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 **JAN VĚRNÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/21173/2017/xverny00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Věrný Jan**
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 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: **Milet Tomáš, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

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

Tato bakalářská práce se zabývá vytvořením grafického intra s omezenou velikostí spustitelného souboru. K dosažení tohoto cíle bylo využito OpenGL společně s metodami procedurálního generování, exe packer a různá nastavení kompilátoru. Výsledkem je spustitelný soubor nepřesahující velikost 64kB, který zobrazuje cestu Marťanských vesmírných lodí k planetě Zemi.

Abstract

This bachelor's thesis deals with creation of graphic intro with limited size of the produced executable file. To reach this goal OpenGL, together with procedural generation techniques, exe packer and various compiler settings were used. The result is an executable file with size not going over 64kB, which shows Martian spaceships travelling towards the planet Earth.

Klíčová slova

grafické intro, 64kB, C++, OpenGL, GLSL, Phongů osvětlovací model, procedurální generace, Perlinův šum, L-systém, MIDI

Keywords

graphic intro, 64kB, C++, OpenGL, GLSL, Phong lighting model, procedural generation Perlin noise, L-system, MIDI

Citace

VĚRNÝ, Jan: Grafické intro 64kB s použitím OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2018

Grafické intro 64kB s použitím OpenGL

Prohlášení

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

……
Jan Věrný
16. května 2018

Poděkování

Tímto bych rád poděkoval svému vedoucímu, Ing. Tomáši Miletovi za odborné rady při tvorbě práce a dále mé rodině a přátelům za psychickou podporu.

Obsah

1 Úvod

Tato bakalářská práce se zabývá problematikou tvorby grafického intra s omezenou velikostí. Toto téma dává řešiteli určitou volnost při výběru použitých metod tvorby grafického obsahu a v rozsahu jeho tvořivé činnosti. Zároveň však autora pevně omezuje velikostním limitem 64kB, konkrétně 65,536 bytů.

Ve svém intru jsem se rozhodl demonstrovat svou kreativitu a zvolené techniky zobrazením Marťanských vesmírných lodí na jejich cestě z planety Mars k Zemi, za doprovodu průběžně gradující hudby.

Termínem grafické intro se zabývá druhá kapitola. Navíc je v ní stručně shrnutá historie a současnost demoscény. Třetí kapitola se věnuje problematice vykreslení scény pomocí grafické knihovny OpenGL, zahrnuje také Phongův osvětlovací model, projekci a skybox. Čtvrtá kapitola popisuje použité techniky a metody generování grafické a hudební složky intra. Především jde o Perlinův šum, L-systémy a hudební formát MIDI. Využití jednotlivých metod v rámci grafického intra je popsáno v kapitole páté. Šestá kapitola se zabývá metodami pro splnění velikostního omezení. Jedná se o nastavení překladače a použití exe packeru. Výsledku a zhodnocení rychlosti vykreslení se věnuje kapitola sedmá. Závěr shrnuje vykonanou práci a rozebírá možný budoucí rozvoj intra.

2 Grafické intro a demoscéna

Grafické intro je určitou formou umění, tvůrce, případně skupina tvůrců, se jeho tvorbou pokouší demonstrovat své programátorské a umělecké schopnosti. V minulosti se rozlišoval termín demo a intro. O intro se jedná v případě, že existuje nějaké omezení jeho výsledné velikosti, jinak jde o demo. Dnes jsou ale tyto termíny tak často zaměňovány že se prakticky jedná o synonyma.

Stará, ale i nová grafická intra se obvykle prezentují na takzvaných demopárty. Na těch se setkávají členové demoscény, tuto komunitu tvoří jak tvůrci inter, tak i jen nadšenci pro tuto formu umění. Většinou se demopárty koná ve formě soutěže, kde se hlasuje o nejlepší intro v dané kategorii.

2.1 Demoscéna

Počátky demoscény se datují do 80. let 20. století. [1] Právě v této době se osobní počítače začaly stávat dostupné široké veřejnosti. Společně s tím se zároveň začal objevovat drahý komerčního software (většinou hry). Ne všichni uživatelé, ale byli ochotní za něj zaplatit. Proto začaly vznikat pirátské skupiny, které se soustředili právě na prolamování ochran proti kopírování.

Jak se začalo vynořovat více a více pirátských skupin, začalo být potřeba se zviditelnit. Pirátské skupiny proto začali k upravenému software přidávat krátká grafická, později hudebně-grafická, intra. Ty měli sloužit jako podpis dané skupiny. Intra se velmi rychle stala populárními a vzrostl tlak na jejich kvalitu zpracování. Pro tvůrce inter se tak postupně začalo snižovat množství času věnovaného prolamování ochran a rostl čas strávený tvorbou inter. Postupem času se někteří úplně přestali věnovat pirátství, vytvořili demoskupiny a vznikla tak demoscéna.

V demoskupině mohou být mimo programátorů, také grafici, hudebníci a občas i scénaristé. Jednotlivé skupiny se setkávají na demopárty, kde prezentují svoji tvorbu a soutěží v jednotlivých

Obrázek 2.1 Foto z německé demopárty Revision 2017. Převzato z [2].

kategoriích. Původně se jednalo o menší událost, kde se setkalo nejvíce pár desítek jedinců. V současnosti jsou demopárty dobře organizované, mezinárodní akce, na kterých se často konají i přednášky a další doprovodný program. Mezi nejznámější a nejprestižnější demopárty patří například německá *Revision* (Obrázek 2.1), dříve *Breakpoint* a finská *Assembly*.

2.2 Grafické intro

Grafické intro je typem neinteraktivní multimediální prezentace (Obrázek 2.2), obvykle ve formě počítačového programu, jehož hlavním cílem je zprostředkovat divákovi co nejpůsobivější audio-vizuální zážitek. Velký důraz je také kladen na optimální využití dostupného hardware. Nejedná se tak o dílo, která přináší nějaký přímý užitek, ale spíše o ukázku programátorských schopností autora, autorů.

Intra mezi sebou soutěží v různých kategoriích dle velikostního limitu, který musí být splněn. Ten se může pohybovat od 32 bytů po více než 1MB, nejběžnější jsou ale 64kB a 4kB intra. Limit zvolený pro tuto práci, 64kB, přesně 65,536 bytů, pochází z tradičního limitu maximální velikosti spustitelného souboru typu COM.

Důvodem, proč existují tato omezení i navzdory všeobecnému technologickému pokroku, je soutěživost členů demoscény. Ti se navzájem předhánějí, kdo lépe dokáže využít limitovaného prostoru, pro vytvoření co nejefektnějšího intra.

Kvůli těmto omezením je nutné vytvářet intra bez využití předpřipravených modelů, obrázků s vysokým rozlišení a dlouhých zvukových stop. Místo toho se grafický obsah procedurálně generuje a hudba se přehrává použitím syntetizátorů. Nakonec se intro zabalí pomocí exe packeru.

Obrázek 2.2 Ukázka z 64kB intra Panic Room od demoskupiny Fairlight

2.3 Téma intra

Tvůrci grafických inter se snaží diváka zaujmout nejen působivou grafickou stránkou, ale i originálním a zajímavým tématem. Obsah inter se liší, menší intra (4kB a méně) jsou obvykle abstraktní, zobrazující nerealistické scény, skládající se z různých geometrických tvarů nebo fraktálů. Větší intra se soustředí na nějakou konkrétní tématiku.

Při návrhu a tvorbě mého intra jsem se nechal inspirovat úvodními titulky z filmu Mars Útočí (1996) (Obrázek 2.3). Intro bude mít několik scén. Na začátku uvidíme Marťanské vesmírné lodě, jak vzlétají z povrchu Marsu. V dalších scénách pak tyto lodě poletí vesmírem, dokud nedoletí k planetě Zemi. Během intra navíc budou na vesmírné obloze ke spatření vzdálené hvězdy, stříbrný pás Galaxie Mléčná dráha a Slunce.

Obrázek 2.3 Úvodní titulky z filmu Mars Útočí (1996)

3 Vykreslování grafiky

Zadání této bakalářské práce vyžaduje použití knihovny OpenGL pro vykreslení veškeré grafiky. V následujících podkapitole se jí podrobně věnujeme. Další podkapitoly se věnují problematice nasvícení scény, projekci a skyboxu.

3.1 Open Graphics Library

OpenGL je na programovacím jazyku nezávislé, multiplatformní rozhraní (API) pro tvorbu grafických aplikací. [3] Umožnuje aplikaci komunikovat s grafickou kartou a využívat ji pro hardwarovou akceleraci. Používá se při tvorbě CAD programů, aplikací virtuální reality, vědeckotechnických vizualizací, počítačových her a dalších.

Vývoj OpenGL započal roku 1991 ve firmě Silicon Graphics Inc. Jak se postupně zvětšoval počet dodavatelů hardware pro 3D grafiku, SGI ztrácela podíl na trhu. Ve snaze ovlivnit tuto situaci se rozhodla změnit jejich Iris GL API na otevřený standard OpenGL. O rok později vzniklo konsorcium OpenGL ARB, skupina společností, jejichž cílem je udržovat a v budoucnu rozšiřovat specifikaci. Roku 2006 OpenGL ARB přenesla kontrolu nad standardem OpenGL pod Khronos Group. V době psaní této práce je specifikace stále aktivně rozvíjena. Přesto se již aktivně pracuje na jejím nástupci, knihovně Vulkan.

3.1.1 Struktura OpenGL

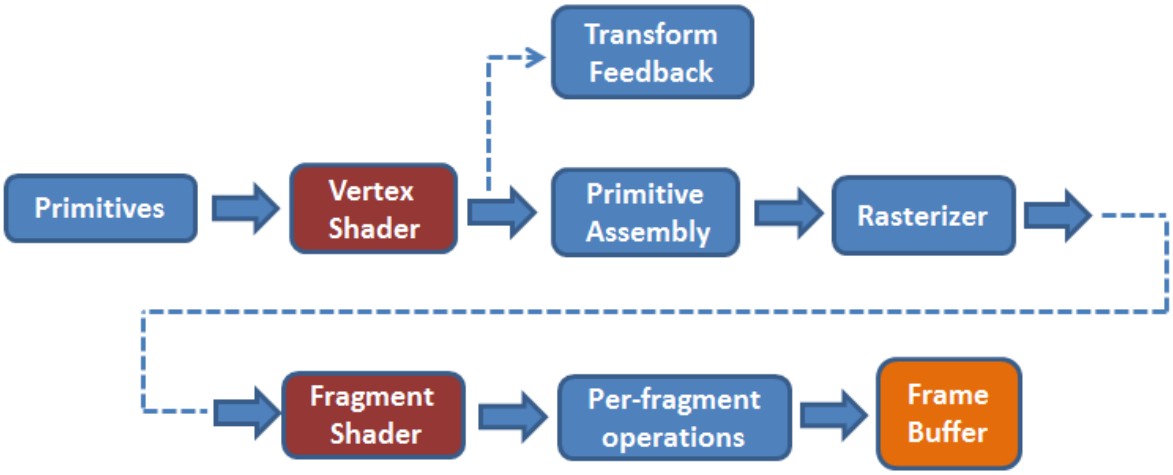
Rozhraní se skládá ze sady pojmenovaných konstant a funkcí. Je založeno na architektuře klient-server. Program (klient) vydává příkazy, které jsou vykonány grafickým adaptérem (server).

V OpenGL se nepoužívá objektově orientované programování. Místo toho pracuje s vertexy (vrcholy, body), každý vertex může mít několik atributů (souřadnice umístění bodu, normály, texturovací souřadnice, barva, …). Z jednotlivých vertexů se vytváří primitiva, obvykle trojúhelníky. Tato vektorová data se posílají ke zpracování do programovatelné pipeline (Obrázek 3.1). Výstupem je rastrový snímek ve Frame Bufferu, který lze zobrazit na monitoru.

Dodatečná funkcionalita je realizována volitelnými rozšířeními poskytovanými například výrobci grafických čipů.

3.1.2 Buffery

Buffery jsou OpenGL objekty, které adresují data uložená v paměti grafické karty. Může se jednat o téměř libovolná data, například vertexy (vrcholy modelu), indexy vertexů tvořících jednotlivá primitiva, normálové vektory, texturovací souřadnice, vykreslené snímky, hloubkové informace a další.



Obrázek 3.1: Programovatelný pipeline použitá v OpenGL. Převzato z [4].

Index Buffer Object

Při skládání primitiv je velmi pravděpodobné že budeme chtít jeden vertex použít vícekrát. Pro tento účel slouží indexované buffery. Společně s vertex bufferem tak vytvoříme index buffer obsahující indexy vrcholů, z kterých se mají složit primitiva. Při nahrávání dat vertexů se tak šetří čas a místo v paměti grafické karty.

Vertex Array Object (VAO)

Jedná se objekt, jenž v sobě drží odkazy na buffery a jejich konfiguraci. Byl navržen tak aby držel informace pro kompletní vykreslení objektu. To zahrnuje nastavení vertex atributů (datový typ, velikost, prokládání, offset) a nastavení pro indexový buffer. Data z něj se použijí jako vstup pro vertex shader.

Framebuffer Object (FBO)

Je objekt, který drží reference na buffery související s vykresleným rastrovým snímkem. Většinou se jedná o barevné buffery, hloubkový buffer anebo stencil buffer. Nahrazuje standardní vykreslení přímo na obrazovku. Umožňuje provést několik průchodů vykreslení, aplikovat postprocessing na finální scénu nebo vykreslit scénu do textury a použít ji pro další vykreslení.

3.1.3 GLSL

Je vyšší programovací jazyk, syntakticky velmi blízký jazyku C, určený pro programování shaderů. [5] Vznikl v rámci postupné transformace původně fixního vykreslovacího řetězce na řetězec programovatelný. Součástí specifikace OpenGL se stal ve verzi 2.0, od té je stále průběžně rozšiřován.

