VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

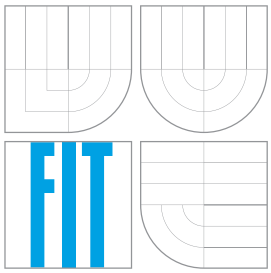
3D HRA V OPENGL S VYUŽITÍM ANALÝZY HUDBY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE HANA BARANOVIČOVÁ
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D HRA V OPENGL S VYUŽITÍM ANALÝZY HUDBY

3D GAME USING OPENGL AND MUSIC ANALYSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

HANA BARANOVIČOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MILET

BRNO 2014

Abstrakt

Tato práce se zaměřuje na analýzu signálu hudby a zakomponování výsledních dat do hry. Cílem je demonstrovat možnosti analýzy hudby, které jsou prakticky využitelné při vytváření hry založené na této analýze, způsoby procedurálního generování grafické reprezentace hudby a využití těchto znalostí na vytvoření hry.

Abstract

This thesis focuses on analysis of music signal and using it in the game. The aim is to demonstrate the possibilities for analysis of music, which are practically applicable in creating games based on the that analysis, procedural ways of generating graphical visualisation of music and use of knowledge to create this type of games.

Klíčová slova

3D, hra, OpenGL, analýza hudby, dolní propust, energie signálu, procedurální generování, Perlinův šum

Keywords

3D, game, OpenGL, music analysis, low pass filter, energy of signal, procedural generation, Perlins noise

Citace

Hana Baranovičová: 3D Hra v OpenGL s využitím analýzy hudby, bakalářská práce, Brno, FIT VUT v Brně, 2014

3D Hra v OpenGL s využitím analýzy hudby

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Hana Baranovičová
21. května 2014

Poděkování

Na tomto místě chci poděkovat vedoucímu mé bakalářské práce, panovi Ing. Tomášovi Miletovi za cenné rady, čas a ochotu při konzultacích.

Obsah

Kapitola 1

Úvod

Spolu s neustálym zlepšovaním technických parametrov a zobrazovacích schopností elektronických zariadení (ako počítače, mobilné telefóny, tablety a herné konzoly) sa zväčšuje počet ľudí, ktorí hrajú elektronické hry. Tým sa vytvára väčší tlak na kvalitu a množstvo hier, ako i nápaditosť hier a rozmanitosť herných žánrov. V súčasnosti sú najznámejšie tieto žánre:

- RPG (Role Playing Game – rolová hra) – *Diablo, Fallout, Elder Scrolls*
- FPS (First Person Shooter – strieľačka z pohľadu prvej osoby) – *Counter strike, Half-Life, Doom*
- Stratégie – *Settlers, Transport Tycoon, The Sims*
- Adventúry – *Myst, Polda*
- Simulátory – *IL-2 Sturmovik, Colin McRae Rally*
- Arkádové hry – *Need For Speed, Grand Theft Auto*

Tlak na nápaditosť hier dáva vzniknúť aj množstvu hier, ktoré sa nedajú presne zaradiť, pretože majú vlastnosti viacerých žánrov. Ako príklad môže poslúžiť hra *Audiosurf*, ktorá podobne, ako táto bakalárska práca využíva hudbu a jej analýzu na vytvorenie hry s dráhou, ktorá je pre každý hudobný záznam jedinečná.

1.1 Hudobné hry

Hlavnou výhodou hier založených na analýze hudby je obrovský potenciál v rozmanitosti výslednej hry. Hra môže byť pomalá, rýchla, náročná, či jednoduchá, závisiac od konkrétnej skladby. Hra je ovplyvnená vlastnosťami hudobného záznamu a vzhľadom na množstvo a rozmanitosť hudby je pravdepodobné, že si každý hráč nájde spôsob hry, ktorý ho bude baviť.

Ďalšou výhodou môže byť aj to, že jedinečnosť hry pre každý hudobný záznam zaručuje dostatočné množstvo verzií hry. Nestáva sa preto po chvíli pre hráča nezaujímavou, čo je jeden z pozitívnych faktorov pre obľúbenosť hry.

V neposlednom rade je plusom aj trvanie jednotlivej hry, ktoré je opäť ovplyvnené dĺžkou zvukovej stopy. Tá je u bežných skladieb zhruba 3 – 4 minúty dlhá, čo je ideálna dĺžka pre krátku oddychovú hru.

V nasledujúcich kapitolách budú bližšie popísané jednotlivé časti, z ktorých sa skladá implementácia hry. Práca je rozdelená do logických celkov. Prvá časť sa venuje formátom hudobných súborov, ich dekódovaniu a následnej analýze signálov. V tejto časti budú priblížené spôsoby analýzy hudby, ktoré ponúkajú dáta vhodné a zaujímavé z hľadiska využitia v hre, ako napríklad energia signálu, alebo pomer výšok a hĺbok v skladbe.

V druhej časti práce budú priblížené jednotlivé techniky generovania prostredia, využité pri vytváraní skyboxu, terénu, dráhy a prekážok, ako je Perlinov šum alebo osvetlenie.

Posledná časť sa bude zaoberať logikou hry. V závere budú načrtnuté ďalšie možnosti vylepšenia hry, hlavne týkajúce sa iných spôsobov analýzy hudby a ich zakomponovania do zobrazovania alebo ovplyvňovania hry.

1.2 Procedurálne generovanie

Charakteristickou črtou hier založených na analýze hudby je nepredvídateľnosť priebehu hry pre rôzne zvukové stopy. Nutným prvkom pri generovaní hry je teda procedurálne generovanie. To v princípe znamená, že z malého množstva dát (vlastnosti zvukovej stopy, ako dynamika, dĺžka a podobne) vygeneruje na základe špecifických algoritmov a procedúr veľké množstvo dát, teda výslednú hru. Vlastnosťou procedurálneho generovania hier je rezultatívnosť, teda vlastnosť, kedy pre jeden konkrétny vstup je vždy výsledkom rovnaká hra.

Kapitola 2

Analýza signálu hudby

2.1 Kódovanie signálu

Najpodstatnejšou časťou vytvárania hudobnej hry je analýza signálu hudby, ktorej sa bude venovať táto kapitola. Samotné spracovanie hry nespočíva iba v analýze signálu a jeho spracovaní. V prvom rade treba dekódovať signál z hudobného súboru.

Hudba sa dá kódovať troma spôsobmi[4]. Hudobný súbor môže obsahovať dáta podobné notovému zápisu. Zástupcom takého kódovania je napríklad formát SMDL (Standard Music Description Language)

Druhou variantou kódovania hudby je kódovanie kontrolné, ktorého najznámejším predstaviteľom je formát MIDI(Musical Instrument Digital Interface). V tomto kódovaní dáta predstavujú informácie, ktoré určujú syntetizátoru, kedy, ako intenzívne, a ako dlho zahrať tón.

Treťou, pre prehrávanie prakticky využiteľnou digitálnou reprezentáciou je digitálny záznam hudby. Dáta takejto reprezentácie predstavujú zmeny v amplitúde zvukových vĺn v čase. Každá dátová vzorka (*sample*) v tomto type kódovania predstavuje hodnotu amplitúdy v určitom čase. Kvalita zaznamenania závisí od počtu vzorkov, ktoré sú zaznamenané v jednej sekunde záznamu, čo sa nazýva vzorkovacia frekvencia (*sampling rate*) . To môže byť napríklad 44,1kHz, čo znamená, že počas jednej sekundy je zaznamenaných 44 100 vzoriek. Čím vyššia vzorovacia frekvencia, tým väčšia kvalita záznamu. Tieto vzorky sú ďalej zakódované rôznymi spôsobmi, aby čo najlepšie vyhoveli požiadavkám na kvalitu záznamu, alebo na kompaktnosť a prenosnosť dát.

2.2 Hudobné formáty kódujúce digitálny záznam hudby

V dnešnej dobe existuje veľké množstvo spôsobov kódovania hudobných súborov. Kódovanie hudobných súborov sa delí na komprimované a nekomprimované. Nekomprimované formáty (napríklad formát WAV – Waveform audio file format) ukladajú zvukové dáta v takom formáte, v akom boli nahrané. Na disku teda zaberajú pomerne veľa miesta. Preto sú rozšírenejšie komprimované formáty, ktoré na disku zaberajú pri porovnateľnej kvalite zvuku menej miesta a posielanie po sieti, alebo kopírovanie na iný disk trvá krátko.

Komprimované formáty sa ďalej delia na stratové a bezstratové formáty. Bezstratové formáty odstraňujú z pôvodných, nekomprimovaných dát iba také množstvo dát, že je vždy možné prekódovať signál do pôvodnej podoby. Ako príklad je možné uviesť formát FLAC(Free Losless Audio Codec), ktorý je zrejme najprenosnejší bezstratový formát. Tento

formát, napriek tomu, že zachováva všetky potrebné informácie pre prekódovanie do pôvodnej, nekomprimovanej verzie, dokáže zmenšiť veľkosť súboru až na 50%-60% pôvodnej veľkosti.

