využíva na tvorbu textúr. Kombináciou tejto metódy s rôznymi matematickými funkciami je možné vytvoriť veľké množstvo procedurálnych textúr, s ktorými je možné napodobniť rôzne prírodné materiály ako napríklad textúra dreva alebo mramoru. Okrem toho sa používa pre tvorbu rôznych efektov ako

sú plamene, dym alebo oblaky, ale tiež aj na generovanie terénu. Väčšinou býva implementovaný ako 2D alebo 3D funkcia, ale môže byť definovaný pre ľubovoľný počet dimenzií. V tejto práci sa využíva 2D implementácia Perlinoveho šumu pri vytváraní profilu terénu popísaná v podkapitole 5.1. Okrem toho sa používa pri tvorbe textúr určených pre terén, ktorá je popísaní v podpodkapitole 5.2.1.

### 4.1.2 Voroného diagram

Voroného diagram popísaný Aurenhammerom [3], je spôsob rozdelenia roviny s niekoľkými bodmi do konvexných polygónov. Každý polygón obsahuje iba jeden bod. Príklad voroného diagramu je možné vidieť na obrázku 4.3, kde pre rozdelenie bola použitá množina náhodných bodov. Tieto body sú takzvané generátory, ktorých poloha a počet určuje rozdelenie danej roviny. Počet bodov a ich pozíciu môžeme stanoviť niekoľkými spôsobmi. Počet a polohu každého bodu môžeme zadať explicitne, ale väčšinou sa tieto body generujú náhodne alebo pravidelne, napríklad pomocou L-systému popísaného v podkapitole 4.5.



Obr. 4.3: Príklad Voroného diagramu. Obrázok je rozdelený na niekoľko farebných oblastí na základe počtu a pozície jednotlivých čiernych bodov. V tomto prípade boli body vygenerované náhodne. Jednotlivé farebné oblasti patria k danému čiernemu bodu, ktorý sa v nich nachádza. Spôsob rozdelenia Voroného diagramom spočíva v tom, že všetky body v danej farebnej oblasti majú menšiu vzdialenosť k tomuto čiernemu bodu než k ostatným (prevzaté z [9]).

Ohraničené polygóny patria k danému generátoru, ktorý sa v nich nachádza. Všetky body v danej farebnej oblasti majú menšiu vzdialenosť k tomuto generátoru než k ostatným. Jednotlivé strany polygónov sú tvorené bodmi, ktoré sa nachádzajú v rovnakej vzdialenosti od dvoch najbližších generátorov. Vrcholy polygónov tvoria body rovnako vzdialené od troch najbližších generátorov. Výpočet vzdialenosti jednotlivých bodov od generátorov, a teda výsledné rozdelenie danej roviny na oblasti závisí od použitej metriky. V tejto práci sa používa euklidovská metrika pre dvojrozmerný priestor:

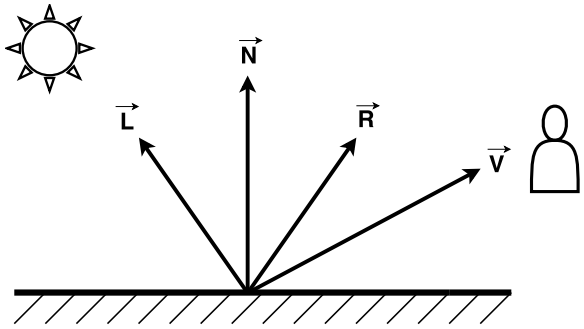
$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}. \tag{4.4}$$

Voroného diagram sa v počítačovej grafike využíva najčastejšie na tvorbu procedurálnych textúr alebo pri generovaní geometrie. Textúry vytvorené pomocou Voroného diagramu sa používajú hlavne pre organické alebo mozaikové materiály, kamene, vysušenú zem alebo lávu. Voroného diagram sa v tejto práci používa na tvorbu textúr pre optický jav – kaustiku popísanú v podkapitole 5.5 a na textúru určenú pre terén popísanú v podpodkapitole 5.2.1.

## 4.2 Phongov osvetľovací model

V počítačovej grafike pojem osvetľovací model popísaný Dohnalom [5], predstavuje model, ktorý popisuje intenzitu svetla v bodoch scény. Hlavné faktory vplývajúce na výsledné osvetlenie danej scény sú svetelné zdroje, vlastností objektov, ktoré sa nachádzajú v scéne a samozrejme tiež vlastnosti pozorovateľa.

V počítačovej grafike existujú rôzne osvetľovacie modely. V tejto práci sa používa Phongov osvetľovací model. Tento model navrhol Bui Tuong Phong v roku 1973 vo svojej dizertačnej práci [15]. Jedná sa o empirický model (nie je založený na fyzikálnych zákonoch), napriek tomu po jeho použití osvetlenie výslednej scény pôsobí veľmi prirodzene. Keďže je výpočtovo nenáročný, má veľké uplatnenie v real-time grafike.




Obr. 4.4: Vektorové zložky vo Phongovom osvetľovacom modeli. Odraz svetla v určitom bode na povrchu materiálu je vo Phongovom osvetľovacom modeli určený štyrmi vektormi. Vektor $\vec{L}$ určujúci smer dopadajúceho svetla, normálový vektor $\vec{N}$ v bode objektu, vektor $\vec{R}$ znázorňujúci odraz vektora $\vec{L}$ a vektor k pozorovateľovi – $\vec{V}$ (prevzaté z [23]).

Na obrázku 4.4 je možné vidieť, že odraz svetla v určitom bode na povrchu materiálu je určený štyrmi vektormi:

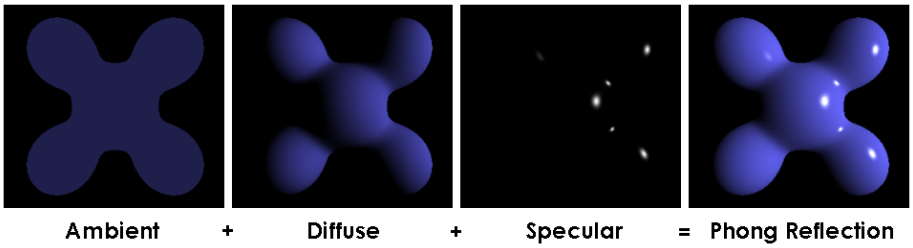
- $\vec{L}$ – vektor určujúci smer dopadajúceho svetla.
- $\vec{N}$ – normálový vektor v bode objektu.
- $\vec{R}$ – vektor znázorňujúci odraz vektora $\vec{L}$.
- $\vec{V}$ – vektor k pozorovateľovi.

Tento odraz svetla sa v Phongovom osvetľovacom modeli skladá z troch svetelných zložiek, ako je možné vidieť na obrázku 4.5. Jedná sa o difúznu, odrazovú a ambientnú zložku. Výsledný odraz je súčtom týchto zložiek:

$$C = C_d + C_s + C_a. \tag{4.5}$$

Difúzna zložka predstavuje časť svetla, ktoré sa po dopade na povrch rovnomerne rozptýli do všetkých smerov. Je určená vzťahom (4.6), kde $I_d$ označuje intenzitu difúzneho osvetlenia scény, $k_d \in (0, 1)$ je odrazový koeficient materiálu objektu. Intenzita je závislá na vzájomnej polohe normály $\vec{N}$ a vektoru dopadu svetla $\vec{L}$.

$$C_d = I_d k_d (\vec{N} \cdot \vec{L}) \tag{4.6}$$




Obr. 4.5: Výsledný odraz svetla je v Phongovom osvetľovacom modeli určený súčtom troch svetelných zložiek. Jedná sa o difúznu, odrazovú a ambientnú zložku (prevzaté z [2]).

Odrazová zložka udáva intenzitu tej časti svetla, ktoré sa od telesa odráža prevažne v jednom smere podľa zákona odrazu. Vypočítame ju podľa vzťahu:

$$C_s = I_s k_s (\vec{V} \cdot \vec{R})^n, \tag{4.7}$$

V tomto vzťahu $I_s$ predstavuje intenzitu odlesku, $\vec{V}$ je vektor pohľadu. Vektor $\vec{R}$ vyjadruje smer ideálneho zrkadlového odrazu a je symetrický k vektoru $\vec{L}$ podľa normály. Je určený vzťahom:

$$\vec{R} = 2(\vec{N} \cdot \vec{L})\vec{N} - \vec{L} \tag{4.8}$$

Odrazový koeficient $k_s \in (0, 1)$ určuje mieru zastúpenia odrazenej lesklej zložky v celkovom odrazenom svetle. Phongov exponent $n \in (0, \infty)$ určuje ostrosť zrkadlového odrazu. Čím väčšie je $n$, tím sú odlesky na povrchu telesa menšie, ale zato intenzívnejšie.

Keďže Phongov osvetľovací model je empirický, ako bolo spomenuté vyššie, neuvažujeme o svetle odrážanom od objektov. Výsledná scéna by teda bola o mnoho tmavšia než by mala byť. Preto sa pridáva ambientná zložka, ktorá tento nedostatok napravuje. Je vyjadrená vzťahom 4.9, kde $I_a$ označuje intenzitu okolitého osvetlenia scény a $k_a \in (0, 1)$ je odrazový koeficient materiálu objektu.