Definuje základní skalární datové typy, a navíc speciální datové typy pro použití v shaderech. Umožňuje pracovat s až čtyř složkovými vektory a maticemi, a také nad nimi definuje matematické operace. Dalším speciálním datovým typem jsou tzv. samplery (např. *sampler2D*), ty usnadňují práci s texturami. GLSL také nabízí širokou škálu matematických funkcí, funkcí pro míchání barev a unikátní operátor *swizzle*, který umožnuje jednoduše přeskládat složky vektoru nebo matice.

3.1.4 Shadery

Jedná se o krátké programy psané v jazyku GLSL. Zejména kvůli přenosnosti a kompatibilitě mezi grafickými kartami různých výrobců, se kompilují a optimalizují až za běhu aplikace. Podle toho, jakou část programovatelné pipeline ovlivňují, rozlišujeme vertex, geometry, tesselační, fragment a compute shadery.

Shadery nahráváme a paralelně zpracováváme na grafické kartě. Shadery jsou velmi izolované, aby mohli běžet paralelně musí na sobě být nezávislé. Nemohou mezi sebou komunikovat, zapamatovat si předchozí stav nebo zjistit výsledek jiného vlákna. Jejich programování je tak zásadně odlišné od běžných programů, a proto může být značně obtížné.

Kód pro jeden shader může být uložen v několika textových řetězcích. Je tak možné mezi shadery sdílet definované funkce. Ve výsledném intru používáme vertex a fragment shader.

Vertex Shader

Vertex shader nejčastěji provádí transformace (násobení) vertexu pomocí projekční, pohledové a modelové matice. Lze jej také využít pro animaci nebo deformaci modelu. Např. deformace povrchu planety nebo pohyb vodní hladiny.

Vertex shader se aplikuje na každý vykreslovaný vrchol. V případě použití indexovaného bufferu, je ovšem možné že se ve vykreslovaném modelu stejný vrchol používá několikrát, v takovém případě se použije předchozí výsledek.

Fragment Shader

Pro každý bod rasterizovaného primitiva se vytvoří takzvaný fragment. Každý fragment obsahuje pozici v prostoru okna, interpolovaný výstup z vertex shaderu a další. Ten je zpracován ve fragment shaderu. Výstupem této fáze je hloubková hodnota a nula nebo více barevných hodnot. V shaderech tohoto typu realizujeme obarvení primitiv, aplikaci textur, osvětlovací model nebo například normálové mapování.

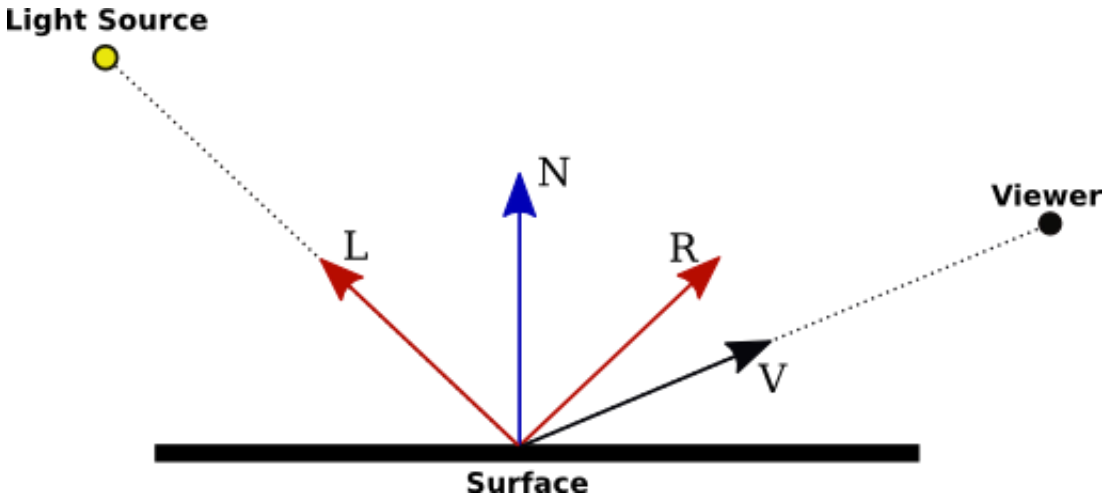
3.1.5 Textury

Textura je obvykle dvou nebo více rozměrné pole texelů. Ten je základní jednotkou textury, reprezentuje její jeden bod. Při vykreslení se texely mapují na pixely, pokud je zapotřebí texel na jiné než celočíselné pozici, provede se filtrování textur. Textury „nanášíme“ na primitiva abychom jim dodali dodatečné detaily. Také lze do textury uložit výstup z programu shaderu.

3.2 Phongův osvětlovací model

Byl navržen vietnamským vědcem Bui Tuong Phongem v roce 1973, rámci jeho disertační práce na University of Utah. [6] Jedná se o empirický osvětlovací model, ten na rozdíl od realistické fyzikální simulace, vyžaduje pouze relativně malé množství výpočetních prostředků.

K výpočtu potřebujeme 4 vektory (Obrázek 3.2). Prvním je vektor $\vec{L}$ přicházející ze zdroje světla a jeho odrazový vektor $\vec{R}$. Vektor $\vec{N}$ je normála v pozorovaném bodě. Úhel pohledu reprezentuje vektor $\vec{V}$.



Obrázek 3.2: Zobrazení vektorů Phongova modelu. Převzato z [7].

Dojmu realistického zobrazení trojrozměrné scény dosahuje rozložením odrazu do tří světelných složek (Obrázek 3.3): ambientní L_a, difuzní L_d a spekulární L_s. Vyjadřujeme jej vztahem 3.1.

$$I = L_a + L_d + L_s \tag{3.1}$$

3.2.1 Ambientní (okolní) světlo

Tato složka udává intenzitu té části světla, která na těleso dopadá se stejnou intenzitou ze všech směrů. Napodobuje tak sekundární odražené a rozptýlené světlo, které vzniká mnohonásobnými odrazy od okolních těles. Tato složka zajišťuje že ani povrchy odvrácené od zdroje světla nebudou úplně černé. S její rostoucí hodnotou roste celková světlost scény.

Lze ji určit vztahem 3.2, kde I_a označuje intenzitu okolního světla a k_a odrazový koeficient materiálu.

$$L_a = I_a\, k_a \tag{3.2}$$

3.2.2 Difúzní světlo

Difúzní složka určuje intenzitu části světla, která se od matného povrchu tělesa odráží do všech směrů. Právě její použití ve scéně vytváří trojrozměrný dojem. Nezávisí na směru pohledu, protože světlo se od difúzního povrchu odráží ve všech směrech rovnoměrně.

Vyjadřujeme ji vztahem 3.3, kde I_d označuje intenzitu difúzního osvětlení scény, k_d určuje odrazový koeficient materiálu, $\vec{N}$ je normála povrchu v místě dopadu světelného paprsku a $\vec{L}$ je směr příchodu světla.

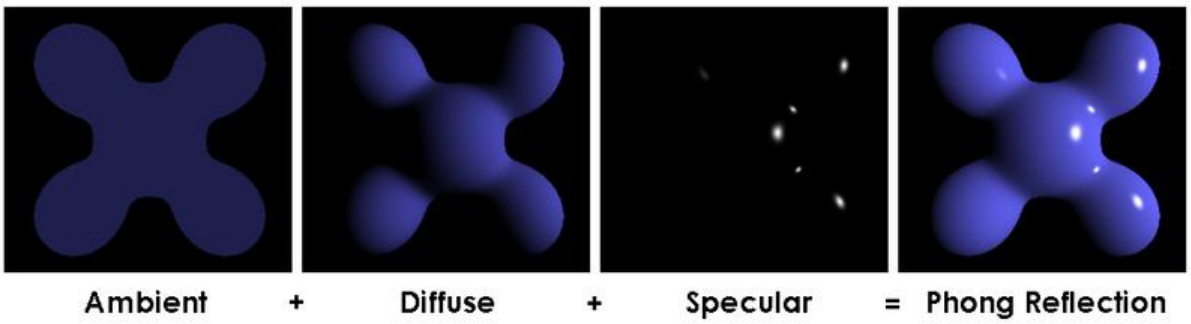
$$L_d = I_d k_d (\vec{N} \cdot \vec{L}) \tag{3.3}$$

3.2.3 Spekulární (lesklé) světlo

Určuje intenzitu světelných odrazů od zdroje směrem k pozorovateli.

K výpočtu této složky použijeme vztah 3.4, kde I_s označuje intenzitu odlesků, k_s je odrazový koeficient, $\vec{R}$ je světlo odražené na povrchu podle zákonu odrazu a $\vec{V}$ určuje směr odkud pozorujeme.

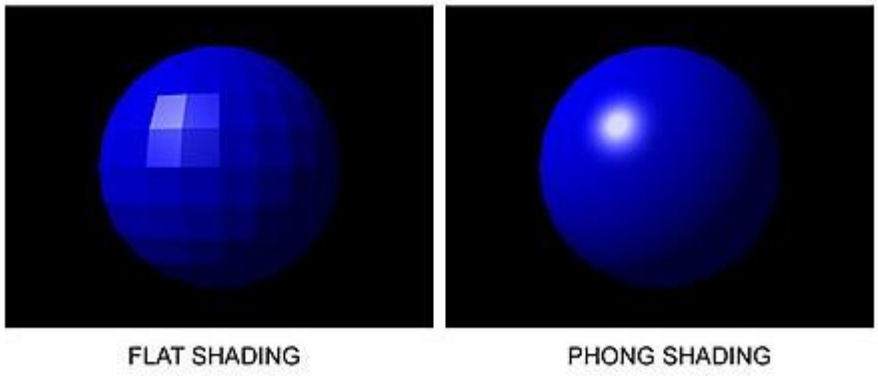
$$L_s = I_s k_s (\vec{V} \cdot \vec{R}) \tag{3.4}$$



Obrázek 3.3 Světelné složky použité pro Phongův osvětlovací model. Převzato z [8].

3.2.4 Phongovo stínování

Vyvinuto společně s Phongovým osvětlovacím modelem, jedná se o metodu určenou ke spojitému stínování těles, jejichž povrch tvoří množina rovinných ploch. Pro určení odstínu výsledného pixelu používá techniku interpolace normálových vektorů, tím se docílí hladkých přechodů mezi sousedními pixely a korektně vykreslených odlesků od světel. Je náročnější na výpočet než Gouraudovo (ploché) stínovaní, které uvažuje konstantní normálu pro celé primitivum, ale zato dosahuje podstatně lepších výsledků. (Obrázek 3.4)



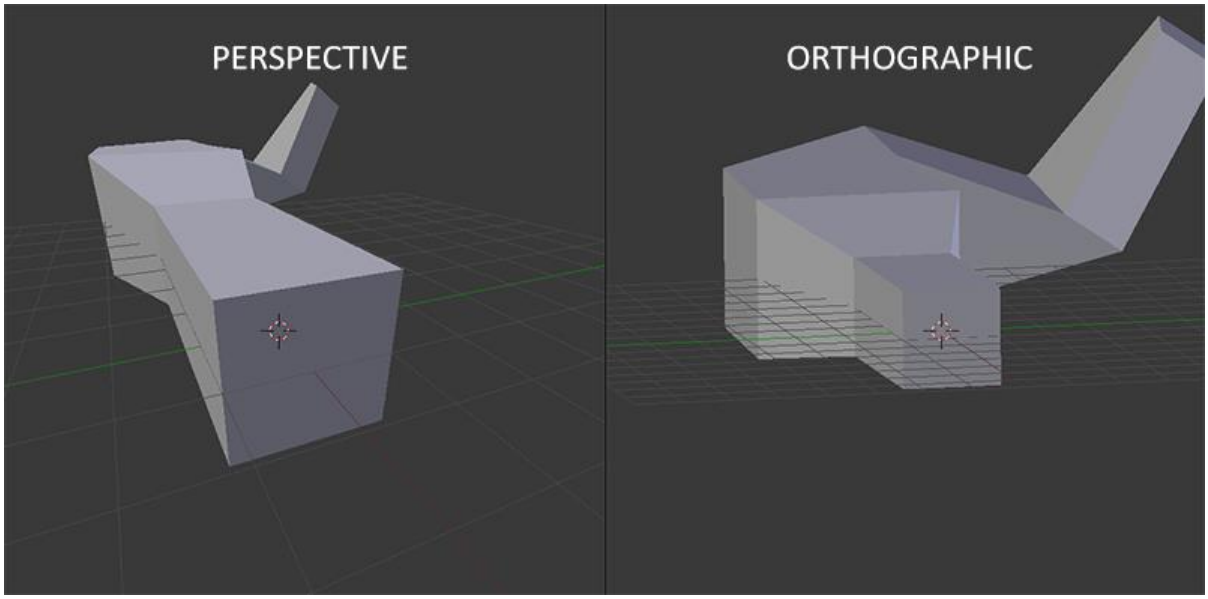
Obrázek 3.4 Gouraudovo (ploché) vs. Phongovo stínování.

3.3 Projekce

Pro korektní vykreslení pomocí OpenGL je nutné transformovat souřadnice jednotlivých vertexů z prostoru souřadnic použitého v našem modelu do prostoru souřadnic používaného v OpenGL. V počítačové grafice k tomuto obvykle používáme projekci ortografickou nebo perspektivní.

V případě projekce ortografické reprezentujeme scénu jako pravoúhlý hranol. To způsobuje že objekty se vzdáleností nemění svou velikost. Tento typ projekce nejčastěji používáme v aplikacích, kde jsou podstatné skutečné rozměry zobrazovaných objektů.