Stratové formáty sú najpoužívanejšie medzi bežnou populáciou, pre malú veľkosť. Bežnými zástupcami sú formáty WMA (Windows Media Audio), RA (Real Audio), Dolby Digital. Z otvorených (*opensource*) formátov je to formát Ogg Vorbis, ktorý vytvorila nezisková inštitúcia Xiph.org Foundation, podobne ako formát FLAC. Najpopulárnejšie formáty sú ale vytvorené podľa štandardu MPEG(Moving Picture Experts Group)[1]. Konkrétnejšie sa jedná o formát AAC(Advanced Audio Coding) definovaný štandardmi MPEG-2 a MPEG-4 a najrozšírejenší formát, MP3 (celé označenie je MPEG-1 alebo MPEG-2 Audio Layer III) ktorý je definovaný v štandardoch MPEG-1 a MPEG-2. Je využívaný pri živom vysielaní hudby a je prehrateľný v takmer všetkých typoch digitálnych hudobných prehrávačoch.

2.3 Dekódovanie signálu z MP3 súboru

Pre rozšírenosť formátu MP3 a zjednodušenie práce program pracuje iba s formátom MPEG. Na samotné dekódovanie sú použiteľné mnohé viac či menej otvorené dekódovacie knižnice ako **LAME**, **FFMPEG**. Práca využíva knižnicu **mpg123** [2] ktorá je súčasťou prenosného konzolového prehrávača a dekódera MPEG-1 a MPEG-2 formátov, vrstiev I-III. Tento program je voľný softvér šírený pod licenciou LGPL 2.1 (Lesser General Public License). Na knižnici **mpg123** je založený napríklad aj **ALSA Player**.

Načítaním hudobného signálu a následným dekódovaním pomocou tejto knižnice vzniknú dáta, ktoré priamo reprezentujú vzorky. Na zakódovanie jednej vzorky bežnej skladby vo formáte mp3 potrebných 16 bitov, vzorky sú kódované znamienkovo. Funkcia `int mpg123_read (mpg123_handle* mh, unsigned char* outmemory, size_t outmemsize, size_t* done)` vracia v premennej`outmemory` pole premenných typu char (8 bitov) o veľkosti `outmemsize`. Z toho vyplýva, že jednu vzorku tvoria dve položky poľa, vzájomne posunuté o 8 bitov a sčítané. Algoritmus:

```
unsigned sample=0;
for(int b=0;b<sample_size;b++)
    sample+=buffer[j+b]<<(8*b);
```

Problém s tým, že vzorky mp3 sú znamienkové a funkcia `mpg123_read(...)` vracia typ unsigned char sa rieši algoritmom 2.3 V premennej `v` je teraz uložená vzorka, ktorá je v

```
short a=0x0000;
a |= sample & 0xffff;
v = (float)(a+32768) / 65536.;
```

rozsahu $< 0, 1 >$. Pre potreby ďalšej analýzy sa ešte upraví na rozsah $< -1, 1 >$.

[1] MPEG je pracovná skupina pri organizáciách ISO (International Standards Organization) a IEC (International Electro-Technical Commision) ktorá sa od r. 1988 zaoberá vývojom štandardov na kódovanie a kompresiu médií.

[2] http://www.mpg123.de/

2.4 Dĺžka skladby

Pre dĺžku trvania hry je nutné z dostupných informácií vypočítať dĺžku hudby v časových jednotkách, vhodné sú sekundy. Táto dĺžka sa dá vypočítať dvoma spôsobmi. Prvý spôsob je založený na princípe vzorkovacej frekvencie 2.1. Správna dĺžka skladby sa dá vypočítať v tomto prípade vzorcom (2.1).

$$dlzka\_skladby = \frac{pocet\_vzoriek}{vzorkovacia\_frekvencia \cdot pocet\_kanalov} \tag{2.1}$$

kde vzorkovacia frekvencia je počet vzoriek za sekundu. Vzorky sa ukladajú pre každý kanál zvlášť, preto je potrebné zohľadňovať aj počet kanálov skladby, teda, či je skladba mono, alebo stereo.

Druhým spôsobom výpočtu je výpočet pomocou bitového toku skladby. Ide o vlastnosť skladby, ktorá určuje, koľko bitov hudobného súboru sa spracuje za jednu sekundu. Mp3 súbor býva zakódovaný s CBR (Constant Bit Rate), čiže konštantným bitovým tokom, s ABR(Average Bit Rate), priemerným bitovým tokom, alebo s VBR (Variable Bit Rate), teda premenlivým bitovým tokom. Veľkosti bytových tokov sú pri MPEG vrstve 3 štandardne v rozsahu 8 kbit/s až 320 kbit/s.[3]

Pri konštantnom bitovom toku, teda CBR, stačí z hlavičky súboru zistiť bitový tok súboru, a z veľkosti hudobného súboru v bitoch sa dá vypočítať dĺžka skladby pomocou vzorca (2.2).

$$dlzka\_skladby = \frac{velkost\_skladby\_v\_B \cdot 8}{bitovy\_tok\_v\_kbps \cdot 1000} \tag{2.2}$$

Veľkosť skladby sa násobí číslom 8, pretože C/C++ načíta bitové súbory po Bytoch. Bitový tok je v hlavičke hudobného súboru zaznamenaný v kbit/s.

Pri premenlivom bitovom toku je výpočet podobný, avšak najprv je potrebné prejsť všetky hlavičky hudobného súboru, zistiť bitový tok každého rámca a z nich vypočítať priemerný bitový tok, ktorý sa dá aplikovať na vzorec (2.2).

Výsledný signál u hudobného súboru vyzerá ako na obrázku 2.1. Detailne vzorky vyzerajú ako na obrázku 2.2

Obrázok 2.1: Ukážka súboru o veľkosti 1 437 696 vzoriek, teda asi 16 s záznamu.

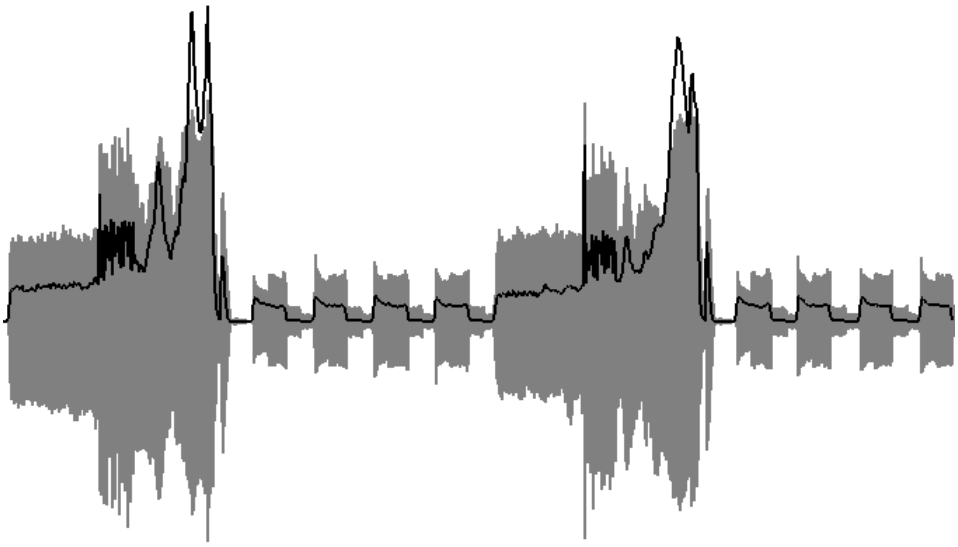
Obrázok 2.2: Detail predchádzajúceho súboru, zobrazených 1 000 vzoriek.

2.5 Energia signálu

Po výpočte dĺžky súboru a získaní vzoriek zo súboru je možné pristúpiť k analýze signálu. Energia signálu je pre výpočet smeru dráhy vhodnou metrikou signálu. Podľa [2] jej výpočet určuje vzorec (2.3),

$$E = \sum_{i=0}^{N-1} |x(i)|^2 \tag{2.3}$$

kde N je v prípade tejto práce vždy počet vzoriek v skladbe delených konštantou 600 (čo je experimentami zvolená konštanta pre vhodný pomer parametrov hladkosť dráhy a náročnosť výpočtu pre rôzne dĺžky skladby). Tento vzorec sa aplikuje postupne na celú skladbu. Výsledkom je funkcia, ktorá vyzerá pre už zobrazovaný signál tak, ako na obrázku 2.3.

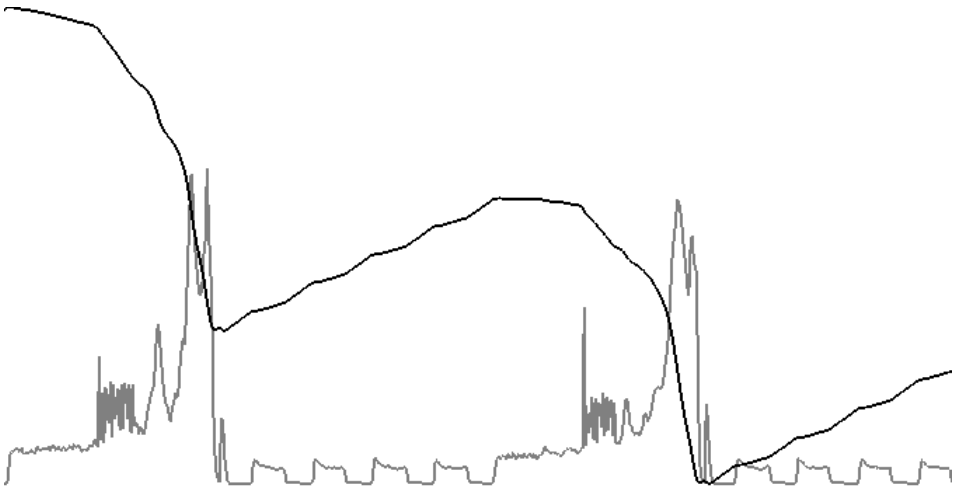


Obrázok 2.3: Energia signálu premietnutá na vykreslené vzorky.

