VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

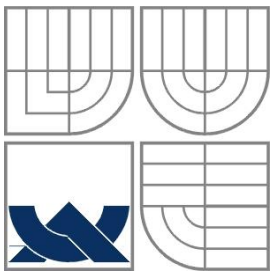
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ANIMOVANÝ PODVODNÍ SKYBOX V OPENGL

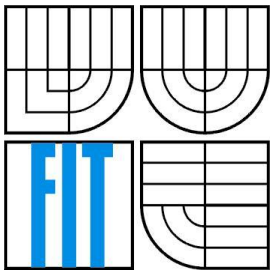
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LENKA LAGOVÁ
AUTHOR

BRNO 2013



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 PODVODNÍ SKYBOX V OPENGL

ANIMATED UNDERWATER SKYBOX USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LENKA LAGOVÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MILET
SUPERVISOR

BRNO 2013

Abstrakt

Bakalářská práce se zabývá problematikou řešení animovaného podvodního skyboxu pomocí OpenGL. Práce obsahuje úvod k OpenGL, popisuje grafické techniky používané pro vykreslování a metody pro procedurální generování objektů a textur.

Abstract

This bachelor thesis deals with the solution of animated underwater skybox using OpenGL. The work contains an introduction to OpenGL, describes graphic techniques which are used for rendering and describes methods for procedural generating of objects and textures.

Klíčová slova

skybox, OpenGL, shader, textura, procedurální generování, Perlinův šum, Voroného diagram, částicový systém, Eulerova numerická metoda, afektor, deflektor, Phongův osvětlovací model, odložené stínování, framebuffer object, kubická mapa

Keywords

skybox, OpenGL, shader, texture, procedural generating, Perlin noise, Voronoi diagram, particle system, Euler numerical method, affector, deflector, Phong lighting model, deferred shading, framebuffer object, cube map

Citace

Lagová Lenka: Animovaný podvodní skybox v OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2013

Animovaný podvodní skybox v OpenGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana inženýra Tomáše Mileta. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Lenka Lagová
14. května 2013

Poděkování

Ráda bych tímto poděkovala Ing. Tomášovi Miletovi za jeho rady a trpělivost v průběhu práce. Také bych ráda poděkovala mé rodině a přátelům za podporu během tvorby práce a za zajímavé nápady.

Obsah

1 Úvod

V dnešní době se jistě každý setkal s nějakou počítačovou hrou, která měla vyobrazenou krajinu či oblohu. Jednou z technik vykreslování této scény je skybox. Tato bakalářská práce se zabývá problematikou vytvoření animovaného skyboxu v OpenGL.

Skybox je umístěn na mořském písečném dně, na kterém se nachází jak kameny, tak mořské řasy a generátory bublinek. Také lze pozorovat mnohé světelné efekty, jako například sluneční paprsky či kaustiky. V několika následujících kapitolách budou blíže popsány principy řešení problematiky této práce. Kapitoly jsou strukturovány tak, aby na sebe logicky navazovaly.

V úvodní kapitole je popsáno, co je to skybox a základní poznatky o OpenGL.

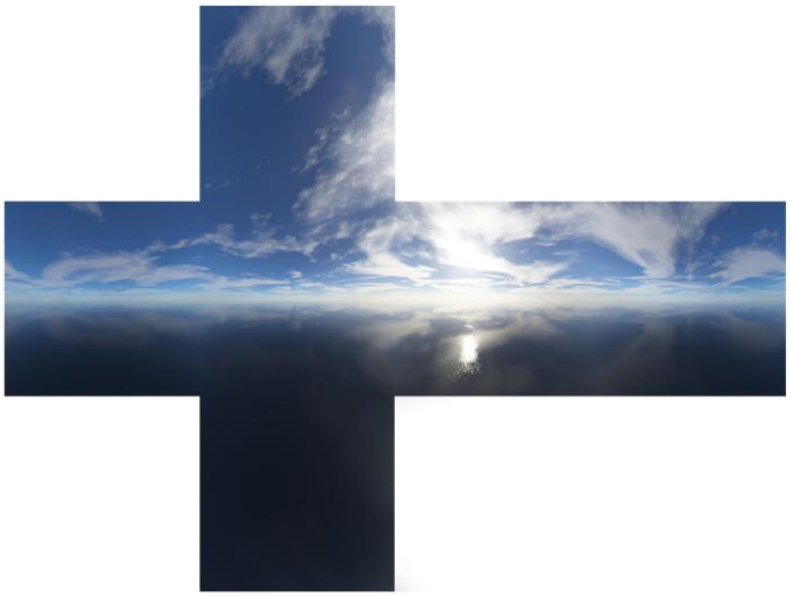
Druhá kapitola obsahuje bližší popis důležitých součástí OpenGL včetně popisu GLSL, shaderů a v neposlední řadě krátký úvod k texturám. Dále se věnuje bližšímu popisu technik používaných při tvorbě procedurálních grafických aplikací a textur. Například je zde popsán Perlinův šum, Voroného diagram, princip částicových systémů, technika deferred shading a Phongův osvětlovací model.

Předposlední kapitola je věnována implementaci jednotlivých částí bakalářské práce. Jednak je popsána tvorba grafických objektů, implementace použitých grafických technik a také popis a použití konfiguračního souboru.

V závěru jsou popsány poznatky získané během tvorby bakalářské práce, jsou zhodnoceny výsledky a diskutovány další možná rozšíření projektu.

1.1 Skybox

Skybox je metoda vytváření scény, která neobsahuje geometrii objektů. Je tvořena pouze sadou 2D obrázků. Scéna bývá uzavřena v krychli, na jejíž strany se obrázky nanášejí tak, aby na sebe navazovaly a tím vytvářejí iluzi okolí. Použití této metody je především v počítačových hrách, kde se vykresluje krajina, hory, obloha aj. Příklad skyboxu vyobrazujícího vodní hladinu a oblohu je na obrázku 1.1.



Obrázek 1.1: *Ukázka šesti navazujících textur tvořících skybox, převzato z [1]*

1.2 OpenGL

Knihovna OpenGL (Open Graphics Library), jak je uvedeno v literatuře [2], je aplikační programové rozhraní k akcelerovaným grafickým kartám. Byla navržena tím způsobem, aby byla využitelná na různých typech grafických karet, ale také pomocí softwarové simulace. Tato knihovna je nezávislá na použitém operačním systému, proto neobsahuje funkce pro práci s okny a jiné, systémově závislé funkce.

Z programátorského hlediska se OpenGL chová jako stavový automat. Příkazy pro vykreslování lze průběžně měnit vlastnosti vykreslovaných primitiv, jako je barva a průhlednost, nebo vlastnosti scény. Tyto vlastnosti zůstávají neměnné do doby, kdy se změní dalším příkazem. Mnoho stavových proměnných odkazuje na módy, které jsou povoleny příkazy `glEnable()` nebo zakázány pomocí `glDisable()`.