V perspektivní projekci je scéna definována jako komolý jehlan. Tím se způsobí že objekt blíže ke kameře se jeví větší než objekty vzdálené. Tento způsob projekce respektuje optický model, který vyjadřuje lidské vidění reálného světa a poskytuje tak dobrý prostorový vjem na průmětné rovině. (Obrázek 3.5)



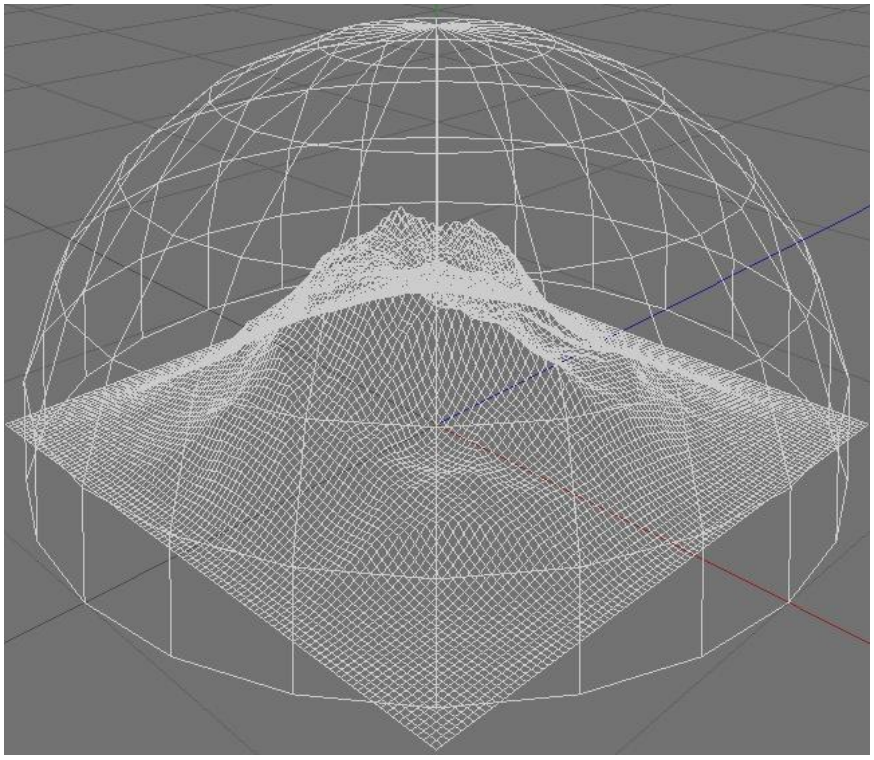
Obrázek 3.5 Perspektivní vs. Ortografická projekce. Převzato z [9].

3.4 Skybox

Skybox se používá k zobrazeních těch nejvzdálenějších oblastí ve scéně. Obvykle se tak jedná, jak už samotný název napovídá, o box s texturou oblohy nebo vzdálené krajiny, který ze všech stran obklopuje scénu. V praxi se stále častěji objevuje takzvaný skydome (Obrázek 3.6), který místo boxu používá polokouli. Vzdálené oblasti je tak možné vykreslit za použití malého množství polygonů s minimálním dopadem na výkon. Při použití vhodných textur, je potom skybox téměř nerozlišitelný od zbytku scény.

Pro interaktivní aplikaci je kamera obvykle umístěna do středu skyboxu a ten následuje její pohyb. Tim že je zajištěno že pozorovatel nikdy nedosáhne horizontu. Tím se vytváří dojem, že scéna je větší, než je ve skutečnosti. Nejčastěji se s takovým chováním setkáme v počítačových hrách.

V našem intru bude skybox implementován jednoduše jako statický dóm obklopující celou scénu, neboť kameru máme plně pod kontrolou a nemusíme se obávat, že by se dostala mimo zamýšlené meze.



Obrázek 3.6 Ukázka modelu skydomu. Převzato z [10].

4 Procedurální generování

Jedná se o způsob, jak na požádání generovat velké až nekonečné množství dat. Můžeme generovat trojrozměrné modely, textury, text nebo cokoliv jiného co nás napadne. Data jsou tak generována až za běhu programu a ve spustitelném souboru tak zabírají pouze tolik místa, kolik potřebujeme pro uložení algoritmu na jejich generaci. Při tvorbě intra je toto obrovskou výhodou, jedna nekomprimovaná textura v běžném rozlišení 2048 x 2048 texelů by spotřebovala celých 16MB paměti. Uložením ve ztrátovém formátu JPEG bychom její velikost mohli znatelně snížit, ale i tak bychom měli podstatný problém dostat se do limitu 64 kilobytů.

Pro procedurální generování je potřeba algoritmus, který generuje nová data, tento algoritmus může ale nemusí využívat náhodnosti. V našim intru chceme docílit vždy stejného výsledku, budeme proto používat pseudo-náhodnost, která je deterministická. Nevýhodou je pouze omezená kontrola nad vygenerovaným obsahem a často složitost implementace algoritmu.

Obrázek 4.1 Snímek ze hry No Man's Sky, zobrazující procedurálně generovanou planetu a vegetaci.

V současnosti i v minulosti je procedurální generace velmi populární, využívá ji velké množství komerčně úspěšných produktů (Obrázek 4.1).

Generátor pseudo-náhodných čísel

Je deterministický efektivní program, který generuje posloupnost čísel, ideálně nerozlišitelnou od náhodné ani za použití statistických testů. Počáteční nastavení X_0 nazýváme *seed* (semínko), od něj se odvíjejí všechny následné vygenerované hodnoty. Zvolením vždy stejného nastavení hodnoty *seed* tak lze zajistit, že při každém spuštění programu bude vygenerována stejná sekvence náhodných čísel.

Jedním z nejstarších a nejpoužívanějších generátorů pseudo-náhodných čísel je lineární kongruentní generátor [11], popsaný vztahem 4.1, kde operace *mod* představuje zbytek po celočíselném dělení a a, c a m jsou vhodně zvolené konstanty. Generátor generuje celá čísla s rovnoměrným rozložením v rozsahu 0 až m.

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

4.1.1 Perlinův šum

Ken Perlin navrhl tento šum v roce 1985 [12], kvůli své frustraci s tehdejším „strojovým“ vzhledem počítačové grafiky. Jedná se o procedurálně generovaný šum, který má poměrně nízkou výpočetní složitost a umožňuje vytvářet textury s přirozeným vzhledem.

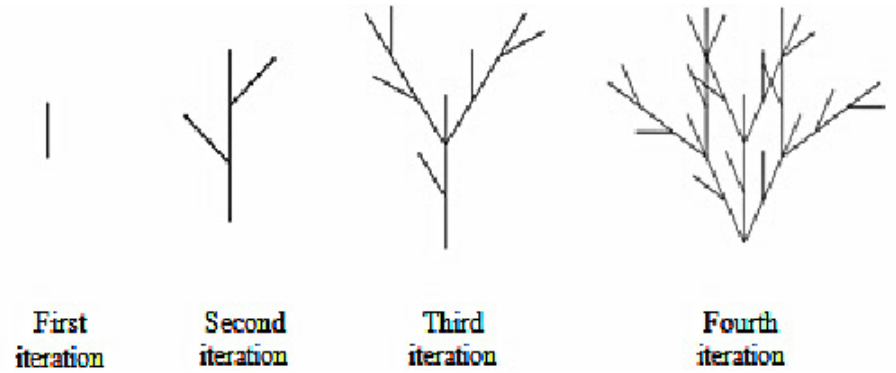
Obrázek 4.2 Ukázka míchání Perlinova šumu.

Nejčastěji se implementuje jako funkce s dvou-, tří- nebo čtyř-rozměrným výstupem, ale může být definován pro libovolné množství rozměrů. Taková funkce má na svém vstupu pozici v prostoru a vlnovou délku výsledného šumu. Základem generovaní Perlinova šumu je generátor pseudo-náhodných čísel. Mezi vygenerovanými hodnotami je následně prováděna interpolace, většinou lineární, to má za následek vyhlazenou funkci. Výsledek pro jednu vlnovou délku však ještě nelze považovat za dostatečně přirozený, obvykle proto mícháme několik vlnových délek (oktáv), nejběžnější je použít oktávy s poloviční vlnovou délkou (dvojnásobnou frekvencí). Tyto oktávy mícháme buď všechny se stejnou váhou, nebo můžeme na některé vlnové délky klást váhu větší. (Obrázek 4.2)

Tento šum má v počítačové grafice široké možnosti použití. V intru použijeme Perlinův šum pro vytvoření textur planety Mars a Země, oblak, mléčné dráhy, a navíc pro deformaci modelů planet.

4.1.2 L-systém

Známý také jako Lindenmayerův systém, jedná se o variantu formální gramatiky, původně vyvinuté pro matematický popis růstu řas. Tento koncept představil v roce 1968 maďarský biolog a botanik Aristid Lindenmayer. [13]



Obrázek 4.3 Několik iterací nad modelem vegetace. Převzato z [14].

L-systém popisuje větvící se moduly a množinu sebe přepisujících pravidel jež se opakovaně aplikují na tyto moduly. V tomto případě modul reprezentuje vše, co se v daném objektu opakuje, roste nebo větví. Hlavní myšlenka tohoto přístupu spočívá v tom, že jak se daný objekt rozrůstá, tak se jeho

jednotlivé moduly stále víc a víc opakují. Takové opakování potom lze vyjádřit pomocí rekurze. Například strom se bude skládat z modulů větev a list, pravidla potom v každé iteraci určí jejich rozmístění.

Z pohledu informatiky lze L-systém definovat jako trojici G = (∑, S, P), kde ∑ je abeceda (neprázdná množina symbolů), P je konečná množina přepisovacích pravidel a S je konečné slovo z abecedy ∑, které definuje počáteční stav systému (axiom). L-systém se nápadně podobá bezkontextové gramatice, drobně se od sebe ale liší. Například tím že L-systém nerozlišuje terminální a neterminální symboly, místo toho se mohou neterminální symboly přepsat sami sebou. Dále gramatika používá jako počáteční stav pouze jeden symbol, ale L-systém povoluje libovolný konečný řetězec symbolů.

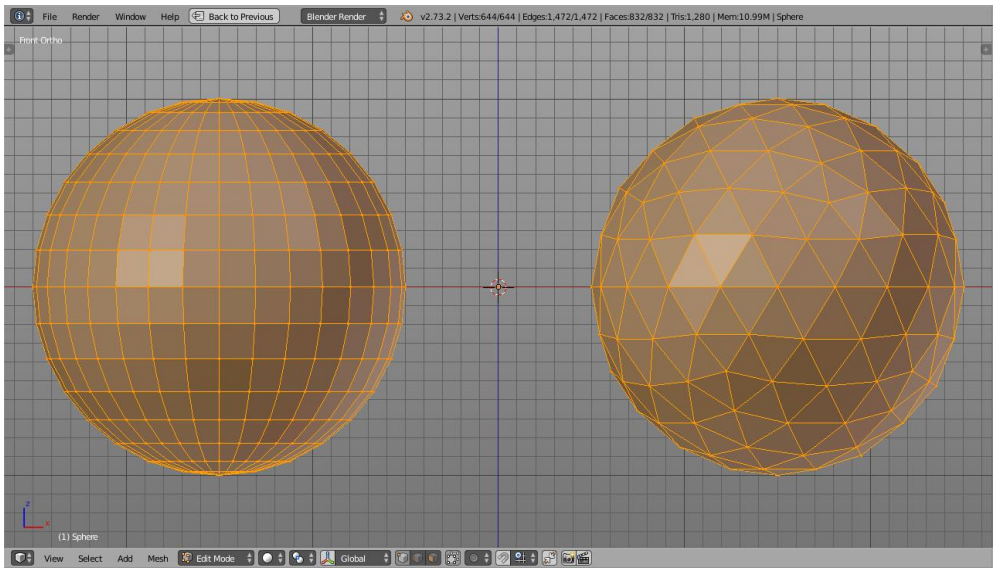
Obrázek 4.4 Konstukce Von Kochovy sněhové vločky. Převzato z [14].

Jednotlivé symboly L-systému interpretujeme různě, někdy je podstatný pouze počet symbolů v i-té iteraci, jindy se symboly chápou jako příkazy pro kreslení nebo symboly reprezentují základní části z kterých se nakonec poskládá výsledný model. Ve 2D je obvyklá interpretace pomocí želví grafiky. Základem této interpretace je virtuální želva nesoucí štětec, ta interpretuje jednotlivé symboly jako příkazy pro pohyb a dle nich pak vykresluje zvolený objekt.

L-systémy s použitím želví grafiky se obvykle používají na tvorbu různých druhů vegetace (Obrázek 4.3). Mohou ale být také použité na generování různých druhů křivek nebo fraktálů (Obrázek 4.4).

4.1.3 Model koule

Pro tvorbu planet v intru budeme potřebovat trojrozměrný model koule, ten poté budeme texturovat a deformovat. Existuje několik způsobů, jak lze takovou kouli modelovat. (Obrázek 4.5)



Obrázek 4.5 UV sphere vs. Icosphere. Převzato z [15].

Icosphere

Základem icosphere je ikosaedr, těleso, jehož stěny tvoří 20 rovnostranných trojúhelníků. Ty můžeme dále rekurzivně dělit na pod-trojúhelníky (teselovat), dokud nezískáme dostatečně hladkou kouli. Výhodou Icosphere je že nevyžaduje tolik trojúhelníků jako ostatní metody a při mapování textur dochází pouze k minimálnímu zkreslení.

Quad sphere

Základem Quad sphere je krychle. Její stěny rozdělíme na požadovaný počet úseků a následně je „vypoukneme“ do tvaru koule. Tato metoda není příliš obvyklá, používá se nejčastěji v modelech, které nevyužívají trojúhelníků, ale takzvaných quadů. Výhodou je že že při mapování textur nedochází ke vzniku singularit na pólech ani nikde jinde na povrchu koule.

