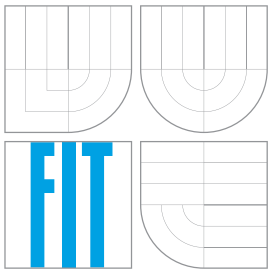
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ANIMOVANÝ SKYBOX V OPENGL

ANIMATED SKYBOX USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÍT PAVLÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MILET

BRNO 2013

Abstrakt

Bakalářská práce se zabývá tvorbou animovaného skyboxu. Cílem práce je demonstrovat metody vytváření skyboxů, především pak metody procedurální. V hlavní části práce jsou rozebrány techniky potřebné pro vykreslení jednotlivých prvků skyboxu. Zaměřuje se na tvorbu barevných přechodů, funkce šumu a na jejich aplikaci. Závěr práce se věnuje rozboru experimentů s nastavením parametrů vykreslování grafického obsahu a implementačním podrobnostem.

Abstract

The Bachelor's diploma thesis addresses the operation of generating an animated skybox. The thesis was intended to demonstrate the methods, especially the procedural methods, for generating skybox applications. The main deal of the paper is devoted to an analysis of fundamental techniques for rendering particular components within a skybox application. The utmost focus is put on processes of generating colour gradients, noise functions and ways of applying thereof. Concluding part comprises an analysis of study trials to investigate different graphics content generating settings, and implementation details.

Klíčová slova

animovaný skybox, OpenGL, generování šumu, deterministický generátor, Perlinův šum, bump mappping, Blinn-Phongův osvětlovací model, barevný přechod, obloha, mraky, hvězdy, hvězdná obloha, textura, GLSL, shader

Keywords

animated skybox, OpenGL, noise generation, deterministic generator, Perlin noise, bump mapping, Blinn-Phong shading model, color gradient, sky, clouds, stars, starry sky, texture, GLSL, shader

Citace

Vít Pavlík: Animovaný skybox v OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2013

Animovaný skybox v OpenGL

Prohlášení

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.

.....
Vít Pavlík
15. května 2013

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce panu Ing. Tomáši Miletovi za jeho podporu, rady a trpělivost při tvorbě práce. Dále bych rád poděkoval své kamarádce Marcele Maškové za odbornou korekturu některých částí práce.

© Vít Pavlík, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**
1.1 Skybox . . . 2
1.2 OpenGL . . . 3

2 Prvky skyboxu **4**
2.1 Slunce a Měsíc . . . 4
2.2 Hvězdy . . . 5
2.3 Mraky . . . 5

3 Generovaní skyboxu **7**
3.1 Barevné přechody . . . 7
3.2 Funkce šumu . . . 10
3.3 Generování mraků . . . 14
3.4 Bump mapping . . . 17
3.5 Generování oceánu . . . 18
3.6 Hvězdná obloha . . . 20

4 Experimenty **23**
4.1 FPS . . . 23
4.2 Vrstvy mraků . . . 23
4.3 Oceán . . . 24

5 Implementační podrobnosti **26**
5.1 Použité knihovny . . . 26
5.2 Generování textury normálové mapy . . . 26
5.3 Výpočet aktuální konfigurace barev . . . 27
5.4 GLSL . . . 29

6 Závěr **30**

A Obsah CD **33**

B Plakat **34**

Kapitola 1

Úvod

Real-time aplikace (aplikace jejichž simulace probíhá v reálném čase) virtuální reality, jako například hry, stále více využívají venkovní prostředí jako okolí vytvářené scény. Důvodem je snaha v těchto aplikacích vytvořit reálně vypadající nebe v závislosti na fyzikálních i obrazových vlastnostech. Programovatelný grafický hardware umožnuje vytvářet reálně vypadající animované scény pomocí efektivních algoritmů s minimálními nároky na výkon [11].

Cílem této práce je vytvoření právě takového okolí scény, jež zachycuje oblohu v průběhu dne, tak aby součástí scény byly animované mraky, hvězdy atd. Převážná část práce pojednává o technikách vytváření funkcí šumu a jejich použití při vykreslování prvků oblohy. Teoretická část se zaměřuje na důležité vlastnosti těchto prkvů vzhledem k jejich simulaci ve virtuální realitě. Ve třetí kapitole jsou popsány algoritmy použité při tvorbě aplikace, vytváření barevných přechodů, výpočet osvětlovacího modelu a *bump mappingu*. V závěru této kapitoly jsou podrobně popsány postupy pro vytvoření jednotlivých částí skyboxu. Následuje sekce, kde jsou zmíněny experimenty s nastavením parametrů vykreslování. V poslední implementační kapitole je popsáno generování textur a programování *shaderů*.

1.1 Skybox

Skyboxem se v počítačové grafice označuje metoda vytváření okolí scény, pomocí níž se docílí iluze, kdy se zobrazená scéna v porovnání se skutečnou velikostí jeví jako větší. Základní myšlenkou je vykreslení vzdálených objektů, například hor, mračen, ale třeba také budov a jiných objektů v závislosti na vytvářené scéně, kterých by při pohybu scénou nemohlo být nikdy dosaženo. Podstatné je, že tyto objekty nejsou ve skutečnosti složeny z polygonů, ale jejich obraz je obvykle namapován na krychli nebo kvádr (dnes se v aplikacích častěji využívájí takzvané skydomy), jenž je vykreslen kolem scény. Jedná se o metodu mapování textury *cube mapping*, která využívá 3D směrový vektor pro naindexování šesti čtvercových 2D textur uspořádaných tak, že tvoří plochu rozložené krychle. Kamera je umístěna uprostřed scény a její pozice se vůči skyboxu nemění ani při pohybu virtuálním světem.

Při vytváření skyboxů lze využít dvou přístupů. U prvního z nich se využije předpřipravená textura, která se následně namapuje metodou cube mappingu, jejíž princip je popsán výše. Je důležité vytvářet texturu pro skybox tak, aby na sebe jednotlivé obrázky navazovaly. Jako obrázky mohou být použity například fotografie reálné scény pořízené z jednoho místa, kdy každá z fotografií je v ortogonální pozici vůči fotografiím, s nimiž sousedí. Na obrázku 2.1 je vidět výsledná textura pro skybox zobrazující denní oblohu. Tímto způsobem

je možné vytvářet jednoduché statické skyboxy.

Obrázek 1.1: Textura reprezentující skybox denní oblohy nad oceánem

Se zvýšením výpočetního výkonu počítačů a především díky pokroku ve výkonosti grafických karet se začalo při vytváření skyboxu využívat procedurálního přístupu. Okolí scény není tvořeno předpřipravenou texturou, ale barva jednotlivých bodů plochy krychle je vypočtena zvlášť. Pro každý pixel se určí paprsek ze středu krychle. Smícháním barev prvků skyboxu, jež tento paprsek protne, se určí výsledná barva v daném bodě. Barva pixelu se počítá znovu při každém překreslení scény, díky čemuž lze snadno měnit vzhled scény v reálném čase. Přávě popsaného postupu bylo využito k vytvoření aplikace skyboxu, jež je předmětem této práce.

1.2 OpenGL

OpenGL (*Open Graphics Library*) je možné definovat jako softwarové rozhraní ke grafickému hardwaru. Využíva se při tvorbě různých 3D grafických aplikací například počítačových her, vědeckotechnických vizualizací, architektonických aplikací a mnoha dalších aplikací. Multiplatformní knihovna OpenGL, již vyvinula společnost SGI (*Silicon Graphics, Inc.*), byla představena v lednu roku 1992. Specifikace OpenGL je spravována průmyslovým konsorciem známým pod zkratkou ARB (*Architecture Review Board*), ve kterém jsou zastoupeny velké firmy počítačového průmyslu jako AMD/ATI, Apple Inc, Nvidia a jiné. Od verze OpenGL 2.0 je podporována programovatelná kreslicí *pipeline*, čehož bylo využito také při vývoji aplikace skyboxu. V roce 2012 byla vydána prozatím poslední verze OpenGL 4.3.

Kapitola 2

Prvky skyboxu

Tato kapitola je zaměřena na teoretický úvod do problematiky skyboxu. Představeny jsou jednotlivé prvky, jež jsou součástí skyboxu. Jejich popis se soustředí především na jejich vzhled a další vlastnosti, které ovlivní postup při vytváření aplikace.

Jak již bylo zmíněno v úvodní kapitole 1.1, pojem skybox se používá jako označení pro vzdálené okolí scény. Nemusí se pokaždé jednat o oblohu, ovšem v této sekci budeme předpokládat právě skybox, na němž je vyobrazena obloha v průběhu dne.

2.1 Slunce a Měsíc

Slunce je hvězda ve středu naší sluneční soustavy. Jedná se kouli žhavého plazmatu s přibližně 330 000krát větší hmotností, než je hmotnost Země. Jakožto hvězda nejblíže zemi představuje Slunce nejjasnější objekt na obloze – její hvězdná velikost je -26,6 m. To je více než dvojnásobek v porovnání s Měsícem, jehož hvězdná velikost je -12,6 m [6]. Hvězdná velikost, neboli magnituda (zkrácenThe Fractal Geometry of Natureě m), je jednotka zářivosti objektu pozorovaného ze Země.

Tvar Slunce připomíná téměř dokonalou kouli. Při východu nebo západu se může jevit větší než v průběhu dne. To je způsobeno tím, že světlo prochází hustší atmosférou. Zemská atmosféra má vliv také na barvu Slunce. Panuje všeobecné přesvědčení, že Slunce má žlutou barvu, ve skutečnosti má však Slunce bílou barvu. Právě průchod paprsků skrz naší atmosféru mění jejich barvu na žlutou. V průběhu dne má Slunce z pohledu pozorovatele na Zemi spíše bílou barvu, kdežto při západu a východu převažuje žlutá barva.

Posledním podstatným faktorem, na který má Slunce vliv a se kterým je potřeba počítat při vývoji aplikace skyboxu, je barva oblohy. Modré zbarvení oblohy je způsobeno průchodem slunečního světla naší atmosférou. Jev, který dané zbarvení vyvolává, je znám jako Rayleighův rozptyl [9]. Jedná se o rozptyl světla na molekulách plynu. Při výskytu drobných částeček v atmosféře je možné při východu nebo západu Slunce zpozorovat jev nazývaný jako červánky, tedy červené, oranžové nebo růžové zbarvení oblohy.

Měsíc je jediná přirozená družice Země, tj. kosmické těleso pohybující se po oběžné dráze jiného většího tělesa. Je k Zemi natočen stále stejnou stranou, označovanou jako přivrácená strana Měsíce. Opačná strana se nazývá odvrácená, někdy temná, strana Měsíce. I přesto, že povrch Měsíce odráží velmi málo světla, jedná se o druhý nejjasnější objekt na obloze. Nejlépe je viditelný v noci, ve fázi úplňku. Občasně je možné jej pozorovat i ve dne. Povrch měsíce je pokryt desítkami tisíc kráterů. Opticky tmavší místa, pozorovatelná z povrchu

Země, jsou takzvaná měsíční moře.

2.2 Hvězdy

Hvězdy můžeme na obloze pozorovat jako zářivé body. Jas jednotlivých hvězd se liší, jak již bylo zmíněno v podkapitole o Slunci 2.1, měříme jej takzvanou hvězdnou vzdáleností (magnituda). Nejjasnější hvězda noční oblohy Sirius má magnitudu -1,6 m. Pouhým okem je možné vidět hvězdy do hvězdné vzdálenosti 6 m, což zahrnuje přibližně 10 000 hvězd. Při pozorování ze Země se zdá, že se hvězdy nepohybují. Zdánlivá nehybnost hvězd je způsobena jejich vzdáleností, ačkoli se hvězdy pohybují poměrně rychle. Gravitační síla spojuje hvězdy do galaxií, v nichž se hvězdy váží do měnších skupin nazývaných hvězdokupy.

2.3 Mraky

Oblak, jinak také mrak, je útvar nepravidelného tvaru, složený z malých částic vody nebo ledu, nacházející se v atmosféře. Kromě vody mohou obsahovat také jiné častice, například různé chemické zplodiny vznikající při průmyslové výrobě, prachové částice a další.

Mračna vznikají kondenzací vzdušné vlhkosti. Tento proces je ovlivněn vzestupným pohybem teplého vlhkého vzduchu. Stoupající vzduch se ochlazuje, dokud nedosáhne hranice, při které se přemění na malé kapičky nebo ledové krystalky. Tato teplotní hranice je označována jako rosný bod.

