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 MAREK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Marek 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 2016
Datum odevzdání: 17. května 2017

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

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá tvorbou grafického intra s omezenou velikostí. Zabývá se metodami využitými pro jeho tvorbu. Popisuje metody pro snížení velikosti spustitelného souboru. Mezi ty se řadí například využití procedurálního generování a využití exe packeru. Výsledkem je grafické intro zobrazující start rakety s velikostí nepřesahující 64 kB.

Abstract

This bachelor's thesis deals with the creation of a graphics intro with a limited size. It deals with methods used for its creation. The thesis describes methods used for reduction of size of executable files. Among them for example the use of prcedural generation and the use of exe packers. The created graphics intro has a size smaller than 64 kB.

Klíčová slova

grafické intro, 64kB, OpenGL, GLSL, procedurální generování, komprese, Phongův osvětlovací model, částicové systémy, billboarding, skybox, Perlinův šum

Keywords

graphics intro, 64kB, OpenGL, GLSL, procedural generation, compression, Phong lighting model, particle systems, billboarding, skybox, Perlin noise

Citace

MAREK, Jan. *Grafické intro 64kB s použitím OpenGL*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Grafické intro 64kB s použitím OpenGL

Prohlášení

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

.....
Jan Marek
17. května 2017

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

Kapitola 1

Úvod

Grafické demo je program, jehož primárním účelem je vytvořit co nejefektnější scénu s pomocí programu s minimální velikostí. Demo by mělo ukázat na programátorské a umělecké schopnosti tvůrce. Grafická intra mohou nabývat různých podob a velikostí, nejběžnějším limitem velikostí je 64 kibibytů. Tato velikost je jako limit nastavena, protože to byla maximální dosažitelná velikost spustitelných souborů typu COM.

1.1 Historie grafických inter

Grafická dema sama o sobě vznikla v osmdesátých letech minulého století, kdy byla využívána jako podpis crackerských skupin. Tyto skupiny připojovaly dema ke hrám, u kterých jimi byla prolomena ochrana. Odtud také pochází důvod jejich limitované velikosti, pokud byla na daném přenosovém médiu data hry, tak už byla značná část prostoru zaplněna. Kvalita výsledného dema byla pro hackerské skupiny otázkou cti a jednotlivé skupiny se předháněly ve využití technik k vytvoření co nejefektnějšího dema. V současnosti se grafická dema stala součástí programátorské subkultury a jednotliví programátoři spolu soutěží na tzv. demoparties v různých kategoriích velikostí dem.

1.2 Cíl práce

Cílem této práce je vytvořit grafické intro o maximální velikosti 64 kibibytů. Intro se bude skládat ze dvou scén. V první scéně bude zobrazen start rakety ze země a druhá scéna se bude zabývat oddělením jednotlivých stupňů rakety a pokračováním letu rakety na jinou planetu. Velikosti menší než 64 kibibytů bude dosaženo využíváním procedurálního generování textur a modelů. Dále v této práci budou popsány techniky využité pro tvorbu dem, bude popsán a zhodnocen stav výsledného programu.

Kapitola 2

Použité techniky

V této kapitole jsou popsány techniky využívané při tvorbě grafického intra s omezenou velikostí. Dále jsou zde popsány metody využité pro osvětlení generované scény.

2.1 OpenGL

Open Graphics Library je multiplatformní programovací rozhraní, které umožňuje renderování 2D a 3D grafiky. Poskytuje funkce, které umožňují programům pracovat s grafickou kartou počítače. Data pro vykreslování jsou nahrávána do bufferů alokovaných v paměti grafické karty a samotné vykreslování je ovládáno pomocí shaderů napsaných v jazyce GLSL. V následující části jsou popsány základní prvky OpenGL využité v této práci.

2.1.1 Buffery

Vertex buffery, tzv. VBO, jsou objekty OpenGL, které slouží k ukládání neformátovaných dat v paměti grafické karty. Tato data mohou obsahovat informace o geometrii objektů, normálové vektory jendotlivých vrcholů, texturovací souřadnice a další data. Při vytváření bufferu je nutné do něj nahrát data a následně předat OpenGL informace o tom, jakými datovými typy má tato data reprezentovat. V praxi jsou jednotlivé VBO agregovány do tzv. vertex array objektu, ve kterém je možné držet referenci na několik vertex bufferů a pracovat s nimi hromadně.

2.1.2 Shadery

Shadery jsou programy, které jsou vykonávány na grafické kartě zřetězeně za sebou. Každý shader má specifickou funkci. Slouží pro zpracování vstupních dat před vykreslením snímku. Ve výsledném demu jsou využívány dva typy shaderů a to: vertex shader a fragment shader.

Vertex shader je odpovědný za zpracování dat o vrcholech polygonů nahraných do paměti grafické karty. Může s využitím transformačních matic měnit jejich pozice.

Fragment shader udává výslednou barvu zpracovaného fragmentu. Je vykonáván po vertex shaderu a přijímá od něj data, která jsou pro každý vykreslovaný fragment interpolována. Barva fragmentu je většinou určena vzorkováním z textury za využití texturovacích souřadnic.

2.1.3 GLSL

OpenGL Shading Language je vysokoúrovňový programovací jazyk určený pro tvorbu shaderů. Jeho syntaxe je velmi podobná jazyku C. Definuje speciální datové typy vhodné pro využití v shaderech, například umožňuje pracovat s vektory a maticemi, zároveň definuje matematické operace nad nimi. Programy shaderů jsou kompilovány za běhu programu pro zajištění přenositelnosti softwaru a zaručení jeho funkčnosti na grafických kartách různých výrobců. Výhodou provádění programů na grafické kartě je využívání paralelního zpracování dat, tím je dosaženo výrazného zrychlení oproti provádění takových programů na procesoru počítače.

2.2 Procedurální generování

Procedurální generování je metoda vytváření dat za pomoci algoritmů. Je využíváno v počítačových hrách pro vytváření velkých unikátních náhodných textur a modelů. V demoscéně je využíváno pro limitování výsledné velikosti spustitelného souboru. Za pomoci technik procedurálního generování lze vytvořit velké modely a textury s malým množstvím kódu. Zároveň je možné výsledek změnit pouze úpravou vstupů algoritmů.

2.2.1 Generování náhodných čísel

V technikách procedurálního generování jsou hojně využívány generátory pseudonáhodných čísel. Jejich využití je dvojí, první z nich je využití náhodných čísel pro generování takových dat, která jsou pokaždé jiná. Tento princip se uplatňuje především v počítačových hrách, kde umožňuje vytvářet unikátní herní světy. Příkladem procedurálního generování ve hrách může být například hra Dwarf Fortress ve které je procedurální generování využito pro tvorbu unikátních světů.

Druhý způsob využití je generování stále stejných sérií dat pomocí generátoru náhodných čísel. Tento způsob využívá faktu, že generátor pseudonáhodných čísel pro stejnou zdrojovou hodnotu, tzv. *seed*, vytvoří vždy stejnou posloupnost čísel. Tato jejich vlastnost je pro jiná použití obvykle nežádoucí a obvykle se do hodnoty seed vkládá současný čas, či jiná měnící se hodnota. V oblasti grafických inter je obvykle využíván konstantní seed, tím je dosaženo generování stejných modelů a textur při každém spuštění programu.

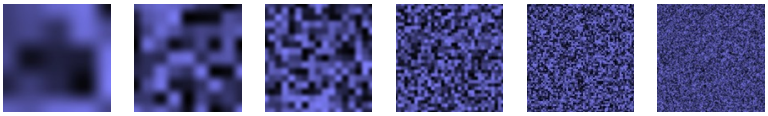
Mezi nejstarší a nejdéle studované generátory pseudonáhodných čísel lze zařadit lineární kongruentní generátory [5].

$$x_{n+1} = (Ax_n + B) \mod M \tag{2.1}$$

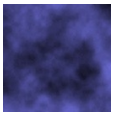
Lze je popsat rovnicí 2.1, kde x_n je předchozí generovaná hodnota, x_{n+1} je nová pseudonáhodná hodnota, x_0 je hodnota seed. Je nutné, aby čísla B a M byla nesoudělná a $A-1$ musí být dělitelné všemi prvočíselnými děliteli čísla M.