Energia signálu určuje primárne sklon dráhy. Princíp vytvárania je ten, že čím vyšší

je rast energie pre signál, tým intenzívnejšie klesá dráha. Preto je ešte aktuálnu energiu signálu, odčítanú od priemernej energie, nutné pričítať do premennej určenej pre výšku dráhy, ako v kóde 2.5. Výsledok je na obrázku 2.4.

```
float v=0;
float help[1024];
    for(signed i=0;i<count_energies;++i){
        v+=(average_energy-energy_buffer[i])/count_energies;
        help[i]=v;
}
```



Obrázok 2.4: Sklon dráhy vypočítaný zo zobrazenej energie signálu.

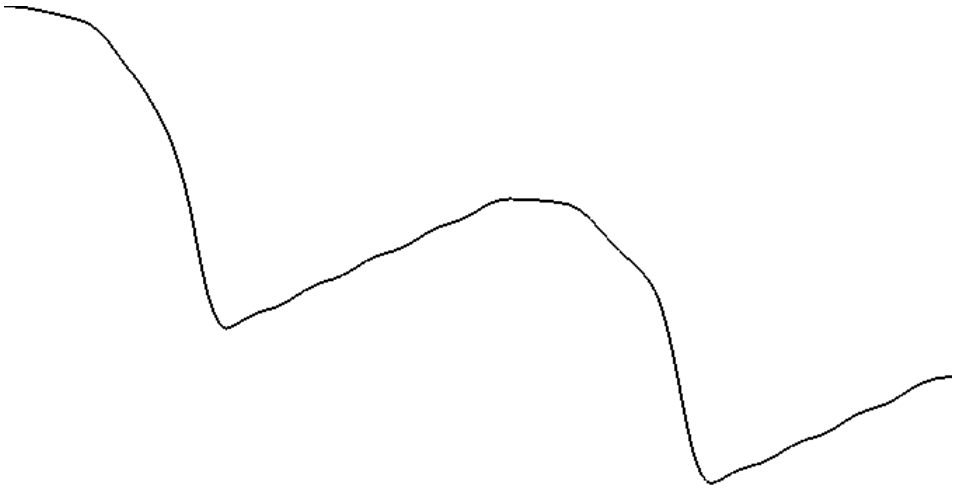
Pri niektorých skladbách sa stáva, že zmena stúpania a klesania je príliš ostrá, výsledný sklon dráhy sa teda ešte vyhladí. Na toto vyhladenie sa použije vyhladzovacie okno, ktorého princíp je, že pre hodnotu v strede okna sa vypočíta priemer zo všetkých hodnôt v okne a nastaví sa ako nová hodnota. Je na to treba využiť pomocné pole pre ukladanie výsledkov, keďže okienko berie aj spätné hodnoty, ktoré už boli vyhladené a teda by to ovplyvňovalo výslednú hodnotu. V bakalárskej práci je použité okno o veľkosti 20 hodnôt. Vyhladzovacie okno je podrobne popísané algoritmom 2.5. Konečný sklon je zobrazený na obrázku 2.6.

2.6 Pomer výšok a hĺbok hudby

Pre výpočet horizontálneho zakrivenia je zvolený pomer energií výšok a hĺbok skladby. Ten určuje, či sa dráha točí vpravo alebo vľavo, aj mieru stáčania v tomto smere. Na zistenie tohoto pomeru je vhodné aplikovať na signál dolnú priepusť, ktorá ideálne prepustí iba frekvencie, ktoré sú nižšie než medzná frekvencia, a frekvencie, ktoré sú vyššie než medzná frekvencia, stlmí. Spôsoby realizácie dolnej priepuste sú mnohé. V princípe záleží na tom, či so signálom pracujeme v časovej doméne (bežné vzorky) alebo v frekvenčnej doméne (frekvencie). Dá sa prechádzať z jednej domény do druhej, pomocou Fourierovej transformácie, prakticky skôr rýchlej Fourierovej transformácie (FFT - Fast Fourier Transformation) [6].

```
for(i=0;i<countOfBuffers;++i){
    for (j=-window_size;j<=window_size;j++){
        if ((i+j)>=0 && (i+j)<countOfBuffers){  //ošetrené zachádzanie za okraj
            counter++;
            average+=input_array[i+j];
            }
        }
        array_out[i]=average/counter;
    }
}
```

Algoritmus 2.5: Finálny sklon dráhy.



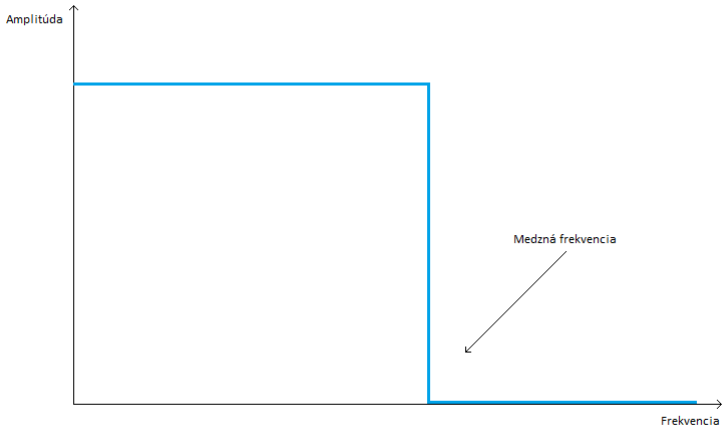
Obrázok 2.6: Finálny sklon dráhy.

Vo frekvenčnom obore sa dá aplikovať na frekvencie ideálna dolná priepusť[11], ktorá je zobrazená na obrázku 2.7 a je popísaná vzorcom 2.4.

$$M(\omega T) = \begin{cases} A & \text{pre } \omega \leq \omega_c \\ 0 & \text{inak} \end{cases} \tag{2.4}$$

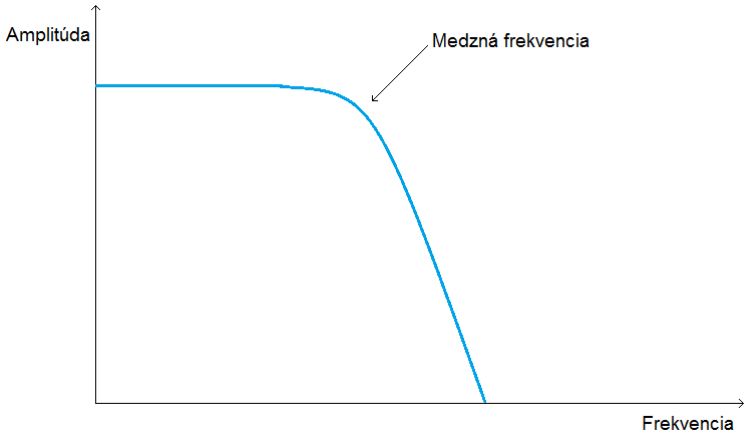
I tu ale nie je v konečnom dôsledku priepusť ideálna. Artefakty v signále v podobe prepustených vyšších a neprepustených nižších frekvencií sú spôsobené prevodom z časovej domény do frekvenčnej pomocou FFT. Ak by bola snaha aplikovať dolnú priepusť vo frekvenčnej doméne, bolo by nutné vykonávať prevod z časovej domény na frekvenčnú a naopak, čo by zdvojnásobilo tieto chyby a boli by to úkony navyš. Preto je v tejto práci snaha aplikovať dolnú priepusť v časovej doméne.

V časovej doméne by sa ideálna dolná priepusť dala matematicky dosiahnuť kovolúciou signálu s jeho impulznou odozvou[11], funkciou *sinc*. V skutočnosti by sa ale ideálna priepusť dala vytvoriť, iba ak by bola konvolúcia nekonečná, preto je tento stav pri súčasných výpočetných možnostiach počítačov prakticky nemožný. Teda sa stáva, že sa niektoré frekvencie nižšie medznej frekvencie skreslia a že filter prepustí aj frekvencie vyššie. Tomuto správaniu sa nedá vyhnúť. Dá sa len čo najviac obmedziť, zúžením prechodovej frekvencie



Obrázok 2.7: Ideálna dolná priepusť.

(šírka, na ktorej je šikmý sklon na obrázku 2.8). Čím je konvolučné okno väčšie, tým je prechodová frekvencia užšia.



Obrázok 2.8: Reálna dolná priepusť.

Dolná priepusť v rejto práci je realizovaná ako FIR (Finite Impulse Response) filter, teda filter konečnej impulznej odozvy, aplikovaný na signál v časovej doméne.