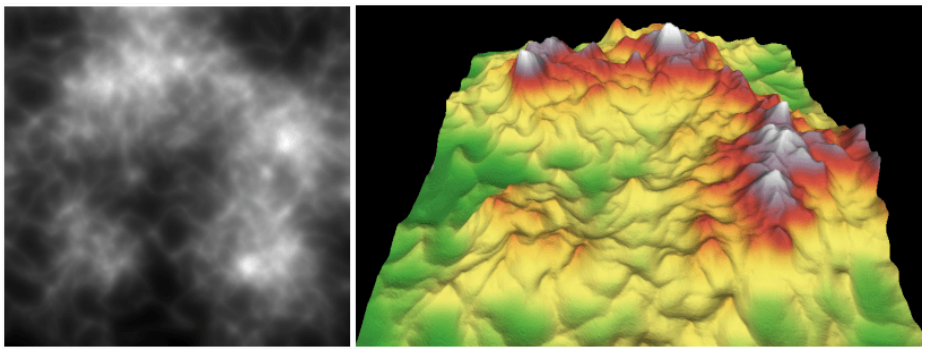
$$C_a = I_a k_a \tag{4.9}$$

## 4.3 Výšková mapa

Jednou z dôležitých častí v mnohých grafických intrách je terén, ktorý diváka obklopuje. Terén môže predstavovať hladký povrch, kopce, hory a iné geografické útvary. Jeho hlavným zmyslom je poskytnúť pozorovateľovi dojem, že sa nachádza v skutočnom svete. Z abstraktného hľadiska predstavuje terén len variáciu na výšku. Napríklad horský terén má veľké výškové rozdiely, ktoré vytvárajú údolia a rokliny medzi jednotlivými končiarmi a zabezpečujú tak ilúziu hôr. Naopak trávnatý rovinatý terén má v podstate konštantnú výšku.

V počítačovej grafike existuje mnoho spôsobov ako vytvoriť realistický terén. V tejto práci je použitá takzvaná výšková mapa, popísaná Barosom [4]. Táto metóda umožňuje vytvoriť prirodzený terén tým, že poskytuje elegantné riešenie pre ukladanie variacií výšok. Výškovú mapu si môžeme predstaviť ako dvojrozmerný obrázok v stupňoch šedi. Príklad je možné vidieť na obrázku 4.6 vľavo. Hodnota každého pixelu v tomto obrázku sa pohybuje od 0 do 255, kde 0 znamená čiernu a 255 bielu. Princíp výškovej mapy spočíva v tom, že

čierny pixel (hodnota 0) reprezentuje najnižšiu výšku a biely pixel (hodnota 255) najvyššiu. Takto dostaneme mapu zachytávajúcu výšky — výškovú mapu. Terén vytvorený pomocou výškovej mapy je možné vidieť na obrázku 4.6 vpravo.



Obr. 4.6: Príklad výškovej mapy. Vľavo je zobrazená výšková mapa, ktorá predstavuje dvojrozmerný obrázok v stupňoch šedi, kde čierny pixel (hodnota 0) reprezentuje najnižšiu výšku a biely pixel (hodnota 255) najvyššiu. Vpravo je vyobrazený terén vytvorený pomocou tejto výškovej mapy (prevzaté z [20]).

Keďže hodnota pixelov sa pohybuje od 0 do 255, dostanete automatické interpoláciu terénu (čím sa vytvára hladký terén) stačí obrázok rozmazať. Štandardným RGB 8-bitovým obrázkom môžeme zobraziť iba 256 hodnôt šedi, a teda iba 256 výšok. Pokiaľ potrebujeme väčší počet výšok môžeme použiť napríklad 24-bitový obrázok, ktorý môže uložiť až 16 777 216 výšok. Vzhľadom k tomu, že výšková mapa definuje pre každý bod iba jednu výšku, nemožno ňou teda vytvárať jaskyne a podobné prírodné útvary.

Výškovú mapu je možné vytvoriť ručne pomocou klasických programov pre maľovanie alebo editormi, ktoré sú zamerané na jej tvorbu. Ďalej ju môžeme získať použitím skutočných údajov napríklad zo satelitných snímok. Tiež ju môžeme automaticky vygenerovať použitím vhodného procedurálneho algoritmu. Táto možnosť vytvorenia výškovej mapy je použitá v tejto práci a je popísaná v podkapitole 5.1. Pre vygenerovanie výšok sa využíva Perlinov šum popísaný v podpodkapitole 4.1.1.

## 4.4 Skybox

Metóda vytvárania vzdialeného okolia scény sa v počítačovej grafike nazýva skybox. Toto okolie nie je reprezentované pomocou polygónov, ale len pomocou obrázkov. Veľmi často aj v tejto práci je skybox vo forme kocky. Na strany kocky sú nanesené jednotlivé časti výsledného obrázka. Je dôležité, aby tieto časti na seba plynule nadväzovali a vytvorili ilúziu okolia. Obrázok môže byť ľubovoľný, ale väčšinou sa jedná o kombináciu medzi oblohou a terénom alebo oceánom. Príklad takéhoto skyboxu je na obrázku 4.7. V tejto práci skybox predstavuje podmorské prostredie a jeho tvorba je popísaná v podkapitole 5.6.

Kamera je umiestnená do stredu skyboxu, ktorý nasleduje jej pohyb. To znamená, že pozícia kamery sa voči skyboxu nemení, takže pozorovateľ nikdy nedosiahne horizont scény. Vďaka tejto metóde nám pripadá scéna väčšia než je v skutočnosti, a preto sa najčastejšie využíva v počítačových hrách. Čím ďalej viac sa v súčasnosti používajú skydomy. Ich princíp je rovnaký ako u skyboxu, ale nemajú tvar kocky, ale pologule.


Obr. 4.7: Príklad skyboxu vo forme kocky. Jednotlivé obrázky na seba plynule nadväzujú a po nanesení na jednotlivé strany kocky vytvárajú ilúziu okolia (prevzaté z [10]).

## 4.5 L-systémy

Lindenmayerov systém (ďalej len L-systém) popísaný Prusinkiewiczom a ďalšími autormi [16], je druh formálnej gramatiky známy najmä vďaka svojmu použitiu pri modelovaní rastlín a to vrátane ich rastu. Tento koncept predstavil v roku 1968 maďarský biológ a botanik Aristid Lindenmayer.

L-systémy sa používajú na objekty, ktoré sa dajú popísať pomocou vetviacich sa modulov a množinou prepisovacích (produkčných) pravidiel. Modul v tomto prípade reprezentuje všetko to, čo sa na danom objekte opakuje, mení, rozrastá alebo rozvetvuje. Hlavná myšlienka tohto pohľadu spočíva v tom, že čím viac sa daný objekt zväčšuje tím viac sa jednotlivé moduly opakujú. Toto opakovanie je možné vyjadriť pomocou rekurzie. Napríklad u stromu by moduly boli konáre a listy, pravidlá by definovali ich rozmiestnenie v každej iterácii.

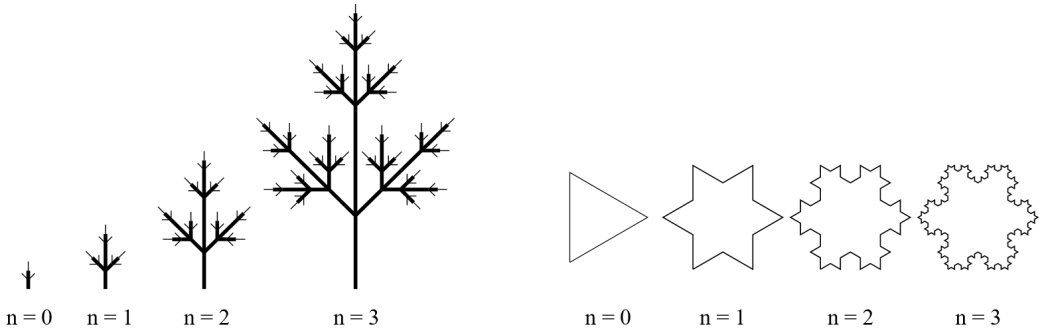
Z pohľadu informatiky L-systémy predstavujú formálnu gramatiku, ktorú môžeme definovať ako trojicu $G = (V, \omega, P)$, kde $V$ je konečná neprázdna množina znakov (abeceda), $P$ je množina prepisovacích pravidiel a $\omega \in V^*$ je počiatočný reťazec (axióm). Pre popisovaný objekt je potrebné definovať gramatiku, určiť prepisovacie pravidlá a zvoliť počiatočný reťazec.

Jednotlivé symboly L-systému môžeme interpretovať rôzne. Najčastejšie ich interpretujeme pomocou takzvanej korytnačej grafiky. Základom tejto grafiky je virtuálna korytnačka, ktorá nesie štetec a podľa jej pohybu sa vykresluje zvolený objekt. Symboly sú interpretované ako príkazy pre jej pohyb.

Pokiaľ chceme pomocou L-systému popísať vetviace sa objekty je potrebné, aby gramatika obsahovala symboly hranatých zátvoriek `[` a `]`, ktoré budú odlišovať vetvu daného objektu. Symbol `[` bude korytnačka interpretovať takto: Ulož aktuálny stav (aktuálnu polohu a orientáciu) na zásobník. Symbol `]` zase takto: Nastav sa podľa stavu na vrchole zásobníku a ten potom z vrcholu odstráň.

L-systémy s použitím korytnačej grafiky sa najčastejšie využívajú na tvorbu rôznej vegetácie, ako je možné vidieť v ľavej polovici obrázku 4.8. Môžu byť tiež použité aj na

generovanie niektorých typov fraktálov, ako napríklad Kochova vločka, ktorá je v pravej polovici obrázku 4.8. V tejto práci sa L-systémy používajú na tvorbu textúr pre korály popísané v podpodkapitole 5.2.2.




Obr. 4.8: Použitie L-systému na tvorbu vegetácie (vľavo) a fraktálu – Kochova vločka (vpravo) (prevzaté z [16]).

## 4.6 Časticové systémy

Časticové systémy popísané Žárem [23, s. 288 – 294] sa v počítačovej grafike používajú k reprezentácii objektov, ktoré sa menia takým spôsobom, pri ktorom nejde jednoznačne definovať ich povrch alebo sú príliš členité. Väčšinou sa jedná o súbor s veľkým množstvom veľmi malých častíc, ktoré majú určitú grafickú podobu a ich vlastnosti sa menia v čase.