2.2.2 Interpolace

Pro vyhlazování nespojitých náhodně generovaných vzorků dat bývá často využívána interpolace. V počítačové grafice využití interpolace pomáhá k dosažení plynulosti zobrazovaných objektů. Vzorce využívané pro interpolaci vyžadují vstupní body mezi kterými má být nalezena hodnota. Dále je pro získání interpolantu nutné znát pozici mezi danými body, pro kterou bude získána hodnota.



Obrázek 2.1: Funkce šumu o různých frekvencích a amplitudách, převzato z [3]



Obrázek 2.2: Výsledný šum vzniklý kombinací šumů na obrázku 2.1, převzato z [3]

Mezi nejběžnější metody interpolace patří lineární interpolace, která vytváří výslednou křivku za pomoci vkládání úseček mezi jednotlivé body. Její výhodou je velmi nízká výpočetní složitost, nevýhodou je to, že lineární interpolace neprodukuje plynulé křivky. Lineární interpolace je popsána rovnicí 2.2. Hodnota t určuje pozici mezi hodnotami y_1 a y_2.

$$y = (1 - t)y_1 + ty_2 \tag{2.2}$$

Další metodou interpolace, která je často využívána je kosinová interpolace, ta produkuje podstatně hladší interpolační křivku za cenu zvýšené výpočetní náročnosti. Je popsána rovnicí 2.4.

$$t_2 = \frac{1 - \cos(t\pi)}{2} \tag{2.3}$$

$$y = (y_1(1 - t_2) + y_2 t_2) \tag{2.4}$$

Hodnoty mohou být také interpolovány za pomoci splinů. Spliny dosahují z jmenovaných metod interpolace nejlepšího vyhlazení za cenu největší výpočetní složitosti. V grafických intrech jsou využívány především pro ovládání pohybů kamery, nebo pohybu objektů ve scéně.

2.2.3 Perlinův šum

Textury vygenerované pouze pomocí generátoru pseudonáhodných čísel často vypadají nepřirozeně, proto je pro tyto účely často využíván Perlinův šum. Jedná se o procedurální šum, který byl vytvořen Kenem Perlinem v roce 1985[7]. Umožňuje vytvářet přirozeně vypadající textury a modely s poměrně nízkou výpočetní náročností.

Princip Perlinova šumu spočívá ve vytvoření několika šumů s různými frekvencemi a amplitudami, takzvaných oktáv, a jejich kombinaci. Každá oktáva má dvojnásobnou frekvenci než předchozí, pro vyhlazení šumů je využívána interpolace mezi jednotlivými náhodně generovanými hodnotami. Pro výpočet amplitudy šumu dané oktávy se využívá hodnota zvaná perzistence, ta ovlivňuje rychlost klesání amplitudy pro jednotlivé oktávy. Výpočet frekvence oktávy i je vyjádřen vzorcem 2.5 a výpočet amplitudy vzorcem 2.6. Hodnoty perzistence se mohou pohybovat v intervalu $(0, 1\rangle$, při zvolení nízké hodnoty perzistence je výsledkem hladší šum.

$$f = 2^i \tag{2.5}$$

$$a = p^i \tag{2.6}$$

Výsledný šum je získán součtem hodnot jednotlivých šumů. Podoba výsledku závisí na zvolených frekvencích, perzistenci a počtu použitých oktáv.

2.2.4 Generování za pomoci matematických vzorců

Některé modely, například ty, které jsou blízké geometrickým tvarům, lze generovat jednoduše pouze za pomoci matematických vzorců.

2.3 Osvětlovací model

Osvětlovací modely jsou v počítačové grafice využívány k určení toho, jakým způsobem se v daném místě odráží od objektu světlo. Osvětlovací modely dodávají renderovaným scénám hloubku. Pokud by nebyl žádný osvětlovací model využit, vykreslená scéna by byla naprosto plochá.

2.3.1 Lambertův osvětlovací model

Lambertův osvětlovací model je jedním ze základních osvětlovacích modelů, popisuje odrazy světla od matných materiálů, tzv. difuzní odrazy. Sílu odráženého světla lze získat pomocí vzorce 2.7, kde I_D značí výslednou intenzitu světla, L je normalizovaný vektor směřující ke zdroji světla, N je normálový vektor povrchu, K_D je odrazivost povrchu a I_L je intenzita světla.

$$I_D = (L \cdot N) K_D I_L \tag{2.7}$$

Tento model nepočítá s lesklými povrchy, z toho důvodu není běžně používán samostatně, ale jen jako součást jiných osvětlovacích modelů.

2.3.2 Phongův osvětlovací model

Phongův osvětlovací model byl vytvořen Bui Tuong Phongem v rámci disertační práce na University od Utah v roce 1975 [8]. Phongův osvětlovací model je aproximací reálného chování lesklých objektů. Není sice úplně realistický, ale je dostačující pro většinu použití. Díky své jednoduchosti a rychlosti je využíván v grafických aplikacích dodnes. Celkové odrážené světlo se skládá ze tří složek: ambientní, difuzní a spekulární, lze jej vyjádřit vzorcem 2.8.

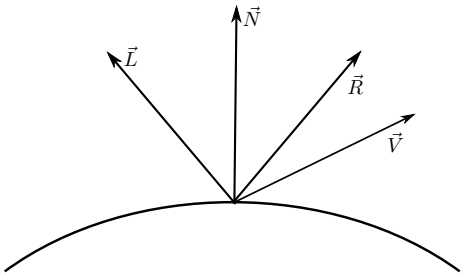
$$I = L_A + L_D + L_S \tag{2.8}$$

Pro výpočet se využívají vektory zobrazené na obrázku 2.3. Vektor $\vec{L}$ směřuje ke zdroji světla, $\vec{N}$ je normálový vektor povrchu, $\vec{R}$ vyjadřuje odraz od zdroje světla a $\vec{V}$ směřuje k pozorovateli.

Ambientní složka

Ambientní složka je vyjádřena vzorcem 2.9, kde I_A vyjadřuje intenzitu ambientního světla a k_A je odrazivost materiálu pro ambientní složku. Tato složka vyjadřuje rozptýlené světlo, které dopadá na všechny strany objektu rovnoměrně.

$$L_A = I_A k_A \tag{2.9}$$



Obrázek 2.3: Vektory v Phongově osvětlovacím modelu

Difuzní složka

Difuzní složka vyjadřuje matné odlesky objektů ve scéně. Tato složka zobrazuje to, že plochy, které jsou přivrácené ke zdroji světla rozptylují více světla do okolí, než ty, které jsou vzhledem ke zdroji světla nakloněné. Lze ji vyjádřit vzorcem 2.10, kde I_D je intenzita difuzního světla a k_D je koeficient odrazivosti pro difuzní světlo.

$$L_D = (\vec{N} \cdot \vec{L}) I_D k_D \tag{2.10}$$

Spekulární složka

Spekulární složka určuje intenzitu odrazů světla od zdroje směrem k pozorovateli. Je vyjádřena vzorcem 2.11, kde I_S je intenzita spekulárního světla, k_S je koeficient odrazivosti pro daný materiál a n je exponent určující lesklost materiálu.

$$L_S = (\vec{V} \cdot \vec{R})^n I_S k_S \tag{2.11}$$

Phongovo stínování

Phongovo stínování bylo vyvinuto zároveň s osvětlovacím modelem. Princip jeho fungování spočívá v tom, že se pro každý vykreslovaný bod interpoluje normála z normál vrcholů polygonu na kterém se nachází. Tato technika umožňuje vykreslit objekty tak, aby vypadaly hladce a zaobleně i přes to že se skládají z plochých polygonů. Phongovo stínování je dnes již standardní funkcí knihoven pro vykreslování 3D grafiky.

