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 — **ZDENKO HANKO**
AUTHOR

VEDOUCÍ PRÁCE — **TOMÁŠ MILET, Ing.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Hanko Zdenko**
Program: Informační technologie
Název: **Grafické intro 64kB s použitím OpenGL**
Graphics Intro 64kB Using OpenGL
Kategorie: Počítačová grafika

Zadání:

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 video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 13. listopadu 2020

Abstrakt

Táto bakalárska práca sa zaoberá problematikou tvorby grafického intra pričom výsledný spustiteľný súbor nepresahuje 64kB. Na jeho vytvorenie sa používa rozhranie OpenGL a rôzne metódy používané pri tvorbe takýchto intier. Výsledkom je grafická scéna zobrazujúca osadu s hradbami. Konečné demo nepresahuje veľkosť 64kB.

Abstract

This bachelor thesis describes the process of making graphical intro, where the executable file on the output is within 64kB. For it's creation an OpenGL is used together with various methods designated for similar problems. The result is a graphical scene, which shows a village surrounded by walls and towers. The final demo is not larger than 64kB.

Klíčová slova

opengl, grafické intro, obmedzená veľkosť, perlinov šum, procedurálne generovanie, skybox, phongov osvetľovací model

Keywords

opengl, graphic intro, limited size, perlin noise, procedural generate, skybox, phong lighting model

Citace

HANKO, Zdenko. *Grafické intro 64kB s použitím OpenGL*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tomáš Milet, Ing.

Grafické intro 64kB s použitím OpenGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Tomáša Mileta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Zdenko Hanko
11. května 2021

Poděkování

Týmto by som sa chcel poďakovať Ing. Tomášovi Miletovi za odbornú pomoc a podporu pri riešení tejto práce kedykoľvek bolo treba. Ďalej sa chcem poďakovať rodine a kamarátom za psychickú podporu.

Obsah

Kapitola 1

Úvod

Bakalárska práca sa zaoberá problematikou tvorby grafického intra s použitím OpenGL, kde výsledné demo nemá veľkosť väčšiu ako 64kB. Jedná sa o grafickú scénu, ktorá vykresľuje svoj obsah v reálnom čase. Grafické intrá majú divákovi poskytovať čo najlepší vizuálny zážitok. Na základe toho existujú aj rôzne súťaže o najlepšie grafické intro. Často býva obmedzená veľkosť výsledného dema a to hlavne na 1, 4 a 64kB.

Práca postupne popisuje techniky využívané pri tvorbe takýchto intier a implementáciu jedného z nich. Zadanie nepopisuje konkrétne ako má výsledné grafické intro vyzerať, čo dáva autorovi voľnú ruku pri tvorbe a zároveň môže použiť svoju fantáziu.

Ako obsah grafickej scény tejto práce som si zvolil grafické zobrazenie ohradenej osady. Kapitola 2 popisuje samotnú grafickú knižnicu OpenGL, na čo nám slúži a čo obsahuje. Technikám na tvorbu grafického intra je venovaná kapitola 3. Postupne popisuje metódy procedurálneho generovania, medzi ktoré patrí aj najviac používaná metóda vo výslednom deme a to perlinov šum, ďalej tu je spracovaná teória vytvárania skyboxu, phongov osvetľovací model, časticové systémy a stručne aj metóda Lindenmayerových systémov. Kapitola 4 popisuje implementáciu výsledného grafického dema a kapitola 5 prezentuje samotný výsledok implementácie ako celok. Samozrejme, ako sa vraví nič nie je dokonalé, tak ďalšie príklady pokračovania práce sú prezentované v kapitole 6. Posledná kapitola 7 zhrňuje výsledné spracovanie práce.

Kapitola 2

OpenGL

OpenGL, skratka pre názov Open Graphics Library, je multiplatformové API, ktoré slúži pre programovanie hlavne 2D a 3D grafických aplikácií. Umožňuje nám pomocou mnohých funkcií pracovať na grafickej karte počítača. Bolo vyvinuté na začiatku 90. rokov firmou Silicon Graphics Inc. Odvtedy už vzniklo niekoľko verzií a najnovšia je OpenGL 4.6, ktorá vznikla v polovici roku 2017. OpenGL patrí medzi najrozšírenejšie grafické API. Umožňuje vývojárom programovať vizuálne pôsobivé softvérové aplikácie. Často sa používa napríklad aj pri vývoji video hier.[11]

2.1 Inicializácia a získanie kontextu

Všetky operácie v OpenGL sa vykonávajú vzhľadom na tzv. kontext, t.j. určitý stav. Toto vyžaduje interakciu s API operačného systemu, popr. správcu okien. Po vytvorení kontextu je nutné dynamicky načítať pamäťové adresy OpenGL funkcii z verzii novších ako 1.1.[7] Z tohto dôvodu existuje viacero knižníc, ktoré abstrahujú a uľahčujú tieto operácie. Umožňuje to podporu viacerých OS a predovšetkým uľahčenie získania kontextu.

2.2 Kontext

Kontext vo všeobecnosti znamená cely stav (state) spojený s danou inštanciou OpenGL a default framebuffer, ktorý reprezentuje okrem iného aj obraz OpenGL programu, ktorý užívateľ vidí na obrazovke.[8] Týchto kontextov môže byť viacero, nejaká operácia sa aplikuje vždy na aktuálny (current) kontext. Kontext je vždy súčasťou nejakého programu, 1 program môže mať viacero kontextov.[8]

2.2.1 Profily

OpenGL do verzie 2.1 poskytoval spätnú kompatibilitu s predchádzajúcimi verziami, čo však viedlo k zapĺňaniu štandardu zastaranými funkciami a bolo potrebné ich vyradiť z novšieho štandardu OpenGL. Na druhej strane bolo potrebné zachovať fungovanie starších programov . Po nie príliš elegantnom pokuse vyriešiť tento problém pomocou rozširenia ARB_compatibility vo verzii 3.0, bol vo verzii 3.2 upravený spôsob vytvárania kontextu. Užívateľovi bolo potom umožnené vybrať si medzi Core a Compatibility profilom.[8] Týmto došlo k rozdeleniu špecifikácií OpenGL na Core a Compatibility varianty. Core špecifikácia označuje funkcie ako zastaralé, ale zatiaľ ich podporuje. Funkcie označené ako zastaralé

v predchádzajúcej verzii zvyčajne odstraňuje a ďalej ich nepodporuje. Compatibility špecifikácia žiadne funkcie neoznačuje ako zastaralé, ani žiadne neodstraňuje. Implementácie OpenGL musia implementovať Core špecifikáciu, ostatné profily nie sú nutné. V praxi výrobcovia grafických kariet implementujú aj Compatibility profil.

2.3 Rozšírenia

Vzhľadom na to, že schopnosti grafických kariet a požiadavky programátorov sa menia a rozširujú, je potrebná "nadstavba"nad samotným OpenGL štandardom. Niektoré rozšírenia sú generické, t.j. nešpecifické pre určitého výrobcu, tieto sa začínajú "GL_EXT".[9] Iné rozšírenia sú označené prefixom, ktorý identifikuje výrobcu, ktorý rozšírenie definuje a implementuje, napr. pre Intel "GL_INTEL". Tieto rozšírenia sú zvyčajne implementované len na hardvéri jedného výrobcu. Generické rozšírenia, ktoré sa ukážu ako užitočné a často používané, môžu byť odsúhlasené ARB (Architecture Review Board).[9] Tieto sa začínajú na "GL_ARB"a často sa stávajú súčasťou štandardu novšej verzie OpenGL.[9]

2.4 Objekty

OpenGL objekty sú konštrukty, ktoré uchovávajú nejaký stav. V prípade, že daný objekt je viazaný na kontext, tento stav je tiež viazaný na daný kontext. Operácie v OpenGL môžu meniť stav kontextu, čo spôsobí zmenu stavu objektu. Inými slovami, objekt v OpenGL je spôsob ako zapuzdriť nejakú skupinu stavov a zmeniť ich jedným volaním funkcie.[10]