2.7 FIR filter

FIR (Finite Impulse Response) filter, teda filter konečnej impulznej odozvy, je prevedením diskrétnej konvolúcie. V práci je použitá jeho nerekurzívna verzia. Jeho všeobecný vzorec je 2.5, kde $y(n)$ je výsledná hodnota signálu, $x(n-i)$ sú vstupné signály násobené konštantami

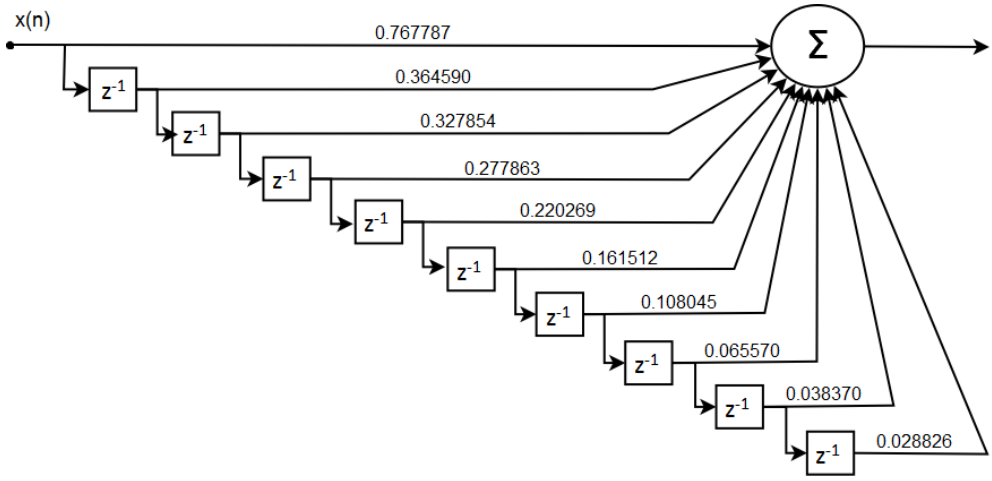
b_i.

$$y(n) = \sum_{i=0}^{N} b_i \cdot x(n-i) \tag{2.5}$$

Pre realizáciu FIR filtra bolo nutné zistiť správne konštanty. Tie sa vypočítali tak, že sa na obdĺžnikový signál (ako na obrázku 2.7) aplikovala rýchla Fourierova tranformácia (aby sme sa dostali do frekvenčnej domény), ktorej výsledok sa potom vynásobil s Hammingovým oknom[11], ktoré je definované vzorcom 2.6.

$$w(n) = 0,54 - 0,46 \cos \frac{2\pi n}{N-1} \quad \text{pre } n \in \langle 0, N-1 \rangle \tag{2.6}$$

Výsledok toho sa potom vrátil naspäť do časovej domény inverznou rýchlou Fourierovou tranformáciou. Prvých 10 koeficientov konečného výsledku predstavuje vhodné konštanty pre aplikáciu FIR filtra na hudobný signál. Tieto konštanty sú zobrazené na obrázku 2.9.



Obrázok 2.9: Nákres FIR filtra s použitými konštantami.

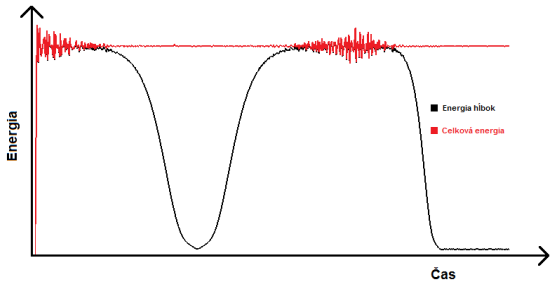
Tento FIR filter je potom aplikovaný na vzorky hudobného signálu, čím získame vzorky, ktoré reprezentujú hĺbky skladby a sú ďalej spracované. Na obrázku 2.10 je zobrazený špeciálny signál), ktorý po aplikácii FIR filtra (čierna farba) a porovnaný s pôvodným signálom (šedá farba) vhodne reprezentuje pomer hĺbok a výšok, obrázok 2.11.

Obrázok 2.10: Pôvodný signál pred aplikáciou FIR filtra.

Obrázok 2.11: Zobrazenie signálu po prevedení FIR filtra.

2.8 Finálne spracovanie signálu

Po aplikácii FIR filtra na signál je možné vypočítať energiu hĺbok skladby, ktorá sa počíta rovnako ako v kapitole 2.5. Pomer celkovej energie a energie hĺbok pre signál z obrázku 2.11 je na obrázku 2.13. Energia signálu hĺbok sa potom znormalizovaná na rozsah $< 0, 1 >$ požije v nasledujúcom algoritme 2.8, ktorý počíta energiu výšok normalizovanú k aktuálnej celkovej energii signálu, čo je zobrazené na obrázku 2.13.

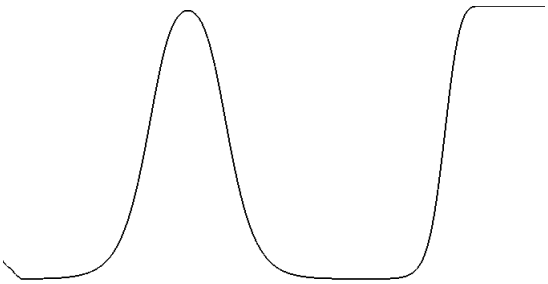


Obrázok 2.12: Zobrazenie celkovej energie a energie hĺbok signálu.

```
for(signed i=0;i<count_energies;++i){
    w=(energy[i]-low_pass_energy[i]+1e-10})/(energy[i]+1e-10);
    low_pass_energy[i]=w;
}
```

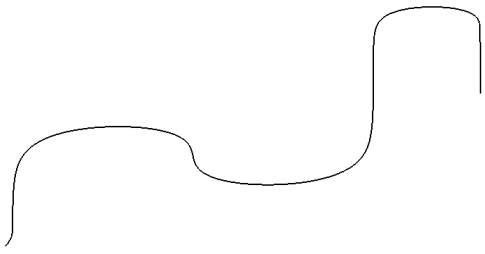
Táto krivka po vyhladení a normalizovaní určuje horizontálne zakrivenie dráhy algoritmom 2.8.

Konštanta π teda 3.14 sa môže nahradiť inou, ktorá priamo ovplyvňuje uhol zatočenia dráhy. Čím vyššia hodnota, tým väčšie zatočenie dráhy. Horizontálne zakrivenie dráhy pre daný signál vyzerá nasledovne 2.14.



Obrázok 2.13: Zobrazenie normalizovanej energie výšok.

```
for(int i=1;i<count_energies;i++){
    x[i]=x[i-1]+(sin((low_pass_energy[i]*3.14)/(float)count_energies);
    z[i]=z[i-1]+(cos((low_pass_energy[i])*3.14)/(float)count_energies);
}
```



Obrázok 2.14: Výsledné horizontálne zakrivenie zobrazené na osiach xy

Kapitola 3

Vizualizácia prostredia

Vizualizácia bakalárskej práce sa skladá z generovania a vykreslovania skyboxu, terénu, dráhy, kruhov a pohybovania sa v teréne. Na vytvorenie okna je použitá knižnica **GLFW 3.0**[1], ktorá sa stará aj o spracovanie ovládania hry. Na vykreslovanie je použité rozhranie **OpenGL**(Open Graphics Library)[2] verzie 3.3 a **GLSL**(OpenGL Shader Lanugage) verzie 330. Z procedurálnych programovacích techník je využitý Perlinov šum.

3.1 OpenGL

OpenGl je podľa [9] definovaný ako sofvérové rozhranie ku grafickému hardvéru. Praktiky je to knižnica pre modelovanie a 3D grafiku, ktorá je vysoko prenosná medzi systémami a zariadeniami a je veľmi rýchla. Je používaná na mnohé účely, od CAD(Computer-aided design) inžinierstva a architektonických aplikácii, až po generovanie počítačových efektov, postáv a strojov vo filmoch. Pevnou súčasťou programovania s knižnicou OpenGL je jazyk GLSL, v ktorom sa vytvárajú shadery, avšak nie je možné pomocou nich vytvoriť celú scénu, preto má programovanie v OpenGL stále svoje pevné miesto pri vytváraní 3D grafiky. Bez neho by sa len ťažko dal nastaviť pohľad, modelovanie, projekčné matice, načítať textúry.

Najhlavnejšou súčasťou OpenGL pre tieto úkony sú VAO (Vertex Array Object), ktoré sú hojne využité aj v tejto práci. VAO sú objekty, ktoré skladujú všetky potrebné informácie o vrcholoch. Neobsahujú priamo vrcholy, ale len odkaz na ich zásobník. VAO nie je špecificky daný, treba ho najprv vytvoriť, nabindovať ho, keď je potreba ho použiť a vymazať až tesne pred koncom aplikácie. Pri vytváraní VAO treba špecifikovať atribút nachádzajúci sa vo VAO, jeho pozíciu, veľkosť a offset. Je totiž možné použiť viac atribútov, ako sú vrcholy, normály, farby a texturovacie koordináty.

3.2 GLSL

Avšak GLSL umožňuje, spolu so súčasným rozvojom grafických kariet, dosiahnuť realistické výsledky v reálnom čase. GLSL je programovací jazyk založený na syntaxi jazyka C, ktorý naviac obsahuje dátové typy a funkcie umožňujúce prácu s vrcholmi, farbami a textúrami. Umožňuje priamo predávať príkazy GPU(graphic processing unit - grafická karta).