Častice sa chovajú podľa daných pravidiel a to vo výsledku vytvára požadovaný efekt. Medzi také efekty najčastejšie patrí oheň, voda, dážď, alebo výbuch. Príklad využitia tohto systému je možné vidieť na obrázku 4.9. V tejto práci sa časticové systémy používajú k reprezentácii bublín popísaných v podkapitole 5.9.

Typické vlastnosti častíc sú aktuálna pozícia, zrýchlenie, smer pohybu, doba trvania, údaje o vzhľade (farba, tvar, priehľadnosť) atď. Časticový systém generuje častice na určitom mieste, ktoré sa priestorovo nemení. Po uplynutý doby trvania častica zaniká a vzniká nová, ktorej sa priradia požadované počiatočné vlastnosti.

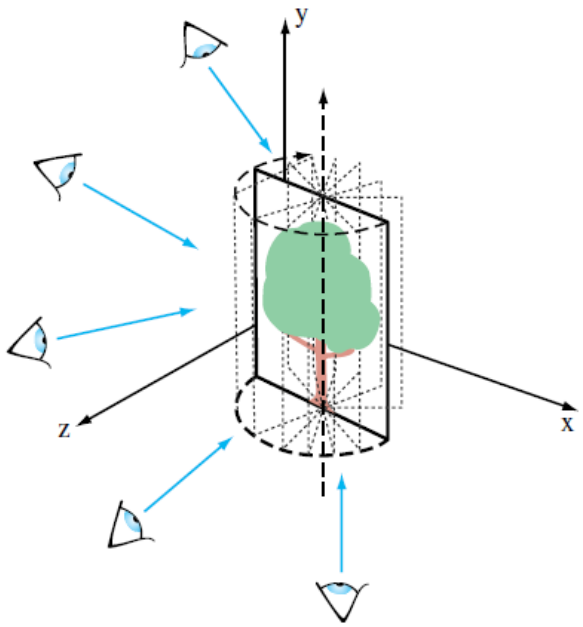
Obr. 4.9: Príklad použitia časticových systémov. Tieto systémy slúžia k reprezentácii objektov, ktoré sú príliš členité alebo sa menia takým spôsobom, pri ktorom nejde jednoznačne definovať ich povrch. Vo videohre Call of Duty: Modern Warfare 3 sú časticové systémy použité k reprezentácii dymu (prevzaté z [18]).

## 4.7 Billboarding

Billboarding je metóda, pri ktorej je orientácia objektu menená závislosti na smere pohľadu. Obvykle je situovaná tak, aby smerovala čelom ku kamere, čo je možné vidieť na obrázku 4.10.

Billboarding sa často používa tam, kde je potrebné zobraziť veľké množstvo polygónov. Princíp spočíva v tom, že zložitá geometria je nahradená jednou textúrou. Tá je nanesená typicky na dva trojuholníky, ktoré tvoria štvorec a tím ušetríme počet polygónov na scéne a taktiež vypočtový výkon. Dojem zložitého 3D objektu sa nestráca, keďže je štvorec s nanesenou textúrou objektu stále natočený čelom ku kamere a pozorovateľ si tak neuvedomí, že sa jedná len o ilúziu 3D objektu.

Táto technika nie je vhodná pre všetky objekty. Napríklad ak kamera preletí nad textúrou stromu, tak je dojem 3D objektu stratený. Naopak billboarding sa často používa v kombinácii s časticovými systémami pre modelovanie ohňa, dymu alebo iných zaujímavých efektov. V tejto práci sa billboarding využíva na bubliny popísané v podkapitole 5.9.




Obr. 4.10: Princíp Billboardingu. Orientácia textúry stromu je menená v závislosti na smere pohľadu pozorovateľa a tým sa vytvára dojem 3D objektu (prevzaté z [21]).

# Kapitola 5

# Návrh a implementácia

Táto kapitola sa zaoberá samostatnou implementáciou grafického intra a spôsobom využitia metód a techník popísaných v predchádzajúcej kapitole a popisuje problematiku pridávania hudby do grafického intra.

## 5.1 Generovanie terénu

Moje intro zobrazuje prírodnú scenériu, v ktorej je terén jedným z dôležitých elementov. Dosiahnutie prijateľného výsledku jeho zobrazenia je možné niekoľkými spôsobmi. Rozhodol som sa použiť výškovú mapu popísanú v podkapitole 4.3.

Najskôr je vygenerovaná pseudonáhodná výšková mapa pomocou Perlinovho šumu. Táto má tvar štvorcovej mriežky bodov o veľkosti $200 \times 200$ políčok. Dané body v nej reprezentujú jednotlivé vrcholy primitív – trojuholníkov. Súradnice týchto bodov sú generované v cykle, v ktorom sú súradnice $x$ a $z$ postupne zväčšované o konštantnú hodnotu a súradnica $y$ je dopočítaná prostredníctvom 2D Perlinovho šumu.

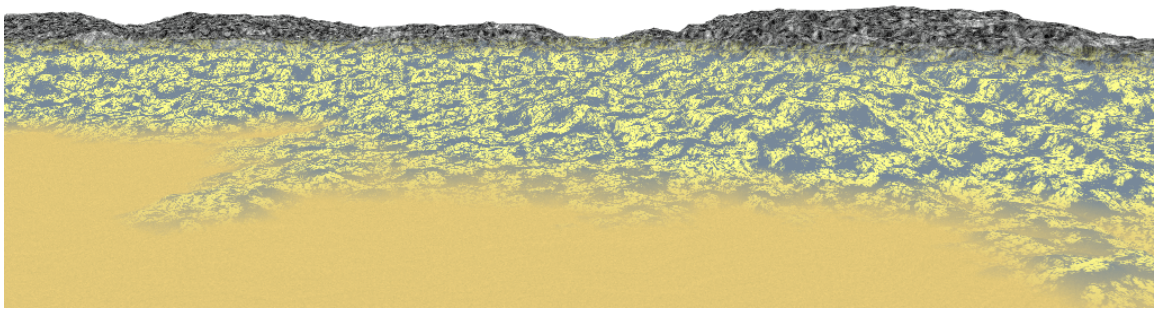
Terén má reprezentovať korálový útes, takže bolo potrebné nastaviť Perlinov šum tak, aby generoval hornatý terén s priehlbňou uprostred, ktorá by predstavovala lagúnu. Keďže som sa rozhodol vytvoriť korálový útes typu Atol, ktorý má prstencovitý tvar s kaskádovitým povrchom, bolo nutné generované hodnoty výšok upraviť. Použil som na to rovnicu kružnice:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{5.1}$$

Kde $x_0$ a $y_0$ sú súradnice stredu kružnice, ktorý je stanovený doprostred výškovej mapy. V tomto prípade majú teda hodnotu $x_0 = 100$ a $y_0 = 100$. Pre každý bod výškovej mapy určený súradnicami $x$ a $y$ je vypočítaný jeho polomer $r$. Podľa hodnoty polomeru sa zistí, v ktorom kruhovom pásme sa daný bod nachádza a jeho hodnota výšky sa predelí určenou hodnotou pre dané pásmo. Čím je kruhové pásmo bližšie k stredu, tým je táto hodnota väčšia. Týmto spôsobom sa vytvorí kruhový a kaskádový terén.

Okrem súradníc bodov pre výškovú mapu sa generujú aj textúrovacie koordináty pre textúry terénu a pre textúry kaustík. Tiež sa generujú indexy jednotlivých trojuholníkov, pomocou ktorých sa terén nevykresluje po samostatných trojuholníkoch ale po trojuholníkových pásoch. Používa sa na to funkcia `glDrawElements()`. Ďalej je potrebné spočítať normály jednotlivých primitív pre výpočet Phongovho osvetľovacieho modelu. Pre správny výpočet normály daného vrcholu je potrebné použiť normály všetkých susedných trojuholníkov a nie len normálu pre daný trojuholník.

Pre povrch terénu sú použité tri textúry, ktorých tvorba je popísaná v podpodkapitole 5.2.1. Sú to textúry pre piesok, korálový povrch a skalu. Aplikujú sa na základe výšky. V spodnej tretine sa použije textúra piesku, v strednej textúra korálového povrchu a nakoniec textúra skaly. Výsledný terén je možné vidieť na obrázku 5.1.



Obr. 5.1: Vygenerovaný terén vytvorený pomocou výškovej mapy a Perlinovho šumu. Terén predstavuje korálový útes.

## 5.2 Generovanie textúr

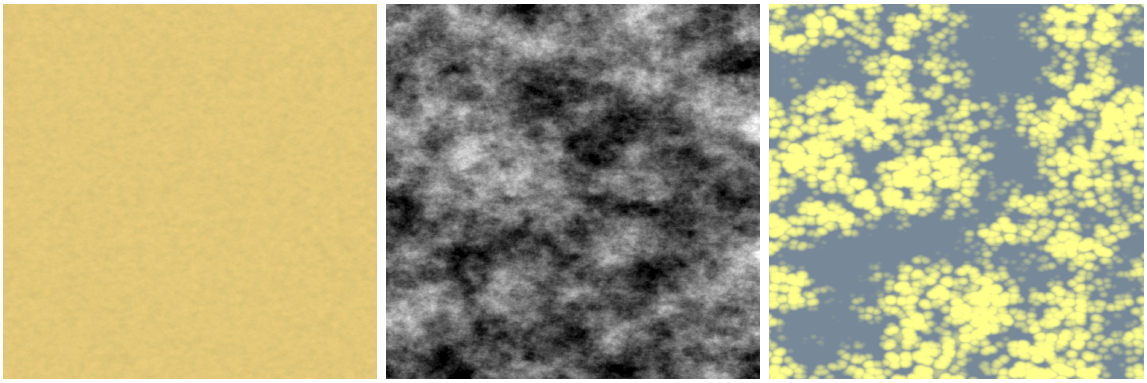
Textúra slúži pre detailný popis vlastností povrchu objektu. Tieto vlastnosti sú dôležité pre vnímanie farby, kvality a štruktúry daného objektu. Kvôli veľkostnému limitu grafického intra nie je možné mať textúry uložené ako statické údaje. Preto sa musia procedurálne vygenerovať.