Mračna jsou neustále v pohybu a jejich tvar se mění. Je to způsobeno několika faktory:

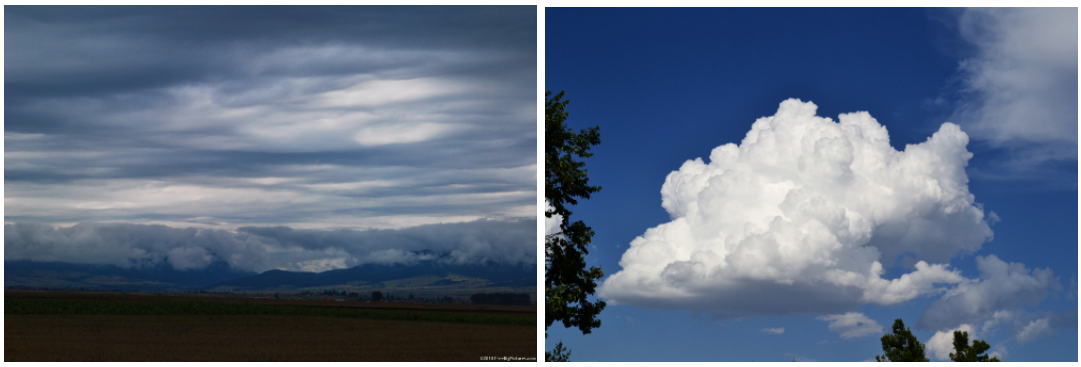
- Gravitací, která působí na částice, jimiž jsou mraky tvořeny.
- Vlivem větru, který vzniká při proudění vzduchu z místa s vyšším atmosférickým tlakem do místa s nižším atmosférickým tlakem.
- Teplotou, protože teplejší vzduch se tlačí vzhůru, a tím vzniká vzdušné proudění.

Mraky dělíme do několika základních kategorií v závislosti na jejich vzhledu, vlatnostech a na tom, v jaké výšce se nachází:

- *Cirrus* (řasa) – jemná vlákna bílých, někdy až hedvábně lesklých oblak obsahujících ledové krystalky, tvořících se ve velkých výškách (8 – 13 km).
- *Cirrocumulus* (řasová kupa) – čistě bílá oblaka složená z ledových krystalků nebo podchlazených kapek vody. Tvoří malé kulovité skupiny obláčků, které nevrhají stíny.
- *Cirrostratus* (řasová sloha) – bílý závoj oblaků. Díky nízké hustotě je skrze vrstvu oblak vidět Slunce nebo Měsíc. Stejně jako předchozí skupiny je tvořen ledovými krystalky. Jejich výskyt bývá doprovázen halovými jevy.
- *Altocumulus* (vysoká kupa) – tvoří menší nebo větší skupiny, někdy i vrstvy mraků, s nepravidelným tvarem, občas připomínajícím vlny. Skládá se z vodních kapek, takže kromě čistě bílé barvy obsahuje částečně i nádech šedé. Vyskytuje se ve výškách 1,5 – 7 km.
- *Altostratus* (vysoká sloha) – šedavá nebo modravá oblačná opona, většinou úplně pokrývá oblohu. Místy tak tenká, že může propouštět sluneční světlo.

- *Stratocumulus* (slohokupa) – tvarem podobný altocumulu, ovšem s většími oblaky, protože se nachází v menších výškách. Tvořen převážně vodními kapkami. Oblaka bývají bělavá s tmavšími šedými odstíny.
- *Stratus* (sloha) – jednolitá šedavá vrstva mračen připomínajících mlhu. Vyskytuje se v malých výškách.
- *Cumulus* (kupa) – obvykle osamocené, husté mraky, převážně s ostrými konturami. Část mraků, jež je osvícena slunečními paprsky, nabývá výrazně bílé barvy. Základna mraků má téměř vodorovný tvar, většinou tmavší šedé barvy. Horní část oblaku, rozvíjející se do výšky, mívá špíše bělavou barvu.
- *Nimbostratus* (deštová sloha) – jednolitá hustá vrstva mračen šedé, olovnaté bavy. Přes jejich velkou mohutnost nelze vůbec pozorovat polohu Slunce.
- *Cumulonimbus* (dešťová kupa) – bouřkový mrak, tvarem i barvou připomínající Cumulus, je však mohutnější a dosahuje velkých výšek. Horní část je zploštělá a podobná kovadlině.

Popis druhů mraků je zaměřen převážně na jejich barvu a tvar. Srážkové vlastnosti pro dané účely nejsou podstatné, protože v aplikaci není simulován déšt ani sněžení.



Obrázek 2.1: Zleva fotky mraků Stratus a Cumulus, jež jsou vzorem pro mračna v aplikaci skyboxu

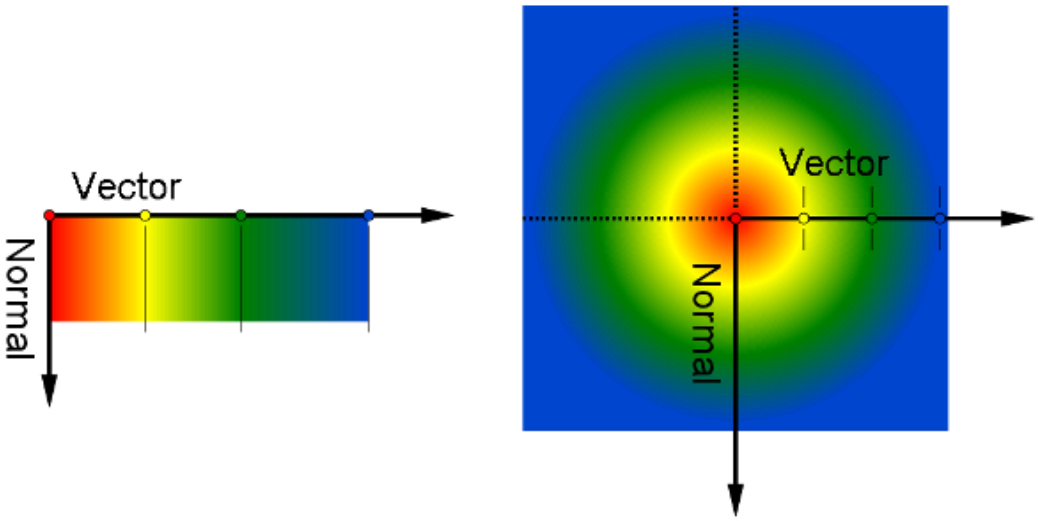
Kapitola 3

Generovaní skyboxu

V této kapitole jsou nejprve představeny algoritmy a techniky potřebné k vykreslení prvků skyboxu, jejichž popisem se zabývá předchozí kapitola. Jsou zde uvedeny informace o funkcích šumu a to, jak se liší od běžných kongruentních generátorů šumu. V dalších částech jsou popsány způsoby tvorby gradientů a metoda bump mappingu. Nakonec jsou ke každému prvku skyboxu uvedeny potřebné kroky k jeho úspěšné vizualizaci. Popis vizualizace je zaměřen na definici matematických vztahů a na grafické znázornění jednotlivých kroků.

3.1 Barevné přechody

Barevný přechod, neboli gradient, je složen z plynulého přechodu mezi kontrolními barvami podél daného vektoru. Barevný přechod mezi dvěma sousedícími barvami je počítán jako lineární interpolace mezi jednotlivými barevnými kanály, případně také alfa kanálem. Rozlišujeme dva druhy gradientů:



Obrázek 3.1: Ukázka lineárního a radiálního gradientu [15]

- Lineární gradient je definován několika body a kontrolní barvou u každého z nich. Barva přechodu se získá interpolací mezi jednotlivými sousedícími body.

- Úhlový gradient je definován jako kruh, určený barvou ve svém středu a okrajovou barvou. Barva přechodu se vypočte na základě vzdálenosti od středu.

Hlavní výhodou barevných přechodů je nízká paměťová náročnost, jelikož stačí uchovávat pouze kontrolní barvy. Barevné přechody jsou použity jako základ aplikace skyboxu při vykreslování oblohy i oceánu. Gradient oblohy se skládá z barvy horizontu, počáteční, základní a koncové barvy oblohy a barvy Slunce [14].

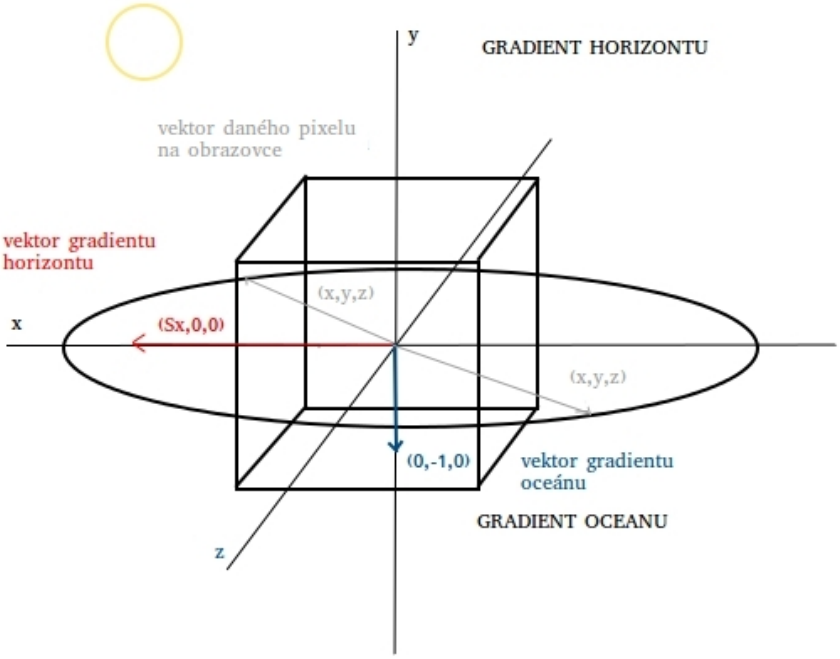
3.1.1 Gradient oblohy

Vytvoření gradientu oblohy se provede v několika krocích postupným smícháním jednotlivých barevných přechodů:

1. Nejprve nastavíme barvu celé scény skyboxu na základní hodnotu $c_0 = (0, 0, 0)$.

2. Jako první se vytvoří barevný přechod pro horizont, jenž je určen dvěma kontrolními barvami a vektorem závislým na pozici Slunce. Gradient horizontu se mění v závisloti na směru pohybu Slunce. Pokud by se Slunce pohybovalo ve směru osy x, byl by vektor gradientu definován jako $\vec{g} = (\vec{s}_x, 0, 0)$. Výslednou barvu horizontu v daném bodě vypočeteme jako $c_h = f(c_{h1}, c_{h2}, \vec{g} \bullet \vec{x})$, kde $\vec{x}$ je vektor pixelu a f reprezentuje funkci lineární interpolace 3.1. V pravé poledne, kdy je vektor ke Slunci roven $\vec{s} = (0, 1, 0)$, má gradient horizontu všude stejnou barvu.

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) \tag{3.1}$$

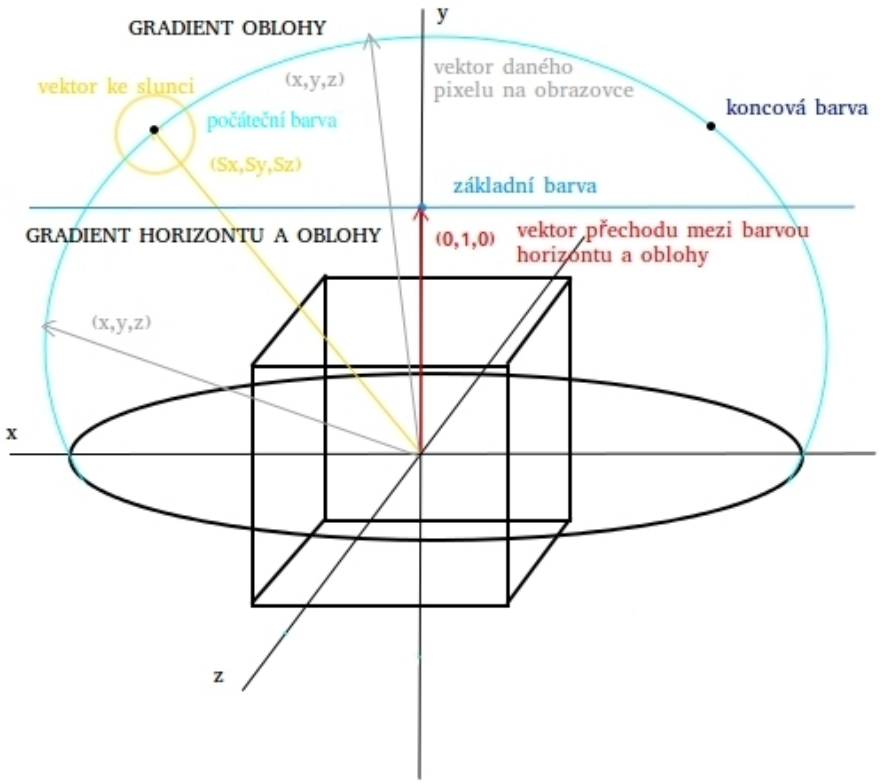


Obrázek 3.2: Znázornění vektoru gradientu oblohy

3. Dále je potřeba vypočítat přechody mezi barvami oblohy. U oblohy počítáme se třemi kontrolními barvami: základní barvou c_{oz}, počáteční barvou c_{op} a koncovou barvou c_{ok}. Nejprve určíme přechod mezi počáteční a koncovou barvou. Počáteční barva je barva blízko Slunce a koncová je barva na opačném konci oblohy, než se Slunce nachází. Výsledná barva je tedy dána vztahem $c_{opk} = f(c_{op}, c_{ok}, \vec{s} \bullet \vec{x})$.