2.4.1 Buffer objekty

Buffer objekty sú objekty OpenGL, ktoré ukladajú neformátované dáta v pamäti grafickej karty. Môžu sa použiť na ukladanie vertexových dát, pixelových údajov získaných z obrázkov alebo framebufferov a podobne.[1] Do buffer objektu sa nahrajú dáta a OpenGL treba dať informáciu podla ktorej vie akými dátovými typmi má tieto dáta prezentovať. Delíme ich na "neupravovateľné"(immutable) a upravovateľné (mutable). Tato "neupravovateľnosť"nie je doslovná, nasledovné operácie sú stále povolené, väčšinou preto, lebo ide o operácie, pri ktorých klient (program) neovplyvňuje obsah bufferu priamo, napr. procesy, ktoré sú súčasťou vykresľovacieho reťazca, vyčistenie (clearing), kopírovanie, zneplatnenie (invalidation). Ďalšie operácie, ako napr. mapovanie bufferu na čítanie alebo zápis, môže užívateľ explicitne povoliť.[1]

2.4.2 Query objekty

Query objekty slúžia na asynchrónne získavanie rôznych druhov informácii, napr. počet vzoriek, ktoré úspešne prešli testom hĺbky alebo čas, za ktorý bola vykonaná určitá ohraničená časť OpenGL operácií.[13]

2.4.3 Textúry

Textúry sú objekty, ktoré obsahujú jeden alebo viac obrázkov, ktoré majú rovnaký formát. Sú použiteľné ako zdroj obrazovej informácie pre shadery, aj ako cieľ, do ktorého je zapísovaný vykresľovaný obraz.[17] Môžu byť jednorozmerné, dvojrozmerné alebo trojrozmerné.

2.4.4 Verzie

1.x	Objekty textúr, 3D textúry, nové formáty pixelov, definovanie konceptu ARB rozšírení, kompresia textúr, multisampling, podpora automatického generovania mipmáp, vertex buffer objekt
2.x	Podpora shaderov
3.x	Nové verzie GLSL, nový mechanizmus vytvárania kontextu, profily, pole textúr, uniformný buffer objekt, geometry shadery, dual-source blending
4.x	Nové verzie GLSL, nepriame vykresľovanie, teselácia so shadermi, kompatibilita s OpenGL ES 2.0, 3.0 a 3.1, viacrozmerné polia v shaderoch, debug výstup, čistenie bufferov na určitú hodnotu (podobne ako memset), compute shadery, zneplatnenie (invalidation) textúr, bufferov a framebufferov, definovanie shaderov v jazyku SPIR-V, nový druh kontextu, ktorý nehlási žiadne chyby

[5]

2.5 Shadery

Shadery sú špecializované programy používané v počítačovej grafike, ktoré prijímajú vstupné údaje, spracovávajú ich, a vracajú výstupné údaje.[6] Údaje, ktoré spracovávajú sú zvyčajne vrcholy, pixely a grafické prvky (napr. body, trojuholníky)[12], v prípade compute shaderov môže ísť o akékoľvek údaje.[3]

2.5.1 Typy shaderov

Vertex shader

Vertex shader je prvotná časť vykresľovacieho reťazca, ktorá spracúva vrcholy vstupnej geometrie grafickej scény. Vrcholy spracúva postupne po jednom a až keď spracovaný vrchol vystúpi z programu tak môže vstúpiť ďalší.

Kontrolný teselačný shader

V prípade, že je definovaný vyhodnocovací teselačný shader, určuje množstvo teselácie, ktorá sa bude vykonávať. Tento shader nemusí byť definovaný, v tom prípade sú použité predvolené teselačné hodnoty.[15]

Vyhodnocovací teselačný shader

Tento shader zo vstupnej teselovanej krivky vypočíta hodnotu vrcholov pre ďalšie operácie vo vykresľovacom reťazci.[16]

Geometry shader

Ďalšou časťou vykresľovacieho reťazca je geometry shader, ktorý umožňuje pridávať a odoberať vrcholy, čím je ovplyvnená výsledná geometria grafickej scény. Geometry shader je čiastočne schopný plniť funkciu teselačných shaderov, avšak jeho použitie na tento účel nie je odporúčané.[4]

Fragment shader

Fragment shader, nazývaný aj ako pixel shader, je vykonaný na každom pixeli rasterizovanej scény, čo znamená, že pracuje aj s 2D obrazom. Slúži na aplikáciu textúr alebo aj úpravu výslednej farby.

Compute shader

Compute shader nespadá do vykresľovacieho reťazca a umožňuje spracovanie ľubovoľných dát. Zvyčajne sa nevyužíva na úlohy priamo spojené s vykresľovaním. Compute shadery nemajú žiadne vstupy a výstupy, okrem údajov udávajúcich polohu danej invokácie shaderu v "oblasti" spustenia. Z tohto dôvodu je na získanie vstupných a zápis výstupných údajov potrebné využiť prístup k textúram, čítanie a zápis obrázkov alebo buffer úložiska shaderov (Shader Storage Buffer Object).[3]

2.5.2 Programovanie shaderov

Shadery sú zvyčajne napísané v jazyku GLSL, ktorý je súčasťou štandardu. Od verzie 4.6 je súčasťou štandardu aj jazyk SPIR-V[14]. Vďaka rozšíreniam je možné použiť aj iné jazyky, napr. Cg (C for graphics), ARB assembler alebo NVIDIA assembler. Tieto jazyky majú obmedzenú podporu výrobcov grafických kariet a boli používané predovšetkým pred štandardizáciou GLSL.

2.5.3 GLSL

GLSL, skratka pre názov OpenGL Shading Language, je vysokoúrovňový programovací jazyk podobajúci sa na jazyk C. Je neoddeliteľnou časťou OpenGL od verzie 2.0.[5] Jazyk slúži hlavne na prácu s matematickými operáciami s vektormi a maticami. Dokáže pracovať s vektormi, ktoré majú 2 až 4 hodnoty (**vec2 - vec4**) a takisto s maticami 2x2 až 4x4 (**mat2 - mat4**). Zároveň slúži aj na prácu s textúrami pomocou dátového typu **sampler**. Disponuje bežnými programovacími konštruktmi ako for, while, if a switch. Rovnako ako OpenGL samotné, GLSL má viacero verzií a je z neho odstraňovaná zastaralá funkcionalita. Z tohto dôvodu je nutné na začiatku kódu GLSL shaderu definovať verziu pomocou direktívy #version. V prípade jej absencie je predpokladaná najstaršia verzia GLSL, 1.1, korešpondujúca s OpenGL 2.0. Rovnako ako OpenGL, aj GLSL má systém rozšírení, hoci fungujú na základe iného mechanizmu.

Typy

V porovnaní s jazykom C, GLSL obsahuje aj typy vyššej úrovne na uľahčenie matematických operácií v počítačovej grafike.

Skalárne typy	bool, int, uint, float, double
Vektory	bvec$n$, ivec$n$, uvec$n$, vec$n$, dvec$n$
Matice	mat$n$xm, mat$n$, dmat$n$xm, dmat$n$
Opaque typy	Sampler, Image, Atomic counter

Premenné

Programátor môže v shaderi definovať vlastné premenné. Ďalej sú v shaderi dostupné vopred definované (built-in) premenné. Začínajú sa predponou "gl_". Vstupné a výstupné

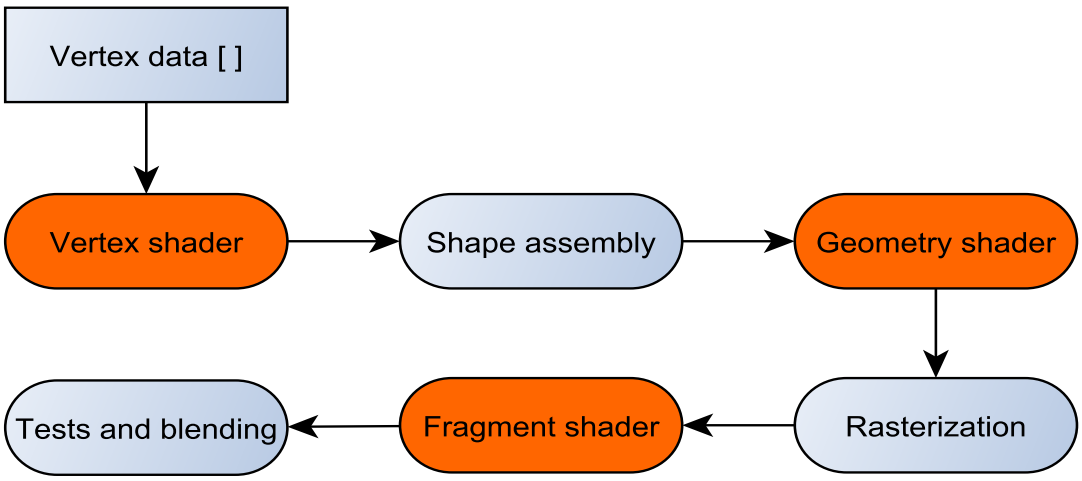
built-in premenné sú závislé od typu shaderu, ďalšie sú rôzne konštanty alebo limity, napr. celočíselná premenná `gl_MaxVertexAttribs` má mať podľa štandardu minimálnu hodnotu 16.[2]

Funkcie

Programovanie GLSL prebieha v takzvaných shaderoch. Shadery sú kompilované za behu programu priamo v ovládačoch grafických kariet, čo výrazne zvyšuje rýchlosť daných operácií. Shadery slúžia na postupné priradzovanie jednotlivých častí vykresľovacieho reťazca, ktorý je znázornený na obrázku 2.1

2.5.4 SPIR-V

SPIR-V je jazyk v binárnom formáte, ktorý môže byť použitý na definovanie shaderov. Na rozdiel od GLSL je jednoduchší. Jeho cieľom je umožniť definovanie shaderov v iných jazykoch, z ktorých sú dané shadery skompilované do SPIR-V.