Vytvárať textúry pre všetky objekty v scéne procedurálne pri každom vykreslení by bolo výpočtovo náročné. Preto je vhodné všetky textúry mapované na objekty, ktoré nemenia svoju farbu a štruktúru vygenerovať iba raz, uložiť a používať pri každom vykreslení scény. Tento spôsob je použitý v práci pre textúry terénu a vegetácie.

Najskôr je potrebné vytvoriť textúry do ktorých sa bude vykresľovať. Každej textúre je nutné zadefinovať požadované vlastnosti – dimenziu, veľkosť, formát. Potom je vytvorený a aktivovaný framebuffer object, ktorý slúži na vykreslenie zadaných dát (farba, pozícia, hĺbka a iné). V tomto prípade sa vykresľuje do vytvorenej textúry, ktorá je pripojená k attachmentom. Sú použité `GL_COLOR_ATTACHMENTi`. Výstup fragment shaderu je zapísaný do i-teho color attachmentu aktuálneho framebufferu. Nakoniec je potrebné definovať zoznam attachmentov.

### 5.2.1 Textúry pre terén

Terén predstavuje korálový útes, pre ktorého povrch sú použité tri textúry. Sú to textúry pre piesok, skalu a korálový povrch. Prvé dve sú vytvorené pomocou Perlinovho šumu popísaného v podkapitole 4.1.1. Voroného diagram, ktorý je popísaný v podkapitole 4.1.2, je použitý pre vytvorenie textúry korálového povrchu. Perlinov šum a Voroného diagram sú implementované vo fragment shadery. Jednotlivé textúry pre terén je možné vidieť na obrázku 5.2.



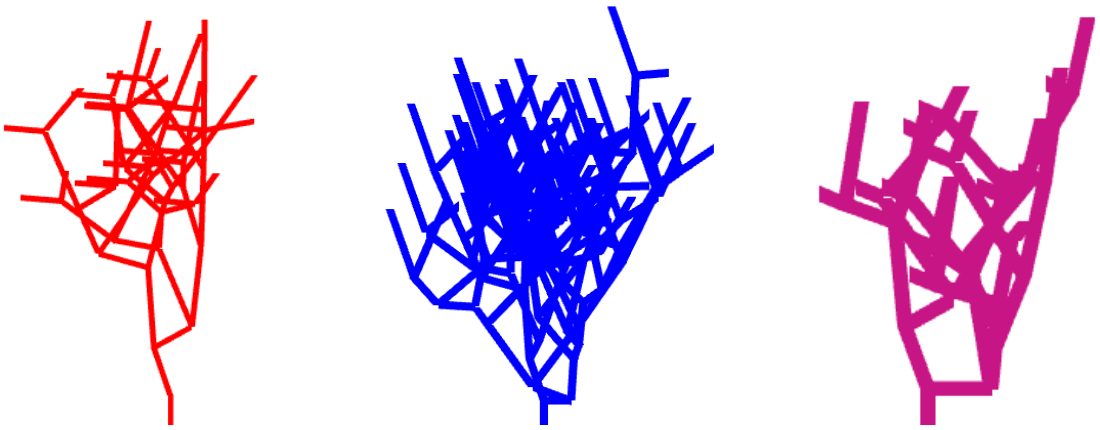
Obr. 5.2: Textúry použité pre povrch terénu. Zľava doprava: textúra piesku, skaly a korálového povrchu. Textúra pre piesok a skalu je vytvorená pomocou Perlinovho šumu. Pre textúru korálového povrchu je použitý Voroného diagram.

### 5.2.2 Textúry pre vegetáciu

Vegetáciu tvoria 3 druhy korálov a morské riasy. Textúry pre korály sú vytvorené na princípe L-systému popísanom v podkapitole 4.5.

Tvoria sa v geometry shadery, kde sa pošle jeden vrchol nastavený na stred textúry. Od jeho pozície sa pripočítaním vhodne zvolenej hodnoty súradnice $y$ vytvorí nový vrchol. Ten je pridaný (vykreslí sa čiara reprezentujúca stonku) k aktuálnemu primitivu pomocou funkcie `EmitVertex()`. Z pozície tohto vrcholu sú vytvorené ďalšie dva pripočítaním vygenerovanej hodnoty súradníc $x$ a $y$. Hodnota závisí od zvoleného uhla natočenia. Po každom vytvorení vrcholu sa volá funkcia `EmitVertex()`. Novo vytvorený vrchol sa uloží na zásobník pokiaľ tento nie je plný. Zo zásobníka sa postupne vyberajú vrcholy, ktorých pozícia slúži na generovanie ďalších dvoch vrcholov. Ak je zásobník prázdny, zavolá sa funkcia `EndPrimitive()`, ktorá dokončí aktuálny výstup primitiva a začne nový. Hrúbka vykreslených čiar je nastavená pomocou funkcie `glLineWidth()`. Takto vytvorené textúry pre jednotlivé korály je možné vidieť na obrázku 5.3.

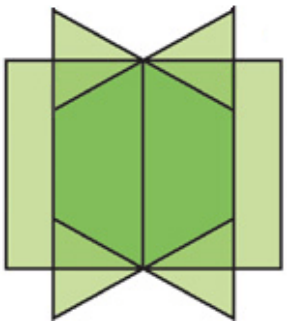
Textúra pre morské riasy je vytvorená vo fragment shadery. Na jej tvorbu som použil program od Erkamana [6].



Obr. 5.3: Textúry pre jednotlivé korály vytvorené na princípe L-systému.

## 5.3 Generovanie vegetácie

V prírode je korálový útes bohatý na rozmanitú podmorskú vegetáciu a preto nesmie chýbať ani v mojom intre. Zobrazujú sa tu tri druhy korálov a morské riasy. Zvlášť vykresľovať každý korál a steblo riasy aby bolo unikátne je kvôli veľkému počtu polygónov výpočtovo náročné. Tento problém je riešený pomocou objektu skladajúceho sa z troch štvorcov, ktorý je možné vidieť na obrázku 5.4. Na každý štvorec sa namapuje textúra pre korál alebo riasy. Pre správny obraz nezávisle od uhla pohľadu sú štvorce prekrížené. Týmto spôsobom budú riasy a korály reprezentované len niekoľkými polygónmi.



Obr. 5.4: Objekt skladajúci sa z troch štvorcov, ktoré sú navzájom prekrížené pre zabezpečenie správneho obrazu nezávisle od uhla pohľadu. Na každý štvorec sa namapuje textúra pre podmorskú flóru, ktorá je takto reprezentovaná len niekoľkými polygónmi.

Predovšetkým je potrebné určiť pozície pre umiestnenie čiastkovej podmorskej flóry. Tie sú získané pri generovaní výškovej mapy. Ako bolo napísané vyššie, výšková mapa má tvar štvorcovej mriežky bodov. Náhodne je vybraných niekoľko bodov tak, aby ich počet nebol príliš veľký, ale ani malý a ich rozmiestnenie pôsobilo prirodzene. Získané body sa pošlú do geometry shaderu. Tu je pre každý bod vytvorený objekt s troma navzájom prekríženými štvorcami. Posúvaním horných vrcholov objektu na jednu stranu a naspäť je dosiahnutý efekt jemného pohybu.

Vo fragment shadery sa na štvorce namapuje náhodne vybraná textúra podmorskej vegetácie. Keďže programovací jazyk GLSL neposkytuje žiadnu funkciu pre generovanie náhodných čísel, bolo nutné si túto funkciu napísať.

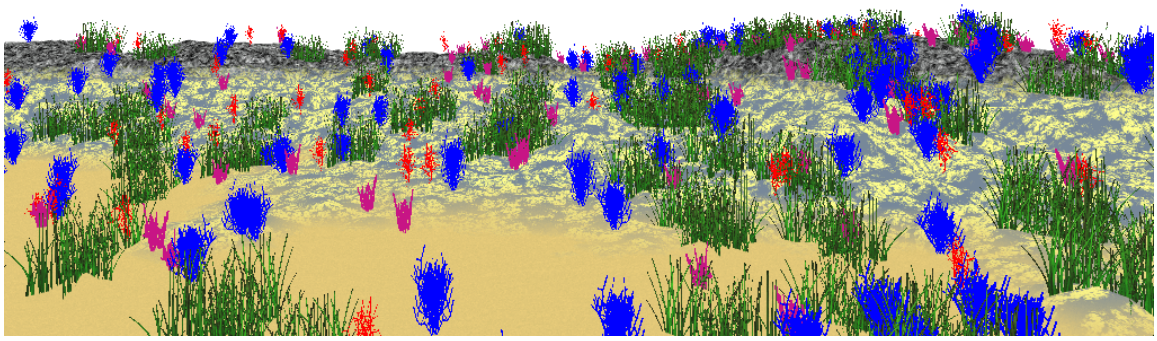
```
//funkcia vrati nahodne cislo z pozicie danej vegetacie
float myRand(vec2 cord){
    return fract(sin(dot(cord.xy ,vec2(12.9898,78.233))) * 3758.5453);
}

// pozicia objektu ktory predstavuje vegetaciu
vec2 cord = vec2(coralSpacePos.x + coralSpacePos.y, coralSpacePos.z);

// ziskane nahodne cislo
float randNum = myRand(cord);
```

Kód 5.1: Ukážka kódu pre získanie náhodného čísla.