4. V dalším kroku vytvoříme přechod mezi výslednou barvou horizontu c_h a základní barvou oblohy c_{ob}. Barva přechází od horizontu vzhůru, ve směru osy y. Vektor gradientu je tedy $\vec{g} = (0, 1, 0)$. Výsledná barva je dána rovnicí $c_{hob} = f(c_h, c_{ob}, (\vec{g} \bullet \vec{x})^e)$, kde exponent e určuje, jak vysoko dosahuje barva horizontu.

5. Nakonec se smíchá přechod barev horizontu k obloze c_{hob} s gradientem oblohy c_{opk}. Stejně jako v předchozím kroku je vektor gradientu roven $\vec{g} = (0, 1, 0)$. Konečná barva gradientu oblohy pro daný pixel se vypočte jako $c_o = f(c_{hob}, c_{opk}, (\vec{g} \bullet \vec{x})^e)$.



Obrázek 3.3: Znázornění vektorů gradientů oblohy

Barevné přechody oblohy jsou závislé převážně na pozici Slunce. Slunce je reprezentováno taktéž pomocí gradientu, přesněji pomocí úhlového gradientu. U Slunce je zapotřebí určit následující vlastnosti: vektor jeho pozice $\vec{s}$, barvu c_s, velikost a velikost okolní záře. Smíchání barvy Slunce a konečné barvy gradientu oblohy je řízeno následnující rovnicí $c_{os} = f(c_o, c_s, (\vec{s} \bullet \vec{x})^e) \cdot k$. Jedná se v podstatě o stejnou rovnici jako u vztahu mezi počáteční a koncovou barvou oblohy, rozdíl je ve velikosti exponentu e. Při vytváření Slunce bude jeho hodnota značně větší než v případě barev oblohy. Změnou velikosti exponentu se

mění také velikost Slunce. Čím je menší hodnota exponentu, tím větší je Slunce. Konstanta k určuje hranici mezi plnou barvou Slunce a okolní září. Čím vyšší hodnota konstanty k se zvolí, tím je tento přechod ostřejší.

Obrázek 3.4: Původní e i k

Obrázek 3.5: Menší e

Obrázek 3.6: Menší k

Nakonec je zapotřebí definovat, jakým způsobem se získají kontrolní barvy. Bylo by obtížné vytvořit konfigurační tabulku, kde by byly určeny všechny kontrolní barvy pro oblohu v průběhu dne. Z toho důvodu se zadefinují pouze kontrolní barvy v několika podstatných bodech například při východu Slunce, v pravé poledne, při západu Slunce a o půlnoci. Kontrolní barvy v reálném čase se pak vypočtou interpolací mezi těmito styčnými body. Postup výpočtu těchto přechodů mezi kontrolními barvami je popsán v implementační části v sekci 5.3.

3.1.2 Gradient oceánu

Barevný přechod oceánu je tvořen přechodem mezi barvou horizontu c_h a základní barvou moře c_s. Vektor gradientu oceánu směřuje opačně, než je tomu u vektoru přechodu barev od horizontu směrem k obloze. Vektor gradientu je tedy $\vec{g} = (0, -1, 0)$. Výsledná barva je dána rovnicí $c_{hs} = f(c_h, c_s, (\vec{g} \bullet \vec{x})^e)$, kde exponent e určuje, jak daleko ještě barva horizontu ovlivňuje barvu oceánu.

3.2 Funkce šumu

Funkcí šumu se v počítačové grafice využívá k dosažení realistického zobrazení některých efektů, zejména přírodních úkazů, jako například simulaci ohně, kouře, mraků, vlnění oceánu, vytváření horských výběžků atd.

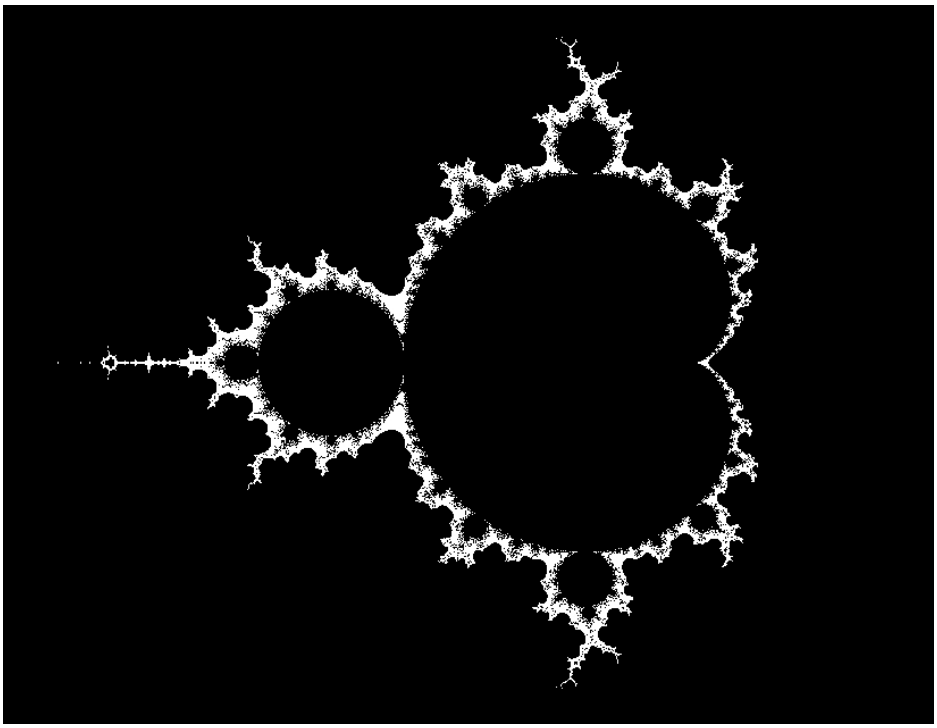
Nejedná se o běžné generátory náhodných čísel 3.2, jichž se využívá v mnoha oblastech informatiky. Jejich použití ke grafické vizualizaci výše zmíněných jevů není vhodné. Problematický aspekt aplikace generátorů náhodných čísel spočívá v tom, že pro stejné vstupní hodnoty nevygenerují totožný výstup. Tedy při opakovaném vykreslení scény bychom vždy dostali jiný výsledek. Takové chování není mnohdy žádoucí. Navíc prudké střídání hodnot na výstupu těchto generátorů má za příčinu nepřirozený vzhled.

Příklad kongruentního generátoru čísel generujícího pseudonáhodná čísla z intervalu $\langle 0, m \rangle$, jenž je definován následujícím rekurentním vztahem:

$$x_n = (a_1 x_{n-1} + \ldots + a_k x_{n-k} + b) \quad mod \quad m; \quad n \leq 0 \tag{3.2}$$

Kde $x_0, \ldots, x_k$ jsou počáteční hodnoty, $a_1, \ldots, a_k$ multiplikativní členy, b aditivní člen a m celočíselné modulo [16].

V přírodě mnoho objektů vykazuje fraktální vlastnosti; podrobnější informace je možné nalézt v knize *The Fractal Geometry of Nature* [7]. Tyto objekty na první pohled vypadají složitě. Pokud bychom však prozkoumali jejich strukturu detailněji, zjistili bychom, že se skládají ze stále se opakujících vzorů invariantních vůči změně měřítka. Tato vlastnost je partná na obrázku 3.7.



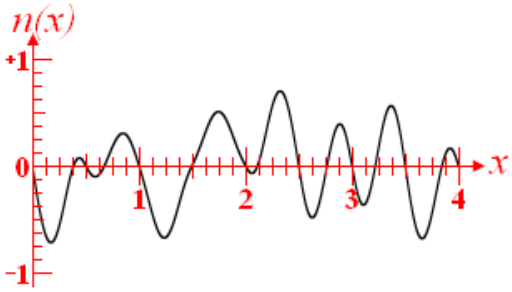
Obrázek 3.7: Ukázka fraktálních vlastností: Mandelbrotova množina

Vhodnou metodu pro počítačovou vizualizaci podobných jevů představuje Perlinův šum. Tato funkce byla představena v roce 1985 Kenem Perlinem na Konferenci o počítačové grafice a interaktivních technikách v New Yorku [8]. V aplikaci bylo využito upravené verze Perlinova šumu, někdy označované jako *Pink noise*. Výpočet původní funkce šumu, jež byla prezentována Kenem Perlinem, je založen na mírně odlišných principech. U funkce Pink noise je simulace fraktálních vlastností dosaženo pomocí sloučení několika funkcí šumu o různé frekvenci.

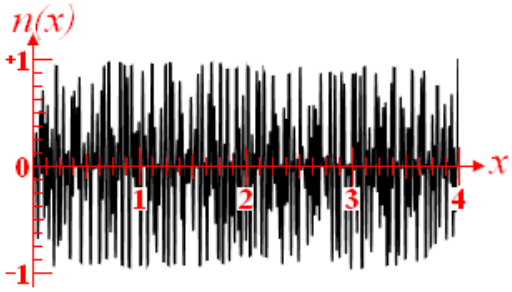
Perlinův šum je možné nahradit například funkcí šumu *Simplex noise*, jejímž autorem je rovněž Ken Perlin. Funkce Simplex noise byla navržena s ohledem na vyřešení daných omezení „klasické“ funkce šumu. Její hlavní výhodou je menší počet hodnot, mezi kterými se interpoluje, potřebných k výpočtu hodnoty šumu v daném bodě. Výraznější zrychlení výpočtů je možné zaznamenat u většího počtu dimenzí. Tento šum je rovněž izotropní, tedy nezávislý na směru, což je vlastnost, která by mohla řešit problém s *aliasingem*, jenž nastane například při simulaci mraků s použití Perlinova šumu, viz 3.16.

3.2.1 Perlinův šum

Metodika Perlinova šumu je obdobná jako u generátoru náhodných čísel. Vstupní hodnotou funkce je reálná hodnota, na jejímž základě je vrácena pseudonáhodná hodnota. Odlišné chování funkce se projeví ve chvíli, kdy jsou funkci opakovaně předány tytéž vstupní hodnoty. Výstupem funkce je poté vždy stejný výsledek. Taktéž budou-li rozdíly mezi čísly, které jsou funkci předány, malé, i výstupní hodnoty si budou podobné. Rozdíl mezi generátorem náhodných čísel a funkcí šumu je patrný na obrázcích 3.8 a 3.9.



Obrázek 3.8: Koherentní šum [1]

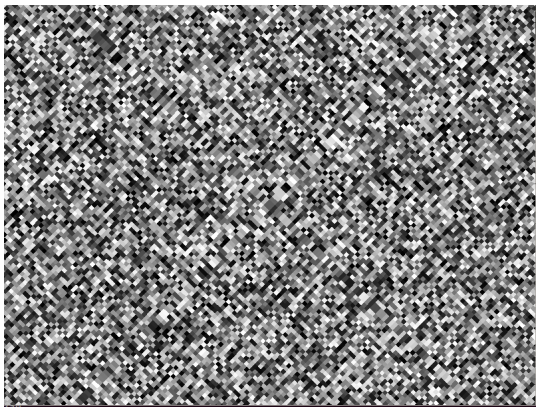


Obrázek 3.9: Nekoherentní šum

Základem Perlinova šumu je deterministický generátor pseudonáhodných čísel. Je deterministický, protože vrací stejný výsledek při totožných hodnotách předaných na vstup, čímž splňuje jeden z požadavků zmíněných výše. Jeho výstupem jsou hodnoty v intervalu $\langle -1, 1 \rangle$. Generátor je definován následujícím vztahem:

$$\begin{aligned} G(x,y,z) &= ((x \cdot 61 + y \cdot 89 + z \cdot 57) \ll 13)^{(x \cdot 61 + y \cdot 89 + z \cdot 57)} \\ Noise(x,y,z) &= 1 - \frac{(G(G^2 \cdot s_1 + s_2) + s_3) \quad \& \quad maska}{1073741824} \end{aligned} \tag{3.3}$$

Kde $\ll$ značí bitový posun doleva a & je bitový operátor logického součinu. Hodnoty s_1, s_2, s_3 nahrazují prvočísla pohybující se řádově v rozsahu $\langle 10^2, 10^3 \rangle$, respektive $\langle 10^4, 10^5 \rangle$, respektive $\langle 10^8, 20^8 \rangle$. Provedením bitové operace součinu s maskou, odpovídající maximální hodnotě 32bitového čísla, je zabráněno případnému přetečení hodnoty [12].

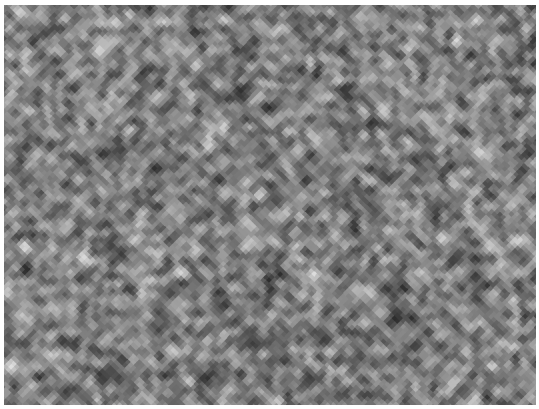


Obrázek 3.10: Výstup generátoru definovaného vztahem 3.3

Je možné nalézt různé definice takového generátoru. Čísla použitá v první části definice 3.3 nejsou striktně dána. Je však nezbytné, aby se jednalo o podobně velká prvočísla. A zároveň

aby zůstaly zachovány určené vlastnosti generátoru. Takto definovaný generátor je základem pro tvorbu 3D šumu. V případě jiného počtu dimenzí postačí upravit výpočet hodnoty G, zbytek definice zůstane stejný.