Obrázek 2.1: Vykresľovací reťazec

Kapitola 3

Techniky upravovania grafickej scény a generovania obsahu

Pre tvorbu tejto práce je nutné používať techniky, ktoré nie sú pre počítač časovo náročné a majú relatívne nízke nároky na výpočtový výkon. Zároveň však musia zachovať dostatočnú kvalitu vzhľadu. S ich pomocou sa dá obmedziť celková veľkosť výsledného súboru, ktorý vykresľuje grafickú scénu, v tomto prípade pod hranicou 64kB. V tejto kapitole sú popísané práve takéto techniky využívané pri tvorbe grafického intra.

3.1 Procedurálne generovanie

Procedurálne generovanie je metóda na vytváranie údajov pomocou počítačových algoritmov namiesto použitia manuálnych operácií. Je to zväčša kombinácia človekom vytvorených algoritmov spojených s počítačovou náhodnosťou. V počítačovej grafike sa používa na generovanie veľkého množstva obsahu (napr.: terén, objekty, textúry,...), ktorý zaberá veľmi málo pamäťového priestoru. Práve pre malú veľkosť je takmer nutnosťou použitie tejto metódy pri tvorbe grafického intra s obmedzenou veľkosťou.

Príkladom použitia procedurálneho generovania sú napríklad efekty ako oheň, materiály ako oblaky a geometria ako terén alebo stromy. Samozrejme všetko má veľa parametrov, ktoré treba nastaviť a podla potreby meniť alebo inak určovať a to prebieha práve algoritmami. Pomocou procedurálneho generovania je vytvorená aj svetoznáma videohra Minecraft, z ktorej ukážku môžeme vidieť na obrázku 3.1.

Obrázek 3.1: Procedurálne generovanie sveta v hre Minecraft od firmy Mojang[22]

Častokrát sú v takýchto programoch použité generátory náhodných čísel na vytvorenie nepredvídateľnosti, prirodzenosti pohybu a správania sa objektov alebo na generovanie textúr. Generátory náhodných čísel majú určite svoje využitie, ale niekedy sa môže ich výstup javiť až neprirodzený. Veľa vecí v prírode je fraktálnych. Majú rôzne úrovne detailov. Bežným príkladom je náčrt pohoria. Obsahuje veľké variácie výšky (hory), stredné variácie (kopce), malé variácie (balvany) a ešte menšie variácie (kamene), podobne sa dá pokračovať aj ďalej. Rovnaký pohľad je na takmer čokoľvek: rozloženie nepravidelnej trávy na poli, vlny v mori, pohyby mravca, pohyb vetiev stromu, mramorové vzory, vetry. Všetky tieto javy vykazujú rovnaké vzorce veľkých a malých variácií. Funkcia Perlinov šum to obnovuje jednoduchým sčítaním hlučných funkcií v rôznych mierkach jak je uvedené v článku Perlin Noise od Huga Eliasa.[20]

3.1.1 Perlinov šum

Perlinov šum je počítačom generovaný šum používaný hlavne ako realistická imitácia náhodnej textúry. Bol vyvinutý v roku 1985 Kenom Perlinom, podla ktorého aj dostal názov. Na vytvorenie Perlinovho šumu musíme najskôr vytvoriť najskôr viac funkcií šumu s rôznymi amplitúdami a rôznou frekvenciou, takzvaných oktáv.

Funkcia šumu

Na vytvorenie funkcie šumu je možné použiť pseudonáhodný generátor čísel, ktorý akceptuje parameter určujúci pozíciu v n-rozmernom priestore. Musí platiť, že pre nejakú hodnotu parametra bude hodnota vygenerovaného čísla vždy rovnaká. Na takýto generátor je najlepšie použiť hashovaciu funkciu.[24] Tento generátor pre každú súradnicu, ktorá je celým násobkom vlnovej dĺžky funkcie šumu, vygeneruje náhodné číslo v rozmedzí amplitúdy funkcie šumu. Hodnoty na súradniciach, ktoré nie sú celými násobkami vlnovej dĺžky sú interpolované.

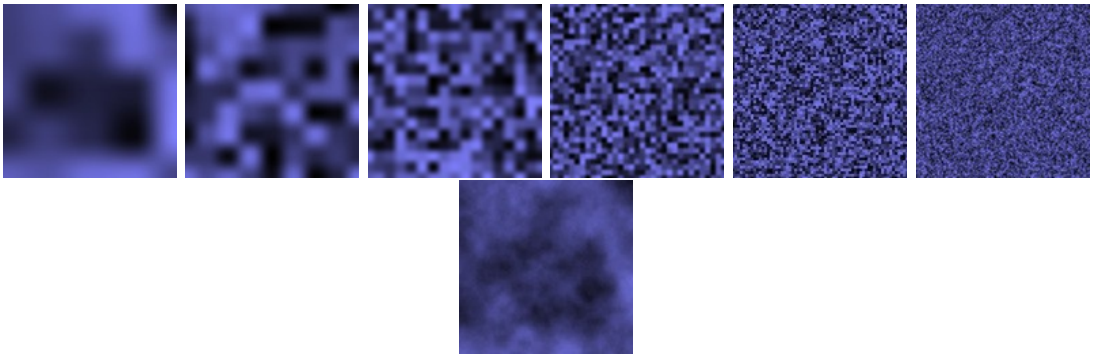
Funkcia Perlinovho šumu

Následne ich sčítaním nám vznikne funkcia Perlinovho šumu. Každá oktáva má dvojnásobnú frekvenciu ako predchádzajúca, čo je znázornené aj obrázkom 3.2. Tento vzťah[23] vyjadruje nasledovná definícia, kde $\vec{x}$ určuje pozíciu v n-rozmernom priestore, pre ktorý vytvárame šum, k určuje počet oktáv, A_i amplitúdu príslušnej oktávy a $Noise(\vec{x}, 2^i)$ hodnotu funkcie šumu s frekvenciou 2^i.

$$Perlin(\vec{x}) = \sum_{i=0}^{k} A_i \cdot Noise(\vec{x}, 2^i)$$

Hoci si v zásade môžeme zvoliť akékoľvek amplitúdy u funkcií šumu, zvyčajne sa na výpočet amplitúdy šumu jednotlivých oktáv používa takzvaná perzistencia, ktorá ovplyvňuje rýchlosť jej klesania pri prechode jednotlivými oktávami. Perzistencia sa pohybuje medzi hodnotami v intervale (0,1>. Čím je perzistencia menšia tým je šum hladší. Pri použití perzistencie sa frekvencia a amplitúda vypočíta nasledovne:

$$f_i = 2^i$$
$$A_i = p^i$$



Obrázek 3.2: Niekoľko funkcií šumu v 2D priestore spojených do Perlinovho šumu[20]

kde i je poradové číslo funkcie šumu, f_i je frekvencia, p je hodnota perzistencie a A_i je amplitúda funkcie šumu.