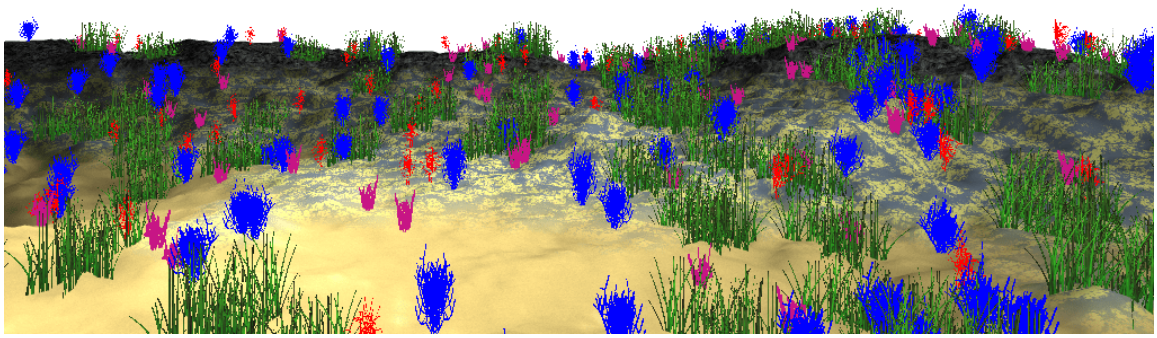
Vstupom tejto funkcie sú súradnicové pozície ($x$, $y$, $z$) daného objektu, pre ktorý sa vyberá textúra. Tieto súradnice sa nemenia a preto funkcia vždy vygeneruje rovnaké číslo, na základe ktorého sa vyberie textúra. Nevznikne tak „blikanie“ objektu zmenou textúry pri každom vykreslení. Vykreslenú podmorskú vegetáciu je možné vidieť na obrázku 5.5.



Obr. 5.5: Korálový útes s podmorskou vegetáciou.

## 5.4 Osvetlenie scény

Terén bez použitia osvetlenia vyzerá neprirodzene, čo je si možné všimnúť na obrázku 5.1. Preto je potrebné naň aplikovať nejaký osvetľovací model. V tejto práci sa využíva Phongov osvetľovací model popísaný v podkapitole 4.2. V tomto modeli je potrebné nastaviť pozíciu, intenzitu a farbu svetelného zdroja a získať pozíciu, normály a farbu osvetľovaného objektu. Potom je možné vypočítať difúznu zložku. K nej sa pripočíta odrazová zložka, ktorej hodnota je vhodne zvolená podľa materiálu osvetľovaného objektu. Keďže Phongov osvetľovací model je empirický, nie je potrebné počítať odraz svetla, ktorý by bol výpočtovo náročný. Hodnota ambientnej zložky je teda adekvátne určená. Scénu s týmto osvetlením je možné vidieť na obrázku 5.6.



Obr. 5.6: Osvetlenie scény pomocou Phonghovho osvetľovacieho modelu.

## 5.5 Kaustiky

Optický jav nazývaný kaustika vzniká zalomením svetelného lúča nejakým zakriveným predmetom alebo plochou, v tomto prípade morskou hladinou. Podobizeň kaustík je vytvorená na princípe Voroného diagramu popísaného v podpodkapitole 4.1.2. Jedná sa o dynamicky meniacu sa 2D textúru, ktorú je možné vidieť na obrázku 5.7. Je implementovaná vo fragment shadery a pomocou funkcie `mix()` sa zmiešava s farbou textúry objektu, na ktorý je aplikovaná. V tejto práci sa používa na terén, morskú hladinu, vegetáciu a ryby.

## 5.6 Skybox

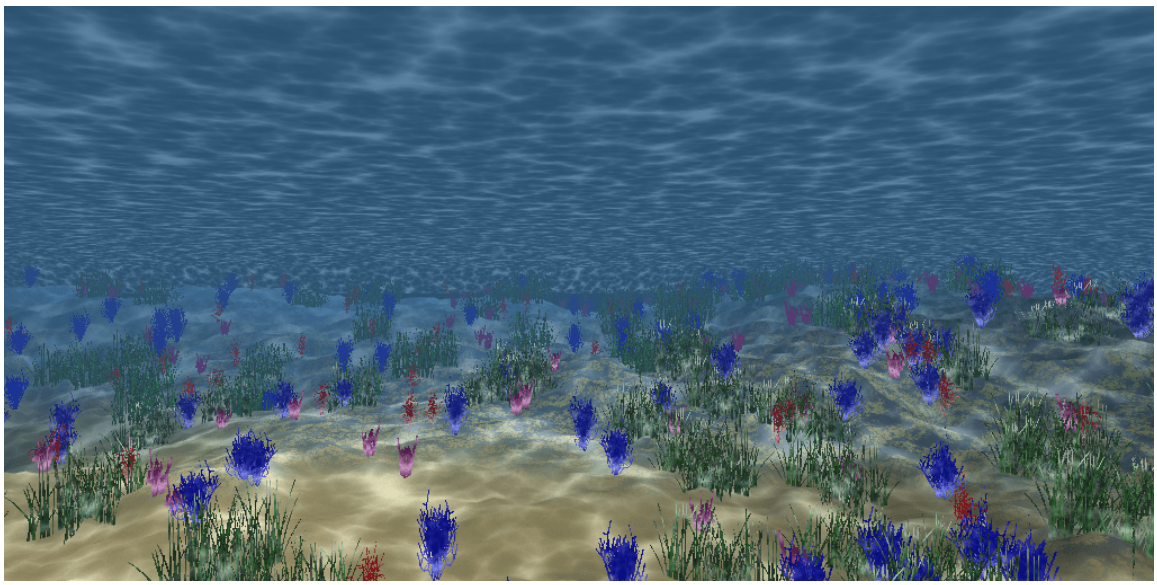
Vzdialené okolie scény má predstavovať podmorské prostredie. V tejto práci je vytvorené pomocou skyboxu, ktorý je popísaný v podkapitole 4.4. Najprv je potrebné vytvoriť vrcholy ktoré budú reprezentovať tvar kocky. Kvôli veľkostnému obmedzeniu grafického intra nie je možné načítavať obrázok pre skybox. Preto je nutné ho procedurálne vytvoriť. Vo fragment shadery je definovaná farba mora, ktorá sa nanesie na všetky strany kocky. Od určitej hodnoty výšky sa namapuje textúra kaustík, ktorá bude simulovať morskú hladinu. Takto vytvorený skybox je možné vidieť na obrázku 5.7.

## 5.7 Hmla

Morská voda nie je dokonale priezračná, takže s rastúcou vzdialenosťou klesá viditeľnosť. Táto skutočnosť je v tejto práci simulovaná pomocou efektu hmly. Hmla je vytvorená zmiešaním farby mora s farbou vykresleného objektu vo fragment shadery pomocou funkcie `mix()`. Hodnota pre interpoláciu týchto dvoch farieb sa určí zo vzťahu:

$$f = \frac{1}{e^{(db)^2}} \tag{5.2}$$

Kde $e$ je Eulerovo číslo (základ prirodzeného logaritmu), $b$ je hustota hmly a $d$ je vzdialenosť od miesta pohľadu. Scénu po aplikovaní hmly je možné vidieť na obrázku 5.7.

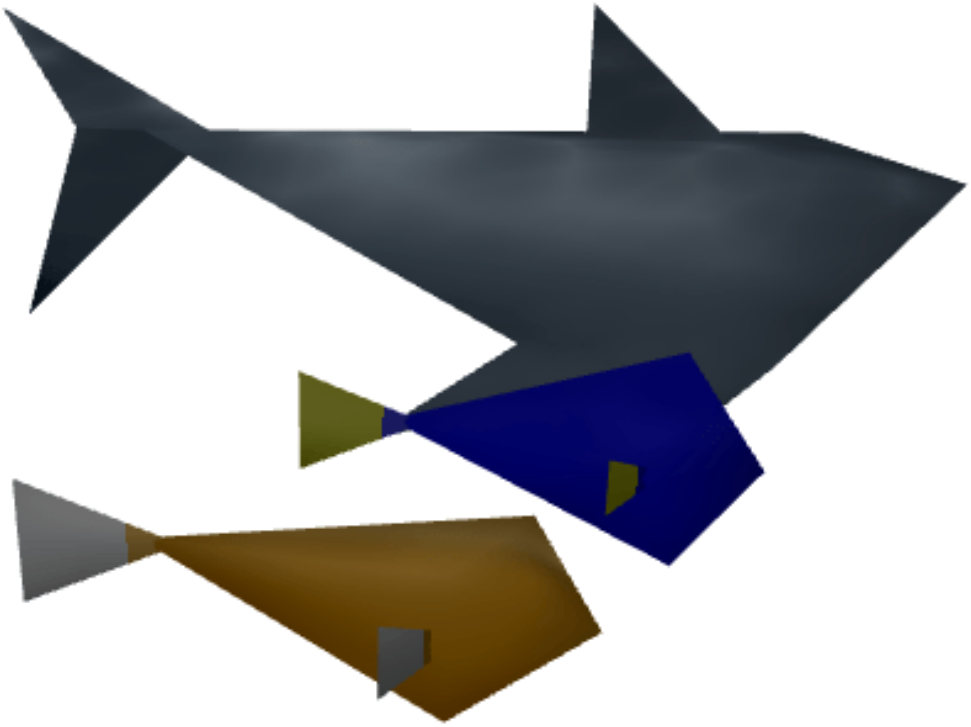


Obr. 5.7: Ukážka podmorskej scény vrátane kaustík, skyboxu a hmly.