Vykreslování scény se provádí procedurálně - voláním funkcí OpenGL se vykreslí výsledný rastrový obrázek uložený ve framebufferu. Ve framebufferu je každému pixelu přiřazena barva, alfa složka, hloubka a další atributy.

Pomocí funkcí poskytovaných knihovnou OpenGL lze vykreslovat obrazce či tělesa složená ze základních geometrických primitiv - bod, úsečka, trojúhelník a další. Na vrcholy tvořící jednotlivá primitiva lze aplikovat různé transformace, jak lineární (zachovávají lineární kombinace), tak afinní (zachovávají kolinearitu a dělící poměr).

2 Teorie

V této kapitole jsou popsány teoretické části bakalářské práce potřebné pro důkladnější seznámení s problematikou. Jsou zde popsány důležité části programování v OpenGL, vysvětlení pojmu procedurálního generování a metody používané pro procedurální generování. Dále se zde popisují částicové systémy, kde je vysvětleno několik principů používaných pří práci s částicovými systémy. V další části jsou popsány grafické techniky, které se při tvorbě grafických aplikací využívají. Mezi tyto metody patří osvětlovaní model a také metoda odloženého stínování.

2.1 Důležité části OpenGL

V úvodní kapitole je popsáno, k čemu se knihovna OpenGL používá a základní informace o ní. V této podkapitole budou vysvětleny důležité součásti této knihovny. Bude popsán jazyk GLSL, principy shaderů, textury a další.

2.1.1 OpenGL Shading Language

OpenGL Shading Language, zkráceně GLSL, je základní a nedílnou součástí OpenGL API. Je to programovací jazyk založený na syntaxi jazyka C. Tento jazyk se používá pro vytváření programů pro programovatelný grafický řetězec. Tyto programy se nazývají shadery.

Jazyk GLSL podporuje oproti jazyku C několik datových typů vhodných k práci s vrcholy, barvami, texturami apod. Prvním datovým typem je vektor. Mohou být jedno-složkové (`vec1`) až čtyř-složkové (`vec4`). Dále podporuje datový typ matice (`mat2` až `mat4`). Dalším datovým typem je `sampler`, což je handler umožňující přístup k texturám.

Pro jednoduchý přístup k jednotlivým složkám vektoru je zaveden operátor swizzle. Pomocí tohoto operátoru je totiž možné přistupovat k jednotlivým složkám vektoru, ale i k jejich různým kombinacím. Například pokud chceme získat složku `z` a `x` (v tomto pořadí) ze čtyř-složkového vektoru `vec4(x,y,z,w)`, můžeme napsat `vectorA = vectorB.zx`.

GLSL navíc obsahuje velké množství vestavěných funkcí, ať už jde o funkce matematické, tak funkce pro práci s texturami, pro míchání barev aj.

2.1.2 Shadery

Shader je program, který řídí jednotlivé části programovatelného grafického řetězce grafické karty (rendering pipeline). U programovatelného řetězce je možné řídit zpracování vrcholů a fragmentů. Jsou navrženy tak, aby byly vykonávány přímo na GPU a často v paralelním zpracování. Počet procesoru grafické karty udává, kolik může být najednou prováděno. Proto jsou shadery velice efektivní.

Jak je popsáno v literatuře [2], dělí se na několik základních typů podle toho, pro kterou jednotku grafického řetězce jsou určeny.

2.1.2.1 Vertex shader

První fází rendering pipeline je zpracování vrcholů. Všechny vrcholy jsou zpracovávány ve vertex shaderu. Data se získávají z vertex array object (VAO). VAO ukládá reference na vertex buffer object (VBO) a nastavení atributů.

Mezi nejčastější operace patří transformace vrcholu - násobení modelovou, pohledovou či projekční maticí.

2.1.2.2 Geometry shader

Po zpracování vrcholu ve vertex shaderu může následovat geometry shader. Ten slouží především k přidávání a odebírání vrcholů, čímž se ovlivňuje výsledná geometrie. Vždy se specifikuje formát vstupních i výstupních dat, včetně počtu vrcholů na výstupu.

2.1.2.3 Fragment shader

Fragment shader je fáze nastávající po rasterizaci primitiva. Pro každý vzorek pixelů vztahujících se k primitivu je vygenerován fragment. Každý fragment obsahuje pozici v okně aplikace a několik dalších hodnot. Výstupem fragment shaderu je hloubka a barva.

2.1.3 Textura

Textura je popis vlastností povrchu důležitá pro vnímání struktury, barvy a kvality objektu. Textura je vzorek, který může být pravidelný i nepravidelný. Může být použita jako zdroj textury v shaderu, nebo jako render target (scéna se nevykreslí na obrazovku, ale do textury). Textura je definována jako pole pixelů příslušné dimenze se specifikovanou velikostí a formátem. Po vygenerování a navázání textury se specifikují parametry. Základní jednotkou textury je texel.

2.1.3.1 Cubemapa

Jak je popsáno v [3], cubemapa, neboli kubická mapa, je speciální technika, která používá sadu šesti dvojrozměrných textur a z nich vytvoří texturovou krychli se středem v počátku. Pro každý fragment jsou souřadnice textury brány jako směrový vektor a každý texel reprezentuje to, co je na texturové krychli vidět z počátku. Kubické mapy jsou ideální pro efekty s okolím, odrazy a osvětlením. Také mohou obalovat kulatý objekt texturami a rozdělovat texely relativně rovnoměrně na jeho stěny.

2.1.3.2 Rastrová textura

Zde je předem připraven rastrový obrázek. U tohoto typu textur je důležité dostatečné rozlišení a detail obrázku, jelikož při aplikování nedostatečně velké textury na příliš velký objekt vede

ke zhoršení kvality obrázku. Paměťové nároky rastrové textury jsou mnohem větší, než u dalšího typu textur - procedurálních.

2.1.3.3 Procedurální textura

Textura vytvořená pomocí matematických funkcí a algoritmů se nazývá procedurální. Využívá se především pro vytvoření přírodních elementů, jako je dřevo, mramor, kámen a další. Výhodou takovéto textury je jak realistický vzhled objektu, tak nízké paměťové nároky. Také je možné vytvářet textury různých rozměrů, multidimenzionální textury a textury s různou průhledností.

2.1.4 Vertex array object

Vertex array object (VAO) je objekt, který obsahuje veškeré údaje potřebné pro specifikaci vertex dat. Sám o sobě neobsahuje pole s vrcholy, pouze drží odkaz na již existující buffer s těmito daty. Je nutné specifikovat, jaký atribut se ve VAO nachází, na jaké pozici, jakou má velikost a offset oproti ostatním atributům. Je totiž možné použít více údajů za sebou, jako například vrcholy primitiv, texturovací koordináty, normály, barvy a další.