Interpolácia

Na vyhladenie jednotlivých hodnôt šumu sa používa interpolácia. Štandardná funkcia interpolácie berie na vstupe 3 hodnoty a to hodnotu **a** a hodnotu **b**, medzi ktorými interpolácia nastane a hodnotu **x**, ktorá sa nachádza medzi hodnotami 0 a 1. Keď sa **x** rovná 0, vráti **a**, a keď **x** je 1, vráti **b**. Ak je **x** medzi 0 a 1, vracia nejakú hodnotu medzi **a** a **b**.[20]

Jednou z najjednoduchších metód je lineárna interpolácia. Funguje na princípe vkladania úsečiek medzi jednotlivé body. Je popísaná rovnicou 3.1.

$$y = a \cdot (x - 1) + b \cdot x \tag{3.1}$$

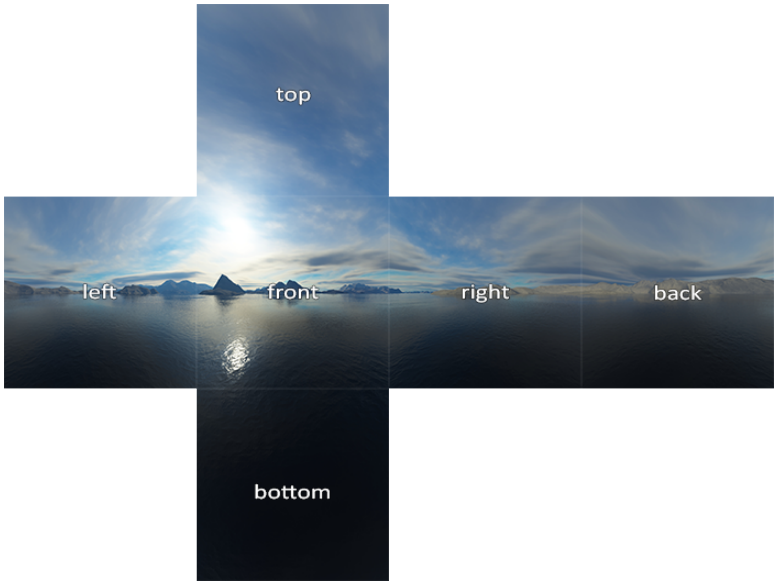
Ďalšou metódou je kosínusová interpolácia, ktorá poskytuje omnoho hladšiu krivku oproti lineárnej interpolácií. Je popísaná rovnicou 3.2.[20]

$$\begin{aligned} f &= \frac{1 - cos(x \cdot \pi)}{2} \\ y &= a \cdot (1 - f) + b \cdot f \end{aligned} \tag{3.2}$$

3.2 Skybox

Skybox je metóda, ktorá vytvára pre diváka na scéne okolie, ako napríklad oblohu, hory v diaľke a podobne. Väčšinou robí danú scénu na pohľad omnoho väčšiu ako v skutočnosti je. Princíp je jednoduchý. Skybox je väčšinou kocka uprostred ktorej sa odohráva celá scéna. Na tejto kocke sú nanesené rôzne textúry pohľadov diváka, pričom je dôležité aby na seba jednotlivé strany kocky nadväzovali. Zároveň musia byť prepojené tak aby nebolo poznať, že je to celé osadené do hranatej kocky. Takéto textúry sa nazývajú aj cube-maps. Príklad textúry skyboxu je znázornený na obrázku 3.3. V prípade, že vzdialenosť skyboxu voči divákovi je konštantná, je vytvorený dojem, že predmety v skyboxe sú vzdialené nekonečne ďaleko. Takéto chovanie nie je vždy žiadúce, hlavne ak sa v skyboxe nachádzajú predmety známej veľkosti. Preto sa niekedy používa metóda, kde sa skybox hýbe spolu s divákom,

ale zlomkom jeho rýchlosti. V závislosti od tohto zlomku je možné navodiť pocit rôznych vzdialenosti.



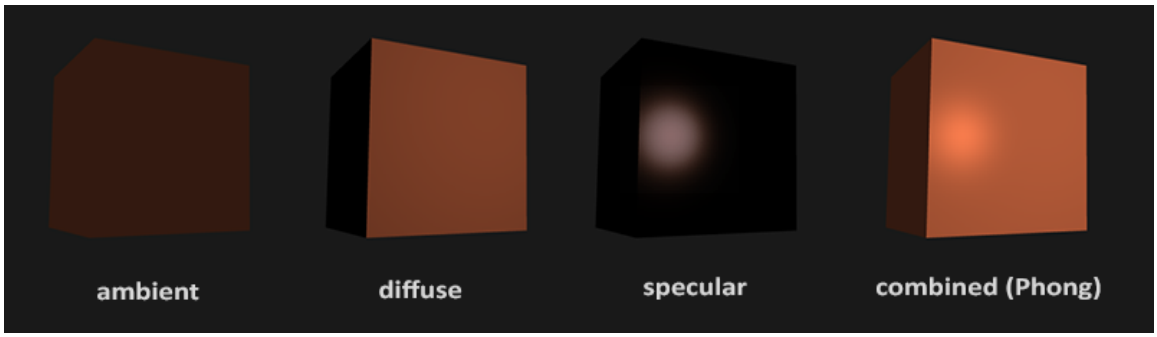
Obrázek 3.3: Textúra skyboxu, tzv. cube-map [27]

Modernejšou metódou skyboxu tzv. skydome, teda nanášanie textúry okolia namiesto kocky na guľu. Výhodou je možnosť nanášania animovaných textúr avšak z dôvodu väčšieho počtu polygónov je vyššia aj výpočetná náročnosť pri zobrazovaní.

3.3 Phongov osvetľovací model

Osvetlenie v reálnom svete je mimoriadne zložité a závisí od príliš veľkého množstva faktorov, ktoré si nemôžeme dovoliť počítať s obmedzeným výpočtovým výkonom, ktorý máme. Osvetlenie v OpenGL je preto založené na aproximácii reality pomocou zjednodušených modelov, ktoré sú oveľa ľahšie spracovateľné a vyzerajú relatívne podobne. Tieto svetelné modely sú založené na fyzike svetla, ako ju chápeme. Jeden z týchto modelov sa nazýva Phongov osvetľovací model.[26]

Tento model vzniká kombináciou troch zložiek a to: ambientná zložka, difúzna zložka a spekulárna zložka.



Obrázek 3.4: Zložky Phongovho osvetľovacieho modelu [26]

Vzťah na výpočet Phongovho osvetľovacieho modelu potom vyzerá nasledovne:

$$I \quad = I_A + I_D + I_S \tag{3.3}$$

,kde I znázorňuje jednotku svietivosti, I_A predstavuje ambientnú zložku, I_D difúznu zložku a I_S spekulárnu zložku svetla.

Ambientná zložka

Vždy používa malú konštantnú (svetlú) farbu, ktorú pridávame k výslednej farbe fragmentov objektu, takže to vyzerá tak, že vždy existuje nejaké rozptýlené svetlo, aj keď neexistuje priamy zdroj svetla. Pridanie tejto zložky svetla na scénu je skutočne jednoduché.[26] Vezmeme farbu svetla, tá je vynásobená malým konštantným faktorom, ktorý určuje intenzitu ambientnej zložky, následne je to vynásobené farbou objektu a výsledok je použitý ako farbu fragmentu v shadere objektu. Tento vzťah je popísaný rovnicou 3.4

$$FragColor = (LightColor \cdot K) \cdot ObjectColor \tag{3.4}$$

Difúzna zložka

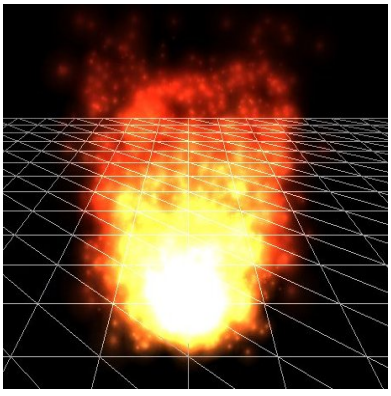
Difúzna zložka dáva objektu tým viac jasu, čím sú jeho fragmenty vhodnejšie otočené k svetelným lúčom zo zdroja svetla. To znamená, že čím menší uhol zdieľa normála plochy objektu so svetelným lúčom, tým je plocha jasnejšia. [26]

Spekulárna zložka

Spekulárna zložka je viac naklonená farbe svetla ako farbe objektu. Jej intenzita je určená odrazom svetla smerom od zdroja svetla k pozorovateľovi. Na rozdiel od predchádzajúcich zložiek svetla, táto zložka je závislá aj od polohy pozorovateľa. [26]