[1] http://glfw.org
[2] http://www.opengl.org/

3.3 Shadery

Shader je program, ktorý riadi časti programovateľného grafického reťazca GPU (rendering pipeline). Môže ovplyvňovať vrcholy a fragmenty. Shadery sa dajú spracovávať paralelne a sú veľmi rýchle. Rozlišujeme 3 druhy shaderov:

- Vertex shadery – spracovávajú vrcholy, ich najčastejšia funkcia je transformácia vrcholov, pomocou pohľadovej, modelovej a projekčnej matice.
- Geometry shadery – umožňujú uberať alebo pridávať vrcholy
- Fragment shadery – vytvárajú fragmenty pre všetky pixely týkajúce sa grafického objektu, ktoré obsahujú pozíciu v okne a iné. Určujú výslednú farbu pixelov. Tieto shadery sa aplikujú na renderovaciu pipeline v poradí, postupne.

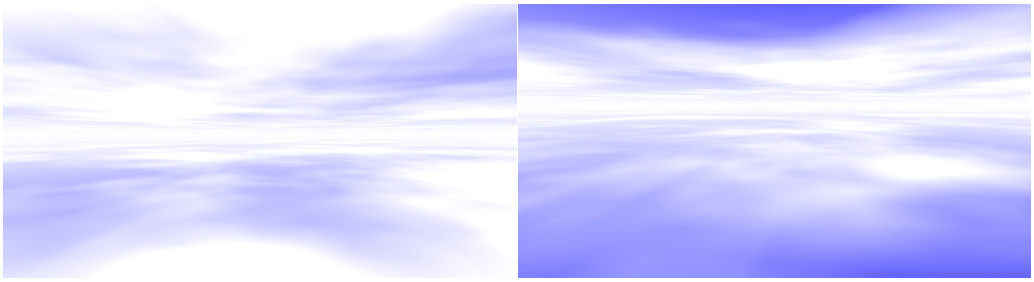
3.4 Skybox

Skybox je metóda vytvárania pozadia pre prostredie (hry), ktoré zabezpečuje pre scénu vizuálne zväčšenie. Nejedná sa o geometriu objektov, ale iba o 2D obrázky, alebo textúry zvyčajne namapované na steny kocky. Kamera je za každých okolností v strede skyboxu a iba rotuje na mieste podľa potreby, čím vzniká ilúzia nekonečnosti prostredia. Textúra mapovaná na skybox môže byť statická aj dynamická, generovaná procedurálne alebo nahraná zo súboru. Alternatívou je skybox v tvare sférickej gule, na ktorej je namapovaná textúra. V prípade mapovania statickej textúry je nutné, aby na seba obrázky mapované na jednotlivé štvorce naväzovali, aby vzniknuté pozadie vyzeralo plynulo a celistvo. Príklad textúry skyboxu je na obrázku 3.1.

Obrázok 3.1: Príklad statickej textúry pre skybox. Prevzaté z [8]

Vyššie technické parametre počítačov umožňujú väčšie výpočty pri vykreslovaní prostredia, preto sa pre väčšiu reálnosť zobrazenia skyboxy nevykreslujú iba staticky, ale je možné textúru vytvárať procedurálne a v reálnom čase jej výzor meniť. Princíp je nasledovný. Pre každý pixel tvoriaci stenu skyboxu sa vytvorí lúč, ktorý vedie od stredu skyboxu a daný pixel pretína. Konečná farba tohto pixelu je potom výsledkom zmiešania farieb všetkých prvkov skyboxu, ktoré pretínajú tento lúč.

Skybox, ktorý je použitý v tejto práci, zobrazuje modrú oblohu s bielymi oblakmi, ktoré sa pohybujú do nekonečna a menia tvar a veľkosť. Premennou, ktorá tento pohyb oblakov ovplyvňuje, je výška dráhy. Skybox z tejto hry je na obrázku 3.2.



Obrázok 3.2: Ukážka samostatného skyboxu použitého v hre a jeho zmena v čase

Oblaky sa generujú pomocou 3D Perlinovho šumu 3.5, kde prvé dve dimenzie tvoria oblaky v 2D prostredí mapované na nekonečnú rovinu a tretia dimenzia je dimenzia času, v tomto prípade ovplyvňovaná dynamikou hudby. Perlinov šum je implementovaný vo fragment shaderi.

3.5 Perlinov šum

Šum je výsledok funkcie, ktorá pre isté hodnoty dáva zdanlivo náhodné výsledky. Jej prínos je v tom, že pre určitý vstup dávy vždy ten istý výstup, preto je odlišná od obyčajného náhodného generovania čísel. Výsledok tejto funkcie, šum, sa dá ďalej využiť na generovanie textúr mnohých prírodných aj umelých povrchov. Kombinovanie šumov do mnohých matematických výrazov vytvára procedurálnu textúru.

Pojem Perlinov šum označuje algoritmus, ktorý vychádza z tvorby Kena Perlina, profesora New-yorkskej Univerzity v roku 1985. Ide o procedurálnu techniku, ktorá je od svojho vzniku využívaná na vytváranie realistického zobrazenia niektorých efektov, najmä prírodných javov ako sú oheň či vodná hladina, či na generovanie textúr.

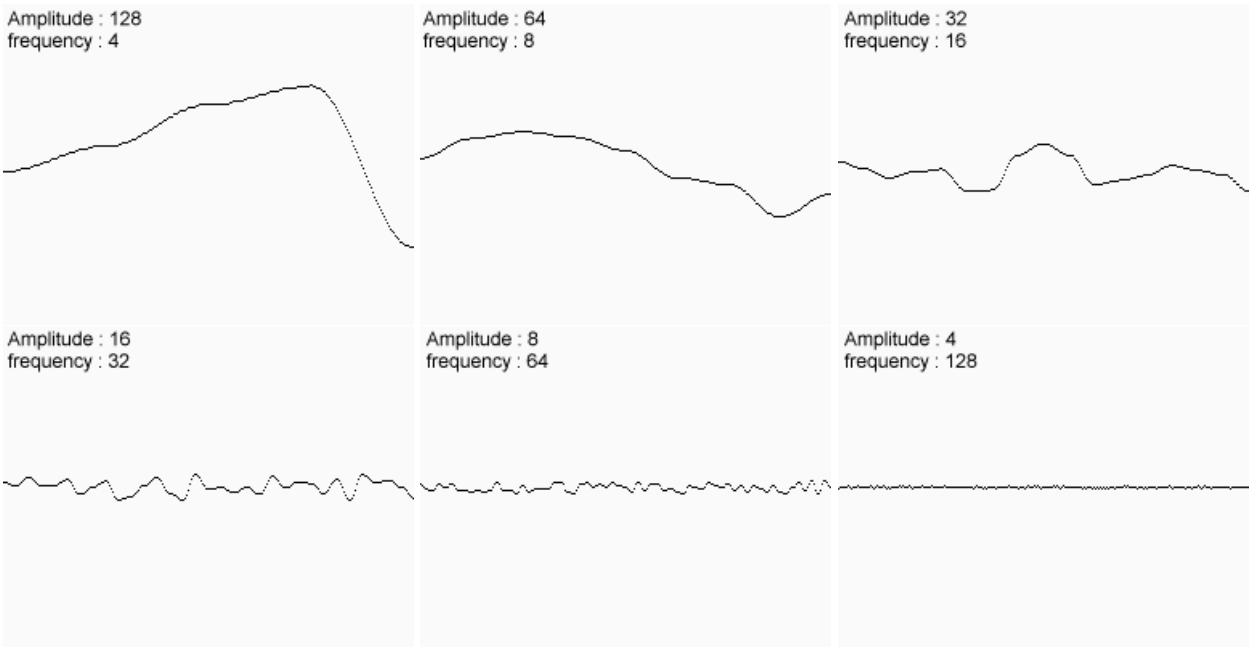
Funkcia Perlinovho šumu pracuje na podobnom princípe ako generátor náhodných čísel. Funkcii sa predá celočíselná hodnota a na jej základe vráti náhodné číslo. Podstatou je, že pre určitý vstup vždy vracia ten istý výstup, čím sa odlišuje od generátora náhodných čísel. Základom je teda deterministický generátor náhodných čísel 3.5.

```
double interpolate1(double a,double b,double x){
  double ft=x * 3.1415927;
  double f=(1.0-cos(ft))* 0.5;
  return a*(1.0-f)+b*f;
 }
```

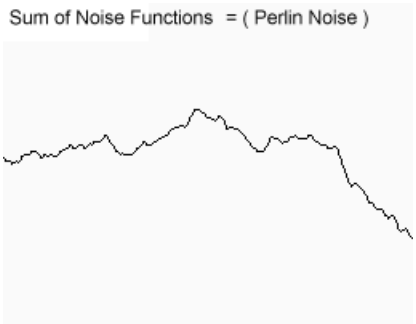
Plynulou interpoláciou medzi vygenerovanými hodnotami môžeme definovať spojitú funkciu. Pri získavaní zašumenej funkcie Perlinovho šumu sa vytvorí niekoľko takýchto funkcií generovaných s rôznymi frekvenciami, amplitúdami, perzistenciami či počtom oktáv. Takto získané funkcie sa následne skladajú. Amplitúda sa získava ako polovica rozdielu maximálnej a minimálnej hodnoty, ktorú môže funkcia nadobúdať. Vzdialenosť týchto bodov

v smere osi x je označovaná ako vlnová dĺžka. Frekvencia je potom definovaná prevrátenou hodnotou vlnovej dĺžky. Počet oktáv zodpovedá počtu skladaných vrstiev. Persistencia je pojem zavedený pre modifikáciu frekvencie a amplitúdy pre každú oktávu.