2.1.5 Framebuffer object

Podle [2], framebuffer object (FBO) je OpenGL objekt, který umožňuje renderovat uživatelské informace (normály, pozice, hloubka, barva) do textur. Skládá se z několika 2D polí, jako hloubkový buffer, stencil buffer a několik barevných bufferů. Také se často nazývá G-buffer.

Před vytvořením FBO je třeba vytvořit textury, do kterých se bude renderovat. Těchto textur může být několik (multiple render targets). Po vytvoření textur a nastavení parametrů je možné přejít k přípravě FBO.

Po získání jména a aktivace FBO se nastaví připojení vytvořených textur k attachmentům. Mezi nejčastěji používané u FBO patří `GL_COLOR_ATTACHMENTi`. Výstupní hodnota fragment shaderu (specifikovaná layoutem) je zapsána do i-tého color attachmentu aktuálního framebuferru.

Dalším krokem je nastavení seznamu attachmentů. Tím definujeme, do kterých barevných bufferů se bude kreslit. Tímto je FBO připraven k renderování.

2.1.6 Renderbuffer object

Renderbuffer object (RBO) obsahuje obrazy. Jsou vytvářeny především zároveň s FBO a optimalizovány pro použití jako render target. Využití nalézá například jako hloubkový buffer.

Po vygenerování jména RBO a jeho aktivace se nevytváří žádné speciální textury. Pouze se přidělí úložiště pro renderbuffer. Vytvořený renderbuffer object se připojí k dříve vytvořenému FBO.

2.2 Procedurální generování

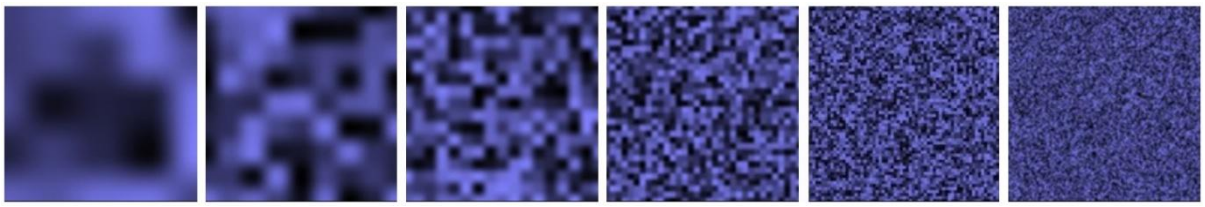
Generování geometrie a textur, jak je popsáno v [4], se v této práci provádí procedurálně. Výhodou takového generování je, že je možné změnou parametrů objektu dosáhnout jiného vzhledu, jak u geometrie, tak textur.

2.2.1 Perlin noise

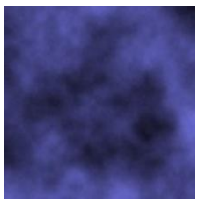
Perlin noise, neboli Perlinův šum [5], je algoritmus vycházející z tvorby Kena Perlina. Řadí se mezi procedurální techniky. Je využíván pro tvorbu různých efektů, jako jsou oblaka a plameny, ale také pro generování textur či zemského povrchu. Využitelnost této metody je především v tom, že se do přesných modelů vnesou prvky náhody, čímž získáme scénu přiblíženou realitě.

Podstatou Perlinova šumu je generování spojitého šumu, který je vypočítáván v diskrétní mřížce. Může být definován v libovolné dimenzi.

Výsledek šumu je závislý na použitých parametrech funkce, konkrétně frekvence, amplitudy, perzistence a počtu oktáv. Princip této metody je takový, že dle zadaných parametrů vytvoří adekvátní počet vrstev, jenž je dán počtem oktáv, a tyto vrstvy se v závěru spojí v jednu výslednou vrstvu.



Obrázek 2.1: *Několik funkcí Perlinova šumu ve 2D (převzato z [6])*



Obrázek 2.2: *Výsledná funkce Perlinova šumu (převzato z [6])*

Perlinův šum počítá hodnotu každého pixelu pro danou texturu (či terén) zvlášť, je tedy poměrně hodně výpočetně náročný. Složitost tohoto algoritmu je $O(2^n)$. S narůstajícím počtem dimenzí (oktáv) narůstá složitost algoritmu. Z obrázku 2.3 je patrné, že od jistého počtu oktáv se výsledná funkce nemění. Zvolením menšího počtu oktáv je tedy možno dosáhnout výsledné textury rychleji bez výrazné změny vzhledu.

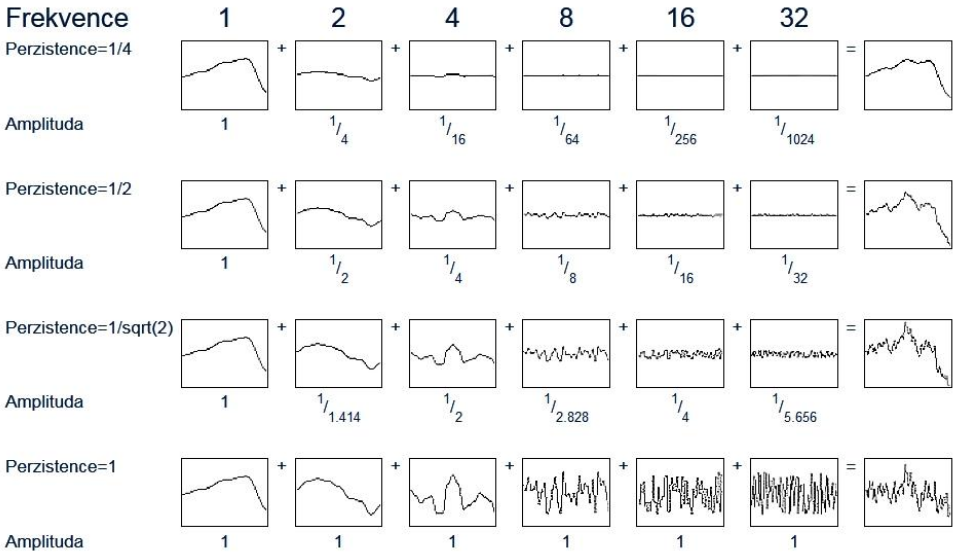
Počet oktáv udává, kolikrát se má algoritmus provést, čili kolik bude vygenerováno vrstev, než se spojí do výsledné vrstvy.