2.4 Skybox

Použití skyboxu je metoda pro vytvoření dojmu velikosti vykreslované scény, reprezentuje okolní prostředí, obvykle oblohu, mraky a okolní terén. Většinou je realizována za pomoci jednotkové krychle, na kterou je namapována textura skyboxu, tzv. *cubemap*. Textura skyboxu obsahuje pohled do jednotlivých směrů okolí, je důležité, aby na sebe textury jednotlivých stran krychle plynule navazovaly. Textura skyboxu musí být také v bodech blížících se k hranám a vrcholům krychle deformována tak, aby nebylo znatelné, že se jedná o texturu namapovanou na krychli. Příklad textury skyboxu je ukázán na obrázku 2.4.

Obrázek 2.4: Textura pro skybox, převzato z [6]

Skybox je vždy vykreslen před ostatními objekty scény a pohybuje se s kamerou tak aby byla kamera vždy ve středu krychle. Tímto postupem je dosaženo efektu rozlehlosti a realističnosti scény. V současnosti jsou využívány i tzv. skydomy, které místo krychle využívají polokouli, na kterou je textura skyboxu namapována. Jejich výhodou oproti tradičním skyboxům je jednodušší tvorba textur a animovaných textur, nevýhodou je vyšší počet polygonů a tím pádem i vyšší výpočetní náročnost jejich zobrazování.

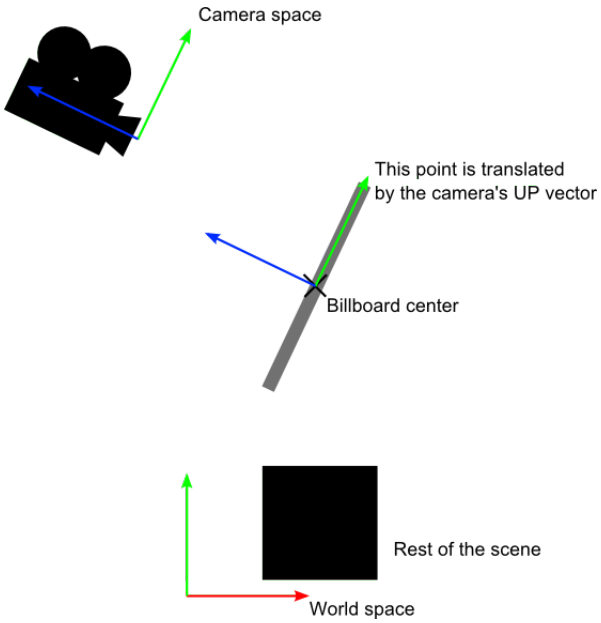
2.5 Částicové systémy

Částicové systémy nachází v grafice dvojí využití, první z nich je pro simulaci kouře, či ohně. V tomto případě je daný jev reprezentován velkým množstvím částic u kterých je simulována fyzika. Částice je vytvořena v tzv. emitoru, je jí přiřazena doba života, počáteční pozice, rychlost a malé množství náhodné síly působící různými směry. Poté je při každém vykreslení scény její pozice přepočítána za pomoci parametrů vzniklých při jejím stvoření a dalších vlivů, například gravitační síly. Po uplynutí doby života částice přestává existovat. Částice jsou obvykle vykreslovány s pomocí techniky billboardingu.

Druhé využití nalézají částicové systémy v tvorbě procedurálních textur. V tomto případě jsou částice vytvářeny stejným způsobem, jen je simulována celá jejich doba života. Stopa částice je vykreslena do textury a ta je poté použita při vykreslování scény.

2.6 Billboarding

Billboarding je v počítačové grafice využíván pro reprezentaci třídimenzionálních jevů pomocí dvoudimenzionálních objektů. Billboard je otexturovaný plochý objekt skládající se obvykle z malého množství polygonů. Jeho pozice je v každém snímku upravována tak aby byl natočen směrem ke kameře, to tvoří dojem trojrozměrnosti objektu, chování billboardů vůči kameře je zobrazeno na obrázku 2.5. V dřívějších dobách, kdy měly počítače nízký výpočetní výkon, byly billboardy využívány k zobrazování objektů, které by byly svým počtem polygonů příliš náročné, například stromů, nebo i nepřátel v počítačových



Obrázek 2.5: Orientace billboardu vůči kameře, převzato z [1]

hrách. V současnosti jsou využívány k reprezentaci složitých třídimenzionálních objektů, jako například kouře, či ohně.

Slovem billboard se často označují i objekty, které technicky billboardy nejsou. Například ploché objekty které se neotáčí směrem ke kameře, ale zůstávají statické. Tímto způsobem se často vytváří například tráva na povrchu terénu.

Kapitola 3

Implementace

Tato kapitola se zabývá implementací výsledného dema a popisuje využití dříve popsaných technik při generování objektů ve scéně.

3.1 Využité knihovny

Projekt byl implementován v jazyce C++, ale většina kódu byla napsána s využitím funkcí dostupných i v jazyce C. Pro vývoj bylo zvoleno prostředí Visual Studio Code. Hlavním z důvodů pro volbu tohoto prostředí byla dobrá integrace debuggeru gdb. Pro překlad práce byl zvolen překladač MinGW g++, což je port GNU verze g++ pro operační systém Windows.

Pro vytvoření okna s demem bylo použito WinAPI. Dále byla využita knihovna glm, která umožňuje používat v C++ datové typy z jazyka GLSL. Knihovna glm implementuje vektory a matice a operace nad nimi, tím výrazně zjednodušuje práci. Tato knihovna zároveň poskytuje další funkce nad rámec GLSL, například tvoření View matice a další.

Pro zpřístupnění novějších rozšíření OpenGL nad verzi 3.0 je nutné načíst funkce, které je umožňují používat. V programu, který by nebyl limitován velikostí, by bylo možné využít například knihovnu GLEW. Bohužel tato knihovna výrazně zvětší velikost výsledného spustitelného souboru. Z toho důvodu byly jednotlivé funkce manuálně načítány za pomoci funkce `wglGetProcAddress`, ta vrací ukazatel na danou funkci z rozšíření OpenGL. Například funkci pro nahrání hodnoty s plovoucí desetinnou čárkou do uniformní proměnné lze získat takto:

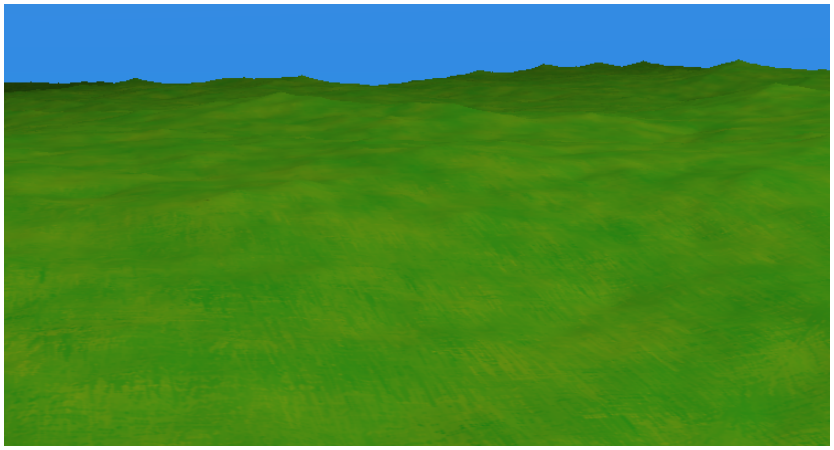
```
glUniform1f = (PFNGLUNIFORM1FPROC)wglGetProcAddress("glUniform1f");
```

3.2 Terén

3.2.1 Generování terénu

Terén je vytvářen za pomoci 2D Perlinova šumu, vygenerovaný šum je použit jako výšková mapa pro generovaný terén. Argumenty pro šum jsou nastaveny tak, aby byl generován mírně kopcovitý terén s množstvím malých hrbolků.