3.4 Časticové systémy

Častica sú malé prvky, ktoré sú vždy natočené smerom do kamery (táto technika sa nazýva billboarding) a väčšinou obsahujú textúru, ktorej veľká časť je priehľadná. Časticové systémy pracujú s veľkým množstvom častíc, ktoré vytvára v takzvanom časticovom emitore (generátore). Každá častica má pridelené prvky definované práve týmto časticovým systémom ako napríklad poloha, smer, rýchlosť, farba, tvar a podobne. Táto technika umožňuje vytvoriť veľmi zaujímavé efekty ako napríklad horiaci oheň, dym, dážď a podobne. Práve preto sú časticové systémy často používané v grafických intrách.[28]

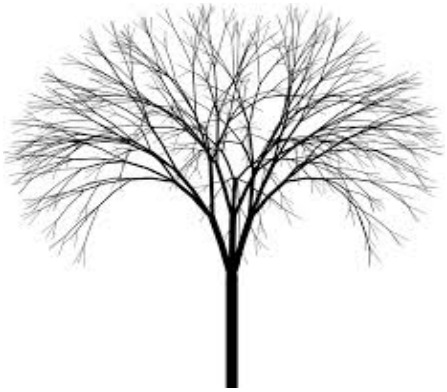


Obrázek 3.5: Ukážka efektu ohňa pomocou časticových systémov [28]

3.5 L-systémy

L-systémy, celým názvom Lindenmayerove systémy, sú skupinou fraktálou definovaných vo svojej najjednoduchšej podobe pomocou regulárnych alebo bezkontextových prepisovacích gramatík. Podstatou tvorby tých najpoužívanejších L-systémov je prepisovanie reťazcov podla určitých pravidiel, ktoré sú buď dopredu zadanou množinou pravidiel, teda gramatikou, alebo sa menia popri generovaní fraktálneho obrazca. Každý symbol v reťazci má určitý geometrický význam, napríklad transformáciu alebo vygenerovanie objektu.[25]

S pomocou L-systémov sa dajú generovať aj fraktálne objekty, ktoré sa podobajú rastlinám, stromom alebo podobným prírodným útvarom. Na tento účel sa aj často používajú práve v grafickom intre.[25]



Obrázek 3.6: Strom vygenerovaný pomocou L-systémov [18]

Kapitola 4

Implementácia

Táto kapitola popisuje jednotlivé prvky a objekty generované vo výslednom grafickom intre a zároveň aj popisuje použité knižnice.

4.1 Použité knižnice

Na implementáciu len samotné OpenGL nestačí a práca sa dá výrazne uľahčiť použitím rôznych knižníc. V tejto práci boli využité nasledovné knižnice:

WinAPI

WinAPI, celým názvom Windows API, je aplikačné rozhranie, ktoré je súčasťou operačného systému Microsoft Windows a tým sa ušetrí miesto v pamäti. Slúži na tvorbu a riadenie aplikačných okien a poskytuje aj riadenie ďalších prvkov ako napríklad vstup klávesnice a podobne. V tejto práci je využité práve na vytvorenie okna, do ktorého sa zobrazí grafický obsah.

Glad

Knižnica Glad slúži k načítaniu OpenGL funkcií počas priebehu programu. V čase prekladu nie je známa poloha funkcií a glad ukladá ukazatele na tieto funkcie pre neskoršie použitie. Táto knižnica slúži na zrýchlenie procesu programovania, nakoľko nahrádza manuálne načítanie jednotlivých OpenGL funkcií napríklad pomocou **wglGetProcAddress**, ktorý je nutné použiť pre každú funkciu samostatne.

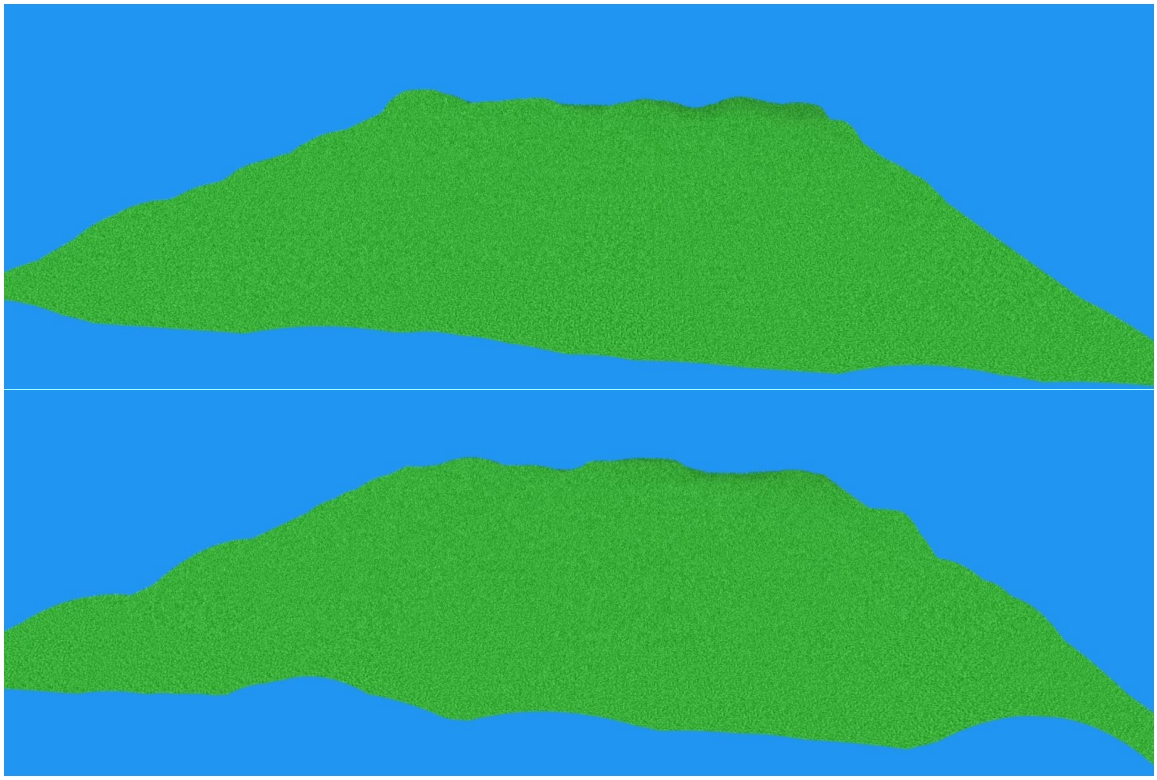
GLM

GLM, celým názvom OpenGL Mathematics, je knižnica pre prácu s matematikou. Je navrhnutá v jazyku C++ a je určená práve pre grafické programy využívajúce OpenGL a GLSL. Umožňuje pracovať s rovnakými dátovými typmi ako v GLSL. Napríklad práca s maticami **glm::mat2()-glm::mat4()** alebo s vektormi **glm::vec2()-glm::vec4()**. Umožňuje napríklad počítať maticové transformácie alebo generovať šum a mnoho ďalšieho.

4.2 Terén

Terén je tvorený pomocou náhodne generovanej výškovej mapy nanesenej na vygenerovanú 3D mriežku. Princíp je jednoduchý. Na začiatku je plochý terén a na ňom sa náhodne vyberie miesto (teda náhodný riadok / stĺpec), ktoré prezentuje vrchol vygenerovaného kopca. Vygeneruje sa náhodný polomer a náhodná výška tohto kopca a tým je kopec hotový. Toto sa zopakuje niekoľko krát a tým vznikne kopcovitý terén. Algoritmus má vopred určené intervaly jak polomeru tak aj výšky kopca, z ktorých náhodne čerpá dáta a takisto je vopred určený počet vzniknutých kopcov v teréne. Táto technika generovania náhodného terénu robí danú grafickú scénu každým spustením trochu inú, čo môže zaujať diváka.

Na vygenerovaný terén je následne aplikovaná procedurálne generovaná textúra za pomoci perlinovho šumu, ktorá kombinuje dva rôzne odtiene zelenej farby a tým vytvára efekt trávnatého povrchu.