Persistence je pojem zavedený pro modifikaci frekvence i amplitudy pro každou oktávu. Je velice časté, že frekvence se každou oktávou mění dvojnásobně. Použije-li se však persistence, získáme rozmanitější průběh funkce (viz obrázek 2.3). Vztah pro výpočet frekvence je tedy dán vzorcem

$$f_i = f_{i-1} \cdot \frac{1}{persistance} \qquad 1$$

a amplituda se počítá vztahem

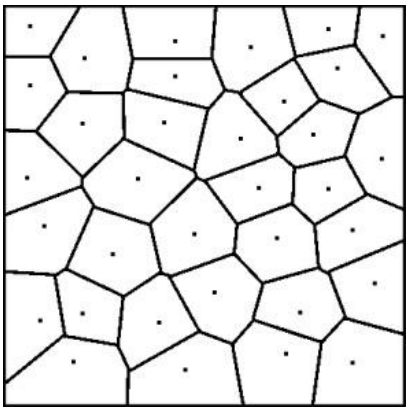
$$a_i = a_{i-1} \cdot persistance \qquad 2$$



Obrázek 2.3: *Diagramy zobrazující vliv persistence na průběh funkce (převzato z [6])*

2.2.2 Voroného diagram

Voroného diagram, popsaný v [7], je rozdělení roviny s několika body do konvexních polygonů tak, že každý polygon obsahuje jeden bod, tzv. generátor. Každý bod v tomto polygonu je blíže ke generátoru v polygonu, než k ostatním generátorům. Strany polygonů tvoří body stejně vzdálené od dvou generátorů, vrcholy tvoří body stejně vzdálené od tří generátorů.



Obrázek 2.4: *Ukázka Voroného diagramu. Body v diagramu jsou generátory, hrany v diagramu vymezují stejnou vzdálenost ke dvěma a více generátorům.*

Množina bodů tvořící hranici mezi dvěma generátory v rovině $p_i = (x_i, y_i)$ a $p_j = (x_j, y_j)$ mají stejnou vzdálenost definovanou euklidovskou metrikou:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad 3$$

Aby bylo možné použít texturu tvořenou Voroného diagramem použít několikrát vedle sebe, je nutné, aby na sebe kraje navazovaly. V tomto případě je řešení v úpravě výpočtu vzdálenosti. Musí se do výpočtu zahrnout možnost, že vzdálenost bodu od generátoru je větší, než polovina rozměru pole. Pak se vzdálenost počítá vztahem:

$$d(p_i, p_j) = \sqrt{min(|x_i - x_j|, whd - |x_i - x_j|)^2 + min(|x_i - x_j|, whd - |y_i - y_j|)^2} \qquad 4$$

Proměnná whd reprezentuje rozměry pole. Funkce min vrací menší hodnotu ze dvou možných hodnot. První hodnota je vzdálenost dvou bodů, která je menší, než polovina rozměru pole. V případě, že je tato vzdálenost větší, než polovina rozměru, použije se pro výpočet bod ze sousedního pole.

2.3 Částicový systém

Jak je v [8] uvedeno, k modelování objektů, jejichž tvar je příliš členitý, nebo se mění takovým způsobem, že ho nelze reprezentovat jako povrch, se používá technika zvaná částicový systém (anglicky particle system). Mezi takové objekty patří padající sníh, tráva, oheň, mlha, bublinky.

Systém částic je reprezentován jako soubor velmi malých prvků, jejichž vlastnosti se mění v čase. Mezi vlastnostmi může být pozice, barva, průhlednost, rychlost, směr pohybu a další. Po určité době života mohou mizet nebo se znovu objevovat.

Na základě parametrů každé existující částice se aktualizuje její poloha a ostatní atributy. V celém systému se vytvoří nové částice. Ty vznikají v nějaké konkrétní oblasti, nebo mohou vznikat jako potomci jiných částic. Každé nové částici se přiřadí vlastnosti. Částice, které překročily svou dobu života, zaniknou.

2.3.1 Affectors

Affectors (česky afektory) jsou síly, které ovlivňují částice. Mezi afektory se řadí například gravitace a vztlak.

2.3.1.1 Tíhová síla

Je to výslednice gravitační síly Země a odstředivé síly vzniklé otáčením Země kolem své osy, síla působící na tělesa na povrchu země. Tato síla se vypočítá vztahem

$$F_G = mg \quad (5)$$

kde m je hmotnost tělesa a g je tíhové zrychlení.

Tíhové zrychlení udává rychlost, kterou nabude těleso na povrchu Země za jednu sekundu volného pádu. Zahrnuje gravitační a odstředivé zrychlení. Hodnota místního tíhového zrychlení závisí na geografické šířce, nadmořské výšce a dalších aspektech. Například v naší zeměpisné šířce je $g = 9{,}81\ \frac{m}{s^2}$.

2.3.1.2 Vztlaková síla

Vztlak je síla, která nadlehčuje těleso v kapalině. Vzniká rozdílem hydrostatických tlaků na spodní a horní části tělesa, protože tlak na spodní část tělesa je větší. Pro těleso obecného tvaru platí vztah

$$F_{VZ} = V\rho g \quad (6)$$

kde V je objem ponořené části tělesa, ρ je hustota kapaliny a g je tíhové zrychlení.

2.3.2 Deflectors

Deflectors (česky deflektory) jsou prostorové deformátory, které mají za úkol odrážet částice nebo vychylovat tok částic. Příkladem deflektoru může být neviditelný objekt ve scéně, případně geometrie scény.

2.3.3 Diferenciální rovnice

Každá částice má svoji rychlost a pozici. Tyto vlastnosti lze popsat diferenciálními rovnicemi. Pro řešení diferenciálních rovnic se může použít několika metod, ať už jednokrokových,

které pro výpočet nového stavu využívají pouze předchozího stavu, nebo vícekrokových, které nový stav počítají z více předchozích stavů. Diferenciální rovnice pro výpočet pozice má tvar:

$$\frac{\partial p}{\partial t} = \vec{v} \quad (7)$$

Po zintegrování této rovnice se získá tvar:

$$p(t) = p(0) + \int_0^t \vec{v}(t) \cdot dt \quad (8)$$

$p(t)$ je pozice v čase t, $p(0)$ je počáteční pozice, $\vec{v}(t)$ je počáteční rychlost a dt je krok času. Pro výpočet rychlosti má diferenciální rovnice tvar:

$$\frac{\partial v}{\partial t} = \vec{a} \quad (9)$$

Rovnice se opět zintegruje a získá se tvar:

$$\vec{v}(t) = \vec{v}(0) + \int_0^t \vec{a}(t) \cdot dt \quad (10)$$

$\vec{v}(t)$ reprezentuje rychlost v čase t, $\vec{v}(0)$ počáteční rychlost a $\vec{a}(t)$ je zrychlení v čase t. S výslednými rovnicemi se nadále pracuje pomocí některé z metod řešení diferenciálních rovnic. Příkladné metody jsou uvedeny v následujících podkapitolách.