Dalším krokem k vytvoření funkce Perlinova šumu je provedení interpolace mezi hodnotami získanými z generátoru 3.3. S ohledem na lepší vizualizaci je však nejprve vhodné provést vyhlazení vygenerovaných dat. Při vyhlazení je pro hodnotu v daném bodě použita průměrná hodnota příslušné hodnoty a hodnot sousedních. Dojde tak k odstanění zrnitosti původního šumu, což je patrné z obrázku.



Obrázek 3.11: Výstup po vyhlazení šumu

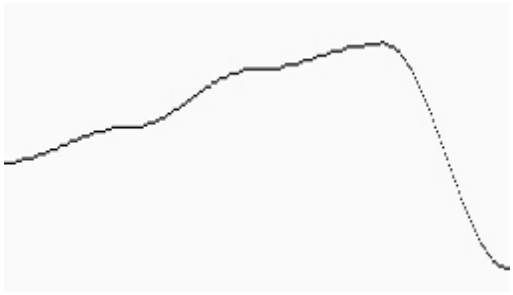
Protože generátor 3.3 generuje hodnoty na základě celých čísel, je hodnota mezi nimi nedefinovaná. Interpolace slouží k výpočtu ostatních hodnot funkce z příslušné diskrétní množiny. Existuje několik druhů interpolací, které se liší ve složitosti výpočtu a výsledné kvalitě. V aplikaci skyboxu byla vyzkoušena lineární a kosinová interpolace, z nichž byla nakonec zvolena kosinová interpolace. Výpočet kosinové interpolace není o mnoho složitější než lineární a výsledná kvalita šumu je vyhovující. Další možností je například kubická interpolace. Ačkoli lze pomocí této metody dosáhnout velmi dobrých výsledků, výrazně poklesne rychlost výpočtu. Výpočet kosinové interpolace:

$$\begin{aligned} f(x) &= (1 - \cos(x \cdot \pi))\frac{1}{2} \\ I_{cos}(a, b, x) &= a(1 - f(x)) + b \cdot f(x) \end{aligned} \tag{3.4}$$

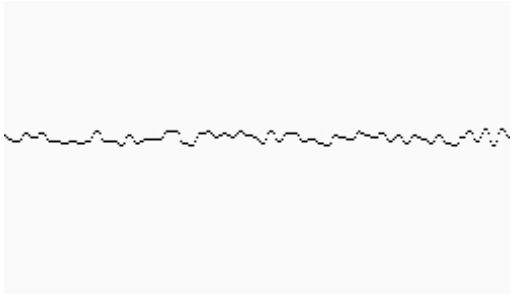
Posledním krokem je sloučení několika interpolovaných funkcí dohromady. Každá postupně přidávaná funkce se nazývá oktáva. Každá z oktáv má jinou amplitudu a frekvenci 3.12. Frekvence je definována jako 1/vlnová délka, kde vlnová délka je vzdálenost mezi vrcholy funkce. Amplitudou se rozumí rozdíl mezi maximální a minimální hodnotou, jichž daná funkce nabývá. Většinou se využívá šumu, kde každá oktáva má dvojnásobnou frekvenci a poloviční amplitudu. Při vizualizaci některých objektů je však vhodnější vztah mezi nimi měnit. Proto se do definice Perlinova šumu zavádí pojem perzistence. Pojem perzistence použil ve své práci o fraktální geometrii [7] francouzský matematik Benoît Mandelbrot. Vztah perzistence vůči frekvenci a amplitudě lze vyčíst z definice 3.2.1. Z obrázku 3.23 lze vypozorovat vliv hodnoty perzistence na výsledný šum. Výslednou hodnotu Perlinova šumu

získáme následovně:

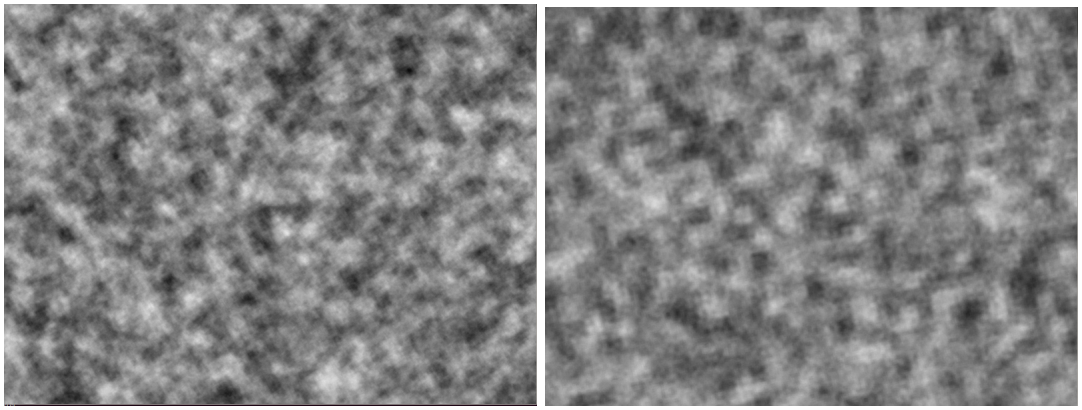
$$\begin{aligned} &PerlinNoise(x,y,z) = \sum_{i=0}^{n} p^i \cdot InterpolatedNoise(2^i x, 2^i y, 2^i z) \\ &frekvence = 2^i \\ &amplituda = perzistence^i \end{aligned} \tag{3.5}$$



Obrázek 3.12: Vysoká amplituda a nízká frekvence [4]



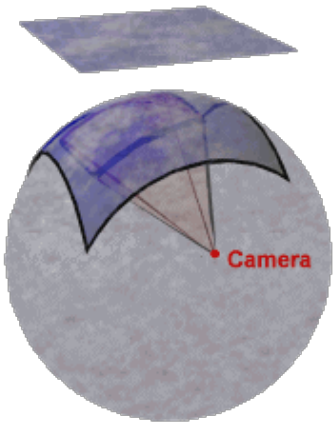
Obrázek 3.13: Nízká amplituda a vysoká frekvence



Obrázek 3.14: Perlinův šum čtyři oktávy; vlevo: persistence = 0,5; vpravo: persistence = 0,25

3.3 Generování mraků

Existuje několik různých technik, jak v počítačové grafice generovat mraky. Zvolení vhodné techniky vytváření mraků pro danou aplikaci je velice důležité. Je potřeba se zamyslet nad tím, jakou míru realističnosti mračen v aplikaci vyžadujeme. Například při vývoji leteckého simulátoru bude určitě zapotřebí namodelovat 3D mračna, která i při průletu skrz ně budou působit reálně. V aplikaci skyboxu, kde je kamera umístěna ve středu scény, postačí 2D mračna. Ušetří se tak výpočetní výkon a při vykreslení více vrstev mraků nad sebou, volbě vhodného osvětlení a úpravě dalších vlastností lze dosáhnout dobrých výsledků.
Mraky jsou reprezentovány nekonečnou rovinou s Perlinovým šumem, definovaným v podsekci 3.2.1. Rovina je umístěna nad scénu skyboxu. Změny tvaru mračen v reálném čase lze docílit použitím 3D šumu, kde právě čas tvoří třetí rozměr. Hodnota Perlinova šumu daného pixelu se zjistí tak, že se dohledá průsečík $P(x, y)$ směrového normalizovaného vektoru



Obrázek 3.15: Příklad mračen reprezentovaných nekonečnou rovinou s naneseným šumem [3]

u tohoto pixelu a roviny v daném čase. Souřadnice $I(x, y, t)$ se poté použijí jako vstupní hodnoty funkce šumu [14]. Přičtením směrového vektoru k těmto souřadnicím ovlivníme pohyb mraků. Výpočet průsečíku pixelu určeného směrovým vektorem $\vec{u} = (x, y, z)$ a roviny ve výšce h je definován nasledujícím vztahem:

$$t = \frac{h}{u_z}$$
$$P(x, y) = u(x, y) \cdot t \tag{3.6}$$

Vypočtená hodnota Perlinova šumu je následně použita při smíchání barvy gradientu oblohy, jejíž výpočet je popsán v sekci 3.1.1, a barvy mraků. Rovnice míchání výsledné barvy vypadá následovně:

$$c = c_1(1 - a) + c_2 \cdot a \tag{3.7}$$

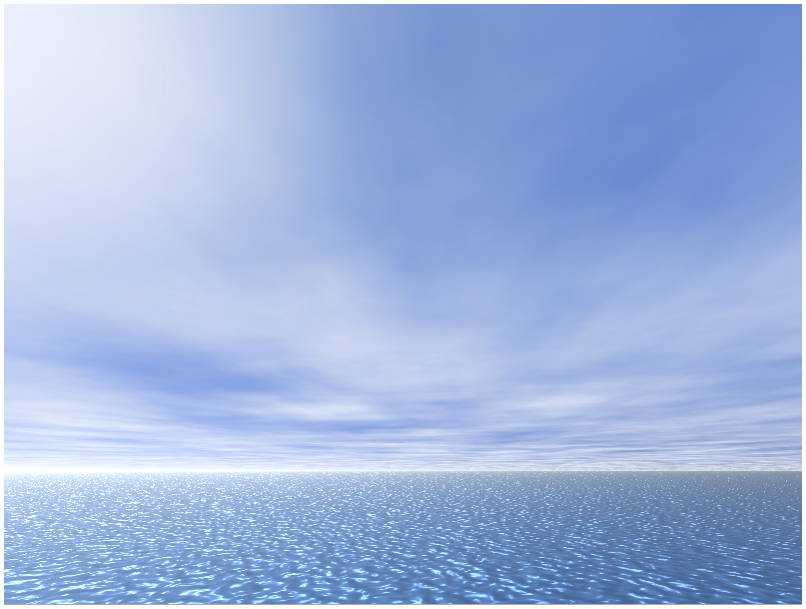
Proměnná a, v tomto případě návratová hodnota Perlinova šumu, určuje poměr mezi barvami c_1 a c_2.

Takto vytvořená mračna pokrývají celou plochu oblohy. Jejich vzhled je ovlivněn barvou a parametry funkce šumu: amplitudou, frekvencí, perzistencí a počtem oktáv.

Oblaka s těmito vlastnostmi jsou nejvíce podobná mrakům typu Stratus. K reprezentaci oblak typu Cumulus je zapotřebí výstup funkce šumu ošetřit vhodným filtrem. Odlišnosti jednotlivých typů mračen jsou popsány v podsekci 2.3.

Při vytváření vrstvy mraků Cumulus tedy není zapotřebí definovat novou funcki šumu s jinými parametry, ale upraví se výstup té stávající tak, aby se zmenšila plocha pokrytá mračny a zvýšila se hustota těchto mraků. Díky tomu se redukuje počet výpočtů, který by s každým voláním nové funkce šumu značně narůstal. Úprava se provede ve dvou krocích, kdy se na výstup fukce šumu aplikují různé filtry:

- V prvním kroku se zvolí konstantní hodnota c z intervalu $\langle 0, 255 \rangle$ (rozsah hodnot osmibitového obrazu v odstínech šedi), která bude hranicí pro odstíny šedi, jež se



Obrázek 3.16: Vrstva mračen Stratus vykreslená v aplikaci. Použité parametry: 2D Perlinův šum, 4 oktávy, persistence = 0.25. U horizontu je patrný mírný aliasing.

propustí filtrem. Pokud hodnota šumu daného bodu bude menší než konstanta c, obarví se tento pixel barvou gradientu oblohy. Tímto se redukuje oblast oblohy pokrytá mračny.

- Druhým krokem je přeměna mlhy připomínající oblak na mraky s vyšší hustotou. Toho lze docílit aplikací filtru definovaného vztahem $f_h = h_{max} - (s^c \cdot h_{max})$. Konstanta s, vybraná z intervalu $\langle 0, 1 \rangle$, udává, jak hustá výsledná mračna budou. Hodnota h_{max} představuje maximální hustotu mraků. Pokud se hodnota konstanty s blíží jedné, výsledná mračna jsou řidší, jak je patrné z obrázku 3.17, 3.18 a 3.19.