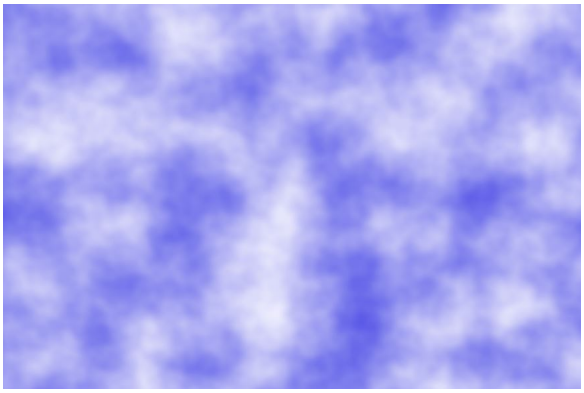


Obrázek 4.1: Ukážka dvoch vygenerovaných terénov s procedurálne generovanou textúrou

4.3 Obloha

Na vytvorenie efektu oblohy bol použitý skybox, teda kocka ktorá obkolesuje priestor kde sa odohráva zvyšok grafickej scény. Postup je pomerne jednoduchý. Prvotne treba vygenerovať kocku, ktorej stred sa nachádza uprostred grafickej scény a následne pomocou **glm::scale** výnásobiť maticu tejto kocky vhodnými vektormi tak aby veľkosťou obkolesovala celú grafickú scénu.

Na túto kocku je následne nanesená procedurálne generovaná periodická textúra vďaka čomu není poznať hrany kocky. Textúra je generovaná pomocou perlinovho šumu s použitím dvoch farieb a to modrej a bielej.



Obrázek 4.2: Textúra oblohy generovaná za pomoci perlinovho šumu

4.4 Budovy a stromy

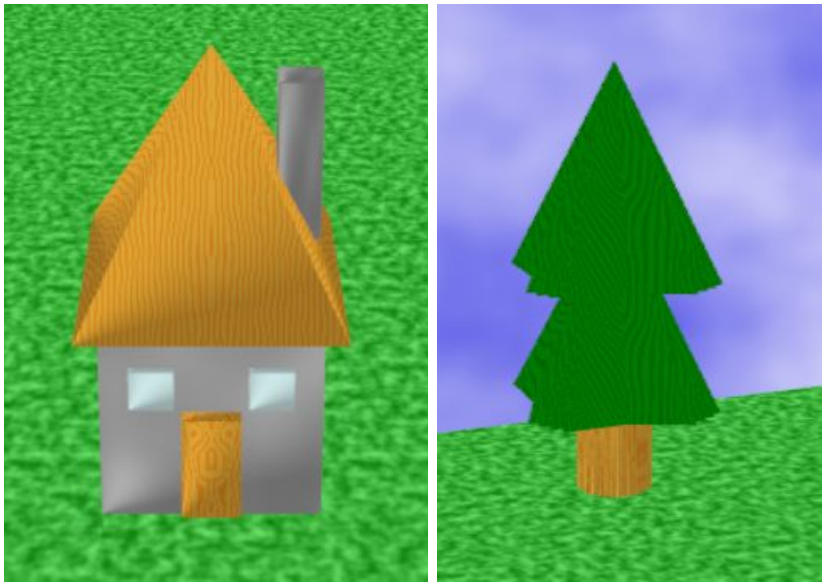
Všetky objekty vložené na povrchu terénu grafickej scény majú vopred pevne stanovené svoje súradnice.

Domy a stromy

Objekty domu sú zložené z jednoduchých geometrických tvarov. Spodná časť domu je kocka s nanesenou textúrou perlinovho šumu imitujúca betónový múr, ktorý tvorí steny domu. Na túto kocku je položený ihlan, na ktorý je taktiež nanesená vygenerovaná textúra upravená tak aby imitovala drevenú strechu. Na strechu je umiestnený kváder predstavujúci komín. Okná a dvere domu sú vhodne umiestnené kvádre s textúrou. Ukážka domu je zobrazená na obrázku 4.3.

Stromy sú vytvorené nezvyčajným spôsobom. Na kmeň stromu je použitých niekoľko kociek, ktoré majú rovnaké súradnice, len je každá otočená pod iným uhlom. Vďaka tomu má kmeň určitú 3D členitosť a není to len hladký valec. Na zafarbenie kmeňu je použitá rovnaká textúra ako pri streche na dome, čo môže vytvárať dojem, že práve z týchto stromov je použité drevo na dome. Zvyšok stromu je generovaný podobne ako kmeň, iba kocky nahradili ihlany v menšom množstve a tým je členitosť stromu výraznejšia. Ako textúra bol použitý perlinov šum kombinujúci dve tmavozelené farby. Ukážka stromu je na obrázku 4.3.

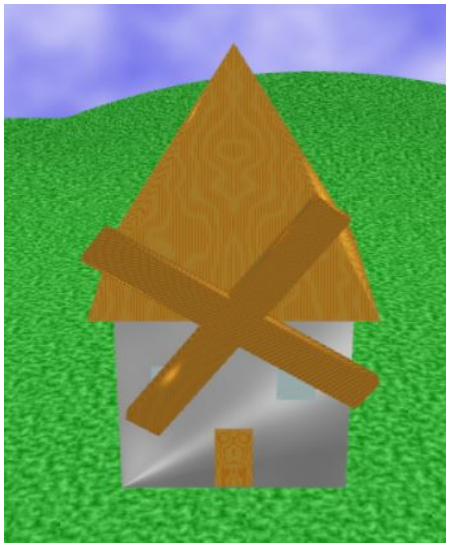
Objekty domov aj stromov sú v teréne umiestnené tak, že výška ich umiestnenia je automaticky prispôsobená výške terénu.



Obrázek 4.3: Ukážka domu a stromu

Mlyn

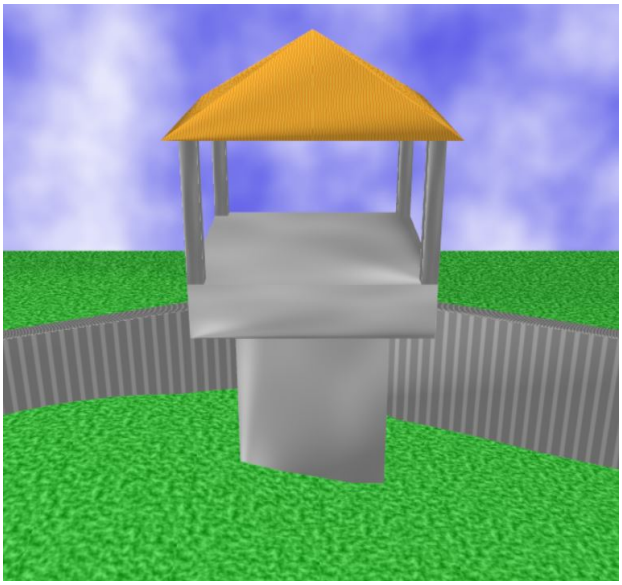
Objekt mlynu je vygenerovaný podobne ako dom, len zväčšený s pridanou vrtuľou. Tá je tvorená dvomi na seba kolmými kvádrami. Textúra je použitá podobná ako pri streche len s kombináciou iných farieb. Efekt točiacej sa vrtule je tvorený pomocou príkazu **glm::rotate** s použitím kosínusu času. Vďaka tomu sa vrtuľa občas točí rýchlejšie a občas pomalšie. Objekt mlynu podobne ako domy aj stromy je automaticky prispôsobený výške terénu. Mlyn je znázornený na obrázku 4.4.



Obrázek 4.4: Ukážka mlynu

Veže a hradby

Poslednými objektami na scénu sú veže a hradby, ktoré obklopujú osadu. Veže sú generované pomocou geometrických útvarov skladaných na seba podobným princípom ako spomínané domy a mlyn. Pri každej veži sa nachádzajú spoje hradieb. Otvor dovnútra osady nakoniec dotvára vstupná brána tvorená dvoma prepojenými vežami. Na obrázku 4.5 je vidieť, že veže a hradby ako jediné objekty na scéne sa neprispôsobujú výške terénu. Terén je zároveň nastavený tak aby nikdy nemal väčšiu výšku ako hradby.