2.3.3.1 Eulerova metoda

Tato metoda je způsob numerického řešení diferenciálních rovnic s danými počátečními podmínkami. Řadí se mezi jednokrokové metody vycházející z rovnic pro změnu polohy a rychlosti určitého objektu. Rovnice pro výpočet nové rychlosti má následující tvar:

$$\vec{v}_{n+1} = \vec{v}_n + dt \cdot \vec{a} \quad (11)$$

Proměnná $\vec{v}_{n+1}$ značí novou rychlost, dt je krok času a $\vec{a}$ je zrychlení, které se odvíjí od druhého Newtonova zákona:

$$\vec{F} = m \cdot \vec{a} \quad (12)$$

Pro výpočet nové pozice je zapotřebí rychlost a krok času. Diferenciální rovnice má tvar:

$$p_{n+1} = p_n + dt \cdot \vec{v}_n \quad (13)$$

Proměnná p_{n+1} je nová pozice, dt je krok času a $\vec{v}$ je rychlost.

Aby bylo možné výpočet začít, je nutné určení počátečních podmínek. Do těchto podmínek spadá rychlost a pozice. Pro obě diferenciální rovnice musí být také zvolen krok času dt.

$$\vec{v}_0 = \vec{v}(0)$$

$$p_0 = p(0)$$

2.3.3.2 Metoda Runge Kutta

Rongovy-Kuttovy metody jsou jedna z nejdůležitějších skupin jednokrokových metod. Obecný tvar této metody je

$$y_{n+1} = y_n + dt \cdot (w_1 k_1 + \cdots + w_s k_s), \quad \textbf{14}$$

kde

$$k_1 = f(x_n, y_n) \quad \textbf{15}$$

$$k_i = f\left(x_n + \alpha_i \cdot dt, y_n + dt \cdot \sum_{j=1}^{i-1} \beta_{ij} \cdot k_j\right), i = 2, \ldots, s \quad \textbf{16}$$

a w_i, α_i a β_{ij} jsou konstanty volené tak, aby metoda měla maximální řád. Nejproslulejší je metoda Runge Kutta 4. řádu. Je přesnější, než metoda Eulerova, avšak výpočetně mnohem náročnější.

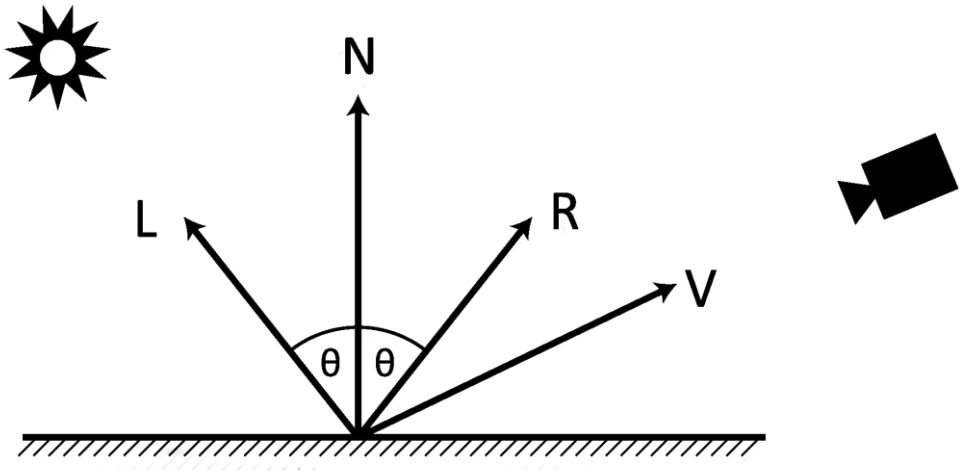
2.4 Grafické techniky

V této podkapitole budou popsány techniky, které se při tvorbě grafický aplikací využívají jak pro optimalizaci programu, tak pro realističtější vzhled. Bude zde popsán osvětlovací model a metoda odloženého stínování.

2.4.1 Phongův osvětlovací model

Bui Tuong Phong navrhl v roce 1973 ve své disertační práci osvětlovací model pro výpočet odraženého světla, který je popsán v [8] a [9] . Není založen na fyzikálních zákonech (je empirický), takže není zcela přesný. Přesto se používá především v real-time grafice, jelikož výsledky jsou velmi zdařilé a výpočet není příliš náročný.

Odraz na povrchu materiálu je určen směrem dopadajícího světla $\vec{L}$, směrem $\vec{V}$ od pozorovatele k bodu bodem na povrchu, normálovým vektorem v místě dopadu $\vec{N}$ a zrcadlově odraženým paprskem $\vec{R}$.



Obrázek 2.5: *Označení vektorů použitých v modelu. L je vektor reprezentující směr světla, V je směr k pozorovateli, R je odražený paprsek a N je normálový vektor v místě dopad (převzato z [8]).*

Phongův osvětlovací model rozlišuje tři druhy odrazu světla od materiálu. Z těchto částí se pak skládá výsledný odraz. Odraz je rozdělen na spekulární, difúzní a ambientní složku. Celková intenzita odrazu je dána vzorcem

$$C = C_a + C_d + C_s \quad (17)$$

2.4.1.1 Ambientní světlo

Ambientní osvětlení je světlo, které bylo rozptýlené prostředím natolik, že nelze určit jeho směr. Zdá se, že přichází ze všech směrů. Jakmile ambientní světlo narazí na povrch, je rovnoměrně rozptýleno všemi směry. Ambientní složka bývá konstantní pro celou scénu. Odraz je určen vztahem

$$C_a = I_a k_a \quad (18)$$

I_a je intenzita okolního osvětlení scény a k_a je odrazivý koeficient (určuje schopnost povrchu odrážet světlo).

2.4.1.2 Difúzní světlo

Difúzní složka je světlo, které přichází na povrch z jednoho směru, z jednoho konkrétního zdroje světla. Udává intenzitu světla, které se od povrchu tělesa odráží rovnoměrně do všech směrů. Její použití vytváří trojrozměrný vzhled ve scéně. Množství odraženého světla závisí na směru dopadu světla L. Tuto složku vypočítáme podle vztahu

$$C_d = I_d k_d (\vec{N} \cdot \vec{L}) \quad (19)$$

Pokud je skalární součin $\vec{N} \cdot \vec{L}$ menší nebo roven nule, povrch je odvrácen od zdroje světla.

2.4.1.3 Spekulární světlo

Spekulární (lesklá) složka světla taktéž přichází z jednoho směru. Udává intenzitu světla odraženého převážně v jednom směru. Výpočet se provádí podle vztahu

$$C_s = I_s k_s \left(\vec{V} \cdot \vec{R}\right)^n \quad \textbf{20}$$

kde n je Phongův exponent určující míru lesklosti.

2.4.2 Deferred shading

Běžný způsob, jak se osvětlování aplikuje na scénu, je provézt výpočet osvětlení při rasterizaci jediným průchodem shaderu. Při metodě deferred shading (v překladu odložené stínování) se provádějí průchody dva.