Obrázek 3.17: $s = 0,83$

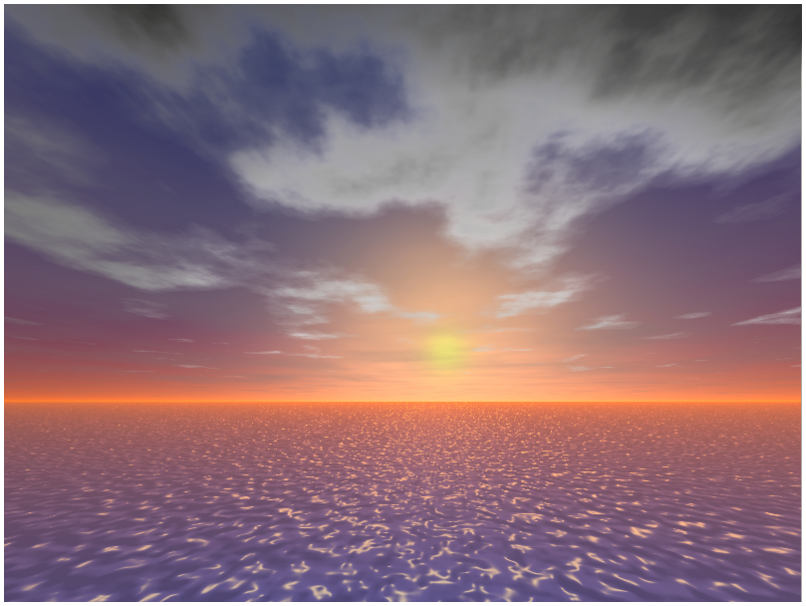
Obrázek 3.18: $s = 0,65$

Obrázek 3.19: $s = 0,35$

Výsledek po sloučení několika vrstev mraků, které se liší barvou a parametry aplikovaných filtrů, je vidět na obrázku 3.20. Mraky nedosahují až k hranici horizontu, protože se směrem k horizontu zvyšuje jejich průhlednost. Důvodem je odstranění aliasingu, který vznikne při použití Perlinova šumu.

Barva mraků je závislá na jejich sytosti a na pozici vůči Slunci. Předpokládáme, že mraky osvětlují pouze paprsky pocházející přímo ze Slunce. Slunce neovlivňuje přímo barvu mraků, ale pouze jejich jas. Nejjasnější budou mraky v blízkosti Slunce a naopak. Hodnota jasu

daného pixelu se určí podle hodnoty skalárního součinu směrového vektoru tohoto bodu a vektoru ke Slunci.



Obrázek 3.20: Výsledná mračna po sloučení několika vrstev a přidání osvětlení

3.4 Bump mapping

Bump mapping, případně *normal mapping*, je technika nanášení textury takovým zbůsobem, že dochází k vytvoření iluze nerovnosti povrchu. Dosáhneme tak zvýšení úrovně detailů, a to beze změny geometrie povrchu. S příchodem moderních grafických karet se začala využívat také technika mapování textur zvaná *displacement mapping*, při které se zasahuje přímo do geometrie povrchu. Takové operace ovšem přispívají ke zvyšování výpočetních nároků.

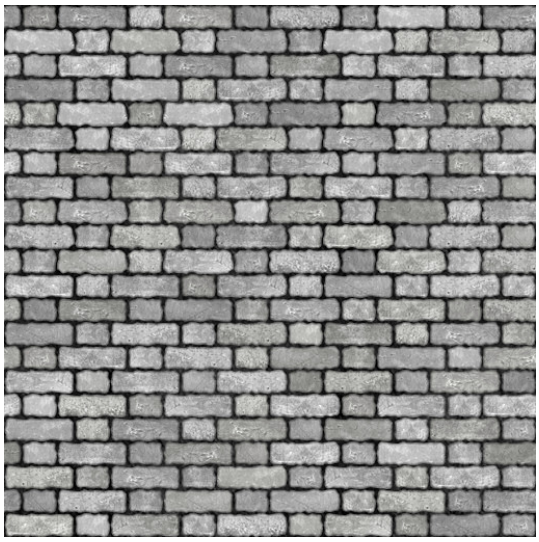
K vytvoření efektu vln oceánu 3.5, nad nímž je vykreslen skybox, bylo využito techniky bump mappingu. Základem bump mappingu je *bump mapa* 3.21, někdy nazývaná jako výšková mapa. Jedná se typicky o texturu, kde všechny složky RGB barvy pixelu mají stejnou hodnotu. Obsahuje tedy pouze odstíny šedi. Světlost pixelu pak určuje, nakolik má daný bod vystupovat z povrchu.

Z takto definované výškové mapy je zapotřebí získat informaci o naklonění povrchu v daném bodě, neboli modifikovaný normálový vektor. Při jeho určení je zapotřebí pro daný pixel vypočítat gradient v osách x a y (výpočet gradientu je znázorněn v rovnicích 3.8). Posuneme se o určitou vzdálenost fE v osách x, respektive y, a zjistíme hodnoty z výškové mapy v daných bodech. Rozdíl mezi těmito hodnotami je hledaný gradient.

$$\begin{aligned} GradX &= h(x + fE, y) - h(x - fE, y) \\ GradY &= h(x, y + fE) - h(fE, y - fE) \end{aligned} \tag{3.8}$$

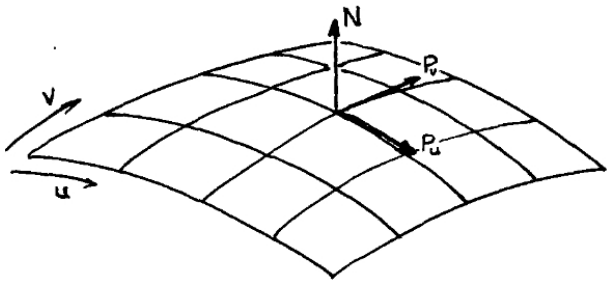
S pomocí gradientů následně upravíme původní normálový vektor polygonu v daném bodě. Modifikovaný normálový vektor lze dopočítat následovně:

$$mN = N + U \cdot (-GradX) + V \cdot (-GradY) \tag{3.9}$$



Obrázek 3.21: Ukázka bump mapy

N je původní normála. $\vec{U}$ a $\vec{V}$ jsou vektory rovnoběžné s osami bump mapy, jak je vidět na obrázku 3.22. Takto získané modifikované normály tvoří takzvanou *normal mapu*. Podobně jako u výškové mapy se jedná o texturu, která má v texelech uloženou normálu místo barvy. Získanou normal mapu lze poté použít při výpočtu Blinn-Phongova osvětlovacího modelu.



Obrázek 3.22: Znázornění vektorů $\vec{U}$ a $\vec{V}$ [2]

3.5 Generování oceánu

Aby ve scéně se skyboxem bylo vidět, v jaké pozici je pozorovatel, jak vzdálený je horizont a jak vysoko nad zemí se nachází kamera, je do scény přidána jednoduchá animace hladiny oceánu.

Oceán je určen svou barvou, směrem pohybu vln, výškou vln a množstvím odráženého slunečního světla.

- Výsledné barvy oceánu je dosaženo smícháním základní barvy oceánu a barvy gradientu horizontu v daném čase na základě vzdálenosti od horizontu.
- Směr pohybu vln lze ovlivnit přičtením směrového vektoru k souřadnicím textury oceánu. Složkou takového vektoru je čas t, upravený podle požadované rychlosti a směru pohybu.
- K výpočtu slunečního záření odráženého od hladiny oceánu slouží Blinn-Phongův osvětlovací model. K výpočtu je potřeba znát normovaný normálový vektor $\vec{N}$. Postup jeho výpočtu je popsán zde 3.4. Jak lze vidět na obrázku 5.1, nejvýraznější barevnou složkou v normálové mapě je zelená.

  Světlo se podle Blinn-Phongova modelu skládá ze tří složek:

  - ambientní složka – zbytkové světlo, rozptýlené mnoha odrazy;
  - difúzní světlo – světlo, jež se po dopadu na povrch rozptýlí do všech směrů;
  - zrcadlový odraz (spekulární složka) – světlo odrážející se od povrchu v určitém směru.

  Ambientní světlo **A** se vypočte pomocí vzorce $A = Ai \cdot Ac$, kde Ai je konstatní ambientní složka světla a Ac je ambientní složka barvy oceánu.

  Difúzní složku **D** světla lze získat jako součin konstatní difúzní složky světla Di, difúzní složky barvy oceánu Dc a skalárního součinu normálového vektoru N a vektoru směřujícího z bodu na hladině oceánu ke Slunci L. Množství odraženého světla je tedy závislé na velikosti úhlu mezi normálou povrchu a vektorem světla. Tento jev popisuje Labertův zákon. Výsledná intenzita difúzní složky světla se vypočte pomocí vzorce $D = Di \cdot Dc \cdot (N \cdot \vec{L})$, kde $\cdot$ značí skalární součin.

Obrázek 3.23: Vlevo: oceán pouze s ambientní sloužkou světla. Uprostřed: oceán s přidanou difúzní složkou světla. Už je patrný 3D efekt. Vpravo: plné osvětlení hladiny oceánu.

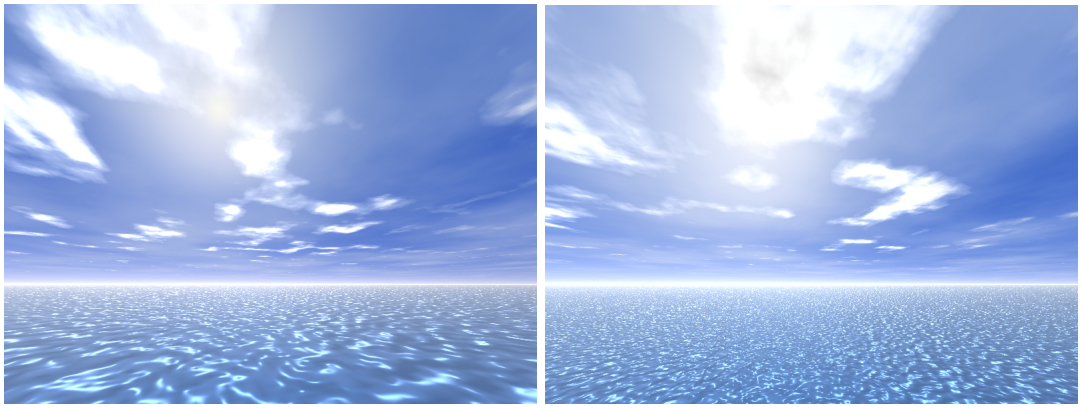
Spekulární složku **S** lze vypočítat pomocí vztahu $S = Si \cdot Sc \cdot (N \bullet \vec{H})^{\alpha}$. Hodnota Si je konstatní spekulární složka světla a Sc zastupuje spekulární složku barvy oceánu. Vektor $\vec{H}$ je takzvaný půlicí vektor (*halfway vector*). Nachází se mezi normalizovaným vektorem pozorovatele V a vektorem ke Slunci $\vec{L}$, je definován následujícím vztahem:

$$H = \frac{(L+V)}{|L+V|} \tag{3.10}$$

Vektor pozorovatele $\vec{V}$ lze získat odečtením souřadnice daného bodu na hladině od souřadnic pozice kamery. Exponent α určuje ostrost zrcalového odrazu. Čím je vyšší, tím jsou odlesky směřovány do menší oblasti a jsou intenzivnější.

Výsledná intenzita osvětlení v daném bodě je definována jako součet všech tří složek $I = A + D + S$.

K vytvoření animovaného oceánu je zapotřebí 3D textura obsahující normály bodů na hladině oceánu (vytvoření takové textury je popsáno zde 5.2). Vektor souřadnic do této textury $I(x, y, t)$ obsahuje souřadnice průsečíku $P(x, y)$, průsečík polopřímky (směrový vektor daného bodu) a roviny (hladina oceánu) a čas t. Postup zjištění souřadnic průsečíku P je definován zde 3.6. Pozice pozorovatele (kamery) je závislá na tom, do jaké výšky vůči scéně je umístěna rovina oceánu, jak je patrné z obrázku 3.24.



Obrázek 3.24: Rovina oceánu umístěná v různých výškách

3.6 Hvězdná obloha

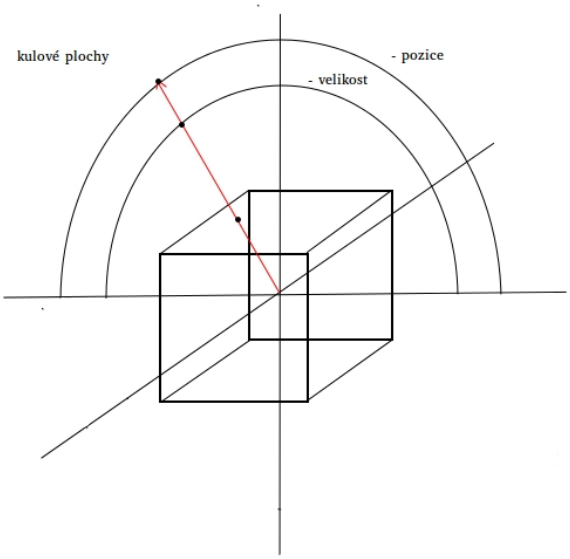
Na noční obloze lze pozorovat širokou škálu hvězd s různou intenzitou záření, jež jsou na první pohled rozmístěny zcela náhodně. Stejně jako Slunce, tak i hvězdy a Měsíc se zdánlivě pohybují po obloze, což je způsobeno rotací Země kolem vlastní osy.