Obrázek 4.5: Ukážka veže s hradbami

4.5 Kamera

Pohyb kamery je pomerne jednoduchý a delí sa na dva typy. Na začiatku kamera prechádza stredom grafickej scény len po y-ovej súradnici, pričom v strede sa pretočí a pozerá stále na stred scény. Po prechode na druhú stranu sa pohyb zmení a kamera sa točí okolo scény po vopred stanovenom radiuse naspäť na kde skončil prvý typ pohyb a odtiaľ sa vráti na štartovaciu polohu. Tento pohyb je rozdelený na základe času, ktorý ak je menší π tak prebieha prvý typ. Ak je väčší ako π a zároveň menší ako 3π tak prebieha druhý pohyb a ak je väčší ako 3π tak prebehne opäť prvý typ pohybu a následne sa čas resetuje. Pri pohybe kamery sa mení len jej pozícia na základe x-ovej a y-ovej súradnice. Výpočet prvého typu pohybu je znázornený vzťahom 4.1

$$\begin{aligned} x &= \sin(time) \cdot \cos(radius) \\ y &= \cos(time) \cdot radius \end{aligned} \tag{4.1}$$

a výpočet druhého pohybu vzťahom 4.2

$$\begin{aligned} x &= \sin(time) \cdot radius \\ y &= \cos(time) \cdot radius \end{aligned} \tag{4.2}$$

4.6 Obmedzenie veľkosti

Veľkosť výsledného spustiteľného súboru tejto práce je limitovaná na maximálne 64kB. Z toho dôvodu je takmer nevyhnutné použiť externý exe-packer, ktorý vykoná bezstratovú kompresiu dát a tým zmenší výslednú veľkosť spustiteľného súboru na minimum. Pre túto prácu je použitý voľne dostupný exe-packer UPX, ktorý vytvorili autori Markus F.X.J. Oberhumer, László Molnár a John F. Reiser. Výsledný spustiteľný súbor má pred zmenšením exe-packerom veľkosť 84kB. Následným použitím UPX sa veľkosť zmenší na 33kB ako môžme vidieť na obrázku 4.7, čím je splnené zadanie.

4.6.1 Metóda komprimovania spustiteľných súborov

Každá metóda samorozbaľovania spustiteľných súborov, ktorá prebieha výhradne v pamäti by sa dala zhrnúť nasledovne:

1. Spustenie programu používateľom.
2. Dekompresia obsahu skutočného programu.
3. Importovanie dynamických knižníc.
4. Skok na skutočný začiatok programu (Original Entry Point - OEP).

Spustenie programu používateľom

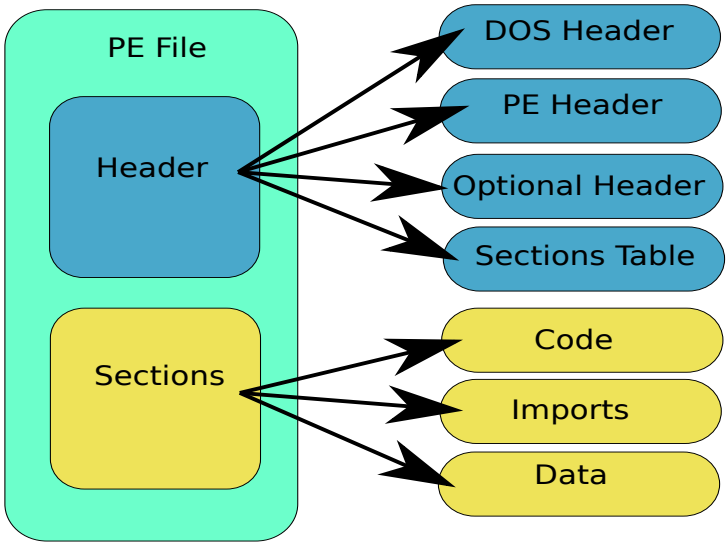
Vzhľadom na to, že ide o samorozbaľovací *spustiteľný* súbor, je nutné aby obsahoval aspoň minimum nekomprimovaného kódu. Veľkosť tohto kódu musí byť minimálna nielen preto, že je nekomprimovaný, ale aj preto, že oproti pôvodnému programu predstavuje dáta naviac.

Dekompresia obsahu

Základnou úlohou počiatočného nekomprimovaného kódu je rozbalenie komprimovaných dát v pamäti. Výber optimálneho kompresného algoritmu je závislý od jeho účinnosti, rýchlosti rozbaľovania a veľkosti rozbaľovacieho kódu. Toto rozbaľovanie môže prebehnúť "in-place," teda priamo prepísaním komprimovaných dát, alebo rozbalením na nové miesto v pamäti. Prvý spôsob predstavuje menšie pamäťové nároky, ale obmedzuje výber kompresných algoritmov.

Importovanie dynamických knižníc

Pôvodný PE (Portable Executable) súbor (obrázok 4.6) obsahoval importovaciu tabuľku (Import Table), ktorá obsahovala údaje o požadovaných knižniciach (.dll súboroch). Táto tabuľka bola skomprimovaná a nahradená zvyčajne prázdnou tabuľkou. Na základe tejto tabuľky operačný systém načíta dané knižnice a doplní adresy funkcií do Import Address Table. Po rozbalení je nutné pre správny chod programu obnoviť obsah Import Table, načítať knižnice a doplniť adresy funkcií, teda vykonať všetky operácie, ktoré by normálne boli úlohou operačného systému.[19]



Obrázek 4.6: Štruktúra PE súboru[21]

Skok na skutočný začiatok programu

V tomto momente by malo byť "prostredie"programu vytvoreného rozbaľovačom nastavené porovnateľne s prostredím, v ktorom sa spúšťa pôvodný nezabalený program. Nasleduje skok na OEP.

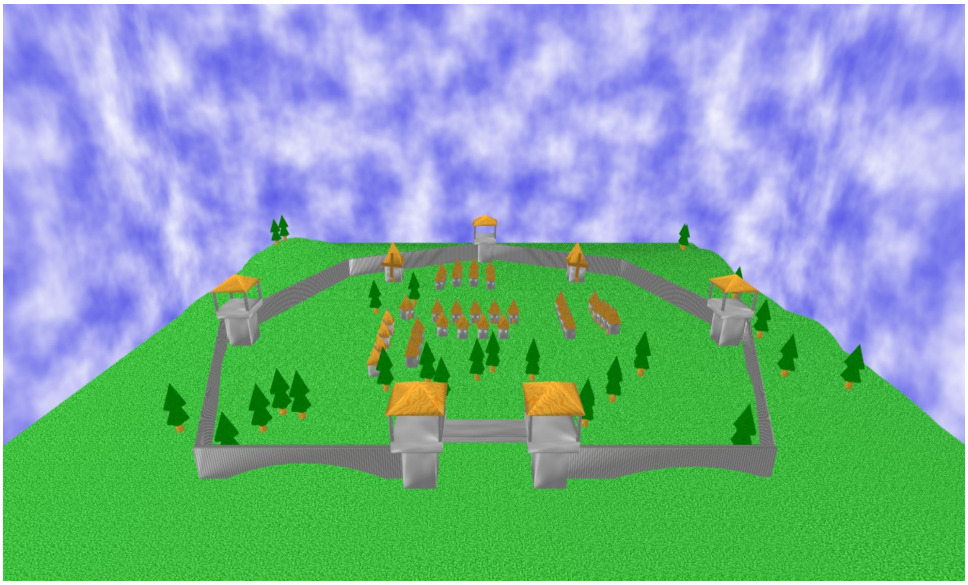
```
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95w       Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     86528 ->     33792   39.05%    win32/pe     OpenGL intro.exe

Packed 1 file.
```

Obrázek 4.7: Výstup exe-packera po kompresii dat spustiteľného súboru

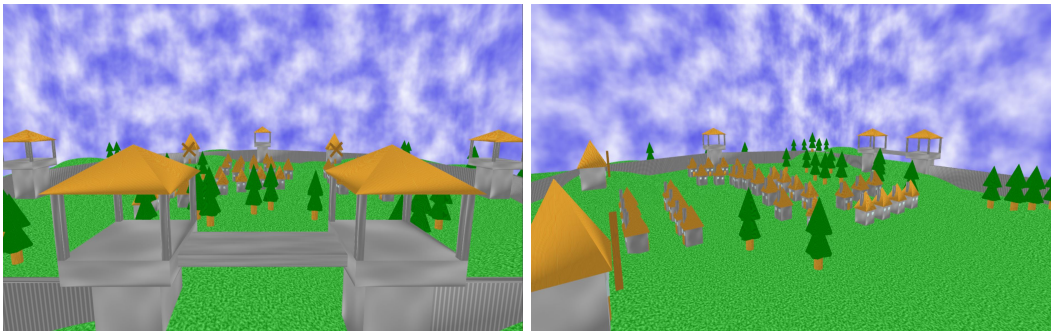
Kapitola 5

Výsledok práce

Zjednotením všetkých implementovaných objektov a techník dostaneme grafické demo, ktoré prezentuje osadu chránenú hradbami, čo bolo aj cieľom tejto práce. Výsledné intro trvá 41s. Celá osada je znázornená na obrázku 5.1



Obrázek 5.1: Zobrazenie výslednej osady grafického intra



Obrázek 5.2: Výstupy kamery z výsledného grafického intra

Kapitola 6

Budúce ciele

Nakoľko veľkosť výsledného spustiteľného súboru je 33kB a limit je 64kB tak stále ostáva dostatok priestoru na vylepšovanie a pridanie ďalších techník, ktoré výsledné grafické demo oživia.