V prvním průchodu se osvětlení neaplikuje, pouze se ve vertex shaderu transformují vrcholy objektů do kamerového systému a ve fragment shaderu se připraví informace potřebné pro vykreslení scény a výpočet osvětlení. Každá informace se vykreslí do zvláštní textury.

Při druhém průchodu se ve fragment shaderu získají potřebné informace z textur vzniklých v prvním průchodu. Dále se pomocí těchto získaných dat a dalších, jako jsou pozice světelných zdrojů, provedou výpočty spojené s osvětlením či dalšími chtěnými efekty a výsledná barva se vykreslí jako finální scéna.

Hlavní výhodou odloženého stínování je oddělení geometrie scény od osvětlení. Díky tomuto způsobu osvětlujeme jen ty pixely, které jsou ve skutečnosti viditelné. Také je možné měnit pozici světla, aniž by bylo nutné překreslovat (a počítat) celou geometrii scény znovu.

3 Implementace

Tato kapitola se zabývá vytvářením jednotlivých částí bakalářské práce, především tvorbou objektů ve scéně. Podkapitoly jsou sestavovány tak, jak bakalářská práce postupně vznikala. Z počátku byly zpracovávány ty části scény, které se vykreslují do FBO, následují světelné efekty a animované objekty.

3.1 Aplikace deferred shading

Aby bylo vykreslování statické scény efektivní a rychlé, a osvětlování této scény také, je vhodné zvolit metodu deffered shading na cube mapě popsanou v kapitole 2.4.2. V tomto případě je nutné vytvořit několik sad textur, do kterých se bude z fragment shaderu vykreslovat pozice, normála a barva. Každá z těchto textur musí být vytvořena pro každou stranu krychle (tedy na každou část cube mapy) a také jim musí být přiřazeny vlastnosti.

```
glBindTexture(GL_TEXTURE_CUBE_MAP, CubeMapPosition);
for (int i = 0; i < 6; i++)
	glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
		GL_RGBA32F, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
```

Kód č. 3.1: *Příklad vytvoření textur pro zápis pozice*

Nejprve se vytvořená textura `CubeMapPosition` ve formátu cube mapy připojí. Následně se v cyklu pro každou stranu krychle zvlášť vytvoří obrázek pomocí funkce `glTexImage2D()`. Interní formát dat pro pozici je `GL_RGBA32F`. `w` a `h` jsou proměnné obsahující šířku a výšku vykreslovacího okna. Normála a barva jsou vytvářeny stejným způsobem, jako pozice. Jediný rozdíl je v interním formátu, kdy oba obrázky mají interní formát `GL_RGBA`.

FBO musí mít celkem tři tzv. color attachmenty. Díky tomu totiž povolíme vykreslování do více objektů. Této metodě se říká multiple render targets. Pořadí attachmentů, které se pomocí funkce `glDrawBuffers()` specifikují pro vykreslení, musí odpovídat pořadí vykreslovaných prvků ve fragment shaderu. Pořadí se určuje následujícím způsobem:

```
layout (location = 0) out vec4 fragColor;
layout (location = 1) out vec4 fragNormal;
layout (location = 2) out vec4 fragPosition;
```

Kód č. 3.2: *Specifikace pořadí výstupních dat ve fragment shaderu*

Výstupem jsou tedy tři čtyř-složkové float vektory.

Aby bylo možné vykreslovat pouze tu část scény, která je opravdu viditelná, je vhodné použít renderbuffer object. Ten se vytváří zároveň s FBO. Vytvoří se další attachment pomocí funkce

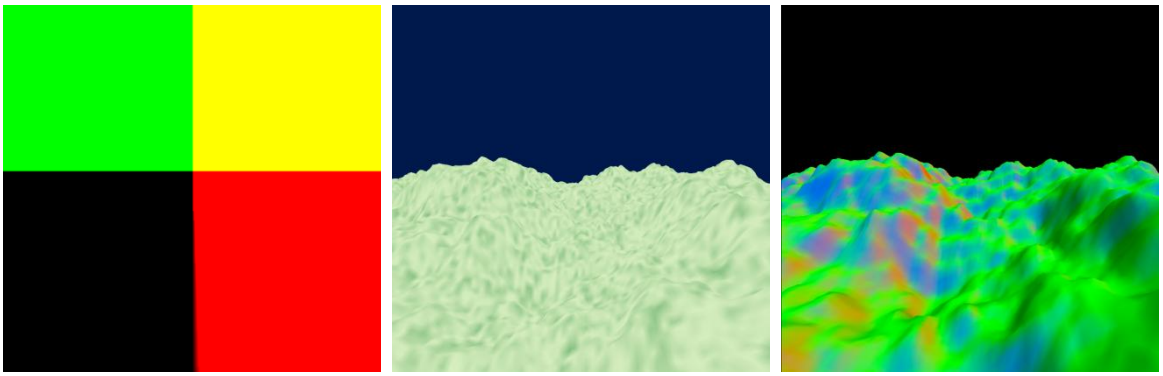
`glFramebufferRenderbuffer()`, tentokrát však nejde o color, nýbrž depth attachment. Bude se zde zapisovat hloubka. Na rozdíl od color attachmentů se depth attachment ani nespecifikuje pomocí `glDrawBuffers()`, ani nevykresluje z fragment shaderu určením layoutu. Pouze se pomocí funkce `glDepthFunc()` specifikuje podmínka, kdy bude pixel vykreslen. V módu `GL_LEQUAL` se vykreslí, pokud je hloubka menší nebo rovna hodnotě uložené.

Pro vykreslení scény z FBO je nutné vytvořit další vertex a fragment shader. Do vertex shaderu se posílají pouze vrcholy krychle, do které se bude scéna vykreslovat. Ve fragment shaderu se vykonávají větší operace.

Nejprve se z každé textury získá potřebná informace, tedy pozice, barva a normála. Pro získání texelu z textury cube mapy se musí vypočítat texturovací koordináty. Ty se vypočítají přímo pro okno, do kterého se scéna bude vykreslovat, z ModelView matice. Tímto je umožněno získat informace pro výpočet osvětlení apod.

```
vec3 ComputeVector() {
    vec2 Ind = - 1 + 2 * gl_FragCoord.xy / vec2(Width, Height);
    vec3 X = modelViewMatrix[0].xyz * Ind.x;
    vec3 Y = modelViewMatrix[1].xyz * Ind.y;
    vec3 Z = modelViewMatrix[2].xyz;
    return normalize(- Z + X + Y );
}
```

Kód č. 3.3: *Výpočet texturovací koordináty pro CubeMapu. Width je šířka okna, Height je výška okna.*



Obrázek 3.1: *Vlevo vykreslená textura pozice, uprostřed textura barvy a vpravo textura normály.*

3.2 Moře

Prvním krokem pro napodobení podmořského světa je vytvořit moře. Vzhledem k tomu, že pozorovatel má pozici pod hladinou moře, je voda všude kolem něj. Jako nejjednodušší způsob vykreslení vody je vytvořit krychli, na kterou se nanese barva vody.