Hvězdná obloha je reprezentována kulovou plochou o poloměru r, jež je umístěna kolem scény. Obdobně jako u mraků je potřeba zjistit průsečík $P(x, y, z)$ směrového vektoru $\vec{u} = (x, y, z)$ daného bodu a kulové plochy. Průsečík je poté použit při výpočtu šumu. Výpočet působečíku je definován následovně:

$$\begin{aligned} t &= u_x + u_y + u_z + r \\ P(x, y, z) &= u(x, y, z) \cdot t \end{aligned} \tag{3.11}$$

Tím, že použijeme takto definovaný průsečík při výpočtu 3D šumu, získáme sférickou mapu šumu. Na rozdíl od mraků, které jsou vytvořeny za pomoci funkce Perlinova šumu, je k vytvoření hvězd potřeba nekoherentního šumu. Vhodný je například generátor pseudonáhodných čísel podobný tomu, jenž je definován zde 3.3. Důležité je, že se jedná o deterministický generátor.

K reprezentaci hvězd jsou zapotřebí dvě kulové plochy. Každá z ploch má jiný poloměr. První plocha slouží k určení pozice, velikosti a počtu hvězd. Velikost hvězd se mění tím, jak velký je poloměr kulové plochy. Řádově je poloměr plochy mnohem větší než výška, v jaké je umístěna rovina reprezentující mraky, z toho důvodu, aby velikost jednotlivých „zrnek“ šumu přibližně odpovídala velikosti hvězd na noční obloze. Pixely určující hvězdy se z původní funkce šumu získají přefiltrováním jejích výstupních hodnot. Zvolí se číslo z intervalu $\langle 0, 1 \rangle$ určující hranici hodnot šumu, jež se nakonec vykreslí na noční obloze. Pokud se zvolí nižší hranice, propustí se filtrem více hodnot šumu a bude vykresleno více hvězd.

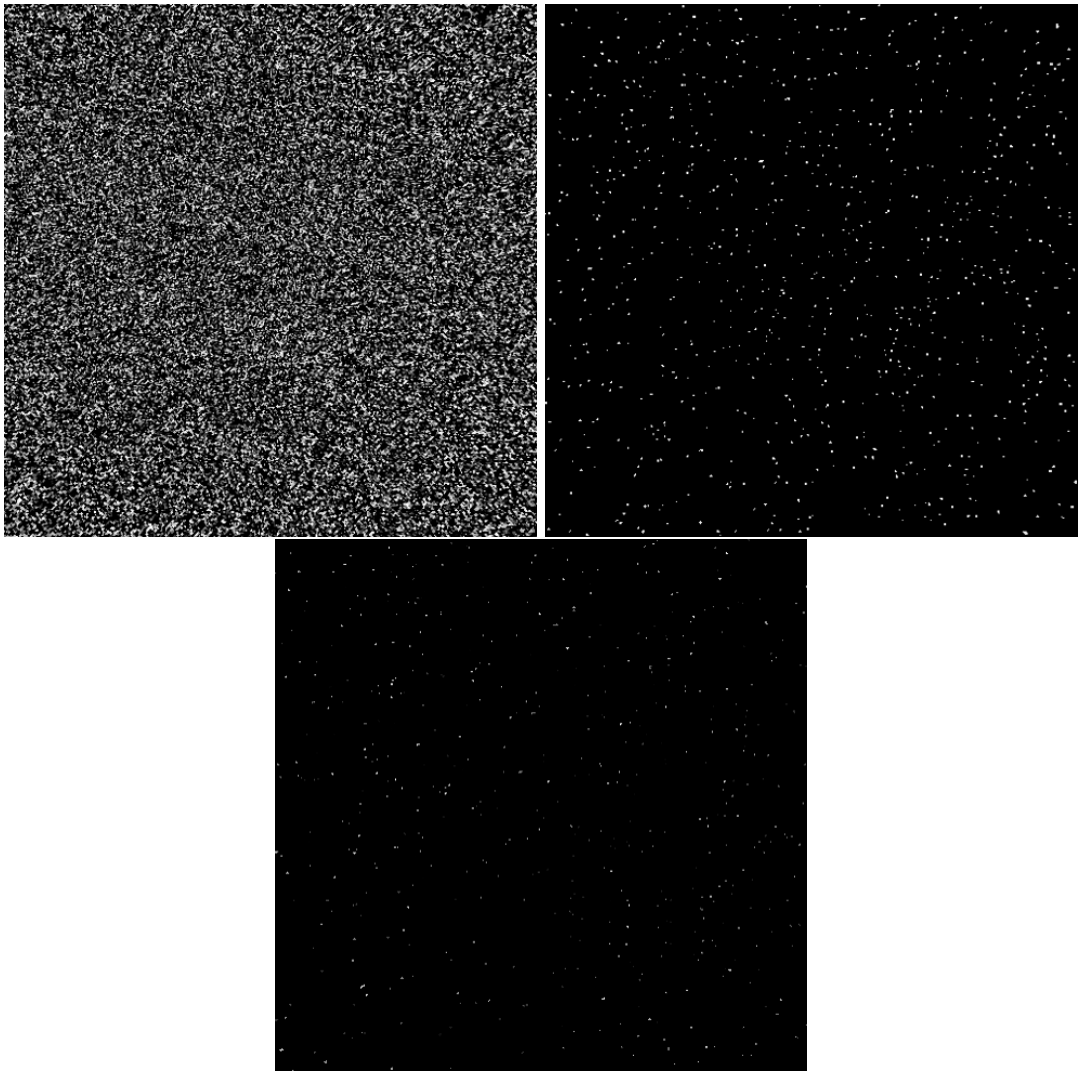


Obrázek 3.25: Znázornění průsečíku směrového vektoru bodu na obloze a kulových ploch reprezentujících hvězdnou oblohu

Druhá kulová plocha má menší poloměr než ta první a slouží k určení hvězdé velikosti hvězd (popsáno v sekci 2.2). Protože intenzita hodnot šumu, pomocí nichž jsou hvězdy reprezentovány, je omezena hranicí zmíněnou výše, je zapotřebí jejich intenzitu upravit jinou funkcí šumu. Jinak bychom neobsáhli celé spektrum hvězd, které je možné pozorovat na noční obloze. První kulová plocha šumu tedy určuje pozici hvězdy a druhá intenzitu v daném bodě. Na následujících obrázcích je vidět přeměna původní funkce šumu na hvězdnou oblohu.

Tím, že je šum mapován na kulovou plochu, narozdíl od mračen, kdy je šum mapován na nekonečnou rovinu, nedochází k efektu aliasingu. Taktéž se liší způsob, jakým se hvězdy pohybují po obloze. Hvězdy se pohybují podobně jako Slunce nebo Měsíc, jejich pohyb je možné definovat jako rotaci kolem určené osy. Rotace hvězd je dosaženo otočením směrového vektoru $\vec{u}$, jímž je hvězda určena. Tedy téhož vektoru, jehož průsečík s kulovou plochou je počítán v rovnici 3.11. Před výpočtem průsečíku je směrový vektor vynásoben následující rotační maticí:

$$R = \begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.12}$$



Obrázek 3.26: Postupná přeměna nekoherentního šumu na hvězdnou oblohu: na prvním obrázku je základní šum, na druhém obrázku jsou pouze přefiltrované hodnoty určující pozici hvězd, na posledním obrázku je výsledná podoba hvězd s určenou hvězdnou velikostí.

Kapitola 4

Experimenty

Aplikace skyboxu byla vyvíjena s cílem dosáhnout dobrých výsledků ve výsledném vzhledu vykreslené scény, pokud možno s co nejmenšími výpočetními nároky. Určitě by bylo možné dále redukovat počet výpočtů například použitím předpřipravených textur nebo prováděním více výpočtů na procesoru místo toho, aby se využívalo výkonu GPU (grafický procesor, *graphic processing unit*). V této kapitole jsou zmíněny experimenty s nastavením vykreslování prvků skyboxu. Dále je uvedeno, jak jednotlivé úpravy ovlivnily výkonostní nároky.

4.1 FPS

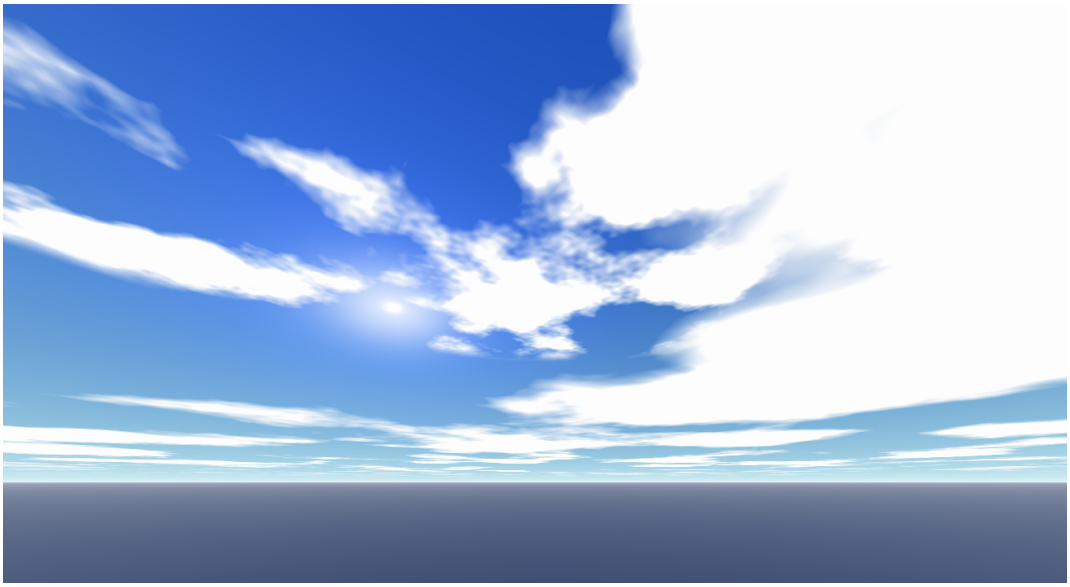
FPS (*frame per second*), neboli snímek za sekundu, je jednotka snímkové frekvence. Jedná se o frekvenci, s jakou vykreslovací zařízení zobrazuje jednotlivé na sebe navazující snímky. Pomocí hodnot FPS jsou v následujících pokusech demonstrovány výsledné výpočetní nároky jednotlivých variant řešení.

4.2 Vrstvy mraků

K vytvoření lepší iluze realných mračen, která jsou vykreslena dvourozměrně, je zapotřebí vykreslit oblaka v několika vrstvách. U vrstev se mění nastavení šumu, parametry filtru šumu a barva mračen. Důležité je však zvolit dobrý kompromis mezi kvalitou výsledného vzhledu a zpomalením simulace.

Při prvních pokusech bylo vykresleno několik vrstev oblak v různé výšce. Využilo se tedy několika rovin a vícero volání funkce Perlinova šumu. Výhoda toho přistupu spočívá v tom, že vrstva mraků definovaná rovinou umístěnou v malé výšce tvoří podlouhlé pásy mračen v blízkosti horizontu. Naopak vrstvou mraků reprezentovaných rovinou umístěnou výše nad scénou jsou vykreslena dobře vypadající oblaka výše od horizontu. Spojením těchto vrstev jsou vytvořena mračna souvisle pokrývající celou plochu oblohy. Problematickým aspektem v tomto ohledu je animace takto definovaných mračen a několikanásobné volání funkce šumu, jež zpomaluje vykreslování. Mraky nelze animovat jako celek a jejich výsledný pohyb tedy vypadá nepřirozeně. Pokud by v aplikaci nebyly vyžadovány animované mraky, postačil by k jejich vykreslení 2D šum, a mohlo by se tedy jednat o použitelné řešení.

Jak je popsáno v kapitole 3.3, k vytvoření vícevrstvých mraků stačí upravit parametry filtru použitého na výstup funkce šumu. Taková úprava by byla dostatečná i na vytvoření vstrvy mračen typu Stratus. Problémem je, že u takto vytvořených mračen lze jen velmi těžce rozlišit jednotlivé vstvy. Je sice viditelný mlhavý závoj v okolí větších oblak, ale



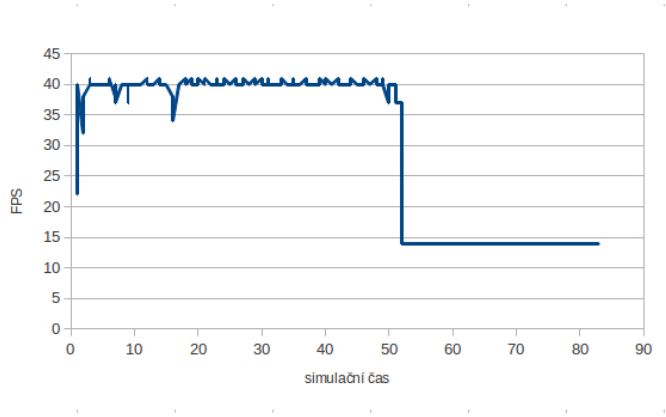
Obrázek 4.1: Ukázka mračen složených z několika různých vrstev

vrstva nepůsobí jako samostatný celek. Navíc se všechny mraky pohybují stejně rychle. Posun mračen je vypočten a předán na vstup funkci šumu, nelze jej tedy dále ovlivnit.