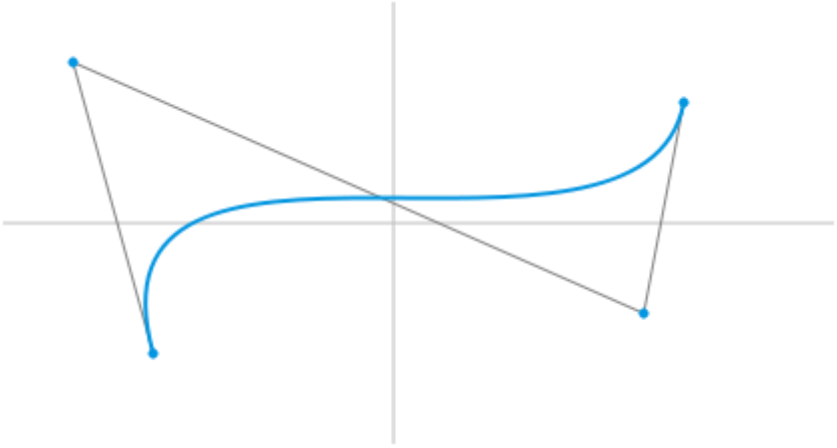
UV sphere

V intru použijeme právě UV sphere. Tu modelujeme pomocí poledníků a rovnoběžek. Vygenerování UV sphere je velmi jednoduché a nevyžaduje žádnou rekurzi. Na výsledný model není složité namapovat vygenerovanou texturu. Drobnou nevýhodou je vyšší hustota bodů a s tím spojené zkreslení v blízkosti pólů koule.

4.1.4 Bézierova křivka

Jedná se o jednu z mnoha druhů parametrických křivek. Pojmenována byla po francouzském inženýru Pierru Bézierovi. Podle počtu řídících bodů rozlišujeme linární, kvadratické a kubické Bézierovy křivky. (Obrázek 4.6)

Tvar křivky měníme změnou polohy řídících bodů. Ve třírozměrném prostoru to ale nemusí být vždy jednoduché. Běžná je tedy metoda, která každému řídícímu bodu přiřadí váhu, reálné číslo, jehož změnou lze měnit tvar křivky. V intru křivku použijeme při generování modelů vesmírných lodí.



Obrázek 4.6 Ukázka 2D kubické Bézierovy křivky

Pro n+1 zadaných kontrolních kontrolních bodů, které tvoří řídící polygon, je pro t € <0,1> definována vztahem 4.2.

$$\mathrm{C(t)} = \sum_{i=0}^{n} B_{i,n}(t) \quad t \in < 0,1 >$$

4.2

Přičemž $B_{i,n}(t)$ je i-tý Bernsteinův polynom n-tého stupně, definován vztahem 4.3.

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

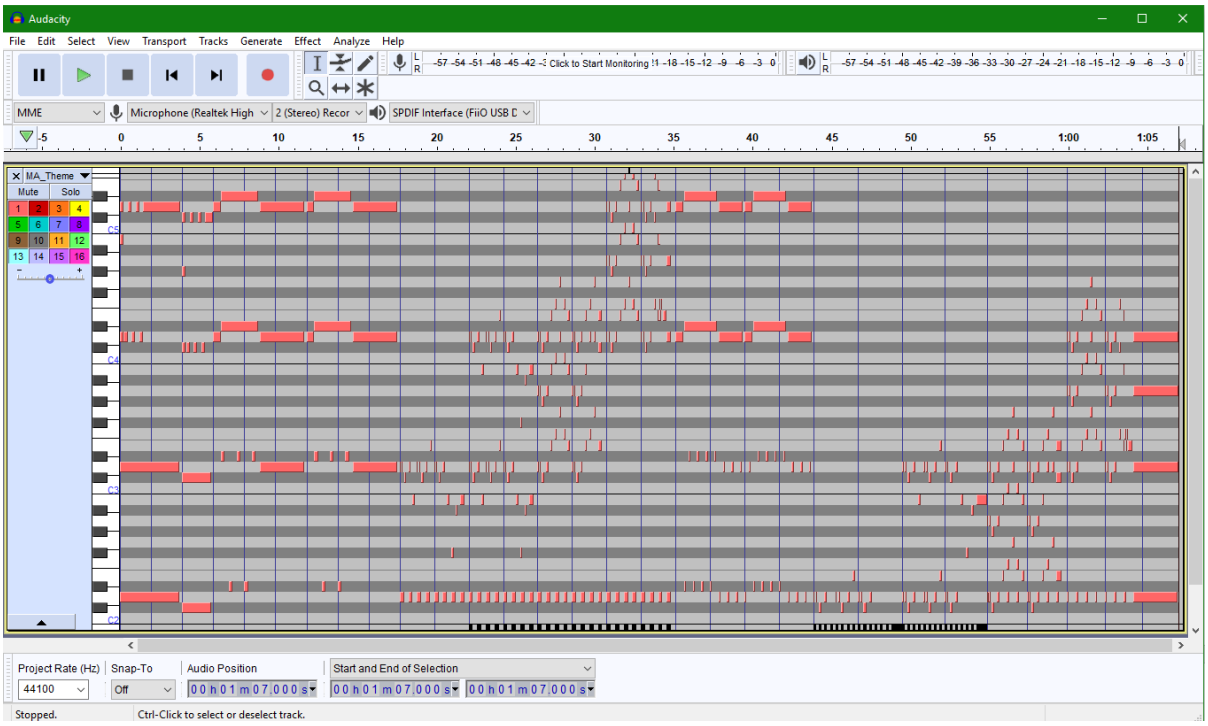
4.3

4.2 MIDI

Musical Instrument Digital Interface, je volně přístupný průmyslový standard, navržený kolem roku 1983. Tento standard v sobě zahrnuje specifikaci hardware a software, pro digitální komunikaci mezi hudebními nástroji a ostatními zařízeními jako jsou počítače, mixery nebo sekvencéry. Komunikace MIDI nahradila dřívější analogovou komunikaci VC/Trig, která nebyla přísně standardizována a zařízení různých výrobců tak mezi sebou nemusela být kompatibilní.

MIDI mezi zařízeními nepřenáší zvukový signál, ale digitální informace o událostech. Většinou týkající se toho, co hudebník se svým nástrojem dělá. Mohou určovat jakou notu nástroj zrovna hraje, její výšku, tempo a další. Tyto události se přenáší v až 16 kanálech, kdy každý kanál představuje jeden hudební nástroj. Některé nástroje mohou události jen vysílat, jiné i přijímat. Samotná událost se skládá ze stavových a datových bytů. V době vzniku MIDI se šetřilo každým byte, protokol je proto navržen velmi úsporně. Stavový byte se vysílá pouze při změně události, je-li událost stejná stačí posílat jen datové byty.

Mezi výhody MIDI patří jednoduchost modifikace a manipulace, široký rozsah nástrojů a syntetizátorů a pro nás nejdůležitější obvykle velmi malá velikost souboru. (Obrázek 4.7)



Obrázek 4.7 Ukázka obsahu MIDI souboru zobrazená programem Audacity

5 Implementace

Tato kapitola se věnuje implementaci výsledného dema, popisuje, jak byly implementovány a použity dříve popsané techniky, k dosažení zamýšlených výsledků. První podkapitola se věnuje vytvoření okna pomocí Windows API, druhá načtením funkcí OpenGL, třetí se věnuje výpočtu transformačních matic, čtvrtá a pátá generováním modelů a textur. Šestá podkapitola popisuje volbu a přehrání MIDI souboru. Sedmá kapitola krátce popisuje implementaci kalendáře událostí.

5.1 C++, Windows API

Výsledný program byl implementován v integrovaném vývojovém prostředí Microsoft Visual Studio 2017. K implementaci byl využit jazyk C++, který rozšiřuje jazyk C o řadu užitečné funkčnosti. [16] Za obzvláště užitečnou považuji možnost přetěžovat funkce a operátory.

Protože je grafické intro limitované velikostí 64kB, pro vytvoření vlastního okna a získání OpenGL kontextu nebylo vhodné použít již existující rozsáhlé multiplatformní knihovny, jako je například knihovna GLFW. Bylo tak zapotřebí využít nativní Windows API. K té lze získat přístup přiložením hlavičkového souboru *windows.h*. To sebou přináší jisté změny, vstupním bodem programu se stane, místo klasické funkce *main*, rozšířená funkce *WinMain*, navíc je taky nutné definovat funkci *WndProc*, která se stará o zpracování zpráv zaslaných oknu. Vlastní okno získáme vytvořením struktury třídy okna typu WNDCLASSEX, ta obsahuje informace o okně. Následně je třeba si vytvořenou třídu okna zaregistrovat pomocí *RegisterClassEx* a poté již lze vytvořit okno zavoláním funkce *CreateWindowEx*. Když máme vlastní okno můžeme následně získat vykreslovací kontext OpenGL zavoláním funkce *wglCreateContext*. [17]

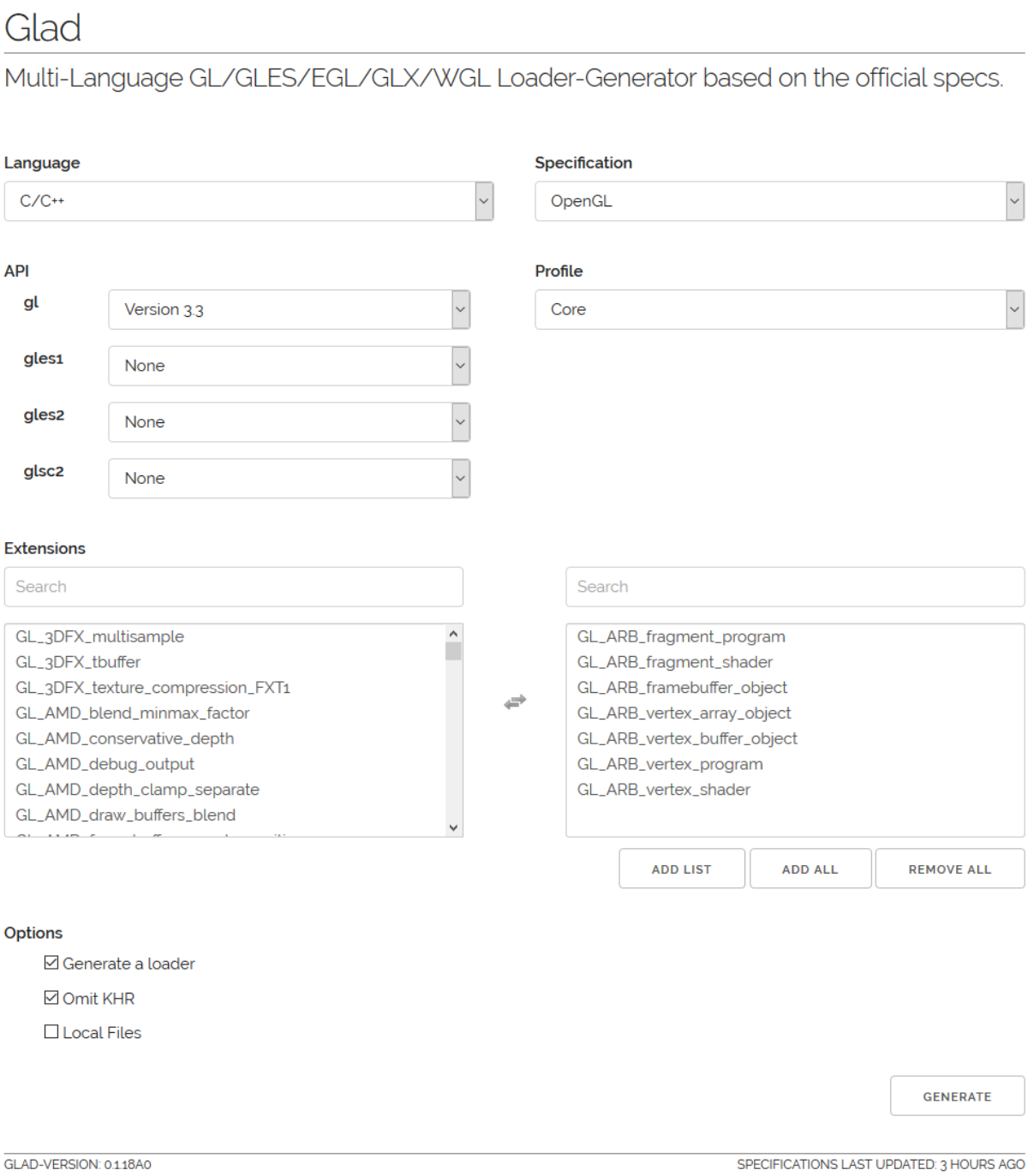
5.2 OpenGL

Pro účely vytvoření intra bylo zvoleno OpenGL ve verzi 3.3. Ze souboru *opengl32.dll* však v operačním systému Windows nelze načíst funkce z verze vyšší než 1.1. Je potřeba načíst je až za běhu aplikace.

Z počátku byla k tomuto účelu využita knihovna GLEW, ta dokáže načíst veškeré dostupné funkce a rozšíření až do, v současnosti poslední, verze 4.6, postupem času se ale ukázala jako nepraktická, protože je příliš rozsáhlá a zabírá tak ve výsledném spustitelném souboru příliš mnoho místa.

Další možností je načíst všechny potřebné funkce manuálně, k tomu je zapotřebí již dříve zmíněný soubor opengl32.dll, v něm se nachází speciální funkce *wglGetProcAdress*, která načte funkci podle jména a parametrů. Proces manuálního načtení je ale velmi zdlouhavý a monotónní na naprogramování.