Geometrie moře je tedy jednoduchá. Stačí vytvořit vrcholy reprezentující geometrii krychle. Tyto vrcholy se ve vertex shaderu transformují až do kamerového systému. Ve fragment shaderu je specifikována barva moře. Tímto získáme základ podmořské scény - modré okolí.

3.3 Písečné dno

Po vytvoření vody následuje vytvoření mořského dna. Těch může být několik druhů, například kamenité, korálové či písečné. V této práci je implementováno dno písečné. Princip vymodelování dna je zkonstruovat plochu, která bude mít v různých bodech různou výšku a na tuto plochu nanést texturu.

3.3.1 Geometrie dna

Aby bylo možné vykreslit plochu napodobující písek, je nutné nejprve vytvořit mřížku bodů reprezentujících pozice vrcholů jednotlivých primitiv, které tvoří výsledný objekt. Povrch je tvořen množinou trojúhelníků.

Souřadnice `x` a `z` trojúhelníků jsou generovány v cyklu. Souřadnice `y`, která reprezentuje výšku, je získávána pomocí 2D Perlinova šumu. Šum je implicitně nastaven tak, aby tvořil velmi mírně zvlněné dno. Získanou hodnotu je třeba zasadit do větší hloubky, než je pozice pozorovatele (která je ve středu souřadnicového systému).

Zároveň se získáváním vrcholu primitiva se provádí výpočet texturovací koordináty pro tento vrchol, aby bylo možné namapovat texturu na dno jako na celek, nikoliv na každé primitivum zvlášť. To zaručí, že se nebude výrazně opakovat motiv.

Poslední výpočet, který se provádí, je výpočet normály k primitivu. Normála je totiž potřebná pro Phongův osvětlovací model. Do shaderů se tedy zasílají jak vrcholy trojúhelníků, tak normály k nim a také texturovací souřadnice.

3.3.2 Textura písku

Slovo písek evokuje představu jasné žluté barvy. Na mořském dnu však tato barva taková není. Voda totiž postupně absorbuje červené světlo a tím ztrácí intenzitu. Barva písku je tedy šedivější a méně výrazná. Na písku se také mohou vyskytovat nečistoty, případně náznaky porostu a podobně.

Výsledná textura písku se vytváří mícháním dvou textur. První textura obsahuje výše zmíněnou barvu písku. Druhá textura má tmavě zelenou barvu, avšak s rozdílným alfa kanálem. Ten je generován Perlinovým šumem, čímž se získají jemné přechody mezi průhlednou částí textury a zabarvenou částí.

Pomocí funkce `glTexParameterf()` se nastaví způsob nanášení textur. Obě textury jsou nastaveny na mód `GL_REPEAT`. Funkcí `glTexParameteri()` a parametrem `GL_LINEAR` jsou

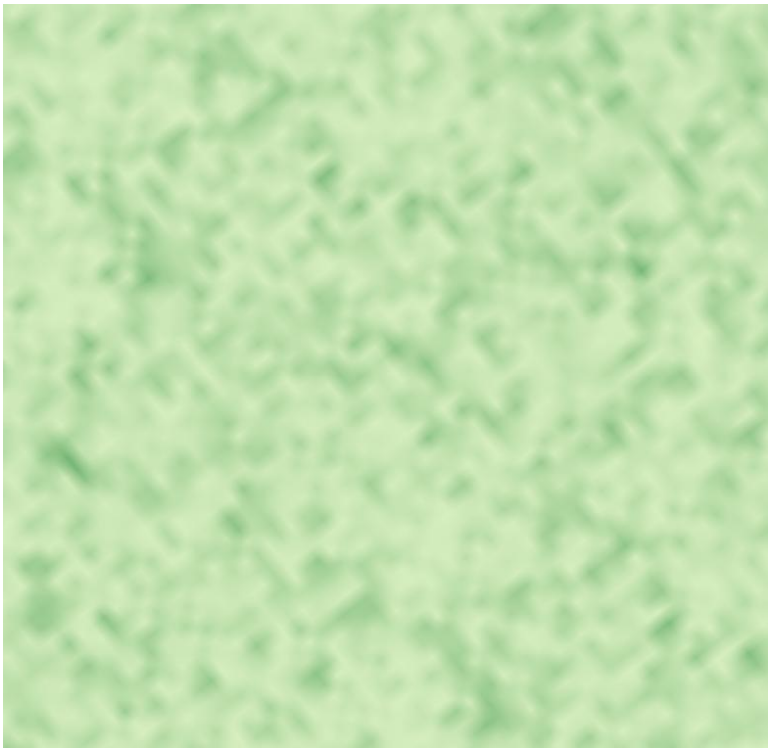
textury parametrizovány na získání texelu textury pomocí váženého průměru čtyř nejbližších texelů v blízkosti uvedené souřadnice pro mapování textury.

Tyto dvě textury se spolu smíchají ve fragment shaderu pomocí funkce `mix()`. Právě alfa kanál zelené textury je využit jako hodnota pro interpolaci mezi texturou písku a zelenou barvou specifikovanou v druhé textuře. Ve výsledku se získá textura, která již napodobuje písek s nečistotami.

```
vec4 tex1 = texture(renderedTexture, coord);
vec4 tex2 = texture(renderedTextureGreen, coord);
fragColor = mix (tex1, vec4(tex2.rgb, 1.0), tex2.a);
```

Kód č. 3.4: *Způsob míchání dvou textur pomocí alfa kanálu.*

Zde `renderedTexture` je handler k textuře žluté barvy, `renderedTextureGreen` je handler textury se zelenými fleky, `coord` jsou texturovací koordináty počítané při vytváření vrcholů a `fragColor` je výstupní barva.



Obrázek 3.2: *Výsledná namíchaná textura písku s nečistotami*

3.4 Kameny

Poslední neanimovanou částí, která se vykresluje do FBO, jsou kameny. Pro jejich vytvoření je potřeba nadefinovat geometrii modelu kamene, dále vytvořit texturu, která bude kámen připomínat a v neposlední řadě specifikovat parametry pro případnou instanciaci kamenů.

3.4.1 Geometrie a textura kamenů

Pro vytvoření modelu kamene se využívá podobného principu, jako je popsáno v podkapitole 3.2.1. Zprvu se vygeneruje mřížka bodů, ve které se budou vytvářet trojúhelníky. Členitost kamene je opět závislá na parametrech Perlinova šumu. Parametry šumu je nutné nastavit tak, aby nebyly vidět okraje, případně dutá místa. Dále se specifikuje hloubka, ve které se kameny nacházejí. Jinak by mohla vzniknout situace, že kameny budou plavat nad dnem, nebo naopak budou pod dnem a nebudou viditelné. Takto vytvořený model se použije jako předloha pro vytvoření ostatních kamenů. Metoda vytváření dalších kamenů je popsána v následující kapitole.