Výstup funkcie šumu je možné zlepšiť vyhladením. Princípom vyhladzovania je použitie hodnoty získanej spriemerovaním hodnoty funkcie šumu pre danú pozíciu a jej susedných hodnôt namiesto použitia samotného výstupu funkcie pre danú pozíciu. Tento postup možno použiť pre ľubovoľnú dimenziu. Pri použití jednej dimenzie sa šum využíva na generovanie kriviek, pri dvoch dimenziách na vytváranie textúr, pri použití troch dimenzií je možné generovať dynamické textúry či terén. [10]

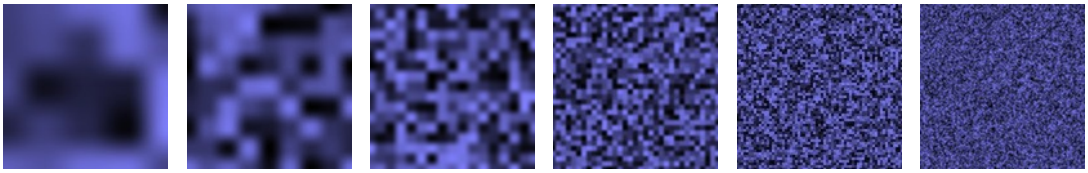


Obrázok 3.3: Ukážky jednorozmerného šumu. Prevzaté z [10]

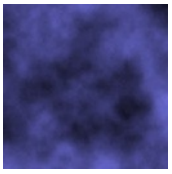


Obrázok 3.4: Výsledný 1D Perlinov šum po poskladaní predchádzajúcich obrázkov. Prevzaté z [10]

Keďže Perlinov šum počíta hodnotu každého pixelu pre danú krivku, textúru či terén zvlášť, je pomerne výpočtovo náročný. Zložitosť tohto algoritmu je $O(2^n)$, pričom zložitosť



Obrázok 3.5: Rôzne funkcie 2D šumu. Prevzaté z [10]



Obrázok 3.6: Výsledný 2D Perlinov šum po poskladaní predchádzajúcich obrázkov. Prevzaté z [10]

algoritmu narastá s rastúcim počtom oktáv. Keďže od istého počtu použitých oktáv sa výsledná funkcia s ďalšími pribúdajúcimi oktávami príliš nemení, je vhodné zvážiť použitie menšieho počtu oktáv pre efektívnejšie generovanie výslednej textúry bez výraznej zmeny vzhľadu.

3.6 Vykreslenie terénu

Terén pre hru je generovaný ako dvojrozmerné pole vrcholov o veľkosti 185×185, kde súradnice x a z vrcholov sú rovnomerne rozmiestnené na rovine. Tvoria teda rovnomernú sieť v tvare štvorca. Súradnica y určuje prevýšenie terénu v danom bode. Jej hodnota je vypočítaná pre každú súradnicu pomocou Perlinovho šumu s amplitúdou 1.5, siedmimi oktávami a perzistenciou 0.4. Na vykreslenie takejto siete vrcholov, ak by nebola použitá indexácia, by bolo nutné skladovať duplicitne niektoré vrcholy a ich vlastnosti, keďže sa vykresluje pomocou trojuholníkov funkciou

```
glDrawElements(GL_TRIANGLES,(XNum-1)*(ZNum-1)*2*3,GL_UNSIGNED_INT,NULL);
```

a ak vezmeme do úvahy obrázok 3.7, štvorec s indexmi 1,2,4,5 by bol vykreslený dvoma trojuholníkmi s indexmi 1,2,5 a 1,5,4, teda už teraz by sa opakovali vrcholy s indexmi 1 a 5. Ak by sme pokračovali ďalej, susedný štvorec s indexmi 2,3,5,6 sa skladá z dvoch trojuholníkov s indexmi 2,3,6 a 2,5,6. Pri vykreslení týchto dvoch štvorcov by bolo potrebné skladovať informácie o bodoch s indexmi 2 a 5 trikrát a bodoch 6 a 1 dvakrát. Toto je veľmi nehospodárne k pamäti počítača, preto je vhodné si uložiť hodnoty každého vrcholu v sieti uložiť len raz a k tomu si vytvoriť pole indexov, v ktorom je uložené, ktorý vrchol sa spája s ktorým, aby vytvorili trojuholníky.

Ďalej je pre každý vrchol potrebné vypočítať jeho normály ktoré sú potrebné pri vyhladzovaní terénu a pri jeho osvetlení. Normály (obrázok 3.8) sú vektory, ktoré sú kolmé na rovinu, ktorá sa dotýka terénu v danom vrchole, ktoré obklopujú daný bod, pre ktorý je normála počítaná.

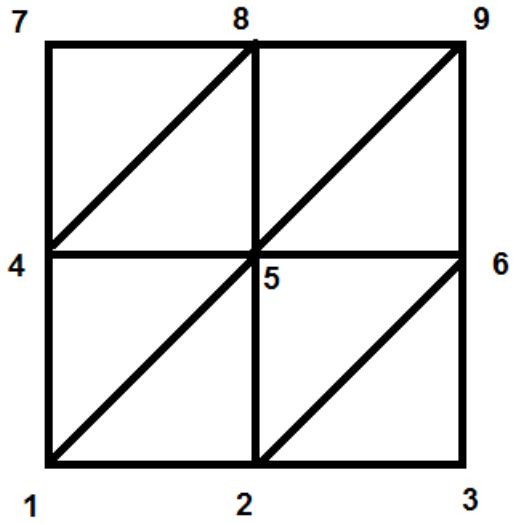
Výsledný terén bez dráhy a namapovanej textúry je na obrázku 3.9.

```
 for(int y=0;y<ZNum-1;++y)
    for(int x=0;x<XNum-1;++x){
        Index[(y*(XNum-1)+x)*6+0]=((x+0))+(y+0)*(XNum);
        Index[(y*(XNum-1)+x)*6+1]=((x+1))+(y+0)*(XNum);
        Index[(y*(XNum-1)+x)*6+2]=((x+0))+(y+1)*(XNum);
        Index[(y*(XNum-1)+x)*6+3]=((x+0))+(y+1)*(XNum);
        Index[(y*(XNum-1)+x)*6+4]=((x+1))+(y+0)*(XNum);
        Index[(y*(XNum-1)+x)*6+5]=((x+1))+(y+1)*(XNum);
}
```

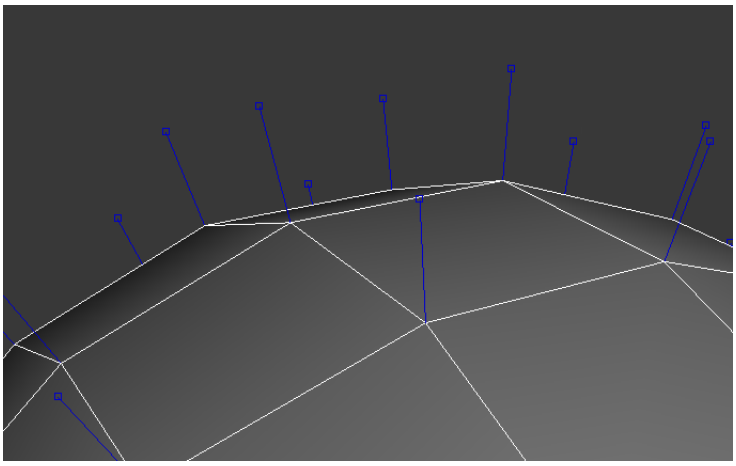
3.7 Vykreslenie dráhy do terénu

Vykreslenie dráhy spočíva v úprave výšky terénu podľa vypočítanej krivky dráhy, kde súradnica y zobrazuje výšku dráhy, ktorá je ovplyvnená energiou signálu a súradnice x a z predstavujú horizontálne zakrivenie, vytvorené pomerom výšok a hĺbok. Táto modifikácia terénu vzniká pomocou pseudokódu 3.10.

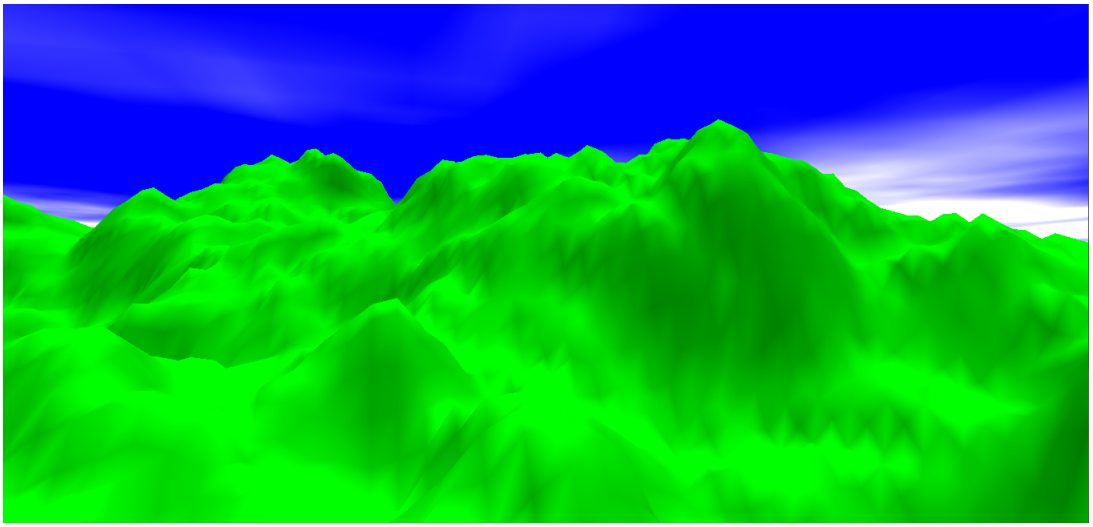
Na terén je namapovaná statická textúra zo súboru, ktorá dodáva celej scéne realistické črty. Výsledok je na obrázku 3.11.