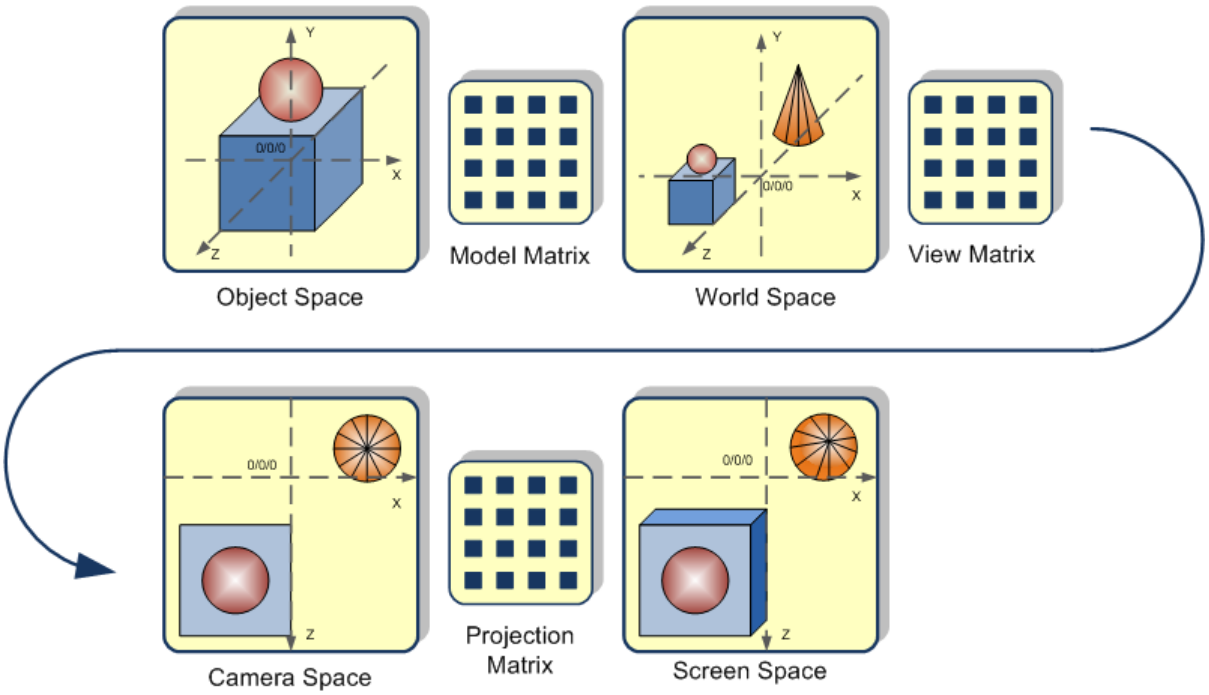
Ideálního kompromisu bylo dosaženo použitím webové služby Glad (Obrázek 5.1), ta umožňuje nechat si vygenerovat knihovnu, která načte pouze funkce ze zvolené verze OpenGL, společně s řadou volitelných rozšíření. Vygenerované soubory potom prostě přiložíme do našeho projektu a po vytvoření OpenGL kontextu pro vykreslení, zavoláme funkci *gladLoadGLLoader*.



Obrázek 5.1 Snímek obrazovky webového generátoru Glad (http://glad.dav1d.de/)

5.3 Projekční, pohledová a modelová matice

Základní princip zobrazení scény lze charakterizovat následujícími kroky. Nejprve jednotlivé modely rozmístíme v prostoru, k tomu slouží modelová matice. Následně nastavíme pozici a směr kamery ve scéně, k tomu slouží pohledová matice. A na závěr scénu transformuje do souřadnicové systému OpenGL za pomoci matice projekční. (Obrázek 5.2) Protože při tvorbě intra nebyly použity žádné knihovny, které by nám poskytly funkce pro výpočet těchto matic (jako například GLM), bylo zapotřebí tyto matice vytvořit manuálně.



Obrázek 5.2 Ilustruje použití modelové, pohledové a projekční matice. Převzato z [18].

Modelová matice

Modelová matice definuje rotaci a pozici modelu v trojrozměrném prostoru. K tomu využívá tří rotačních matic $R_x(\varphi)$, $R_y(\theta)$, $R_x(\psi)$ vyjadřujících rotace v jednotlivý osách, a matici T vyjadřující posun v prostoru. Vyjádříme je vztahem 5.1. [18]

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{T} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 5.1$$

Z hlediska implementace bylo vhodnější použít pouze jednu matici, byly proto mezi sebou roznásobeny rotační matice, následně se přičetla posuvná matice a vznikla nám jedna modelová matice M vyjádřená vztahem 5.2.

$$M = \begin{bmatrix} \cos\theta & \sin\theta\sin\psi & \sin\theta\cos\psi & T_x \\ \sin\theta\sin\varphi & \cos\psi\cos\varphi - \cos\theta\sin\psi\sin\varphi & -\cos\varphi\sin\psi - \cos\theta\cos\psi\sin\varphi & T_y \\ -\sin\theta\cos\varphi & \cos\psi\sin\varphi + \cos\theta\cos\varphi\sin\psi & -\sin\psi\sin\varphi + \cos\theta\cos\psi\cos\varphi & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2

Pohledová matice

Pohledová matice V rotuje a posouvá prostor kolem kamery tak aby vypadalo jako že se kamera pohybuje a otáčí. K jejímu výpočtu jsou potřeba čtyři vektory. Vektor $\vec{P}$ určuje pozici kamery v prostoru, vektor $\vec{F}$ určuje směr, kterým se kamera dívá, vektor $\vec{U}$ ukazuje směrem nahoru z pohledu kamery, a vektor $\vec{R}$ získáme vektorovým součinem vektorů $\vec{F}$ a $\vec{U}$. Vypočteme ji vztahem 5.3. [19]

$$\mathrm{V} = \begin{bmatrix} R_x & R_y & R_z & -(\vec{R}\cdot\vec{P}) \\ U_x & U_y & U_z & -(\vec{U}\cdot\vec{P}) \\ F_x & F_y & F_z & -(\vec{F}\cdot\vec{P}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.3

Projekční matice

Pro intro byla zvolená perspektivní projekce. K tomu je použita projekční matice P. K jejímu výpočtu je zapotřebí čtyř hodnot, *near* a *far* reprezentující vzdálenost dvou podstav komolého jehlanu, *aspect* udává poměr stran jehlanu a *fov* určuje šířku zorného pole. Lze ji vypočítat vztahem 5.4. [20]

$$range = \tan(fov/2) \times near$$
$$Sx = (2 \times near)/(range \times aspect + range \times aspect)$$
$$S_y = near/range$$
$$S_z = -(far + near)/(far - near)$$
$$P_z = -(2 \times far \times near)/(far - near)$$
$$\mathrm{P} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & P_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

5.4

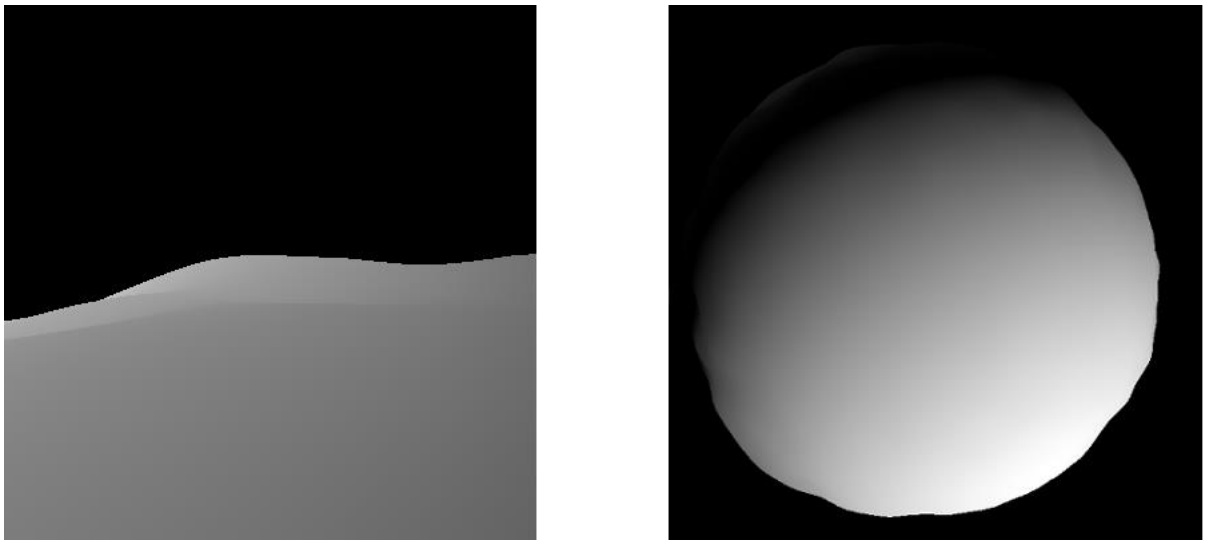
5.4 Generování modelů

Pro účely intra generujeme modely jako indexované buffery vertexů a normál. Texturovací souřadnice ukládáme do separátního bufferu. V případě že chceme použít Gouraudovo (ploché) stínování je nutné vertexy sdílené mezi více primitivy uložit do bufferu pro každé primitivum zvlášť.

5.4.1 Planety

V intru chceme mít dvě planety, Mars a Zemi, jako základ si pro ně vygenerujeme model UV sphere. Skutečné planety ovšem nejsou dokonale hladké koule, pro použití v intru potřebujeme dodat našim modelům trochu více detailu. Pro tento účel se skvěle hodí trojrozměrný Perlinův šum. Ten na model

aplikujeme ve vertex shaderu, smícháme čtyři vzorky šumu, jeden nízko a tři vysoko frekvenční. Při vykreslení planet využíváme Phongův osvětlovací model i Phongovo stínování. (Obrázek 5.3)

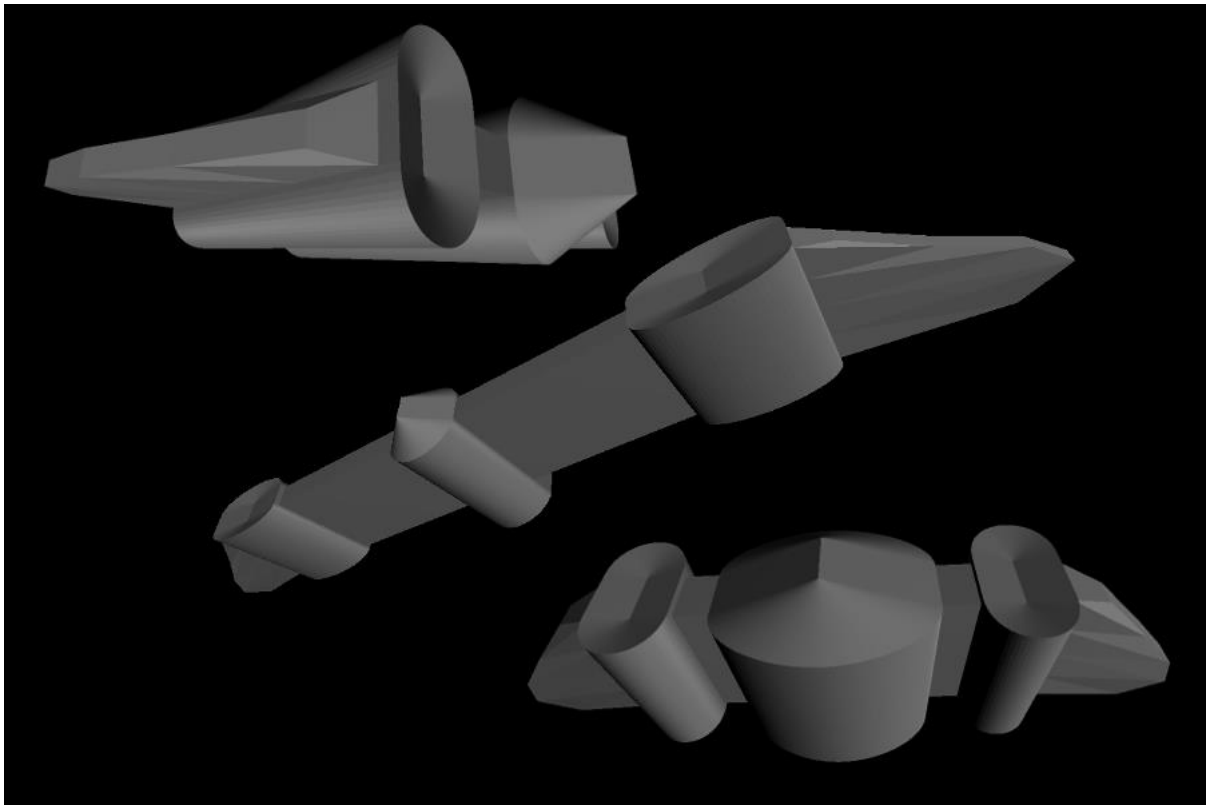


Obrázek 5.3 Deformace koule, vlevo detail, vpravo přehled.

5.4.2 Vesmírné lodě

Žádná mimozemská invaze se neobejde bez vesmírných lodí. Pro jejich generování je použita varianta L-systému, mírně zjednodušená tak, aby vyhovovala našim potřebám.