Jako nejvhodnější řešení se ukázalo vytvoření vrstvy mračen Stratus pomocí vlastní funkce 2D Perlinova šumu. Jelikož se jedná o mlhu připomínající mraky, není důležité, aby se měnil jejich tvar, a postačí 2D šum. Rozdíl v FPS není prakticky žádný a tím, že je možné tuto vrstvu umístit níže nad scénu, se zaplní prostor poblíž horizontu.

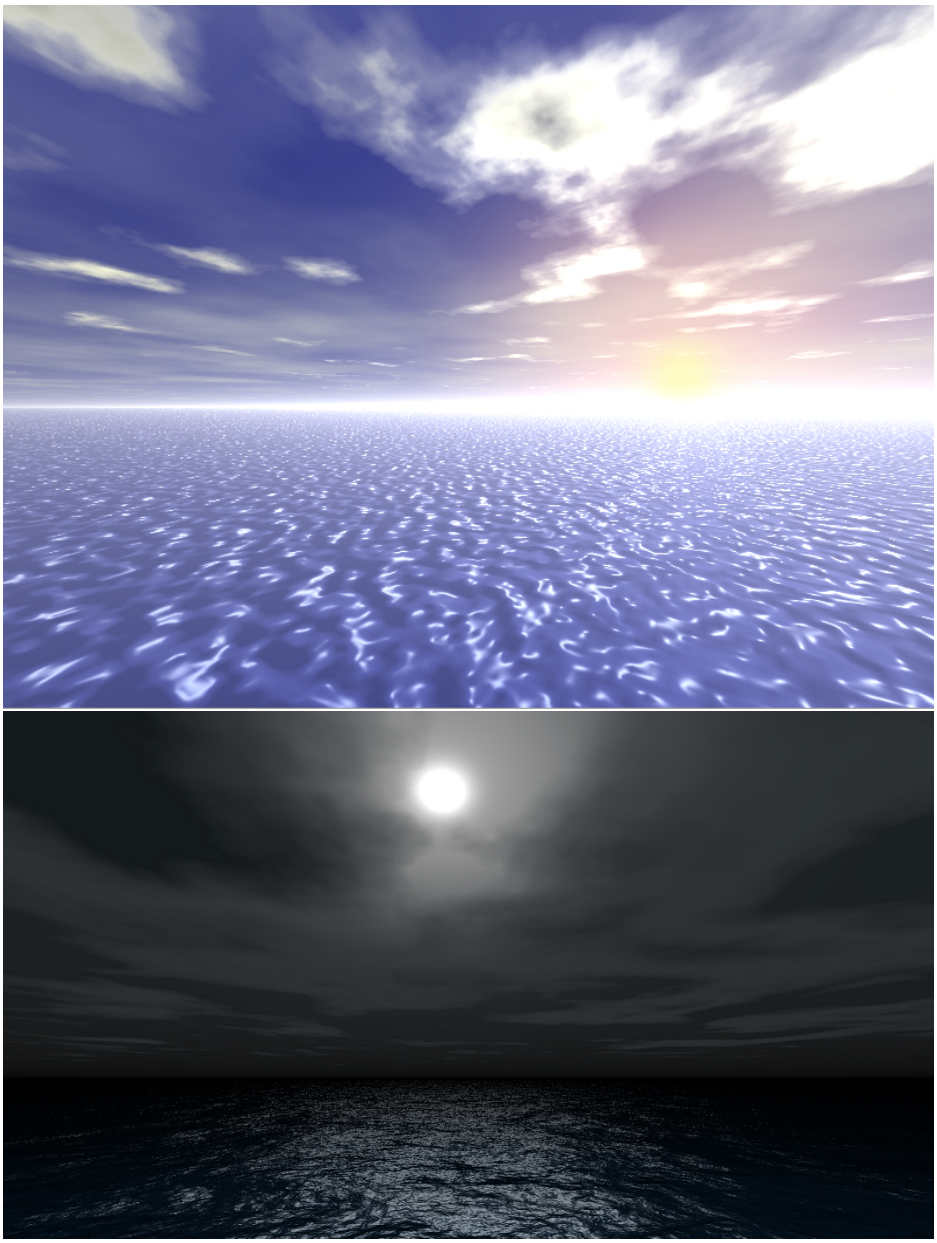
4.3 Oceán

Postup vytvoření oceánu popsaný v kapitole 3.5 představuje pouze jednu z možností. Normálu bodu na hladině oceánu lze rovněž počítat v reálném čase přímo v *pixel shaderu*. Pokud je algoritmus bump mappingu proveden přímo v shaderu, je možné pozorovat na hladině oceánů větší míru detailů. Na rozdíl od použití textury, jež se stále opakuje, aby pokryla celou plochu oceánu, je v shaderu vypočtena normála pro každý pixel zvlášť.



Obrázek 4.2: FPS v průběhu simulace se změnou způsobu vykreslování oceánu

Nevýhoda tohoto přístupu je patrná z grafu 4.2, na němž je zobrazeno kolísání FPS v průběhu simulace. Přibližne v době, kdy simulační čas překročil hodnotu 50, se místo načítaní normál z textury přešlo na výpočet normál v shaderu. Hodnota FPS poklesla o více než polovinu. Vysvětlení příčiny vyšší náročnosti výpočtu normály v reálném čase lze nalézt v implementační kapitole 5.2.



Obrázek 4.3: Rozdíl ve výsledném vzhledu oceánu. Nahoře: normály oceánu uloženy v textuře. Dole: normály oceánu vypočteny ve fragment shaderu.

Kapitola 5

Implementační podrobnosti

Tato kapitola se zabývá přesnějším popisem implementace některých zajímavých částí projektu. Jsou zde zmíněny informace o vybraných knihovnách, jež byly použity v aplikaci, stručný popis jazyka GLSL a jeho vyžití při vývoji aplikace.

5.1 Použité knihovny

Projekt byl vyvíjen v systému Ubuntu Linux. Okno aplikace je vytvořeno pomocí knihovny GLUT (*OpenGL Utility Toolkit*), nadstavby knihovny OpenGL. Dále je využita nadstavba GLEW (*OpenGL Extension Wrangler Library*), jež poskytuje mechanismy k určení podporovaných rozšíření OpenGL na cílové platformě. K výpočtu tranformačních matic, perspektivní projekce a práci s vektory, mimo program shaderu, je využita knihovna GLM (*OpenGL Mathematics*), která poskytuje podobné třídy a funkce jako GLSL. K projektu je připojena také knihovna SDL (*Simple DirectMedia Layer*), jejíž funkce jsou použity při výpočtu FPS.

5.2 Generování textury normálové mapy

Textury je možné vytvářet několika způsoby. Jedním ze způsobů je použití klasického rastrového obrázku jako textury. Další možností je vytváření takzvaných procedurálních textur, jež se generují pomocí různých algoritmů. Procedurální textury je možné vytvořit předem a uložit do rastru, nebo se jejich výpočet provede až při samotném vykreslování.

Grafické architektury, u kterých se hojně využívá rastrově uložených textur, mají většinou velký paměťový prostor, jenž je spravován rychlými procesory. Získání dat rastrové textury je nenáročné na výpočet, vyvažuje to však množství paměti potřebné k jejich uložení. V minulosti takovou architekturu zastupovala například společnost *The Silicon Graphics Skywriter*. Dnes už paměťové nároky na uložení textur nejsou takovým problémem [10]. Nevýhodou rastrových textur je jejich pevně dané rozlišení. Pokud je zapotřebí provést zvětšení či změnšení rastrové textury v době vykreslení, může docházet k aliasu.

Nevýhodou procedurálních textur generovaných matematickými funkcemi v reálném čase je jejich poměrně náročné vytváření, naopak následná manipulace s nimi je podstatně jednodušší. Procedurální textury je možné parametrizovat, například měnit jejich rozlišení.

V aplikaci je využita 3D textura, v níž jsou uloženy normály oceánu. Prvně byly normály

potřebné k vytvoření hladiny oceánu počítány přímo ve *fragment shaderu*. Jak vyplývá z algoritmu bump mappingu popsaného v sekci 3.4, k výpočtu normály v dáném bodě je nutné znát také hodnoty čtyř okolních bodů z výškové mapy. To znamená, že k určení normály jednoho bodu na hladině oceánu je nutné volat čtyřikrát funkci Perlinova šumu, pokud výpočet probíhá přímo ve fragment shaderu. Při větším počtu oktáv funkce šumu je zpomalení aplikace neúnosné. Proto jsou v aktuální verzi aplikace normály nejprve vypočteny na procesoru a v pixel shaderu se dopočítá pouze osvětlení hladiny oceánu.

Textura pro normály oceánu je definována následovně:

```
glTexImage3D(GL_TEXTURE_3D,0,GL_RGB16F,TSize,TSize,TSize,...
```

První parametr definuje, že se jedná o 3D texturu. Parametr `GL_RGB16F` určuje, že pro každou složku barvy, červenou, zelenou, respektive modrou, jsou vyhrazeny dva bajty paměti. Celkově tedy šest bajtů paměti na texel, v němž je místo barvy uložena normála. Velikost textury je určena hodnotou `TSize`. Nelze zadefinovat příliš velkou texturu, jelikož u trojrozměrné textury se doba výpočtu značně prodlouží za předpokladu, že vzroste její velikost. Ve výsledku je tedy hladina oceánu tvořena jedním vzorem, jenž se stále opakuje. Pokud není textura vytvořena tak, aby na sebe navazovala, jsou místy patrné přechody mezi jednotlivými opakováními. Jak je vidět na obrázku normálové mapy 5.1, jednotlivé textury na sebe úplně ideálně nenavazují. U textury jsou dodatečně definovány další dva parametry:

```
glTexParameteri(GL_TEXTURE_3D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_3D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
```

Podle úrovně detailu se textura při vykreslování zmenšuje, nebo zvětšuje. Předešlými paramtery se nastaví, jaká funkce je použita v případě těchto operací. Podrobnější informace o dostupných funkcích lze nalézt na oficiálním webu OpenGL [13]. V aplikaci je v obou případech využito funkce `GL_LINEAR`, jež počítá s váženým průměrem čtyř nejbližších texelů.

Před vytvořením textury normál je nejprve nutné vygenerovat výškovou mapu oceánu. Výšková mapa je definována polem, v němž jsou uloženy hodnoty funkce Perlinova šumu.

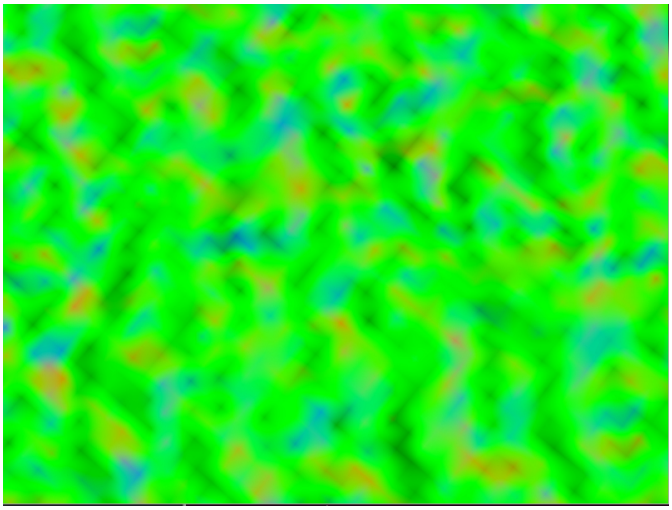
```
OceanNoise[TSize*TSize*TSize];
```

Z výškové mapy jsou poté vypočteny normály a následně uloženy do textury.

5.3 Výpočet aktuální konfigurace barev

Jak bylo řečeno v sekci 3.1.1, z konfiguračního souboru jsou načteny pouze konfigurace barev skyboxu ve čtyřech základních kontrolních bodech. K výpočtu aktuální konfigurace barev v reálném čase je tedy zapotřebí získat průběžné hodnoty mezi těmito kontrolními body. Postup je následující:

1. Nejprve se podle pozice Slunce zjistí, mezi kterými dvěma kontrolními body se momentálně nachází simulační čas. První kontrolní bod určuje, jaká je aktuální denní doba `TOD` (*time of day*). V proměnné `time_dy` je uložena délka časového intervalu mezi těmito body. Poslední hodnota, kterou je potřeba znát k získání aktuální konfigurace,



Obrázek 5.1: Normálová mapa hladiny oceánu

je posun pozice Slunce vůči pozici, v jaké se Slunce nacházelo v prvním kontrolním bodě `sunShift`.

2. Tako definované hodnoty jsou předány funkci:

   ```
   getSkyConfInTOD (int dayTime,float time_dy, float sunShift, int varCl);
   ```

   Poslední parametr `varCl` funkce `getSkyConfInTOD` určuje, pro jakou barvu se má aktuální hodnota vypočítat. Barevná konfigurace aplikace skyboxu obsahuje celkově devět základních barev: základní barva oblohy, koncová barva oblohy, počáteční barva oblohy, počáteční barva horizontu, koncová barva horizontu, barva oceánu, barva Slunce, barva první vrstvy mračen a barva druhé vrstvy mračen. Samozřejmě vykreslená scéna není tvořena jenom těmito barvami, tyto barvy slouží jako základ při tvorbě barevných přechodů. Tvorba barevných přechodů v sekci 3.1.