## 5.8 Modely rýb a ich animácia

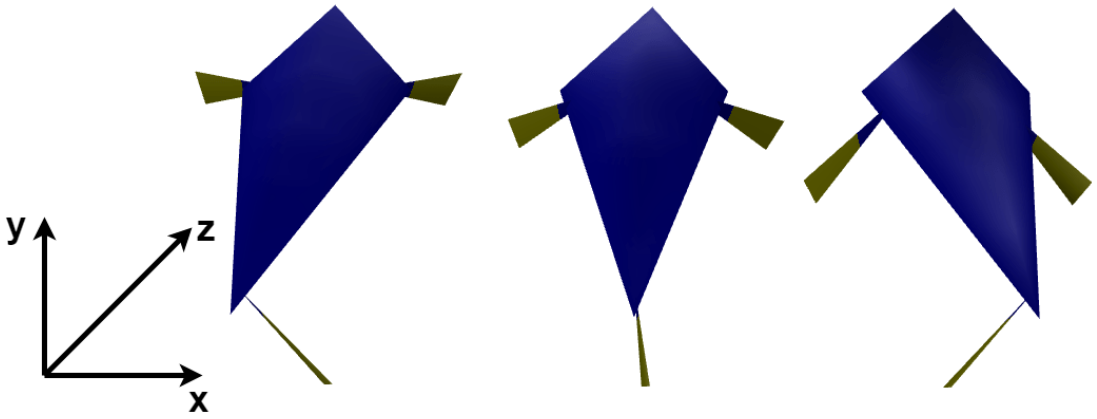
Pre oživenie scény sú vytvorené dva modely rýb a model žraloka. Každý model je zložený z dvoch ihlanov, ktoré sú spojené svojimi podstavami. Jeden je kratší a predstavuje hlavu ryby. Druhý je dlhší a reprezentuje telo. Jeden druh ryby má ihlany kosoštvorcovej podstavy

a druhý druh ryby a žralok má ihlany štvorcovej podstavy. Plutva aj chvost pre ryby sú vytvorené pomocou jedného trojuholníka. Chvost pre žraloka je tvorený dvoma spojenými trojuholníkmi. Jednotlivé modely je vidieť na obrázku 5.8.



Obr. 5.8: Vytvorené modely rýb a žraloka.

Telo, chvost a plutvy rýb sú animované. Pri každom vykreslení sú menené len hodnoty súradníc vrcholov určených trojuholníkov tvoriacich model ryby. Trojuholníky sú určené tak, aby sa časť tela a chvost pohybovali zľava doprava. Plutvy zasa spredu dozadu. Na obrázku 5.9 je možné vidieť, že pri pohybe tela a chvostu je menená len $x$–ová súradnica a pri pohybe plutiev $z$–ová. Celý model sa pohybuje pripočítavaním určenej hodnoty k $z$–ovej súradnici všetkých vrcholov.




Obr. 5.9: Ukážka animácie modelu ryby.

Začiatočné umiestnenie a následný pohyb rýb sú nastavené tak, aby preplávali od jedného konca korálového útesu k druhému. Tento koniec je určený preplávanou vzdialenosťou daného modelu. Na ňom je model ryby horizontálne rotovaný o požadovaný uhol a premiestnený na druhý koniec útesu. Farba jeho tela, plutiev a chvostu je tiež zmenená o určenú hodnotu. Toto je možné vidieť na nasledujúcom úseku kódu.

```
// kontrola preplavanej vzdialenosti
if (currentDistance >= finalDistance)
{
        moveFish.z -= currentDistance; // zmena umiestnenia
        angleRotation += angle; // zmena uhla rotacie
        colorFish += colorValue; // zmena jednej farebnej zlozky
}

// horizontalna rotacia modelu
Model = glm::rotate(model, angleRotation, glm::vec3(0.0f, 1.0f, 0.0f));

// premiestnenie modelu
Model = glm::translate(model, move);

// zmena farby tela
glm::vec3 colorFishBody = glm::vec3(1.0f, 0.0f, colorFish);

// zmena farby chvostu a plutiev
glm::vec3 colorFishFins = glm::vec3(1.0f, 1.0f, 1.0f - colorFish);
```

Kód 5.2: Ukážka kódu pre premiestnenie, rotáciu a zmenu farby modelu ryby.

Premiestnenie rýb na začiatok a zmenu ich farby si divák nevšimne, keďže kamera je vždy natočená tak, aby nezachytila tento jav. Týmto spôsobom sa môže pozorovateľovi zdať, že nad korálovým útesom plávajú vždy iné ryby.

Okrem modelov rýb sa v scéne nachádza aj model truhlice. Truhlica má tvar kvádra a hnedú farbu. Na konci intra sa otvorí. Animácia otvorenia je vykonaná podobným spôsobom ako animácia pohybu rýb. Scénu vrátane rýb a truhlice je možné vidieť na obrázku 5.10.

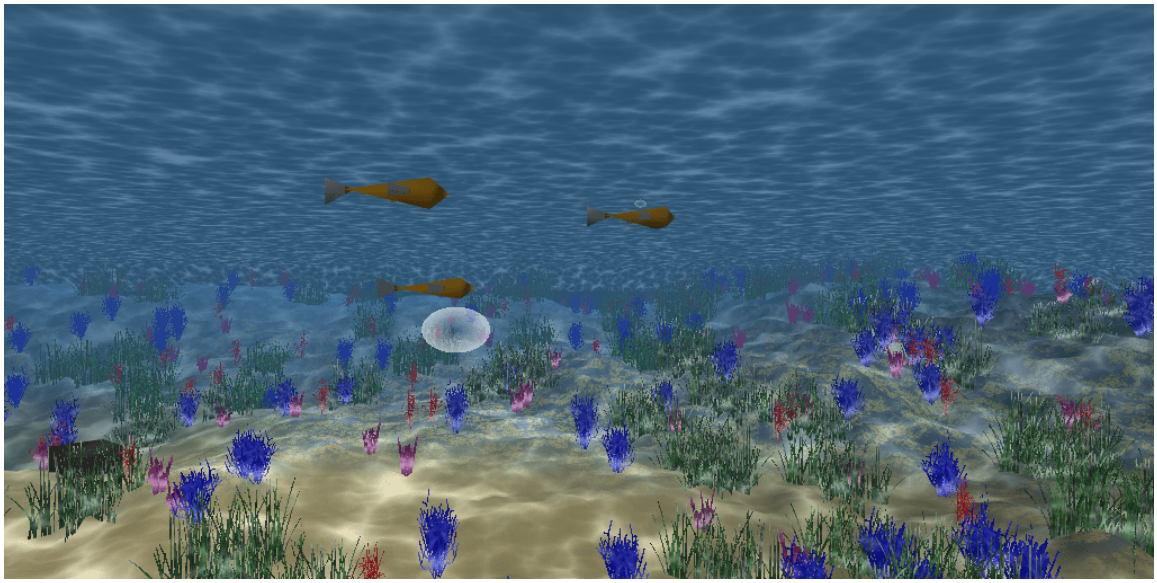
## 5.9 Bubliny

Bubliny sú vytvorené pomocou časticového systému popísaného v podkapitole 4.6 a billboardingu popísaného v podkapitole 4.7. Model bubliny sa skladá z 2 trojuholníkov tvoriacich štvorec. Na tento štvorec je namapovaná textúra bubliny. Textúra bubliny sa vytvára vo fragment shadery a má tvar kruhu. Ten je tvorený pomocou rovnice pre kružnicu (5.1), pričom čím je daný pixel bližšie k stredu kružnice, tým je priehľadnejší. Tiež je využitý efekt hmly – vykreslovaná bublina menej vzdialená od kamery je viditeľnejšia. Okrem textúry bubliny sa na vytvorený štvorec aplikuje princíp billboardingu dodávajúci 3D dojem.

Ďalšie bubliny sú vytvorené pomocou inštancovania tohto modelu. Tento spôsob zabezpečí vykreslenie jedného modelu viackrát do scény. V tejto práci sa na to používa funkcia `glDrawArraysInstanced()`, ktorá navyše obsahuje parameter určujúci počet inštancií k vykresleniu. Keďže sa jedná o polopriehľadné objekty, je pre správne vykreslenie potrebné ich zoradiť od najvzdialenejšieho k najbližšiemu.

Na určenom mieste pod terénom sa generujú bubliny (častice). Každej bubline je priradená náhodná hodnota veľkosti a smeru pohybu. Rýchlosť a dobu trvania majú všetky

bubliny rovnakú. Bublina sa pohybuje vygenerovaným smerom počas určenej doby trvania, ktorá sa zmenšuje pri každom vykreslení. Scénu s bublinami je možné vidieť na obrázku 5.10.



Obr. 5.10: Ukážka podmorskej scény po pridaní rýb, bublín a truhlice.

## 5.10 Kamera

Grafické intro je rozdelené do štrnástich časových intervalov. V každom sú nastavené požadované hodnoty pre smer pohybu kamery (súradnice $x$, $y$ a $z$), rýchlosť pohybu a uhol pohľadu. Tieto hodnoty v jednotlivých intervaloch sú zvolené tak, aby natáčanie a pohyb kamery bolo počas intra väčšinou pozvolné.

## 5.11 Hudba

Väčšina intier poskytuje divákovi nielen grafický zážitok, ale aj zvukový. Hudobný doprovod dokresľuje atmosféru a zvyšuje výsledný dojem z celej prezentácie. Samozrejme je potrebné brať do úvahy veľkostné obmedzenie celej aplikácie. Preto nie je možné použiť bežné zvukové formáty typu MP3 alebo WAWE, keďže priemerne dlhá skladba v nich by dosiahla veľkosť niekoľko megabajtov.

Riešeným tohto problému by mohol byť napríklad formát MIDI, v ktorom je uložený v podstate len zoznam nôt danej melódie. Hlavnou nevýhodou takéhoto riešenia je skutočnosť, že programátor nemá plnú kontrolu nad výsledným zvukom. Všetky výpočty sú totiž ponechané zvukovému hardwaru. Z tohto dôvodu sa tento spôsob používa len ojedinele a to väčšinou pri intrách s obmedzením na 4kB a menej.