Při generaci lodě si vyhradíme trojrozměrný prostor s náhodně určenými délkami os. Tento prostor následně rozdělíme podél osy y na náhodně určený počet různě dlouhých úseků. Tyto



Obrázek 5.4 Ukázka vygenerovaných vesmírných lodí

podprostory již lze považovat za jednotlivé díly lodě, trup, motor, spoj a křídlo. Následně podprostory rozdělíme na několik stejně dlouhých úseků. Zde rozlišujeme tři druhy úseků, první, mezi-úsek a poslední. Každý takovýto úsek je popsán vlastní specifickou sadu trojúhelníků. Při generování trupu, motorů a křídla využíváme Bézierovy křivky. Při generaci jednotlivých vertexů, generujeme pouze čtvrtinu lodě a symetrické části získáme zrcadlením v ose y a z. Při vykreslení využíváme Phongův osvětlovací model, ale protože model skládáme z několika menších modelů, používáme pouze ploché stínování. (Obrázek 5.4)

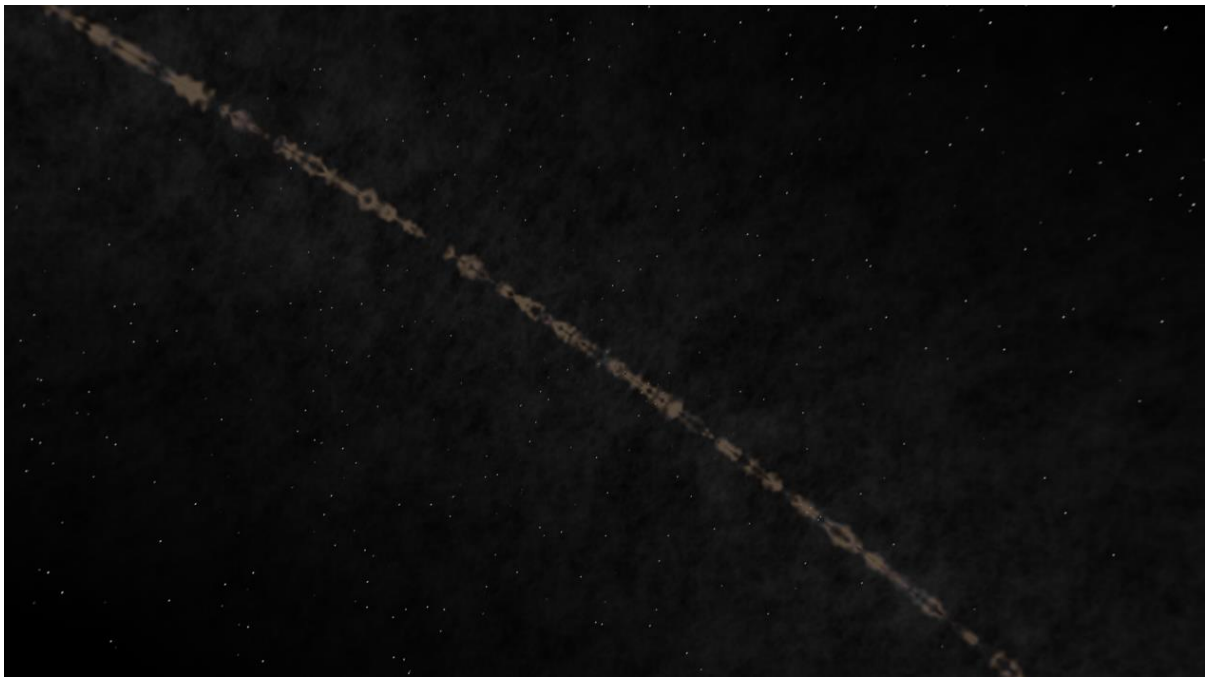
5.5 Generování textur

Procedurálně generovat textury pro všechny objekty ve scéně pokaždé, když chceme vykreslit snímek, by bylo výpočetně zbytečně náročné. Pokud nechceme, aby objekty během intra nějak neměnily svou barvu nebo strukturu, je vhodnější si textury vygenerovat pouze jednou, uložit a použít při každém vykreslení scény. V intru tohoto způsobu využijeme pro všechny textury.

Pro vykreslení scény do textury si musíme nejprve vytvořit novou texturu v paměti grafické karty. U každé textury musíme definovat její velikost, počet rozměrů a barevný formát. Následně si vytvoříme a aktivujeme framebuffer objekt. Barevná data se ve framebuffer objektu ukládají do GL_COLOR_ATTACHMENTi. Do některého z těch připojíme texturu, do které chceme vykreslit. Potom stačí standardně vykreslit scénu. Pro generaci procedurálních textur používáme dva trojúhelníky vyplňujících celou plochu scény. Výslednou texturu můžeme namapovat na libovolný model.

5.5.1 Skybox

Texturu pro skybox generujeme ve dvou průchodech. V prvním fragment shaderu naneseme do textury vzdálené hvězdy, Ty vytvoříme navzorkováním několika offsetů řídkého tečkového šumu (*sparse dot noise*) [21].

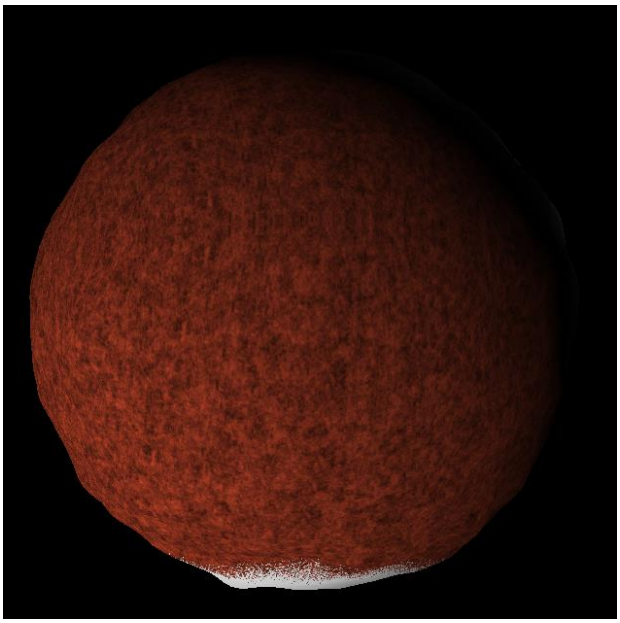


Obrázek 5.5 Ukázka textury skyboxu

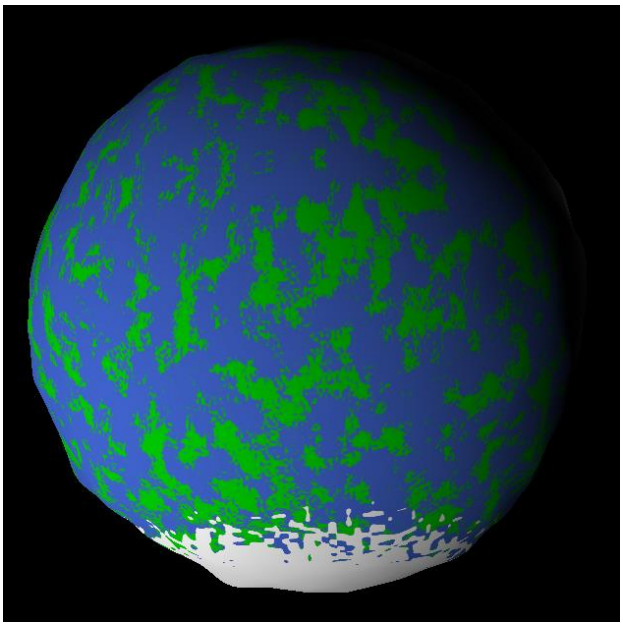
V druhém fragment shaderu vygenerujeme mléčnou dráhu. Tu vytvoříme ze dvou složek. První je slabě svítíce bílé mračno, to vytvoříme smícháním tří vzorků vysokofrekvenčního Perlinova šumu. Druhá složka je tmavé jádro, to vytvoříme použitím Perlinova šumu pro turbulenci kružnice. (Obrázek 5.5)

5.5.2 Mars

Pro texturu Marsu smícháme několik vzorků Perlinova šumu a zabarvíme je do červeno-hnědé. Na pólech chceme vidět sněhovou čepici. Tu vytvoříme tak že použijeme funkci, která určí, zda pozice daného fragment leží uvnitř kružnice zvolené velikosti, se středem v obrazovce. [22] K pevně dané minimální velikosti této kružnice přičteme hodnotu Perlinova šumu pro souřadnici fragmentu. Tím vznikne dojem nepravidelného přechodu. (Obrázek 5.7)



Obrázek 5.7 Ukázka textury Marsu



Obrázek 5.6 Ukázka textury Země

5.5.3 Země

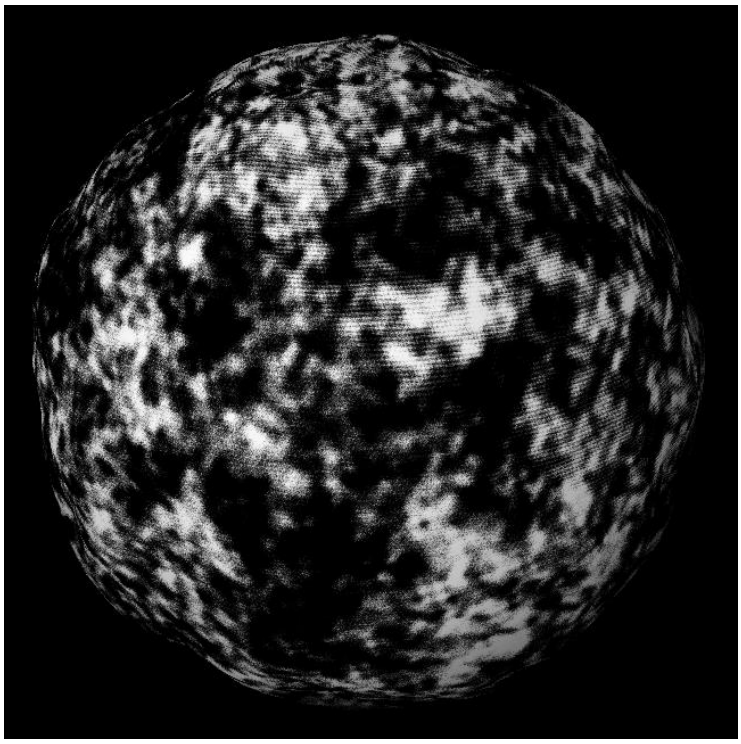
Texturu Země opět vytvoříme za pomoci míchání Perlinova šumu. Chceme napodobit reálný povrch Země a vytvořit středně velké oblasti zelené a modré. Smícháme několik nízko frekvenčních vzorků Perlinova šumu, tím ovšem získáme víceméně rovnoměrnou distribuci šumu. Jednolité oblasti bílé a černé vytvoříme tím, že šum „prahujeme“. Ve fragment shaderu toho dosáhneme výpočtem 5.5, kde *noise* zastupuje hodnotu šumu pro daný fragment, *fraction* určuje procento plochy, která se bude rovnat 1, *transition* učuje rychlost přechodu mezi 0 a 1.

$$\text{noise} = \text{smoothstep}(\text{fraction} - \text{transition}, \text{fraction} + \text{transition}, \text{noise}) \quad 5.5$$

Následně rozdělené oblasti obarvíme modře a zeleně. Na pólech vytvoříme sněhové čepičky stejným způsobem jako u textury Marsu. (Obrázek 5.6)

5.5.4 Mraky

Texturu mraků vytvoříme smícháním několika nízkofrekvenčních vzorků Perlinova šumu. Protože chceme vidět planetu pod mraky, znova použijeme výpočet 5.5 pro „prahování“ a vytvoříme víceméně jednolité oblasti. Tento šum uložíme do alfa kanálu čistě bílé textury. Texturu naneseme na nijak nedeformovaný model koule umístěný ve stejném místě jako model Země, ale s měřítkem zvoleným tak aby ji obklopoval. (Obrázek 5.8)



Obrázek 5.8 Ukázka textury mraků

5.5.5 Slunce

Pro texturu Slunce, jednoduše ve fragment shaderu vygeneruje mnohostěn, kombinací funkce *cos*, *floor* a *length* modulujeme jeho stěny, tak abychom vytvořili paprsky do vrcholů mnohostěnu. [22] Následně takto získanou hodnotu uložíme do alfa kanálu mírně nažloutlé textury. Texturu naneseme na rovnou plochu umístěnou v blízkosti skyboxu. (Obrázek 5.9)

Obrázek 5.9 Ukázka textury slunce

5.6 Přehrání MIDI

Jako hudební doprovod použijeme cover hudby z originální inspirace pro toto intro, filmu Mars Útočí (1996). Ten byl vytvořen uživatelem *brickboy2x2* a je volně šířen ve formátu MIDI. [23]

64kB grafické intro nesmí používat žádné dodatečné soubory, veškerá použitá data musí být ve spustitelném souboru. MIDI soubor proto přiložíme v poli unsigned charů. Textový řetězec pro toto pole si vytvoříme pomocí krátkého a jednoduchého programu, ten v jednom cyklu projde celý soubor MIDI a vypíše hodnotu každého jeho byte v šestnáctkové soustavě.

Po spuštění intra, toto pole uložíme do dočasného souboru, jméno dočasného souboru vygenerujeme pomocí funkce *tmpnam,* takový soubor je automaticky smazán s ukončením programu a nezobrazuje se ani v průzkumníku souborů.

K samotnému přehrání použijeme funkci *playMIDIFile* z dokumentace k Windows API. [24]

5.7 Kalendář událostí

Technika běžně používaná v modelování a simulaci. Jedná se o uspořádanou datovou strukturu uchovávající aktivační záznamy událostí.

Každý záznam události musí mít počáteční a finální čas, odkaz na objekt, kterého se událost týká, vektor tří čísel s plovoucí desetinou čárkou a ukazatel na funkci která se má použít.

Kalendář pracuje tak, že měří čas uplynulý mezi vykreslením jednotlivých snímků. Před každým další vykreslením kalendář projde všechny aktivní události a zavolá s nimi spojenou funkci společně s dodanými parametry a hodnotou vyjadřující v procentech, kolik procent doby trvání události uplynulo od poslední aktualizace času.