Implementace Perlinova šumu využívá pro interpolaci hodnot šumu metodu lineární interpolace. S využitím kosinové, či kubické interpolace by bylo možné dosáhnout vlnitého terénu s menším počtem hrbolků. Využití těchto interpolačních technik se ukázalo jako nevhodné z důvodu neúměrně zvýšené výpočetní náročnosti. Terén je generován pomocí



Obrázek 3.1: Otexturovaný terén

trojúhelníkové sítě, tak aby se střed terénu nacházel ve středu souřadného systému. Pro každý vrchol polygonu je za hodnotu souřadnice na ose y zvolena hodnota z výškové mapy.

Po vygenerování geometrie terénu jsou pro každý bod vypočítány normálové vektory pro využití v osvětlovacím modelu. Pro výpočet normál jsou vytvořeny dva vektory vedoucí z jednoho z vrcholů ke zbývajícím dvěma. Pomocí těchto dvou vektorů je vektorovým součinem vypočítána normála. Normála je nastavena celému polygonu. Poté jsou pro každý vrchol vytvořeny nové normály zprůměrováním hodnot normál sousedních polygonů. Dále jsou vypočítány uv souřadnice pro texturování tak, aby se textura na celém terénu šestnáctkrát zopakovala. Otexturovaný a vystínovaný terén je zobrazen na obrázku 3.1.

3.2.2 Generování textury terénu

Při generování textury terénu je nejprve spočítán Perlinův šum, který ovlivňuje barvu trávy v daném bodě. Na texturu je nejprve nanesena jednolitá zelenohnědá barva. Poté jsou generovány jednotlivá stébla trávy. Každému stéblu je vygenerován náhodný směr, tímto směrem je poté vykreslena čára s využitím algoritmu DDA. Pokud stéblo začne v průběhu vykreslování přesahovat přes okraj textury, pokračuje stéblo stejným směrem na jejím druhém konci. Tím je dosaženo toho, že texturu lze na terénu opakovat bez toho, že by na ní byly viditelné hrany. Textura vytvořená tímto způsobem je podobná efektu, který vznikne při malbě štětcem.

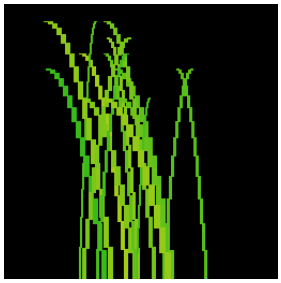
Opakování textury na geometrii terénu je dosaženo tím, že je jí při nahrávání do GPU nastavena vlastnost opakování v horizontálním směru následujícím příkazem, pro opakování ve vertikálním směru je tento příkaz zopakovat znovu s jinak zvoleným druhým argumentem.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

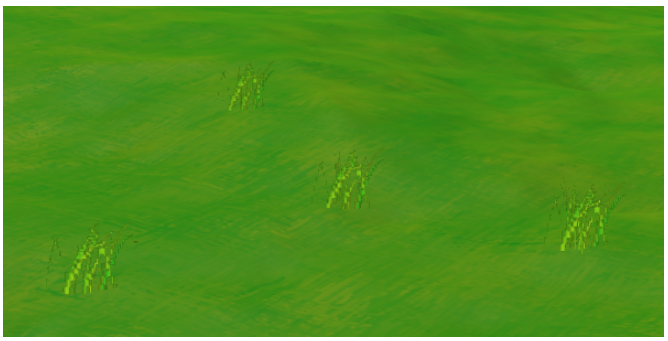
3.3 Tráva

3.3.1 Generování textury trávy

Pro generováni textury trávy byl využit částicový systém. Pro každé stéblo trávy je vygenerována jedna částice. Pro tu je vygenerován vektor náhodné síly, který ovlivňuje směr letu



Obrázek 3.2: Textura trávy vygenerovaná pomocí částicového systému



Obrázek 3.3: Trsy trávy umístěné na terénu

částice. Náhodná síla je přičtena k vektoru rychlosti, který je pro všechny částice stejný a směřuje vzhůru. Poté je odkrokován pohyb částice, v každém kroku je na částici aplikována gravitační síla. Simulace částice pokračuje dokud je vertikální složka jejího vektoru rychlosti větší než nula. Stébla jsou vykreslena za pomoci obdélníků, které jsou tvořeny podél trajektorie částice. Barva právě kresleného stébla je vybrána z gradientu zelených barev dle hodnoty jednodimenzionálního Perlinova šumu. Vygenerovaná textura je zobrazena na obrázku 3.2.

Na obrázku můžete vidět, že značná část textury je zabrána černou barvou, ve skutečnosti se jedná o průhlednou černou barvu, tzn. složka alpha je nastavena na nulovou hodnotu. Během aplikace této textury na model je ve fragment shaderu dosaženo průhlednosti černé části textury zahozením těch fragmentů, které jsou průhledné. Vykresleny jsou jen fragmenty, které reprezentují jednotlivá stébla. Zahazování fragmentů je dosaženo využitím následujícího programu pro fragment shader:

```
void main()
{
    vec4 col = texture(GrassTextureSampler, uv);
    if(col.a < 0.5) discard;
    color = col;
}
```

3.3.2 Umístění trávy

Textura trávy je namapována na model sestávající se ze dvou čtverců, které se uprostřed protínají, tento model je ale třeba umístit několikrát do scény. Jednotlivá umístění trávy

jsou vybrána hned po vygenerování terénu. O výběr jednotlivých umístění se stará funkce `PlaceGrass`, té je předána geometrie terénu. Funkce náhodně vybere n vrcholů polygonů na které bude tráva umístěna. Umístění jsou vybírána tímto způsobem z toho důvodu, aby bylo dosaženo faktu, že tráva musí vždy růst přímo z terénu. „Vznášející“ se trsy trávy by působily nerealisticky a kazily by celkový dojem z dema. Jednotlivé trsy trávy umístěné na terénu jsou zobrazeny na obrázku 3.3. Trsy trávy jsou poté vykresleny na terén pomocí instancingu. Využití instancingu značným způsobem redukuje výpočetní náročnost vykreslování velkého množství objektů. V případě výsledného dema se jedná o tisíc instancí. Všechny trsy trávy jsou vykresleny využitím následujícího příkazu:

```
glDrawArraysInstanced(GL_TRIANGLES, 0, GRASS_VERTEX_COUNT, 1000);
```

3.4 Raketa

Raketa je hlavním bodem ve výsledném intru, skládá se ze dvou stupňů. Ty jsou v první fázi vykreslovány jako jeden model. V druhé fázi intra, po opuštění zemského povrchu na oběžné dráze, dochází k oddělení jednotlivých stupňů a odletu druhého stupně směrem k jiné planetě.

3.4.1 Generování rakety

Tvorba modelu rakety využívá faktu, že raketa tvarem připomíná válec. Je generována pomocí matematického vzorce, ve kterém je s měnící se výškou upravován poloměr válce. Každý bod je generován podle následujících vzorců. Úhel θ je nejdříve vypočítán za pomoci vzorce 3.1, kde $w_{\max}$ označuje celkový počet polygonů po obvodu válce a w_i značí současný polygon. Ve vzorcích 3.2 a 3.4 značí r poloměr válce. Souřadnice na ose y je vypočítána pomocí vzorce 3.3, kde h_i značí pořadí kroku po ose y a k je velikost kroku, odečtením hodnoty 0.5 je celý model posunut tak aby byl vycentrovaný. V programu jsou zvoleny hodnoty $h_{\max} = 200$ a $w_{\max} = 100$.

$$\theta = 2\pi \frac{w_i}{w_{\max}} \tag{3.1}$$

$$x = r\cos(\theta) \tag{3.2}$$

$$y = h_i k - 0.5 \tag{3.3}$$

$$z = r\sin(\theta) \tag{3.4}$$

Během tvorby geometrie pro raketu je poloměr válce měněn tak, že výsledný válec má tvar rakety. Pro tvar hlavice rakety je použit tzv. *power series* vzorec. Tvar vznikne rotací parabolické křivky popsané vzorcem 3.5, kde R je konečný poloměr hlavice a L je délka hlavice. Tento vzorec je u reálných raket využíván pro tvorbu hlavic s nízkým odporem vzduchu.

$$y = R\left(\frac{x}{L}\right)^{0.5} \tag{3.5}$$

Po dokončení geometrie hlavice je poloměr konstantní, tím je vykresleno tělo druhého stupně rakety. Dále je tvořen první stupeň rakety, poloměr je postupně zvětšován, až dosáhne hodnoty poloměru prvního stupně. Po dokončení těla druhého stupně je poloměr snížen na polovinu a je dále zvětšován pro vytvoření trysky motoru.