V tomto intre som sa rozhodol použiť formát v2m, ktorý oproti MIDI, poskytuje väčšie možnosti a dosahuje lepšiu kvalitu. Pre interpretáciu tohto formátu je potrebný prehrávač, takzvaný syntetizátor. Jedná sa o program, ktorý je schopný pomocou rôznych matematických algoritmov vytvoriť zvuk rôznych hudobných nástrojov. Príklad takéhoto prehrávača je dodávaný spolu s knižnicou libv2, ktorá poskytuje funkcie vďaka ktorým je možné súbor s hudbou otvoriť, vložiť do spustiteľného programu a prehrať. Táto knižnica bola vyvinutá

nemeckou skupinou Farbrausch špeciálne pre grafické intrá s obmedzenou veľkosťou. Keďže niektoré časti kódu sú napísané v programovacom jazyku assembler, bolo nutné pridať prekladač pre tento jazyk. Rozhodol som sa pre prekladač YASM.

Samostatný výber hudobného podkladu je veľmi dôležitý. V prípade nevhodne vybranej skladby sa môže celkový umelecký dojem z intra skaziť. Zložiť vlastnú hudbu by bolo náročné, a preto som sa rozhodol nájsť skladbu tematicky blízku k môjmu intru. Po vypočutí viacerých hudobných diel som sa rozhodol pre skladbu s názvom Patient Zero od Melwyn and Little Bitchard.

# Kapitola 6

# Techniky a metódy pre obmedzenie veľkosti aplikácie

Výsledná veľkosť programu hrá dôležitú úlohu u aplikácií typu grafické intro. Existuje mnoho spôsobov, ako docieliť zmenšenie výsledného programu. Dôležitú úlohu hrá nastavenie prekladača. Ďalšou metódou, ktorá sa hojne využíva pri tvorbe grafického intra je skomprimovanie výslednej aplikácie pomocou špeciálneho baliaceho nástroja – Exe packera.

Ušetriť miesto je možné aj výberom programovacieho jazyka a používaním vhodných konštrukcií v ňom. Medzi štandardy dobrého programátorského štýlu patrí využívanie funkcií. Prioritne je dobré mať na mysli to, aby sa veľkosť aplikácie zmenšila, keďže nevhodným použitým funkcie môže naopak narásť.

Tiež je vhodné používať pri práci s číslami s plávajúcou desatinou čiarkou dátový typ `float` (ak je rozsah dostačujúci) namiesto dátového typu `double`. Pri použití dátového typu `float` je vždy nutné písať za desatinným číslom postfixovú značku `f`, napr. `0.5f`. Pokiaľ by sme ju neuviedli, tak by prekladač spravil implicitnú konverziu na `double`. Dané číslo by teda neuložil na 4, ale na 8 bitov.

## 6.1 Nastavenie prekladača

Pri vývoji tejto práce som pracoval v prostredí Microsoft Visual Studio 2013. Toto vývojové prostredie poskytuje veľké množstvo nastaviteľných parametrov pre kompilátor [11] a linker [12]. Každý z nich má určitý vplyv na aplikáciu. Keďže mojím cieľom bolo dosiahnuť čo najmenšiu veľkosť výsledného programu, tak som sa zameral na optimalizáciu veľkosti.

Použité parametre s najvýznamnejším vplyvom na veľkosť sú:

- Parameter `-Os` – minimalizuje veľkosť na úkor rýchlosti.
- Parameter `-s` – eliminuje nevyužitý kód programu a tiež všetky statické údaje pre ladenie programu.
- Parameter `/INCREMENTAL:NO` – zabezpečí vypnutie inkrementálneho linkovania.
- Parameter `/DEBUG` – vypne všetky informácií určené pre debugger.

## 6.2 Exe packery

I napriek optimalizácii veľkosti programu pomocou nastavenia prekladača, bola veľkosť výslednej aplikácie skoro dvojnásobne väčšia. Preto som použil Exe packer. Jedná sa o aplikáciu, ktorá slúži na komprimáciu skompilovaných programov. Pri spustení takto zmenšeného súboru, dekomprimačný algoritmus vytvorí pôvodný kód z komprimovaných dát a následne ho spustí. Doba rozbaľovania býva tak väčšinou veľmi krátka, že ju ani nezaregistrujeme. Činosť aplikácie po použití komprimačného nástroja zostáva rovnake.

Existuje mnoho dostupných exe packerov, ktoré sa medzi sebou líšia použitými algoritmami kompresie a špecializáciou na rôzne typy aplikácií. Preto bolo potrebné zistiť, ktorý z nich bude pre túto prácu najlepší.

Celkovo som otestoval tri voľne dostupné komprimačné programy. Dva boli úspešne aplikované – skomprimované intro sa bezproblémovo spustilo. Po použití komprimačného programu kkrunchy 0.23 alpha 2 sa moje intro nespustilo. Je to sklamaním, pretože sa jedná o program vyvinutý nemeckou demoskupinou Farbrausch, ktorý je prispôsobený práve na 64kB grafické intrá. Príčinu tohto stavu som nehľadal, keďže komprimačný nástroj s názvom MPRESS vo verzii 2.19 dostatočne zmenšil veľkosť výslednej aplikácie. Dosiahol najlepší komprimačný pomer a to 29,5%. Porovnanie úspešnosti komprimácie použitých komprimačných nástrojov je možné vidieť v tabuľke 6.1.

| Nástroj | Verzia | Použité parametre | Výsledná veľkosť v [kB] | Komprimačný pomer v [%] |
|---|---|---|---|---|
| MPRESS | 2.19 | `-s` | 62 | 29,5 |
| UPX | 3.93 | `--ultra-brute` | 64 | 30,5 |

Tabuľka 6.1: Porovnanie úspešnosti komprimácie jednotlivých komprimačných nástrojov na súbore s pôvodnou veľkosťou 209kB

# Kapitola 7

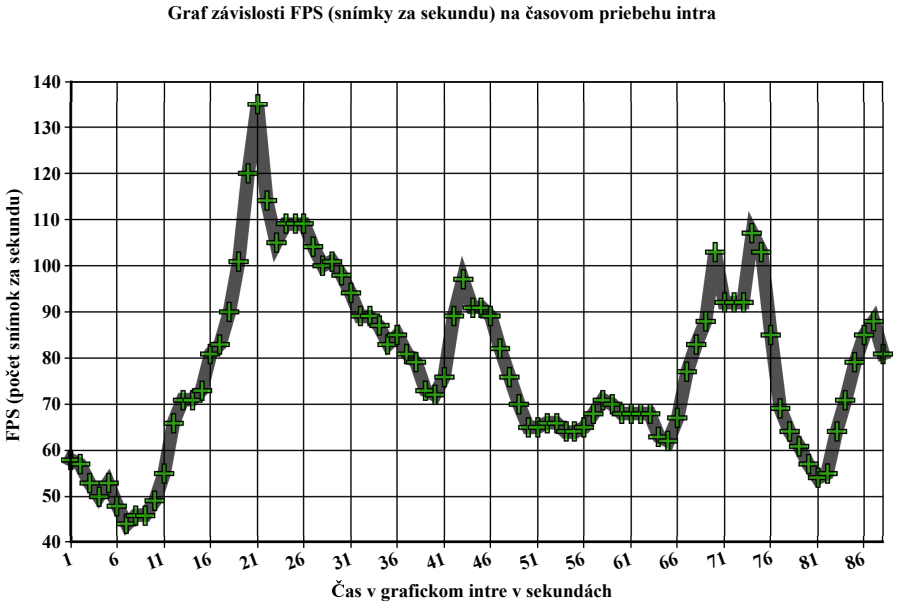
# Výsledok

V tejto kapitole je prezentovaný finálny výsledok vytvoreného programu a zhodnotenie rýchlosti jeho vykreslenia.

## 7.1 Zhodnotenie rýchlosti zobrazovania

Jedným s cieľov tejto práce bolo vytvoriť grafické intro, schopné plynulého behu v reálnom čase. Preto na výslednej aplikácií bola otestovaná závislosť FPS (snímky za sekundu) na časovom priebehu intra.