V grafickém intru kalendář používáme pro řízení běhu intra v čase. Jednotlivé události ve specifických časových intervalech pohybují a rotují modely, řídí pohyb kamery, spouštějí hudbu a další.

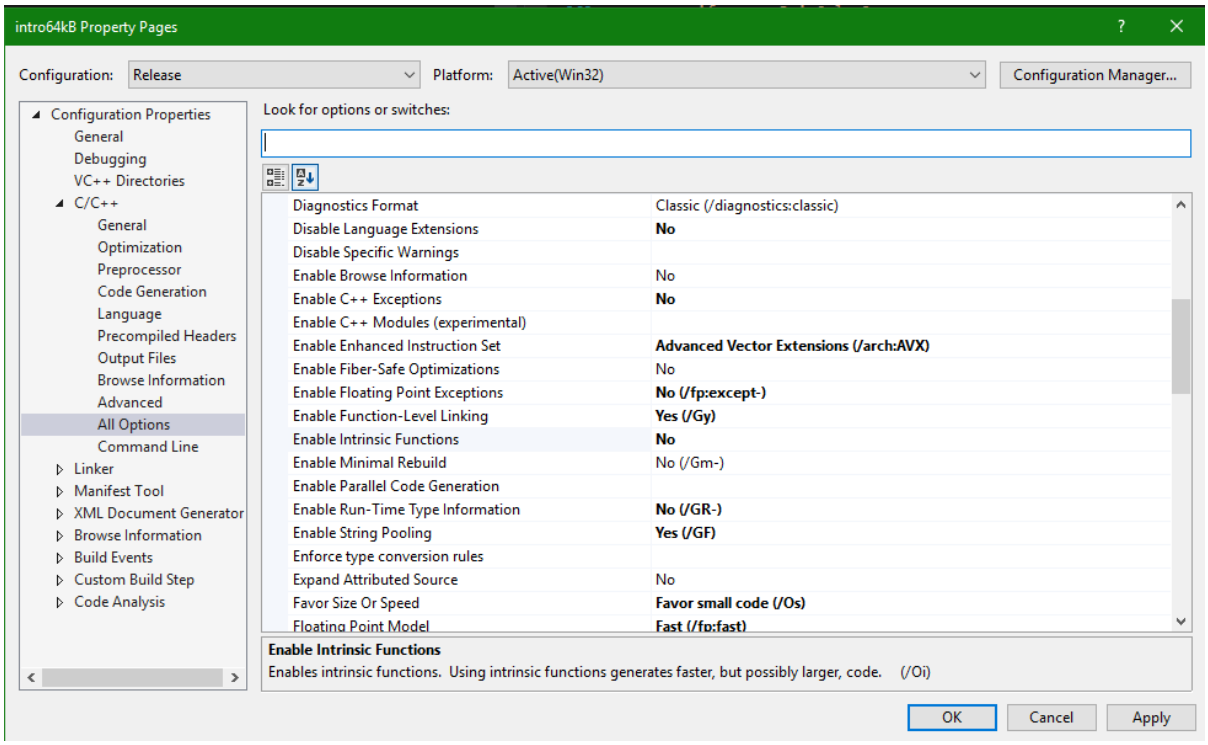
6 Omezení velikosti spustitelného souboru

U aplikace typu grafické intro, hraje výsledná velikost programu důležitou roli. Zmenšení velikosti výsledného programu lze dosáhnout mnoha způsoby. Zásadní úlohu hraje správné nastavení překladače. Další dostupná metoda je komprimace výsledného programu pomocí speciálního balícího nástroje takzvaného exe packeru.

Dále lze velikost výsledného programu podstatně ovlivnit volbou programovacího jazyka a psaním kódu v něm tak, aby byla na prvním místě znovupoužitelnost. Místo lze také šetřit volbou malých jednoúčelových knihoven, nebo případně tím že žádné knihovny nepoužijeme.

6.1 Nastavení překladače

Při překladu programu provádí kompilátor a linker spoustu různých úkonů. Jejich správným nastavením lze podstatně ovlivnit velikost výsledného souboru, rychlost vykonání programu a další. Pro tvorbu grafického intra bylo použito Microsoft Visual Studio 2017, následující nastavení se tak vztahují zejména na překladač firmy Microsoft (Obrázek 6.1), je ale velmi pravděpodobné, že konkurenční překladače (MinGW) obsahují podobná nastavení.



Obrázek 6.1 Ukázka nastavení kompilátoru ve Visual Studiu

Jako první je nutné si zvolit cílovou platformu, moderní aplikace už dnes ve velké míře používají architekturu x86-64. Ta má oproti x86 výhody jako možnost adresovat více než 4GB RAM, využít 64bitové registry a další drobné vylepšení. Nevýhodou je že oproti x86 používá ukazatele s velikostí osm bytů místo čtyř. Kvůli tomu má x86 menší velikost výsledného souboru a zvolíme si ji pro naše grafické intro.

Následují nastavení pro kompilátor, v rámci této práce se budeme věnovat pouze těm, které mají dopad na velikost výsledného programu. [25] První je nastavení optimalizace, chceme, aby při překladu, byla dána přednost malému kódu, zároveň také vypneme všechny debugovací informace. Protože jsme při psaní intra nepoužili C++ výjimky, můžeme vypnout jejich zpracování. Dalšího zmenšení můžeme dosáhnout použitím některé z vektorových instrukčních sad SIMD, největší úsporu nabízí AVX2, ale procesor, na kterém bylo intro vyvíjeno ho nepodporuje, kompromisem je použití AVX, které by mělo mít na moderním hardware podporu běžně. Drobné úspory místa lze dosáhnout zvolením jiné volací konvence než standardní konvence jazyka C.

Linker nám také nabízí spoustu možností, jak zmenšit velikost programu. [26] Důležité je zakázat postupné linkování, generování manifestu, debugovacích informací a metadat systému Windows.

6.2 Exe packer

Po důkladném nastavení překladače má zkompilované intro 84,0kB. Docela se tak blíží limitu 64kB, nicméně, pro zbavení se těch posledních kilobytů, je zapotřebí použít exe packer. Ten slouží právě ke kompresi spustitelných souborů. Ke zkomprimovanému souboru přidá packer krátký program pro rozbalení. Po spuštění se tak nejprve spustí dekompresor, který rozbalí původní program do paměti RAM a následně teprve spustí cílový program. Chování programu se tím nijak nezmění, navíc doba rozbalení je většinou tak krátká že si jí ani nevšimneme. Nevýhodou je možná nekompatibilita s některým anti-virovým softwarem.

Obrázek 6.2 Ukázka grafického rozhraní a jednotlivých nastavení packeru MEW

Existující packery se mezi sebou liší zaměřením na různý druh aplikací a použitým kompresním algoritmem. Většina packerů pro maximální efektivitu používá kompresní metodu LZMA. Je proto vhodné najít packer, který bude pro naše intro vyhovovat nejvíce.

Celkově byly vyzkoušeny čtyři volně šířené komprimační nástroje. Všechny až na jeden se povedlo úspěšně použít. Ke všeobecnému zklamání, se nepodařilo grafické intro zkomprimovat pomocí, v rámci demo scény velmi populárního, nástroje kkrunchy. Příčinu tohoto stavu se ani přes snahu nepodařilo odhalit. Nepomohlo ani nastavení pevné bázové adresy při kompilaci. Ostatní packery naštěstí dosahují dostatečných výsledků. Nejlepšího kompresního poměru dosáhl MEW11 (Obrázek 6.2). Porovnání výsledků použitých packerů můžeme vidět v tabulce Tabulka 6.3.

Nástroj	Verze	Použité parametry	Výsledná velikost	Kompresní poměr
MEW11	1.2	-	29,1kB	34,69 %
MPRESS	2.19	-s	31,5kB	37,50 %
UPX	3.94	--lzma --ultra-brute	32,0kB	38,10 %
kkrunchy	0.23	-	-	-

Tabulka 6.3 Porovnání jednotlivých komprimačních nástrojů na souboru s původní velikostí 84,0kB.

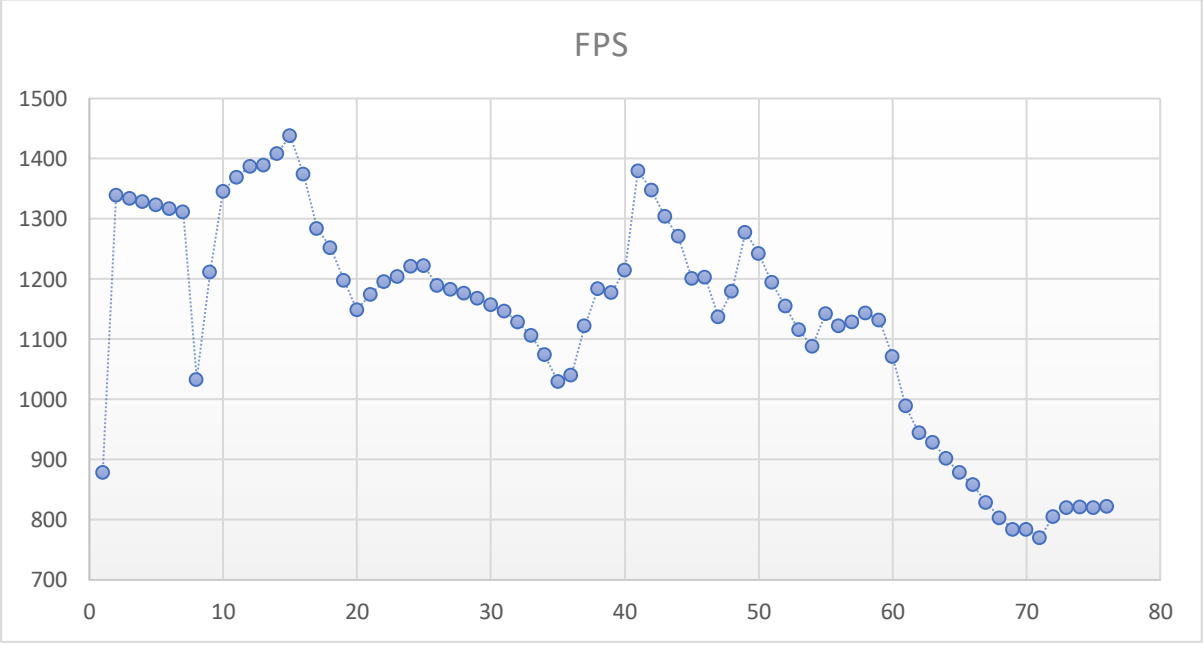
7 Výsledky práce

Tato kapitola se věnuje prezentaci finálního výsledků práce a zhodnocení rychlosti vykreslení grafického intra.

7.1 Zhodnocení rychlosti vykreslení

Grafické intro bylo vyvinuto a otestováno na stolním počítači s operačním systémem Windows 10 v 64 bitové verzi, osazeném procesorem Intel® Core™ i5-3470 společně s grafickou kartou AMD® RX 470. Pro měření byla použita neregistrovaná verze programu Fraps 3.5.99.

Výsledky měření můžeme vidět v grafu 7.1. Při rozlišení 1440p je na testovací sestavě dosaženo průměrné snímkové frekvence 1132.466 snímků za sekundu, s přijatelným minimem 769. Z toho usuzujeme že by intro mělo běžet plynule i na málo výkonných počítačích. Tyto hodnoty jsou také dostatečně vysoké, abychom mohli intro dále rozšiřovat, např. o částicový systém. Snímková frekvence převážně závisí na počtu objektů ve scéně, jak intro probíhá, roste počet objektů ve scéně a náročnost vykreslení se tak zvyšuje. Tím postupně klesá snímková frekvence. Za zmínku stojí nápadný propad v 8 sekundě intra, kdy se spustí hudba. Ten je způsoben, tím že intro zapisuje hudbu k přehrání do dočasného souboru.