Obrázok 3.7: Indexy vrcholov v sieti terénu.



Obrázok 3.8: Vizuálne zobrazenie normál, prevzaté z [5]

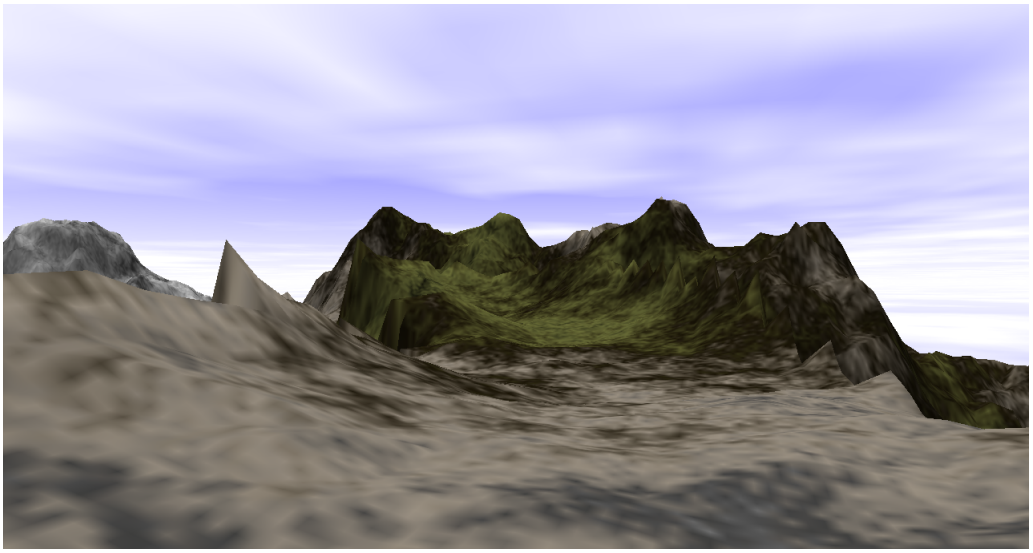


Obrázok 3.9: Ukážka vygenerovaného terénu.

```
for (all points of terrain)
        for (all points of track){
            distance=sqrt(pow(terrain.x-track.x,2)
                        +pow(terrain.z-track.z,2));
                if (distance < max_distance){
//vypocet vah pre blizke body drahy, pre urcity bod
                    near_points[counter]=count;
                    distances[counter]=distance;
                    counter++;
                }

            }
        for(int i=0;i<counter;i++){
            help_distances[i]=(max_distance-distances[i])/max_distance;
            sum_dist+=help_distances[i];
            if (distances[i]<middle_distance){
                help_mid_distances[i]=(middle_distance-distances[i])/middle_distance;
                sum_mid_dist+=help_mid_distances[i];
            }
        }
        for(int i=0;i<counter;i++){
            if (distances[i]>=middle_distance)
                height+=(
//vypocet vysky bodov vzdialenejsich od drahy
                    track.y*(1-distances[i]/max_distance)
                    +track.y*(distances[i]/max_distance))
                    *help_distances[i]/sum_dist;
            else
//vypocet vysky bodov blizkych drahe
                if (Music_values->vector[near_points[i]*3+1]<min_dist){
                    min_dist=track.y;
                    height=track.y;
                }
            }
        terrain.y=height;
    }
}
```

Algoritmus 3.10: Algoritmus na vykreslenie dráhy do terénu.



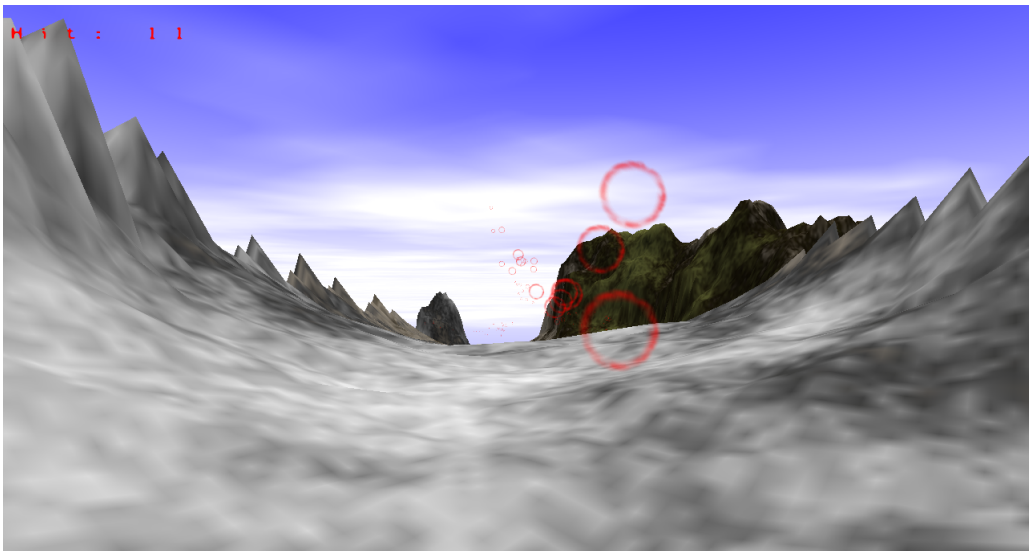
Obrázok 3.11: Vykreslenie scény s textúrou a dráhou.

Kapitola 4

Hra a vizualizácia jej prvkov

Princípom hry je prechádzať čo najväčším počtom kruhov počas prelietania krajinou. Pohyb kamery sa ovláda pohybom myši. Počítadlo sa vykresluje na obrazovku pomocou knižníc prevzatých z [1](nachádzajú sa v zdrojovom kóde v zložke `/2Dtext`) spolu s knižnicami prebratými z [7] (zložka`/tga`), ktoré načítavajú obrázkový súbor s príponou `tga`, keďže knižnica **GLFW** od verzie 3.3 nepodporuje načítavanie obrázkových súborov.

Program začína otvorením dialógového okna pre výber hudobného súboru. Po vybraní nasleduje načítanie obrazovky, ktoré je znázornené aj graficky, pomocou progress baru. Po načítaní nasleduje samotná hra, ktorej ovládanie už bolo popísané. Ukončiť hru je možné v tejto fázy hry stlačením klávesy Escape. Názorná ukážka hry je na obrázku 4.1. Po prejdení celej dráhy sa zobrazí konečná obrazovka, na ktorej je napísaný počet prejdených kruhov, počet všetkých kruhov v hre a percentuálne vyjadrenie úspešnosti. Táto obrazovka časom mení farbu a po piatich sekundách sa hra automaticky vypne. Viď obrázok 4.5.



Obrázok 4.1: Snímka aktuálnej hry

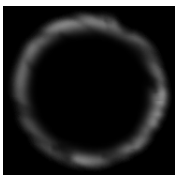
4.1 Vykreslovanie kruhov do terénu

Kruhy, cez ktoré je potrebné prechádzať, aby hráč mohol získať body, sú textúry (obrázok 4.3) namapové na štvorce. Tie sú pozicované na dráhe, kolmo na krivku určujúcu dráhu. Ich poloha je ešte upravená, teda posunutá v rovine kolmej na krivku dráhy. Výsledná poloha uložená v dvojrozmernom poli Vertex je určená algoritmom 4.2, kde je využitý generátor jednorozmerného šumu, zvlášť pre vertikálne a horizontálne posunutie.

```
for(int i=0;i<4;i++){
    Vertex[j][i]=-(pozicia_kamery()+radius*kamera.x*(2.f*(i%2==0)-1)
                  +radius*kamera.y*(2.f*(i>=2)-1));
    Vertex[j][i]+=kamera.x*(float)noise2(120,j,0.4, 0.1,6)*0.07f;
    Vertex[j][i]+=kamera.y*(float)noise2(80,j,0.4, 0.1,6)*0.14f;
        }
```

Algoritmus 4.2: Algoritmus na posun pozície kruhu.