   Ukázka barevné konfigurace pro východ Slunce:

   ```
   1.0,0.5,0.5 0.2,0.2,0.55 0.1,0.1,0.0 0.2,0.2,0.2 1.0,1.0,1.0
   1.0,1.0,0.4 0.0,0.0,0.1 1.0,1.0,0.85 0.15,0.15,0.42
   ```

   Barvy jsou uvedeny ve stejném pořadí, v jakém jsou zmíněny ve výčtu základních barev definovaném v předchozím odstavci.

3. Ve funkci `getSkyConfInTOD` jsou z konfiguračního souboru načteny barvy v kontrolních bodech pro danou denní dobu `TOD`. Z konfigurace jsou načteny pouze barvy určené parametrem $varCl$. Poté jsou postupně předány všechny tři barevné kanály vektorů těchto barev na vstup funkce:

   ```
   getVectorValue(float time_dy,float sunShift, float vec1Val, float vec2Val);
   ```

4. Ve funkci `getVectorValue` je vypočtena aktuální hodnota barevného kanálu na zá-

kladě předaných parametrů. Zjistí se rozdíl hodnot mezi barevnými kanály hraničních barev `value_dy` a následně poměr mezi tímto rozdílem a posunem v čase `dy`.

Kód 5.1: Výpočet aktuální hodnoty barevného kanálu

```
value_dy = abs(vec1Val - vec2Val);
dy = time_dy/value_dy;

if(vec1Val < vec2Val)
  return vec1Val + sunShift/dy;
else
  return vec1Val - sunShift/dy;
```

5. Výsledná barva složená z barevných kanálu, jež byly vypočtěny podle uvedeného postupu, představuje aktuální barevnou konfiguraci pro daný simulační čas.

5.4 GLSL

Jak bylo řečeno v úvodní kapitole, novější verze OpenGL dovolují programovat kreslicí pipeline, což bylo umožněno zejména pokrokem ve vývoji grafických karet. Programování na úrovni OpenGL pipeline je prováděno pomocí shaderů. Shader je nezávisle překládaná jednotka napsaná pomocí jazyka GLSL. Shader program je soubor shaderů, jež jsou přeloženy a spojeny dohromady [5]. Rozlišuje se několik druhů shaderů. V aplikaci skyboxu bylo využito *vertex* a fragment (neboli pixel) shaderů. Společně tvoří jednu nebo více programovatelných úrovní kreslicí pipeline.

Jazyk GLSL má základ v syntaxi jazyka C a byl vyvinut konsorciem ARB za účelem jednoduššího programování na úrovni GPU. Ke spuštění aplikace skyboxu je zapotřebí, aby byla podporována minimálně verze GLSL 3.3.0. Jakou verzi podporuje grafická karta na cílové platformě, je možné zjistit příkazem `glxinfo | grep string`.

Převážná část výpočtů vedoucích k vytvoření skyboxu je prováděna ve fragment shaderu. Ve vertex shaderu se provede transformace scény pomocí transformační matice, jež je vypočtena na procesoru a předána do shaderu jako parametr. Také jsou zde určeny některé vektory potřebné k výpočtu osvětlovacího modelu.

Ve fragment shaderu jsou počítány funkce šumu, Blinn-Phongův osvětlovací model, různé filtry a další. Postupně se zde získají barvy daného bodu pro jednotlivé prvky skyboxu a jejich smícháním se určí výsledná barva, která je poté vykreslena do *framebufferu*. Míchání barev se provádí pomocí vestavěné funkce *mix*.

```
T = mix(T x,T y,T a);
```

Funkce provede lineární smíchání hodnot `x` a `y` na základě hodnoty `a`.

Kapitola 6

Závěr

V této práci jsem se zabýval metodami tvorby skyboxu. Jak je zmíněno v úvodu, při vytváření skyboxu lze využít dvou přístupů. Vzhledem k tomu, že zadáním bylo vytvořit animovaný skybox, věnuji se v práci popisu procedurálního přístupu. Zvolil jsem scénu zachycující širý oceán, z toho důvodu je část práce zaměřená na vytváření vodní hladiny pomocí metody bump mappingu, i když se nejedná o součást zadání. Hlavní část práce na vývoji aplikace měla zahrnovat implementaci funkcí šumu a jejich použití při generování animovaných mraků. Mým původním plánem bylo naimplementovat jak funkci Perlinova šumu, tak její upravenou verzi Simplex noise a poté demonstrovat rozdíly v jejich použití při vykreslování. Z časových důvodu jsem implementoval pouze funkci Perlinova šumu, neboli její variaci někdy označovanou jako Pink noise. I přes problém s aliasingem se Perlinův šum ukázal jako vhodná volba při generování grafického obsahu, ať už mraků, nebo hladiny oceánu. Jako nepoužitelná se ukázala varianta vykreslování hladiny oceánu přímo v shaderu. Použitím textury nedosáhneme takové úrovně detailů. Zároveň na hladině vznikají drobné artefakty, protože na sebe textura nenavazuje, je však možné aplikaci rozšířit o navazující texturu. Navíc díky použití 3D textury je oceán animovaný.

Při řešení projektu jsem objevil několik metod a technik jež bych rád vyzkoušel, ať už přímo v aplikaci skyboxu, nebo v jiných projektech. Jednou z těchto technik jsou částicové systémy. Částicové systémy se využívají například k tvorbě 3D oblak v leteckých simulátorech. V aplikaci jsem se pokusil pomocí částic vykreslit letící kometu na noční obloze. I když jsem její implementaci nedokončil, pokročil jsem poměrně daleko při práci s krelicí pipeline. Hlavně s procesem *transform feedback*, pomocí něhož je dosaženo generování nových částic. A také jsem si alespoň částečně vyzkoušel práci s *geometry shaderem*.

Při vytváření oceánu jsem narazil na algoritmus Rychlé Furierovy transformace (*Fast Fourier transform*, zráceně FFT). Pomocí FFT je možné vykreslovat vlny na oceánu v reálném čase. Dosažené výsledky jsou velmi realistické. Algoritmem FFT by bylo možné v aplikaci nahradit použitou metodu bump mappingu. Protože se Furierova transformace vypočítá na procesoru před samotným vykreslením scény, neměl by nastat problém ani s poklesem FPS.

Při pokusech s aplikací jsem si uvědomil, že důležitým kritériem při výtváření jakékoliv grafické aplikace je rovnováha mezi mírou realističnosti a rychlostí vykreslování.

Literatura

[1] Bevins, J.: What is coherent noise?
`http://libnoise.sourceforge.net/coherentnoise/index.html`, 2003 [cit. 2013-05-10].

[2] Blinn, J. F.: Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.*, ročník 12, č. 3, Srpen 1978: s. 286–292, ISSN 0097-8930, doi:10.1145/965139.507101.
URL `<http://doi.acm.org/10.1145/965139.507101>`

[3] Elias, H.: Cloud Cover.
`http://freespace.virgin.net/hugo.elias/models/m_clouds.htm`, 2000 [cit. 2013-05-10].

[4] Elias, H.: Perlin Noise.
`http://freespace.virgin.net/hugo.elias/models/m_perlin.htm`, 2000 [cit. 2013-05-10].

[5] Kessenich, J.; Baldwin, D.; Rost, R.: *The OpenGL Shading Language*. The Khronos Group Inc., 2012.

[6] Kulhanek, P.: Základní vlastnosti hvězd. Aldebaran Group for Astrophysics, [cit. 2013-05-10].

[7] Mandelbrot, B. B.: *The Fractal Geometry of Nature*. W. H. Freeman and Co., 1982, ISBN 0-7167-1186-9.

[8] Perlin, K.: An image synthesizer. *SIGGRAPH Comput. Graph.*, ročník 19, č. 3, Červenec 1985: s. 287–296, ISSN 0097-8930, doi:10.1145/325165.325247.
URL `<http://doi.acm.org/10.1145/325165.325247>`

[9] Philip Gibbs: Why is the sky blue? In *The Physics and Relativity FAQ*, 1997 [cit. 2013-05-10].

[10] Rhoades, J.; Turk, G.; Bell, A.; aj.: Real-time procedural textures. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, I3D '92, New York, NY, USA: ACM, 1992, ISBN 0-89791-467-8, s. 95–100, doi:10.1145/147156.147171.
URL `<http://doi.acm.org/10.1145/147156.147171>`

[11] Roden, T.; Parberry, I.: Clouds and stars: efficient real-time procedural sky rendering using 3D hardware. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '05, New York, NY, USA: ACM, 2005, ISBN 1-59593-110-4, s. 434–437, doi:10.1145/1178477.1178574.
URL `<http://doi.acm.org/10.1145/1178477.1178574>`

[12] Sepcot, M. J.: Perlin Noise for 2-Dimensional Clouds.
`http://www.sepcot.com/blog/2006/08/PDN-PerlinNoise2d`, 2006 [cit. 2013-05-10].

[13] Silicon Graphics, Inc.: *glTexParameter - OpenGL 4 Reference Pages*. [cit. 2013-05-10].

[14] Tomáš Milet: *Grafické intro 64kB s použitím OpenGL*. Diplomová práce, FIT VUT v Brně, 2012.

[15] W3C Recommendation: *Gradients and Patterns*. 2011 [cit. 2013-05-10].

[16] Z. Mikulka: *Generátory náhodných čísel*. Bakalářské práce, Fakulta elektrotechniky a komunikačních technologií VUT v Brně, 2008.

Příloha A

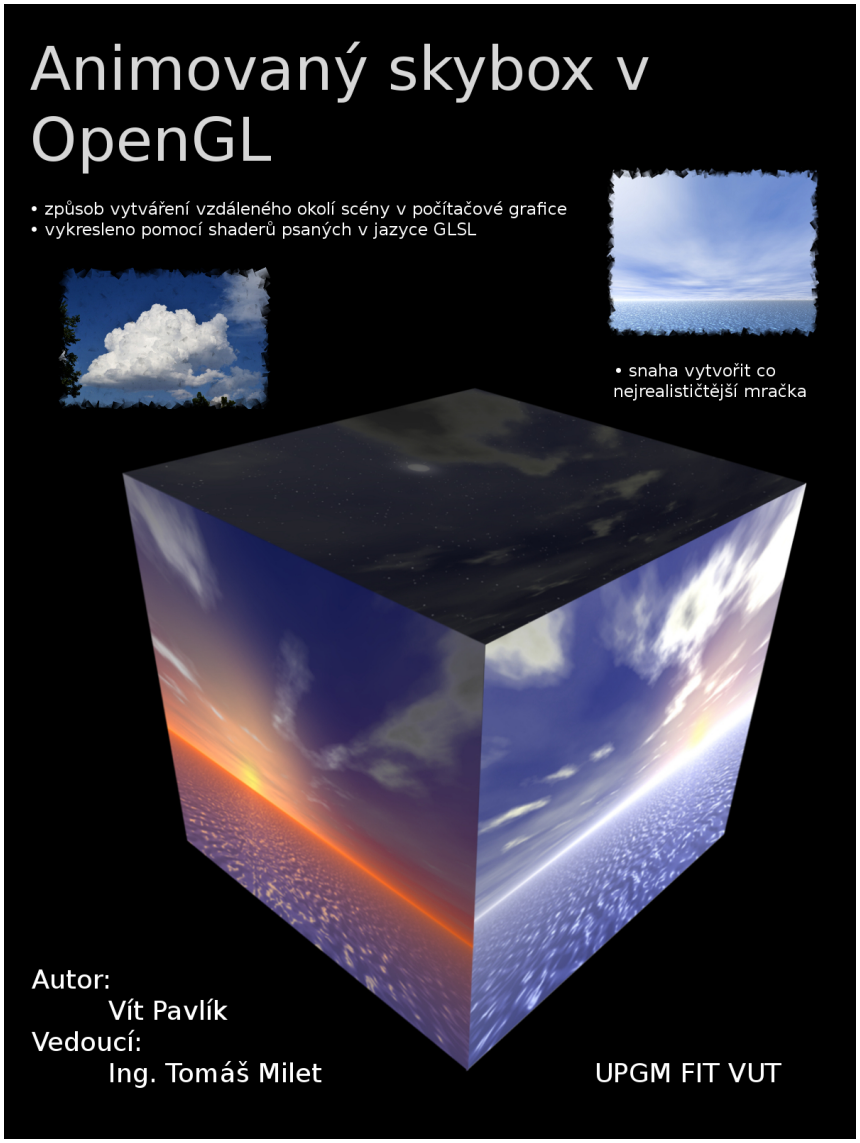
Obsah CD

Přiložené CD obsahuje:

- `app-src/` – Zdrojové soubory aplikace, manuál k aplikaci v soubor README.txt a Makefile.
- `doc-src/` – Zdrojové soubory technické zprávy.
- `doc/` – Technická zpráva ve formátu PDF.
- `demo/` – Demonstrační video a plakát.

Příloha B

Plakat



Obrázek B.1: Plakát reprezentující vytvořenou aplikaci skyboxu.