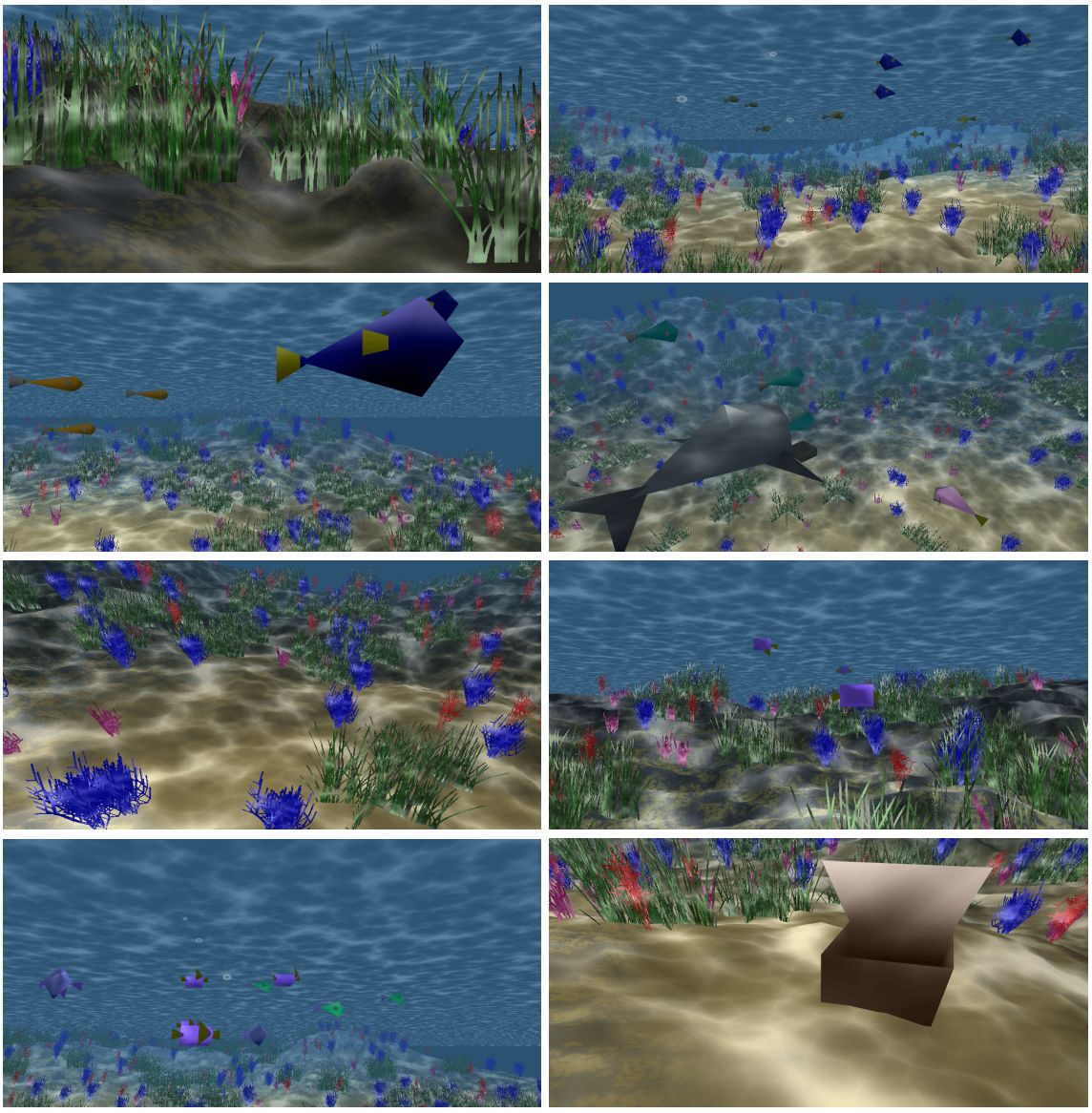
Výsledok je možné vidieť na grafe 7.1. Z neho vyplýva, že hodnoty FPS kolíšu počas celého priebehu intra, čo je dané počtom vykresľovaných objektov v zobrazenej scéne v danom čase. Napriek tomu, že hodnoty FPS oscilujú, rýchlosť vykreslenia je vždy väčšia ako 30 FPS, zobrazovanie intra je plynulé. Meranie hodnôt FPS bolo vykonané na notebooku Ultrabook HP ENVY 6-1150ec Midnight Black s grafickou kartou AMD Radeon HD 7600M.




Obr. 7.1: Graf závislosti FPS (snímky za sekundu) na časovom priebehu intra.

## 7.2 Výsledná scéna grafického intra

Vytvorené grafické intro má veľkosť 62kB a trvá skoro 90 sekúnd. Začína na úpätí vonkajšej strany korálového útesu. Pokračuje pomalým pohybom cez korály a morské riasy do stredu útesu. Tu sa objavujú ryby v rôznych farebných kombináciách, neskôr aj žralok. Pohyb a natáčanie kamery je počas intra väčšinou pohodovo relaxačný. Intro končí otvorením truhličky na morskom dne. Pár vybraných snímok z intra je možné vidieť na obrázku 7.2.



Obr. 7.2: Ukážky z výsledného grafického intra.

# Kapitola 8

# Záver

Cieľom tejto bakalárskej práce bolo vytvorenie grafického intra s obmedzenou veľkosťou. Požadovaný veľkostný limit bol splnený, výsledné intro nepresiahlo veľkosť 64kB. Pri tvorbe som sa zoznámil s knižnicou OpenGL a jej nadstavbami. Ďalej som naštudoval rôzne metódy a techniky pre generovanie grafického obsahu. Pre ich demonštráciu v grafickom intre som si vybral podmorskú scenériu s korálovým útesom a plávajúcimi rybami.

Korálový útes je vytvorený pomocou výškovej mapy a Perlinovho šumu. Mapujú sa na ňom tri textúry. Pre ich tvorbu je použitý Voroného diagram a Perlinov šum. Korály sú tvorené na princípe L-systému. Podarilo sa mi vytvoriť dva druhy jednoduchých modelov rýb a model žraloka. Ich plutvy a chvost sú animované. Pre okolie scény je použitý skybox. Kaustiky sú vytvorené pomocou Voroného diagramu. Pre tvorbu bubliniek je využitý časticový systém a billboarding. Osvetlenie scény je realizované pomocou Phonghovho osvetľovacieho modelu.

Ďalšie pokračovanie projektu je možné v zlepšení súčasných detailov textúr pre terén a vodnú hladinu a tiež pre vytvorenie šupín modelov rýb aplikovaním normal mappingu. Pre zobrazenie tieňov môže byť použitý shadow mapping, korály môže reprezentovať 3D model, morské riasy a svetelné lúče môžu byť vytvorené pomocou časticových systémov. Najprv však bude nutné získať potrebné miesto pre uvedené zlepšenia. To je možné dosiahnuť použitím WinAPI pre prácu s oknom namiesto knižnice GLFW.

Pri vývoji tejto práce som získal mnoho znalostí a skúseností z oblasti tvorby grafických aplikácií pomocou knižnice OpenGL a jej nadstavieb. Tiež som sa zoznámil s problematikou vytvárania programov s obmedzenou veľkosťou a možnosťami ich kompresie. Vývoj grafického intra bol veľmi zaujímavý a veľkým prínosom v rozvoji mojich programátorských schopností v tejto oblasti.

# Literatúra


[1] 2ohm: *Panda3D et HeightMap*. [Online; navštívené 20.04.2017].
URL https://zestedesavoir.com/tutoriels/334/panda3d-et-heightmap/

[2] Aggarwal, V.; Garima: Minutiae Fingerprint Recognition Using Mahalanobis Distance. *International Journal of Scientific & Engineering Research*, ročník 4, č. 9, 2013: s. 2330–2335, ISSN 2229–5518.

[3] Aurenhammer, F.: Voronoi diagrams-a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, ročník 23, č. 3, 1991: s. 345–405, ISSN 03600300, DOI: 10.1145/116873.116880.

[4] Baros, M.: Heightmap Terrain Rendering. *Dr. Dobb's Journal*, ročník 31, č. 6, 2006: s. 51–54, ISSN 1044789X.

[5] Dohnal, M.: *Osvětlovací modely v počítačové grafice*. [Online; navštívené 13.04.2017].
URL http://home.zcu.cz/~mika/MM/Galerie%20studentskych%20praci%20MM/2010/Dohnal-Osvetlovac%C3%9D%20modely.pdf

[6] erkaman: *ugly grass*. [Online; navštívené 11.04.2017].
URL https://www.shadertoy.com/view/MlSXRV

[7] Hörmann, W.; Derflinger, G.: A portable random number generator well suited for the rejection method. *ACM Transactions on Mathematical Software (TOMS)*, ročník 19, č. 4, 1993: s. 489–495, ISSN 00983500.

[8] Kessenich, J.; Baldwin, D.; Rost, R.: *The OpenGL Shading Language*. [Online; navštívené 28.04.2017].
URL https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.50.pdf

[9] Longstreet, D.: *Voronoi As Art*. [Online; navštívené 23.04.2017].
URL http://www.davidlongstreet.com/voronoi.html

[10] Meiri, E.: *Tutorial 25: SkyBox*. [Online; navštívené 19.04.2017].
URL http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html

[11] Microsoft: *Compiler Options Listed by Category*. [Online; navštívené 10.04.2017].
URL https://msdn.microsoft.com/en-us/library/19z1t1wy(v=vs.120).aspx

[12] Microsoft: *Linker Options*. [Online; navštívené 10.04.2017].
URL https://msdn.microsoft.com/en-us/library/y0zzbyt4(v=vs.120).aspx

[13] Perlin, K.: *Noise Hardware*. [Online; navštívené 15.04.2017].
URL https://www.csee.umbc.edu/~olano/s2002c36/ch02.pdf

[14] Perlin, K.: An image synthesizer. *ACM SIGGRAPH Computer Graphics*, ročník 19, č. 3, 1985: s. 287–296, ISSN 00978930, DOI: 10.1145/325165.325247.

[15] Phong, B. T.: *Illumination of Computer-Generated Images*. Dizertačná práca, University of Utah, UTEC-CSs-73-129, 1973.

[16] Prusinkiewicz, P.; Hammel, M.; Hanan, J.; aj.: *L-systems: from the Theory to Visual Models of Plants* . [Online; navštívené 26.04.2017].
URL http://algorithmicbotany.org/papers/sigcourse.2003/2-1-lsystems.pdf

[17] Reunanen, M.: *ComputerDemos–WhatMakesThemTick?* Dizertačná práca, Aalto University School of Science and Technology, Faculty of Information and Natural Sciences, Helsinki, 2010.

[18] Schroeder, B.: *Modern Warfare 3 Break Down – INFR 2350U Week 1.* [Online; navštívené 19.04.2017].
URL https://brandenschroeder.wordpress.com/tag/particle-systems/

[19] Segal, M.; Akeley, K.: *The OpenGL Graphics System: A Specification (Version 4.5 (Core Profile) - October 24, 2016)*. [Online; navštívené 28.04.2017].
URL https://khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf

[20] Spacerat: *Procedural Terrain Heightmap Generation using DLA (Diffusion Limited Aggregation)*. [Online; navštívené 21.04.2017].
URL http://voxels.blogspot.sk/2014/01/procedural-terrain-heightmap-generation.html

[21] Stosic, D.: *Project Dwarf*. [Online; navštívené 22.04.2017].
URL http://ddstosic.50webs.com/dwarf.htm

[22] Zdrojewska, D.: *Real time rendering of heterogenous fog based on the graphics hardware acceleration*. [Online; navštívené 20.04.2017].
URL http://old.cescg.org/CESCG-2004/web/Zdrojewska-Dorota/

[23] Žára, J.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80–251–0454–0, 609 s.