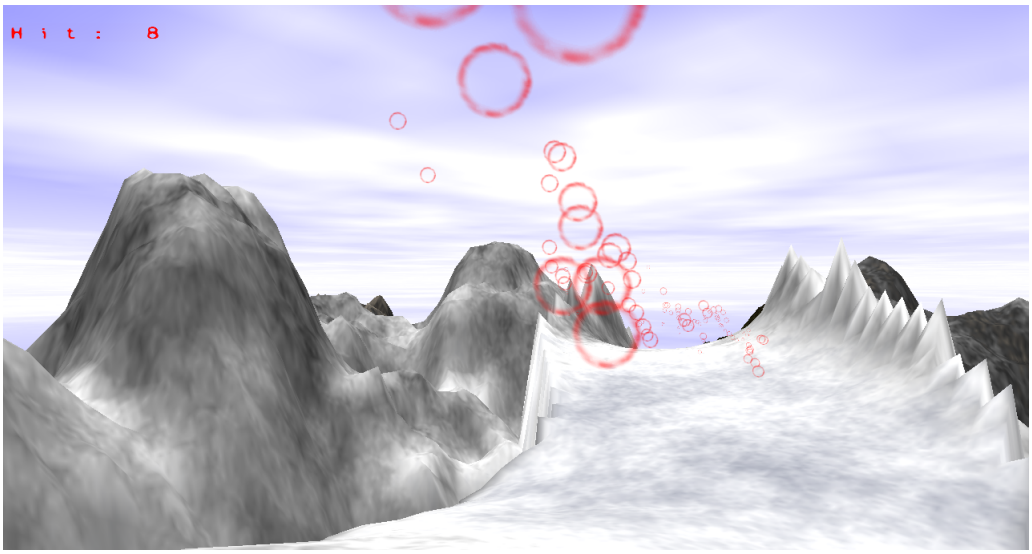
Radius v algoritme 4.2 určuje polomer kruhu, ktorý je rozdielny pre každý kruh a je tiež výsledkom jednorozmerného Perlinovho šumu. Tento polomer sa uchováva ďalej, pre zisťovanie kolízie kamery a kruhu. Kolízia vedie k prejdeniu kruhom, čo zvyšuje skóre. Kolízia kruhu s kamerou je implementovaná ako kolízia bodu s kruhom, kedy sa pri prechádzaní prostredím zisťuje, či je vzdialenosť od stredu najbližšieho kruhu menšia než jeho polomer, a ak hej, došlo ku kolízii a zvyšuje sa skóre. Ku kolízii s rovnakým kruhom už nikdy prísť nemôže. Kruhy vykreslené v prostredí sú znázornené na obrázku 4.4.



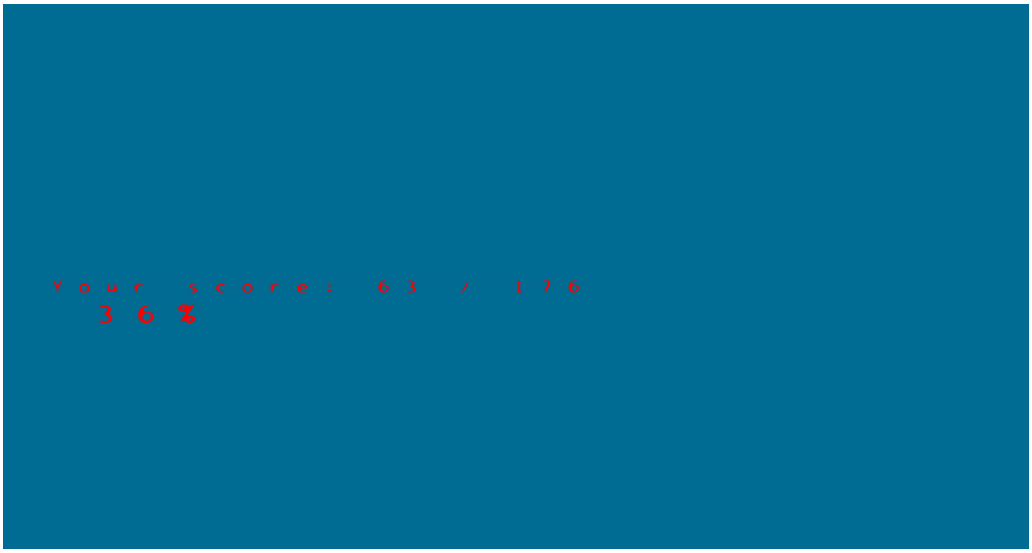
Obrázok 4.3: Textúra kruhu.

4.2 Synchronizácia hudby a hry

Poslednou témou je vplyv hudby a jej parametrov na hru. Dĺžka skladby ovplyvňuje skladbu, keďže dráha je vždy roztiahnutá cez celý terén a teda by mala byť vždy rovnaká, avšak čas jej prejdenia sa vždy odvíja od dĺžky skladby. To vo výsledku určuje to, že pri krátkej skladbe sa preletí dráhou veľmi rýchlo a pri dlhej pomaly. Pri krátkej je teda problém ovládať myšou kameru natoľko aby sa ľahko dalo triafať do kruhov. Na druhú stranu to neznamená, že by boli dlhé skladby príliš ľahké. Od dĺžky skladby sa tiež odvíja počet kruhov na dráhe. to znamená, že na rovnakej dráhe je oveľa viac kruhov, ak je skladba dlhá, než keď je krátka. Vzniká teda takmer pomer medzi obtiažnosťou krátkych a dlhých skladieb. Avšak, každá má svoje pre a proti.



Obrázok 4.4: Snímka aktuálnej hry.



Obrázok 4.5: Obrazovka po ukončení hry so skóre a percentuálnou úspešnosťou.

Kapitola 5

Záver

Táto práca sa zaoberala možnosťami analýzy hudby a spôsobmi získavania vhodných dynamických informácií o hudobnom súbore, ktoré sa dajú využiť pri procedurálnom vytváraní 3D grafickej hry založenej na analýze hudby. Zistila som, že najvhodnejšou vlastnosťou hudby je jej lokálna energia. Samozrejme existuje aj viac vlastností hudby, ako napríklad pomer výšok a hĺbok v skladbe, poloha a intenzita jedotlivých nástrojov, alebo melódia skladby. Pre nedostatok času hra využíva pre svoje generovanie energiu signálu a pomer energií výšok a basov, ktoré sú však použité v rôznych fázach generovania hry.

Hra implementuje dekódovanie a analýzu hudby, konkrétne zistenie energie signálu a aplikáciu dolnej priepuste pomocou FIR filtra na daný signál. Potom sa zaoberá generovaním prostredia, teda dynamického skyboxu, kde je využitý pri generovaní oblakov trojrozmerný Perlinov šum, kde je tretím rozmerom čas, ktorý je modifikovaný premennou, ktorá je rovná výške dráhy. Ďalej terénu, ktorého výška je tiež dielom Perlinovho šumu, ktorý je ďalej modifikovaný tak, aby obsahoval dráhu, ktorá je výsledkom modifikácie energie signálu a energie výšok signálu. Poslednými generovanými objektami sú kruhy, ktorými je treba prechádzať, a ktoré sú implementované ako textúra s alfa zložkou namapovaná na štvorce umiestnené v rámci dráhy a otočené kolmo na smer pohľadu kamery. Zároveň sa rieši aj osvetlenie scény, teda terénu s dráhou, keďže kruhy sú prevedené ako 2D textúra. Ďalej sa rieši automatický pohyb kamery po scéne podľa dráhy a vedomý pohyb kamery ovládaný užívateľom pomocou myši. Princíp hry je vcelku jednoduchý, skóre hráča sa zvyšuje, ak prejde kamerou cez kruh.

Bolo by ešte možné urobiť nejaké vylepšenia, napríklad prispôsobiť hudbe nielen generovanie dráhy, ale aj terénu, využiť viaceré vlastnosti skladby pri generovaní terénu (zvlniť dráhu podľa úderov basov), v menu umožniť užívateľovi ovplyvňovať parametrami obtiažnosť hry, detail vykreslovania či zakrivenie dráhy. Ďalej sa dá vylepšiť vzhľad menu, fyzika cesty dráhou a ovládanie pohľadu, pridať objekt, pre lepšiu vizualizáciu prejazdu dráhou. Je možné pridať ovládanie šípkami na klávesnici pre spokojnosť väčšej časti užívateľov.

Literatura

[1] Tutorial 11 : 2D text [online]. http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-11-2d-text/, 2013 [cit. 2014-05-21].

[2] Andrla, P.: *Segmentace řečového signálu.* bachelor's thesis, Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008.

[3] Brandenburg, K.; Popp, H.: An introduction to MPEG Layer-3. http://www.mp3-tech.org/programmer/docs/trev_283-popp.pdf, June 2000[cit. 2014-05-21].

[4] Brent, L.: A Brief Introduction to Digital Music File Formats. http://www.interpares.org/display_file.cfm?doc=ip2_f1_digital-_music_file_formats.pdf, 2001[cit. 2014-05-21].

[5] Drahoňovský, T.: Smoothing, stínování [online]. http://vertex-cz.blogspot.cz/2011/02/smoothing-stinovani_4972.html, 2011.02.22 [cit. 2014-05-21].

[6] Jan, J.: *Číslicová filtrace, analýza a restaurace signálů.* VUTIUM, 2002, iSBN 80-214-1558-4.

[7] Molofee, J.; Pipho, E.: Loading Compressed And Uncompressed TGA's [online]. http://nehe.gamedev.net/tutorial/loading_compressed_and_uncompressed_tga's/22001/, 2014 [cit. 2014-05-21].

[8] Reijerse, R.: Skyboxes [online]. http://reije081.home.xs4all.nl/skyboxes/, 2006 [cit. 2014-05-21].

[9] Richard S. Wright, J.; Haemel, N.; Sellers, G.; aj.: *OpenGL SuperBible Fifth Edition.* Addison-Wesley, 2010, iSBN 0-32-171261-7.

[10] Turek, M.: Perlin noise [online]. http://farao.czweb.org/perlin.htm , [cit. 2014-05-21].

[11] Vích, R.; Smékal, Z.: *Číslicové filtry.* Academia, 2000, iSBN 80-200-0761-X.

Dodatek A

Plakát