Medzi to patrí napríklad implementovanie hudby pomocou knižnice LibV2 od skupiny Farbrausch, ktorá pracuje so zvukovým formátom .v2m. Daná knižnica je priamo prispôsobená pre použitie v 64kB grafických demách. Pri implementácií prebehlo viac pokusov o použitie danej knižnice ale doposiaľ sa to nepodarilo úspešne implementovať kvôli zatiaľ neznámej chybe v nastavení.

Ďalším pokračovaním by mohlo byť nahradenie súčasných stromov vytvorených geometrickými útvarmi pomocou lindenmayerových systémov popísaných v kapitole 3.5. Počas implementácie tejto práce bolo s L-systémami experimetované ale výsledný objekt sa nepodaril vyrenderovať podľa predstáv. S ohľadom na čas nakoniec boli stromy pomocou L-systémov nahradené jednoduchšou metódou. Taktiež by mohol byť pridaný dažďa alebo dym z komínov pomocou časticových systémov popísaných v kapitole 3.4, ktoré v práci zatiaľ využité neboli.

Kapitola 7

Záver

Cieľom tejto práce bolo vytvorenie grafického intra s použitím OpenGL s obmedzením veľkosti výsledného spustiteľného súboru na maximálne 64kB. Na demonštráciu grafického dema som sa rozhodol vyrenderovať osadu s hradbami. Zadanie bolo splnené a výsledné demo má veľkosť 33kB. Pri tvorbe tejto práce bolo nutné najprv oboznámenie sa s implementáciou grafických scén a aplikačným rozhraním OpenGL, nakoľko som s tým nemal takmer žiadne predchádzajúce skúsenosti.

Pri vývoji grafického intra som sa postupne oboznamoval s možnosťami ako dosiahnuť minimálnu veľkosť výsledného dema, aké knižnice možno použiť. Dôležité bolo aj použitie vhodných implementačných metód. V tejto práci sú použité metódy procedurálneho generovania, textúry tvorené pomocou perlinovho šumu, okolie grafickej scény za použitia skyboxu, výšková mapa na generovanie terénu a taktiež tieňovanie pomocou phongovho osvetlovacieho modelu. Rovnako dôležité bolo aj použitie externého exe-packeru na zmenšenie výslednej veľkosti.

Tvorba práce mi dala mnoho nových skúseností s programovaním grafických scén a rovnako aj znalostí rôznych techník a metód, ktorých štúdium bolo nevyhnutné pre splnenie zadania.

Literatura

[1] *Buffer Object* [online]. [cit. 2020-5-21]. Dostupné z: https://www.khronos.org/opengl/wiki/Buffer_Object.

[2] *Built-in Variable (GLSL)* [online]. [cit. 2021-4-19]. Dostupné z: https://www.khronos.org/opengl/wiki/Built-in_Variable_(GLSL).

[3] *Compute Shader* [online]. [cit. 2021-4-13]. Dostupné z: https://www.khronos.org/opengl/wiki/Compute_Shader.

[4] *History of hardware tessellation* [online]. [cit. 2021-4-14]. Dostupné z: https://rastergrid.com/blog/2010/09/history-of-hardware-tessellation/.

[5] *History of OpenGL* [online]. [cit. 2021-4-13]. Dostupné z: https://www.khronos.org/opengl/wiki/History_of_OpenGL.

[6] *LearnOpenGL - Shaders* [online]. [cit. 2021-4-13]. Dostupné z: https://learnopengl.com/Getting-started/Shaders.

[7] *Load OpenGL Functions* [online]. [cit. 2021-4-11]. Dostupné z: https://www.khronos.org/opengl/wiki/Load_OpenGL_Functions.

[8] *OpenGL Context* [online]. [cit. 2021-4-11]. Dostupné z: https://www.khronos.org/opengl/wiki/OpenGL_Context.

[9] *OpenGL Extension* [online]. [cit. 2021-4-11]. Dostupné z: https://www.khronos.org/opengl/wiki/OpenGL_Extension.

[10] *OpenGL Object* [online]. [cit. 2021-4-11]. Dostupné z: https://www.khronos.org/opengl/wiki/OpenGL_Object.

[11] *OpenGL Overview* [online]. [cit. 2020-4-25]. Dostupné z: https://www.khronos.org/opengl/.

[12] *Primitive* [online]. [cit. 2021-4-13]. Dostupné z: https://www.khronos.org/opengl/wiki/Primitive.

[13] *Query Objects* [online]. [cit. 2021-4-11]. Dostupné z: https://www.khronos.org/opengl/wiki/Query_Object.

[14] *SPIR-V* [online]. [cit. 2021-4-19]. Dostupné z: https://www.khronos.org/opengl/wiki/SPIR-V.

[15] *Tesselation* [online]. [cit. 2021-4-14]. Dostupné z: https://www.khronos.org/opengl/wiki/Tessellation.

[16] *Tesselation Evaluation Shader* [online]. [cit. 2021-4-14]. Dostupné z: https://www.khronos.org/opengl/wiki/Tessellation_Evaluation_Shader.

[17] *Texture* [online]. [cit. 2021-4-13]. Dostupné z: https://www.khronos.org/opengl/wiki/Texture.

[18] *Tree* [online]. [cit. 2020-5-15]. Dostupné z: http://www.malsys.cz/Gallery/Detail/JFI1IzqW.

[19] *Understanding the Import Address Table* [online]. [cit. 2021-4-13]. Dostupné z: http://sandsprite.com/CodeStuff/Understanding_imports.html.

[20] Elias, H. *Perlin Noise* [online]. [cit. 2020-5-02]. Dostupné z: https://web.archive.org/web/20160325134143/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm.

[21] Grunzweig, J. *Basic Packers: Easy As Pie* [online]. [cit. 2021-4-28]. Dostupné z: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/basic-packers-easy-as-pie/.

[22] Lee, J. *How Procedural Generation Took Over The Gaming Industry* [online]. [cit. 2020-04-28]. Dostupné z: https://www.makeuseof.com/tag/procedural-generation-took-gaming-industry/.

[23] Mount, D. *CMSC 425: Lecture 14 - Procedural Generation: Perlin Noise*. [cit. 2021-4-19]. Dostupné z: https://web.archive.org/web/20210415014732/https://www.cs.umd.edu/class/fall2018/cmsc425/Lects/lect14-perlin.pdf.

[24] Perlin, K. An Image Synthesizer. *ACM SIGGRAPH Computer Graphics*. 1985, [cit. 2021-4-19].

[25] Tišnovský, P. *L-systémy: přírodní objekty i umělé artefakty* [online]. [cit. 2020-5-15]. Dostupné z: https://www.root.cz/clanky/l-systemy-prirodni-objekty-i-umele-artefakty/.

[26] Vries, J. de. *Basic Lighting* [online]. [cit. 2020-5-04]. Dostupné z: https://learnopengl.com/Lighting/Basic-Lighting.

[27] Vries, J. de. *Cubemaps* [online]. [cit. 2020-5-04]. Dostupné z: https://learnopengl.com/Advanced-OpenGL/Cubemaps.

[28] Vries, J. de. *Particles* [online]. [cit. 2020-5-15]. Dostupné z: https://learnopengl.com/In-Practice/2D-Game/Particles.

Příloha A

DVD

Obsah priloženého DVD

- /src - Zdrojové kódy programu
- /bin - Výsledný spustiteľný súbor
- /packer - Použitý exe-packer UPX
- /latex - Zdrojový tvar písomnej správy
- /video - Videozáznam grafického intra
- Táto písomná správa v PDF
- readme