Obrázek 3.4: Otexturovaný model rakety

V druhé fázi intra bylo třeba, aby se od sebe oddělily jednotlivé stupně rakety. Toho bylo dosaženo tím, že byl pro každý stupeň vygenerován samostatný model a před jejich oddělením byly tyto modely umístěny přesně na místo modelu reprezentujícího oba spojené stupně.

3.4.2 Textura rakety

Pro raketu je generována textura skládající se z několika barevných horizontálních pruhů. Hlavice rakety je obarvena na červeno, tělo druhého stupně rakety je bílé, tělo prvního stupně je oranžové a tryska rakety je tmavě šedá. Otexturovaná raketa je na obrázku 3.4.

3.5 Startovní plošina

Model startovní plošiny je jediným modelem, který není generován za běhu programu. Není generován protože by funkce potřebná pro jeho vygenerování byla větší, než samotný model. Model je tvořen kvádrem se zkosenými hranami.

Textura pro startovní plošinu je vytvořena ve dvou průchodech. V prvním průchodu je vytvořena textura betonu za pomoci Perlinova šumu. Výstupní hodnoty šumu jsou využity pro vytvoření gradientu šedých barev, které tvoří texturu betonu. V druhém průchodu je za pomoci indexů pro daný pixel vytvořena textura mříže, která se nachází ve středu startovní plošiny.

3.6 Skybox

V demu je využit skybox na který jsou za pomoci techniky mutlitexturingu v závislosti na čase mapovány tři různé textury. Je nutné, aby textura skyboxu byla vygenerována tak, aby pozorovatel nebyl schopen rozeznat, že se jedná o jednotkovou krychli, ale měl dojem, že scéna má rozlehlé okolí. Dojem mohou také kazit viditelné hrany krychle, ty vznikají pokud je vzorkován bod, který leží přesně na hranici mezi jednotlivými stěnami krychle. Tento problém lze vyřešit při generování textury využitím následujícího příkazu pro všechny texturovací dimenze:

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
```

Obrázek 3.5: Gradient oblohy

3.6.1 Textura oblohy

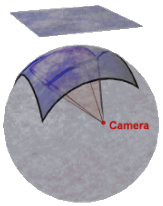
První částí skyboxu je textura modré oblohy. V reálném světě není obloha zabarvena pouze jedním odstínem modré barvy, ale barva oblohy vytváří tzv. gradient.

Gradient je plynulý přechod barvy z jednoho odstínu barvy do jiného. Obloha vytváří gradient ze světle modré barvy do tmavšího odstínu modré barvy směrem zespodu nahoru. Barva oblohy pro část zasahující nad horizont je lineárně interpolována po jednotlivých barevných složkách pomocí následujících vzorců, kde faktor m nabývá hodnot od nuly do jedné, výsledkem je gradient na obrázku 3.5.

$$r = (112 - 70)m + 70 \tag{3.6}$$
$$g = (178 - 138)m + 138 \tag{3.7}$$
$$b = (229 - 223)m + 223 \tag{3.8}$$

3.6.2 Textura oblak

Důležitou součástí oblohy jsou také oblaka. Oblačnost je ve skyboxu generována za pomoci Perlinova šumu. Mračna tvoří rovinu, která se nachází ve výšce horní roviny skyboxu, ilustrace využitého přístupu je na obrázku 3.6.



Obrázek 3.6: Rovina s oblačností, převzato z [2]

Pro získání hodnoty Perlinova šumu v daném bodě je nutné pro každý pixel vypočítat směrový vektor ze středu skyboxu. Tento vektor je po normalizaci využit pro získání správného indexu do pole s šumem. Pro nalezení indexu je nutné zjistit kde vektor daného pixelu protíná rovinu s Perlinovým šumem. Výpočet pozice průsečíku je vyjádřen následujícími rovnicemi, kde h je výška roviny nad středem skyboxu, x_v,y_v a z_v jsou jednotlivé komponenty směrového vektoru daného pixelu a x, y jsou souřadnice bodu na rovině s Perlinovým

šumem. Souřadnice x, y jsou poté dále transformovány na indexy ukazující do pole. Využitím tohoto přístupu je dosaženo toho, že oblaka vypadají jako by se skutečně na obloze nacházela a mapování textury na skybox není rozpoznatelné.

$$t = \frac{h}{y_v} \tag{3.9}$$

$$(x, y) = (tx_v, tz_v) \tag{3.10}$$

Ve skyboxu jsou vykreslována oblaka pouze tak, že okraje oblak sahají jen do určité výšky nad horizont. Z důvodu zvýšeného aliasingu textury byla oblakům blížícím se horizontu postupně zvyšována průhlednost.

3.6.3 Textura vesmírné hvězdné oblohy

Pro druhou část intra, která se nachází mimo pozemskou atmosféru bylo nutné vygenerovat texturu, která zobrazuje hvězdy. Pro generování této textury bylo využito podobného přístupu, jako při generování oblačnosti.

Prvním problémem, který bylo nutné vyřešit byl způsob generování pozic hvězd. Pouhé náhodné vygenerování jednotlivých komponent směrového vektoru směřujícího ze středu skyboxu k pozici hvězdy na skyboxu se ukázalo jako nevhodné. Na skyboxu byl viditelný tvar krychle a hvězdy byly znatelně nerovnoměrně uspořádány.

Řešení tohoto problému spočívá v generování uniformních sférických souřadnic a jejich transformaci do třídimenzionálního kartézského souřadnicového systému. V tomto případě byly hvězdy nejdříve mapovány na jednotkovou kouli a poté byly promítnuty na texturu skyboxu.

Pro generování souřadnic jsou nejdříve využity následující rovnice, pomocí kterých jsou vytvořeny úhly λ a ϕ, kde u_1 a u_2 jsou náhodné hodnoty s uniformním rozložením v intervalu $\langle 0, 1 \rangle$.

$$\lambda = \arccos(2u_1 - 1) - \frac{\pi}{2} \tag{3.11}$$

$$\phi = 2\pi u_2 \tag{3.12}$$

Tyto úhly jsou dále využity v následujících rovnicích pro výpočet souřadnic daného bodu na jednotkové kouli.

$$x = \cos(\lambda)\cos(\phi) \tag{3.13}$$

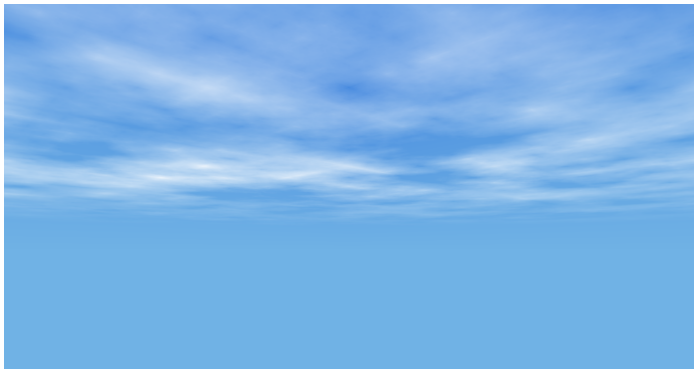
$$y = \cos(\lambda)\sin(\phi) \tag{3.14}$$

$$z = \sin(\lambda) \tag{3.15}$$

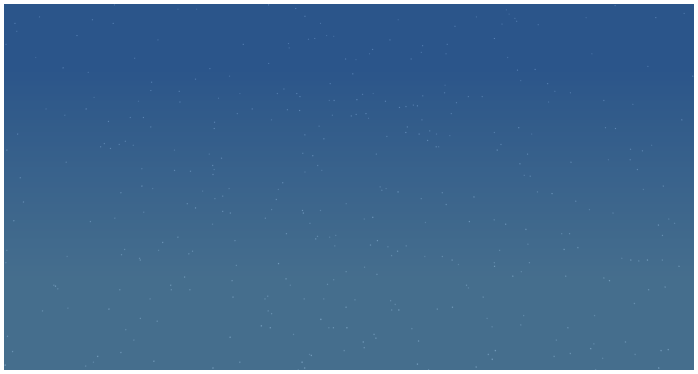
Výsledné souřadnice jsou využity pro tvorbu vektoru směřujícího k danému bodu. Vektor je nejdříve normalizován, poté je zjištěna jeho největší komponenta. Největší komponenta vektoru určuje stěnu krychle, na které bude vykreslena hvězda. Dvě zbývající komponenty jsou využity pro vypočítání pozice hvězdy na dané stěně krychle. Pro každou hvězdu je dále vypočítán náhodný jas, to přispívá realističnosti celého skyboxu.

3.6.4 Míchání jednotlivých textur

Protože je nutné v různých fázích intra zobrazovat různé skyboxy a je nutné je i míchat, bylo využito techniky multitexturingu. Jednotlivé skyboxy jsou vygenerovány samostatně a za běhu programu jsou míchány ve fragment shaderu.



Obrázek 3.7: Skybox oblohy s mraky



Obrázek 3.8: Smíchané textury pro pozemskou a vesmírnou oblohu

Ve finálním intru jsou v první fázi před startem rakety vykresleny textury pro oblohu a mračna. Po startu rakety mračna postupně mizí a později se na úkor modré oblohy začne postupně vykreslovat hvězdný skybox. Míchání textur jednotlivých skyboxů ovládá následující kód prováděný ve fragment shaderu:

```
vec4 CloudColor = mix(vec4(1.0, 1.0, 1.0, 0.0),
                      texture(Cloud, TexCoords), CloudAlpha);
vec4 SkyColor   = mix(texture(Star, TexCoords),
                      texture(Sky, TexCoords), SkyAlpha);
color = vec4(mix(SkyColor.rgb, CloudColor.rgb, CloudColor.a), 1.0);
```

Textura mraků je míchána s průhlednou bílou barvou, tím je dosaženo plynulého mizení oblačnosti. Faktory míchání jsou definovány uniformními proměnnými `CloudAlpha` a `SkyAlpha`, které jsou programem nastavovány v závislosti na čase. Na obrázku 3.7 jsou zobrazeny smíchané textury pro oblačnost a gradient oblohy, na obrázku 3.8 jsou míchány textury pro pozemskou oblohu a hvězdnou oblohy.

3.7 Částicové efekty

Částicové efekty jsou v demu využity pro simulaci kouře vycházejícího z raketového motoru. Jednotlivé částice tvoří billboardy kouřové textury.

Ta je generována za pomoci 2D perlinova šumu. Nejdříve je vygenerována mapa šumu. Pro každý bod je během generování vytvořen vektor vedoucí ze středu textury. Podle délky

vektoru je pak hodnota průhlednosti pixelu upravena tak, aby výsledkem byla kruhová textura částice. Průhlednost pixelů získaných z Perlinova šumu není blíže ke středu upravována a se zvyšující se vzdáleností od středu je postupně snižována.

Jednotlivé částice jsou v paměti reprezentovány pomocí datové struktury obsahující současnou pozici částice, její rychlost, zbývající dobu života a vzdálenost od kamery. Při vygenerování částice jsou tyto hodnoty inicializovány tak, aby se částice vždy vytvořila v motoru rakety. K určité stálé hodnotě rychlosti částice je také při jejím vygenerování přidána náhodná hodnota, aby se každá částice během simulace jejího života ubírala jinou trajektorií. Protože se při každém vykreslení snímku může změnit jejich počet, nelze při nahrávání dat do paměti grafické karty použít pouze funkci `glBufferData` s argumentem `GL_STATIC_DRAW`, ale je před každým vykreslením nutné nejdříve daný buffer vynulovat a data nahrát s použitím argumentu `GL_STREAM_DRAW`.

Při každém vykreslení snímku jsou všechny částice aktualizovány. Je jim snížena životnost o dobu vykreslení posledního snímku, je upravena jejich pozice. Dále jsou částice seřazeny vzestupně dle vzdálenosti od kamery, seřazení umožní správné vykreslení průhlednosti při překrytí několika částic, seřazení ovšem nevyřeší všechny konflikty při překrývání částic, proto je nutné před vykreslením vypnout zápis do z-bufferu a po vykreslení jej zase zapnout, toho lze dosáhnout následujícím příkazem:

```
glDepthMask(GL_FALSE);
```

Živé částice jsou po dokončení simulace vykresleny za využití instancingu. Před samotným vykreslením je ve vertex shaderu upravena orientace částice tak, aby byla svým čelem přivrácena ke kameře. Tato úprava dosáhne toho, že se částice budou chovat jako billboardy. Kód zodpovědný za tuto transformaci je následovný:

```
vec3 particlePos = posttl.xyz+CameraRight*vertices.x+CameraUp*vertices.y;
```

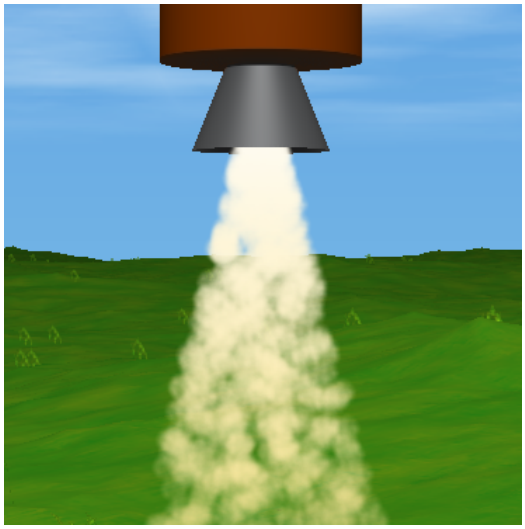
Ve fragment shaderu je dále upravena barva částice v závislosti na zbývající délce života částice. Pro kouř jsem zvolil počáteční bílou barvu přecházející do žluté. V závislosti na zbývající délce života se mění také průhlednost částice, se snižující se životností je průhlednost zvyšována. Tím je dosaženo toho, že když částici končí doba života tak jen nezmizí, ale postupně se rozplyne. Finální vzhled částicových efektů je zobrazen na obrázku 3.9.

3.8 Planety

Součástí druhé scény implementovaného intra jsou dvě planety, jedna pozemského typu a druhá podobná planetě Mars.

3.8.1 Generování planet

Pro vytvoření modelů planet je použit podobný přístup, jako pro generování modelu rakety. Každá planeta je reprezentována koulí. Pro výpočet souřadnic jednotlivých vrcholů polygonů jsou využity následující rovnice.



Obrázek 3.9: Částice vycházející z motoru rakety

$$\theta = \pi \frac{h_i}{h_{\max}} \tag{3.16}$$

$$\phi = 2\pi \frac{w_i}{w_{\max}} \tag{3.17}$$

$$x = r \sin(\theta) \cos(\phi) \tag{3.18}$$

$$y = r \cos(\theta) \tag{3.19}$$

$$z = r \sin(\theta) \sin(\phi) \tag{3.20}$$

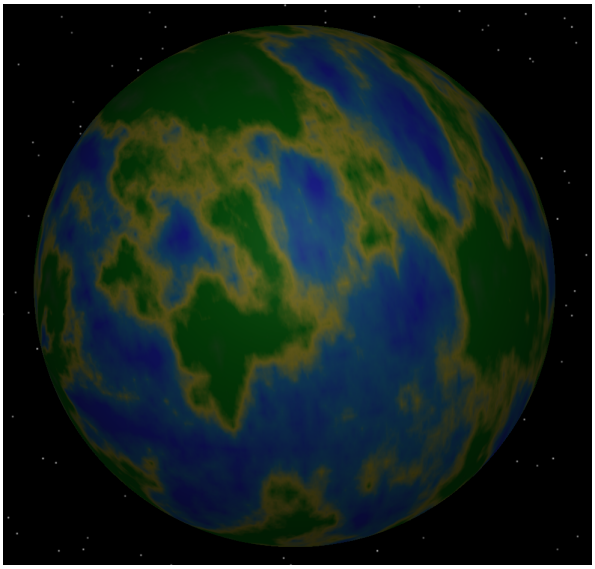
Při počítání pozice vrcholu je nejdříve nutné vypočítat úhly θ a ϕ, úhel θ označuje azimut vrcholu a úhel ϕ označuje odchylku od horizontální osy. Pro jejich výpočet je využito hodnot h_i a w_i, které označují pořadí kroku po dané ose, kde $h_{\max}$ a $w_{\max}$ jsou jejich maximální hodnoty. Ve výsledném demu jsou tyto hodnoty 1000 pro planetu pozemského typu a 100 pro planetu podobnou Marsu, tyto hodnoty udávají celkový počet „pásků“ využitých pro sestavení planety.

3.8.2 Generování textur planet

Textury planet jsou generovány s využitím Perlinova šumu v tomto případě je pole s perlinovým šumem využito jako pole vstupních hodnot pro několik barevných gradientů.

Při generování textury pro planetu pozemského typu jsou využity čtyři barvy a gradienty mezi nimi. Pro generování oceánů jsou využity dva odstíny modré, pro hluboký oceán tmavá, pro oceán u pobřeží světlejší odstín modré. Pro generování terénu jsou využity tři další barvy, světle žlutá pro pobřežní oblasti, světle zelená pro oblasti mezi pobřežím a vnitrozemím a tmavě zelená pro oblasti hluboko ve vnitrozemí. Textura planety nanesená na model je zobrazena na obrázku 3.10. Využitím několika gradientů je dosaženo realistického vzhledu textury.

Generování textury pro planetu podobnou Marsu probíhá podobným způsobem, jako generování předchozí textury. V tomto případě jsou využity jen dvě barvy, jeden odstín červené a tmavý načervenalý odstín hnědé. Textura pro tuto planety je generována jen



Obrázek 3.10: Textura planety nanesená na modelu

za pomoci dvou gradientů protože se planeta nachází ve značné vzdálenosti od kamery a využití většího počtu barev nebylo rozeznatelné.

3.9 Kamera

Vzhledem k tomu, že výsledné demo nemá být interaktivní musela být zvolena metoda pro ovládání kamery, která nevyžaduje žádné uživatelské vstupy. Pouhá lineární interpolace pozice kamery mezi řídícími body by byla velmi nevhodná, protože by byly všechny změny směru znatelné a celé demo by působilo kostrbatě.

Proto byla pro ovládání pozice kamery zvolena interpolační křivka Catmull-Rom spline[4]. Tento spline je definován čtyřmi řídícími body, přičemž křivka vytvořená splinem je vykreslena pouze mezi prostředními dvěma body. Catmull-Rom spline je vyjádřen vzorcem 3.21.

$$\mathbf{P}(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \frac{1}{2} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_{i-1} \\ \mathbf{P}_i \\ \mathbf{P}_{i+1} \\ \mathbf{P}_{i+2} \end{pmatrix} \tag{3.21}$$

V tomto vzorci jsou $\mathbf{P}_{i-1}$ až $\mathbf{P}_{i+2}$ řídící body křivky a hodnota t je vzdálenost mezi body $\mathbf{P}_i$ a $\mathbf{P}_{i+1}$. Ve výsledném intru je za hodnotu t dosazován čas. Protože se ukázalo, že pokud by kamera měla každou sekundu urazit vzdálenost mezi dvěma řídícími body, byl vytvořen kamerový čas, který plyne osmkrát pomaleji, než čas reálný. Tím je dosaženo snížení počtu nutných řídících bodů s udržením plynulého pohybu kamery.

Pozice kamery není po celou dobu běhu dema ovládána Catmull-Rom splinem. V některých částech intra je kamera statická, nebo se pohybuje současně s raketou. Při přechodech mezi těmito dvěma ovládacími mechanismy na sebe jednotlivé prvky ovládající pohyb kamery navazují tak aby nebyl znatelný žádný trhavý pohyb. I když se pozice kamery v průběhu času mění, je celou dobu zaměřena na model rakety. Raketa je tedy po celou dobu intra středobodem scény.

Pro generování view matice je využita funkce `glm::lookAt`, s její pomocí je v každém snímku view matice aktualizována. Této funkci jsou předány tři argumenty typu `glm::vec3`, jsou to v tomto pořadí: pozice kamery, bod určující směr pohledu kamery a tzv. up vektor. Up vektor označuje osu, která má ve výsledném pohledu kamerou směřovat vzhůru. V intru je up vektor nastaven na hodnotu `glm::vec3(0,1,0)`, což značí standardní orientaci v souřadném systému OpenGL.

Kapitola 4

Omezení velikosti spustitelného souboru

Vzhledem k tomu, že výsledný program je od počátku limitován maximální velikostí, bylo nutné během psaní kódu přihlížet k výsledné velikosti spustitelného souboru. Mezi techniky využitelné pro snížení výsledné velikosti souboru patří například psaní kódu tak aby byla maximalizována znovupoužitelnost, využívání funkcí a cyklů místo opakování kódu. Dále lze ušetřit velikost spustitelného souboru nevyužíváním knihoven, které neúměrně zvětšují výsledný program. V grafických intrech je pro ušetření velikosti hojně využíváno procedurální generování, vygenerovaná data mnohonásobně převyšují velikost samotného spustitelného souboru. Dále je možné velikost spustitelného souboru zmenšit za pomoci využití vhodného nastavení parametrů překladače a využitím exe packerů.

4.1 Parametry překladače

Pro překlad intra byl využit port překladače g++ pro Microsoft Windows. Tento překladač nabízí mnoho různých možností optimalizace, jednotlivé argumenty jsou popsány v následující tabulce:

Parametr	Efekt
`-O1`	Základní úroveň optimalizace, kompilátor se snaží balancovat rychlost programu a jeho velikost
`-O2`	Pokročilejší úroveň optimalizace, kompilátor se snaží minimalizovat velikost programu, ale ne za cenu snížení rychlosti
`-Os`	Zapíná všechny optimalizace z předchozí úrovně, které nezvyšují velikost kódu. Dále vypíná některé optimalizace, které mají za následek zrychlení kódu na úkor jeho velikosti.
`-m32`	Kompilátor generuje kód pro 32bitovou architekturu, to má za následek poloviční velikost všech pointerů a tím i snížení velikosti.
`-s`	Kompilátor ke kódu nepřikládá informace pro debugger.

Tabulka 4.1: Tabulka popisující vhodné argumenty překladače

Pro sestavení výsledného programu byly využity poslední tři jmenované parametry. Jejich využití snížilo velikost spustitelného souboru přibližně o sedmdesát procent.

4.2 Exe packery

Dalším krokem pro snížení velikosti spustitelného souboru bylo využití exe packeru. Exe packery jsou programy, které umožňují kompresi spustitelných souborů, fungují na principu samorozbalujícího archivu. Po kompresi je na začátek spustitelného souboru vložen kód pro dekompresi a následné spuštění programu. Exe packery jsou v současnosti využívány především pro kompresi grafických inter.

Pro kompresi grafického intra byl zvolen program UPX, Ultimate Packer for eXecutables. Pro kompresi intra byl využit následující příkaz:

```
upx demo.exe -ocdemo.exe -9
```

Výstupem tohoto programu je zkomprimovaný soubor `cdemo.exe`. Použitím UPX bylo dosaženo redukce spustitelného souboru z původní velikosti 91 kB na výslednou velikost 41 kB.

Kapitola 5

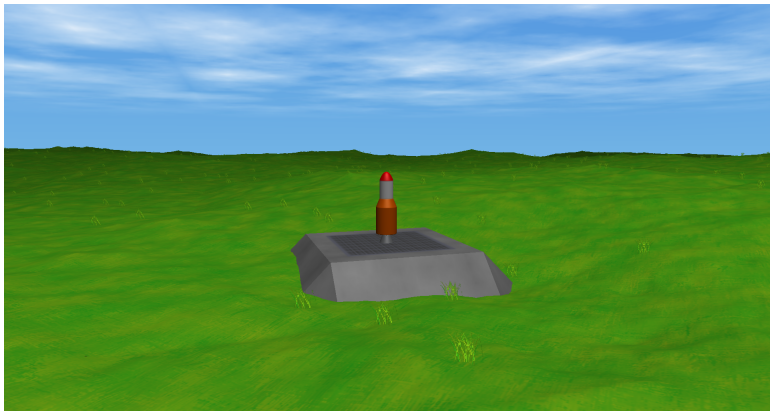
Výsledky práce

V této kapitole jsou zhodnoceny výsledky práce. Je popsán výsledný program a jsou prezentovány výsledky měření.

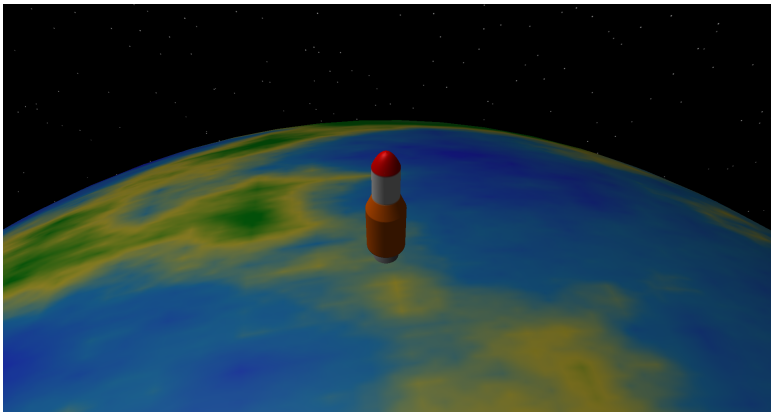
5.1 Implementované intro

Konečná velikost spustitelného souboru je 41 kB. Za využití tohoto prostoru je vykresleno grafické intro o celkové délce 160 sekund. Při spuštění programu se intro pokusí přejít do režimu zobrazení na celou obrazovku, pokud tato funkce není dostupná, spustí se v režimu okna s upravitelnou velikostí. Prvních čtyřicet sekund je zabráno první scénou, kde je ukázána raketa před startem na povrchu planety. Druhá scéna začíná startem rakety. Poté je zobrazeno opuštění atmosféry raketou a vstup do vesmírného prostoru. Poslední část intra je strávena záběrem na oddělení druhého stupně rakety a jeho následný odlet směrem k jiné planetě.

Intro končí po uplynutí 160 sekund, zavření okna nebo stisku klávesy escape. Na obrázcích 5.1 a 5.2 jsou zobrazeny scény z konečného programu. Intru lze předat dva argumenty, `-w`, tento přepínač vypne spouštění aplikace v režimu celé obrazovky. Druhým argumentem je `-f` který zapíná výpis počtu snímků za sekundu do konzole.



Obrázek 5.1: Snímek z úvodu intra



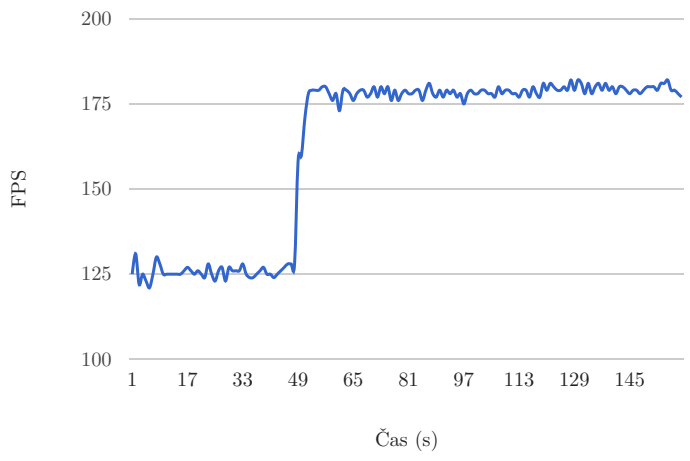
Obrázek 5.2: Snímek z druhé fáze intra

5.2 Měření

Na výsledném programu bylo provedeno měření počtu vykreslovaných snímků za sekundu. Pro měření uplynulého času byla po otestování několika různých funkcí využita funkce `GetSystemTime`. Ta vrací současný čas s přesností na milisekundy.

Výpočet snímků za sekundu probíhá za pomoci dvou čítačů. K prvnímu z nich je při každém vykreslení snímku přičtena jednička, druhý čítač je vždy zvýšen o počet milisekund uplynulých od začátku vykreslování posledního snímku. Pokud druhý čítač překročí jednu sekundu, je vypsána hodnota počtu snímků a oba čítače jsou vynulovány.

Na grafu 5.3 je znázorněna hodnota FPS v závislosti na čase. Po celou dobu běhu programu dosahovaly počty snímků za sekundu vysokých hodnot, což znamená, že intro běželo velmi plynule. Zvýšení počtu snímků za sekundu v čase 50 s je způsobeno tím, že intro přechází do druhé scény, ve které přestává být vykreslován terén s trávou.



Obrázek 5.3: Počet vykreslených snímků za sekundu v závislosti na čase

Měření bylo provedeno na počítači s operačním systémem Microsoft Windows 10. Počítač disponuje procesorem Intel Core i5 6500 a grafickou kartou NVIDIA GeForce GTX 1060.

Vzhledem k tomu, že hodnoty počtů snímků za sekundu dosahují velmi vysokých hodnot, by bylo možné dále demo rozšiřovat, nebo zvýšit detaily generovaných modelů a textur.

Kapitola 6

Závěr

V rámci této práce mělo být vytvořeno grafické intro, jehož velikost nebude přesahovat velikost 64 kB. Výsledné intro má po kompresi velikost 41 kB, takže byl velikostní limit splněn. Během tvorby tohoto projektu byly nastudovány a implementovány následující techniky:

- Perlinův šum
- Částicové systémy
- Generování pomocí matematických vzorců
- Phongův osvětlovací model
- Billboarding
- Skybox

Před začátkem práce na programu jsem neměl téměř žádné zkušenosti s tvorbou grafických aplikací. Během tvorby intra jsem měl možnost se důkladně seznámit s prací s 3D grafikou a OpenGL.

Mezi možnosti pro pokračování projektu patří například implementace shadow mappingu, animovaného skyboxu, nebo vytvoření vodní hladiny.

Literatura

[1] *Billboards*. [Online; navštíveno 11.5.2017].
URL http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/billboards/

[2] Elias, H.: *Cloud Cover*. [Online; navštíveno 8.5.2017].
URL https://web.archive.org/web/20150921131550/http://freespace.virgin.net:80/hugo.elias/models/m_clouds.htm

[3] Elias, H.: *Perlin Noise*. [Online; navštíveno 8.5.2017].
URL https://web.archive.org/web/20160325134143/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

[4] Joy, K. I.: *Catmull-Rom splines*. [Online; navštíveno 12.5.2017].
URL
http://graphics.cs.ucdavis.edu/~joy/ecs278/notes/Catmull-Rom-Spline.pdf

[5] Kapadia, A.: *Linear Congruential Generators*. [Online; navštíveno 8.5.2017].
URL https://www.cs.indiana.edu/~kapadia/project2/node7.html

[6] Meiri, E.: *SkyBox*. [Online; navštíveno 12.5.2017].
URL http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html

[7] Perlin, K.: An image synthesizer. *ACM SIGGRAPH Computer Graphics*, ročník 19, č. 3, 1985: s. 287–296, ISSN 00978930, doi:10.1145/325165.325247.
URL http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/325165.325247

[8] Phong, B.: Illumination for computer generated pictures. *Communications of the ACM*, ročník 18, č. 6, 1975: s. 311–317, ISSN 00010782, doi:10.1145/360825.360839.
URL http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/360825.360839

Příloha A

DVD

DVD přiložené k této bakalářské práci obsahuje následující přílohy:

- Zdrojové kódy k programu
- Spustitelný soubor programu
- Readme
- Video