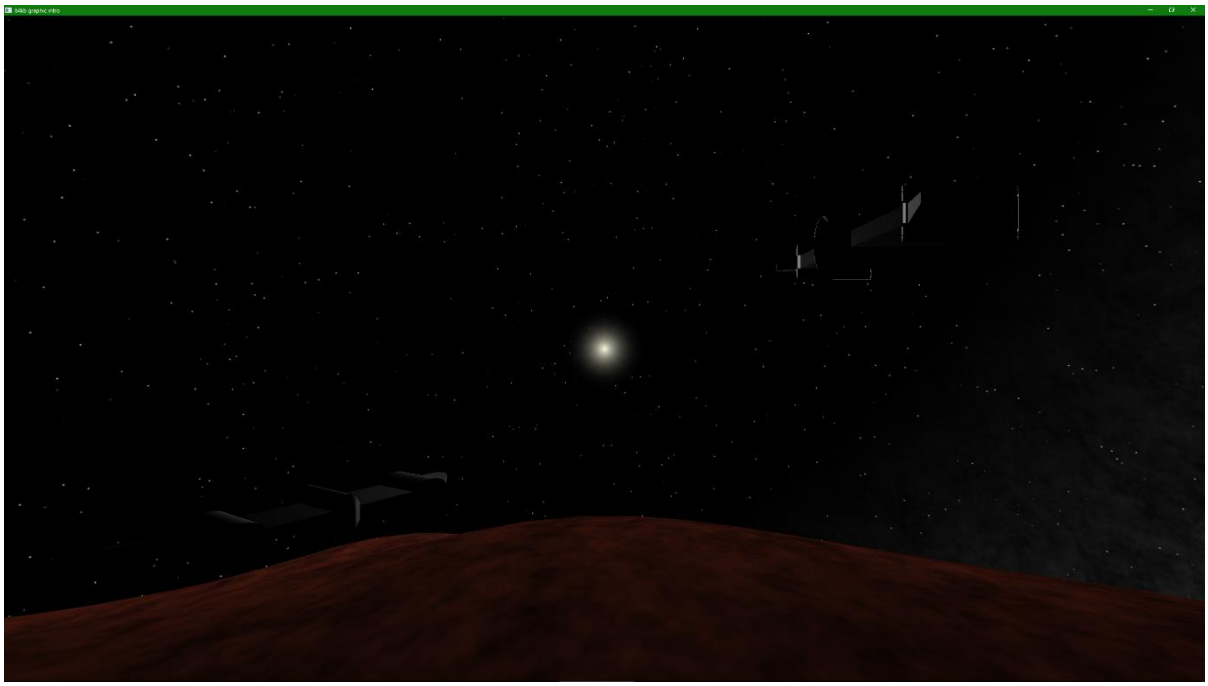


7.1 Graf závislosti snímkové frekvence na časovém průběhu intra

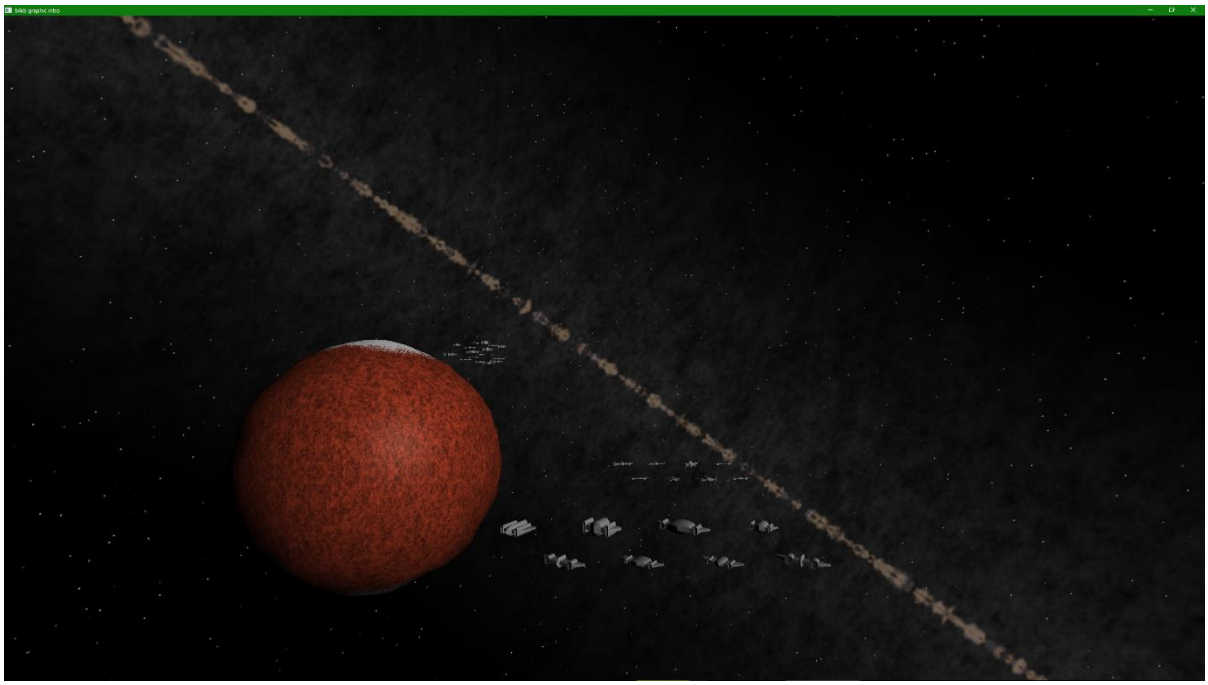
7.2 Výsledné grafické intro

Výsledné grafické intro má 29,1kB a přehrání trvá přibližně 75 sekund. Skládá se ze čtyř scén, v první scéně pozorujeme vycházející slunce nad Marťanskou krajinou (Obrázek 7.2), krátce poté se z povrchu planety začnou zvedat mimozemská vesmírná plavidla. V další scéně vidíme formace vesmírných lodí, jak odlétají z Marsu (Obrázek 7.3). Následuje krátká scéna, v které plavidla letí volně vesmírem.

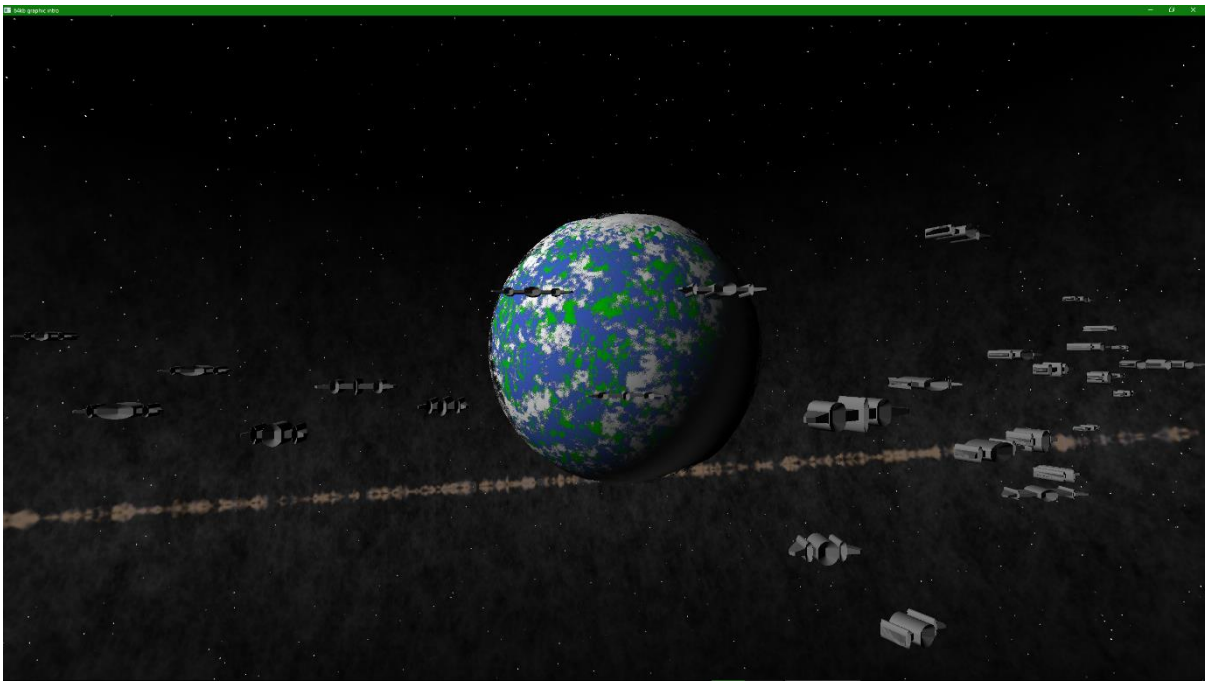
V poslední scéně vidíme mimozemské lodě, jak pomalu obkličují bezbrannou planetu Zemi (Obrázek 7.4).



Obrázek 7.2 První scéna výsledného intra



Obrázek 7.3 Druhá scéna výsledného intra



Obrázek 7.4 Poslední scéna výsledného intra

8 Závěr

Cílem této práce bylo vytvoření grafického intra s omezením výslednou velikostí pod 64kB. Tento cíl byl splněn, výsledné grafické intro má velikost pouhých 29,1kB. Dalším požadavkem bylo použití grafické API OpenGL, s tou jsem měl pouze minimální předchozí zkušenosti a bylo proto nutné se s ní podrobně seznámit.

Pro tvorbu intra bylo také zapotřebí nastudovat různé techniky procedurální generace, textury jsou generovány pomocí Perlinova šumu. Pro generování modelů vesmírných lodí byly inspirací L-systémy. Okolí scény je obklopeno skyboxem. Její nasvícení je realizováno pomocí Phongova osvětlovacího modelu. Pro ozvučení byl použit soubor ve formátu MIDI.

V projektu lze nadále pokračovat, pro rozšíření je dostatek prostoru a intro běží na vysokých snímkových frekvencích. Vhodné by bylo například použít částicové systémy pro zobrazení běhu motorů vesmírných lodí. Další možnost je dále rozvést mimozemský útok na Zemi, například zobrazit vystřelení rakety a následně animovat výbuch planety. Dalším možným krokem by bylo zapracování pokročilejších grafických technik jako je teselace, pokročilé stínování, bump mapping, lens flare a další.

Při vývoji grafického intra jsem získal spoustu nových znalostí z oblasti tvorby grafických aplikací pomocí knihovny OpenGL a procedurálního generování. Dále jsem se důkladně seznámil s nastaveními překladače a možnostmi tvorby programů s omezenou velikostí.

Vývoj intra mi pomohl rozvinout své programátorské schopnosti pro vývoj grafických aplikací a byl pro mě cennou zkušeností.

Literatura

[1] Reunanen, M.: How Those Crackers Became Us Demosceners. [Online; navštívené 04.05.2018].
URL http://widerscreen.fi/numerot/2014-1-2/crackers-became-us-demosceners/

[2] Revision: General Information. 2017. [Online; navštívené 04.05.2018].
URL https://2017.revision-party.net/about/general

[3] Segal, M.; Akeley, K.: The OpenGL® Graphics System: A Specification (Version 3.3 (Core Profile)). 2010. [Online; navštívené 04.05.2018].
URL https://www.khronos.org/registry/OpenGL/specs/gl/glspec33.core.pdf

[4] ARM: ARM® Guide to Unity: Enhancing Your Mobile Games. 2014. [Online; navštívené 04.05.2018].
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.100140_0100_00_en/nic1408619466543.html

[5] Kessenich, J.; Baldwin, D.; Rost, R.: The OpenGL Shading Language. 2010. [Online; navštívené 28.04.2017].
URL https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.3.30.pdf

[6] Phong, B. T.: Illumination of Computer-Generated Images. Disertační práce, University of Utah, UTEC-CSs-73-129, 1973.

[7] Eck, D. J.: Introduction to Computer Graphics. 2018. [Online; navštívené 04.05.2018].
URL http://math.hws.edu/graphicsbook/c7/s2.html

[8] Smith, B.: Phong components. 2006. [Online; navštívené 04.05.2018].
URL https://commons.wikimedia.org/wiki/File:Phong_components_version_4.png

[9] Vries, J.: Coordinate Systems. 2014. [Online; navštívené 04.05.2018].
URL https://learnopengl.com/Getting-started/Coordinate-Systems

[10] Azerdark: XNA – Skydome. 2010. [Online; navštívené 04.05.2018].
URL https://azerdark.wordpress.com/2010/01/31/skydome/

[11] Kapadia, A.: Linear Congruential Generators. 2001. [Online; navštívené 04.05.2018].
URL https://www.cs.indiana.edu/~kapadia/project2/node7.html

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

[13] Lindenmayer, A.: Mathematical models for cellular interactions in development. I. Filaments with one-sided inputs. Journal of theoretical biology, 18.3.1968. s. 280-99.

[14] Amir, S.; Mohamed, N. S.; Hashim Ali, S. A.: Ion Conductive Polymer Electrolyte Membranes and Fractal Growth. 2012. DOI: 10.5772/26922.

[15] Gumster, J.: Modelling an eye. 2015. [Online; navštívené 04.05.2018].
URL http://www.blenderbasics.com/post/modeling-an-eye/

[16] Stroustrup, B.: The C++ Programming Language. 4th edition. vyd. Addison-Wesley, 2013. ISBN 978-0-321-56384-2, 1368 s.

[17] Nehe: Creating an OpenGL Window (Win32). 2000. [Online; navštívené 04.05.2018].
URL http://nehe.gamedev.net/tutorial/creating_an_opengl_window_(win32)/13001/

[18] Kholodov, I.: Matrices. [Online; navštívené 04.05.2018].
URL http://www.c-jump.com/bcc/common/Talk3/Math/Matrices/Matrices.html

[19] Badiou, V.: View Matrix. 2015. [Online; navštívené 04.05.2018].
URL http://in2gpu.com/2015/05/17/view-matrix/

[20] Badiou, V.: Projection Into the Matrix. 2015. [Online; navštívené 04.05.2018].
URL http://in2gpu.com/2015/05/23/enter-the-matrix-and-projection/

[21] Renk, T.: From random number to texture - GLSL noise functions, 2015. [Online; navštívené 04.05.2018].
URL http://www.science-and-fiction.org/rendering/noise.html#sparse_dot

[22] Vivo, P. G.; Love, J.: The Book of Shaders: Shapes. 2015. [Online; navštívené 04.05.2018].
URL https://thebookofshaders.com/07/

[23] Brickboy2x2: Mars Attacks Theme. 2017. [Online; navštívené 04.05.2018].
URL https://musescore.com/user/20941876/scores/4820601

[24] Microsoft: Playing a MIDI File. [Online; navštívené 04.05.2018].
URL https://msdn.microsoft.com/en-us/library/windows/desktop/dd743673(v=vs.85).aspx

[25] Microsoft: Compiler Options Listed by Category. 2015. [Online; navštívené 10.04.2017].
URL https://msdn.microsoft.com/en-us/library/19z1t1wy(v=vs.120).aspx

[26] Microsoft: Linker Options. 2015. [Online; navštívené 10.04.2017].
URL https://msdn.microsoft.com/en-us/library/y0zzbyt4(v=vs.120).aspx

Přílohy

Příloha A

DVD

Na přiloženém DVD jsou uloženy veškeré zdrojové soubory, návod na překlad, výsledné intro, soubor README a video s intrem. Zde je uveden přesný popis adresářové struktury, který je uložena na DVD disku:

- **packer/** Použitý exe packer
- **pdf/** Tento soubor PDF
- **result/** Výsledné intro se souborem README
- **src/** Zdrojové soubory grafického intra
- **video/** Video se záznamem intra
- **word/** Zdrojový tvar písemné zprávy
- **word/img/** Obrázky použité při tvorbě písemné zprávy