Textura kamene se skládá ze tří textur. Základní texturu tvoří neprůhledná hnědá barva. Na ní se nanáší ostatní textury. Druhou texturou je šedá textura s proměnlivým alfa kanálem. Poslední texturou je zelená textura, taktéž s proměnlivým alfa kanálem. Všechny textury mají stejné nastavení parametrů, jako je uvedeno v podkapitole 3.3.2. Míchání těchto textur je také zprostředkováváno funkcí `mix()`. Nejprve se smíchá základní hnědá textura s šedou barvou z druhé textury, kde pro interpolaci slouží alfa kanál šedé textury. Po získání barvy z tohoto kroku se provede mixování podruhé, tentokrát se vezme získaná hodnota z předešlého kroku a smíchá se spolu se zelenou barvou poslední textury. Pro interpolaci opět poslouží alfa kanál zelené textury. Výsledná textura namíchaná ze tří textur je na obrázku 3.3.

Obrázek 3.3: *Textura kamenů namíchaná z hnědé, šedé a zelené barvy.*

3.4.2 Instanciování

Další kameny jsou vytvářeny tzv. instancováním tohoto modelu. Je to způsob, jak jeden model vykreslit vícekrát do scény. Provádí se pomocí standardních funkcí pro vykreslování, avšak s koncovkou `Instanced` (`glDrawElementsInstanced()` např.), kde se navíc v parametrech této funkce určí počet instancí k vykreslení.

Aby bylo možné jednotlivé instance umístit různě po prostoru, je nutné pro každou instanci upravit pozici vrcholu. Toto je možné například ve vertex shaderu. Nejprve se připraví data, která zastupují posun pozice instancí oproti původnímu objektu. Zde se vytvoří pole, do kterého se vloží náhodně vygenerované pozice instancí.

```
for (int i = 0, x = 0; i < instanceCount; i++, x += 4) {
	glm::vec3 p = getUnicateLocation();
	centerPtr[x + 0] = p.x;
	centerPtr[x + 1] = p.y;
	centerPtr[x + 2] = p.z;
	centerPtr[x + 3] = 0.0f;
}
```

Kód č. 3.5: *Vytvoření unikátních poloh instancí, které se budou k původně vytvořenému modelu přičítat.*

Velikost uniformního pole ve shaderech je značně omezená. Je proto vhodné najít jiný způsob, jak data o posunu do shaderu předat. Jednou z možností je vytvořit si texturu, do které se obsah pole vloží. Pro tento princip je vhodné nastavit texturu na mód `GL_CLAMP_TO_EDGE`

a `GL_NEAREST`. Takto připravenou texturu je možné předat shaderu a čerpat z ní potřebné informace. Ve vertex shaderu existuje proměnná `gl_InstanceID`, která má v sobě vždy uloženou hodnotu aktuálně zpracovávané instance. Pomocí této proměnné provedeme výpočet texturovací koordináty pro získání posunu.

3.5 Mlha ve vodě

Počítačové obrázky se někdy zdají nerealisticky ostré a umělé. Jedním z efektů, který přiblíží scénu realitě, je efekt mlhy. Ta způsobí, že objekty se v dálce ztrácejí. Jejich barva se s rostoucí vzdáleností přibližuje barvě mlhy.

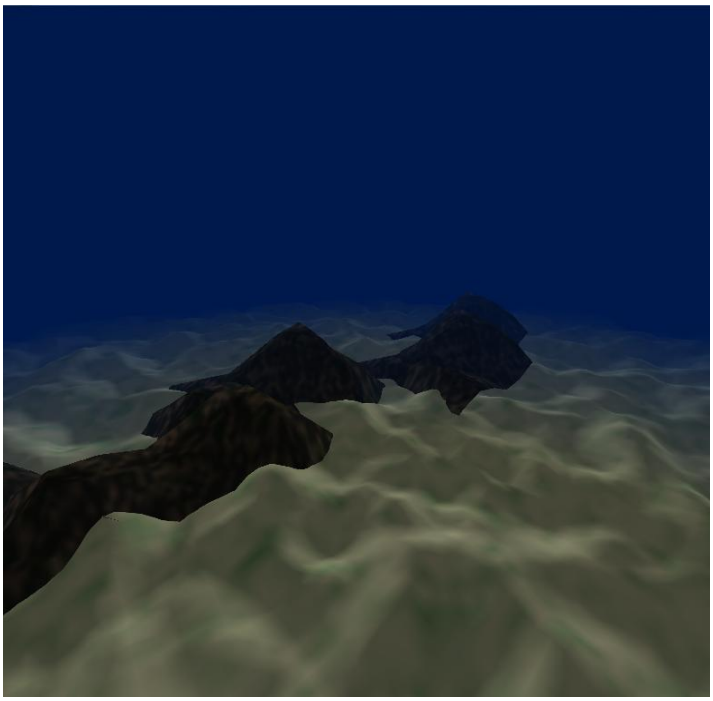
Jelikož mořská voda není průzračná, tak s rostoucí vzdáleností klesá viditelnost. Tento jev se dá simulovat právě aplikováním mlhy. Mlha se napodobí smícháním barvy vykreslovaného objektu s barvou moře ve fragment shaderu pomocí funkce `mix()`. Pro získání hodnoty pro interpolaci těchto dvou barev se využívá vzdálenosti vrcholu od počátku souřadného systému. Výsledná scéna je patrná z obrázku 3.4.

3.6 Světelné efekty

Další částí bakalářské práce jsou světelné efekty. Slunce nad mořskou hladinou způsobuje různé světelné efekty. Například paprsky, které se objevují a zase mizí, dále kaustiky, které se neustále pohybují a samozřejmě celkové osvětlení scény.

3.6.1 Osvětlení scény

Po získání pozice, normály a barvy je možné začít počítat osvětlení. Využívá se Phongova osvětlovacího modelu. Vypočítá se difúzní složka, která se vynásobí barvou světla a barvou získanou z textury. Se spekulární složkou se nepracuje, jelikož písek ani kameny nejsou lesklé materiály. Pokud je skalární součin normály a směru zdroje světla menší než nula, osvětlovací model se neaplikuje. Scéna s takovýmto osvětlením je vidět na obrázku 3.4.



Obrázek 3.4: *Scéna vyobrazující písečný terén s kameny, aplikováním osvětlení do scény a aplikováním mlhy*

3.6.2 Paprsky

Jedním ze světelných efektů viditelných pod mořskou hladinou jsou sluneční paprsky. Implementace tohoto problému spočívá ve využití částicových systémů a také využití geometry shaderu. Mezi vlastnosti paprsku jako částice patří pozice, doba života a průhlednost.

3.6.2.1 Geometrie paprsku

Jak bylo zmíněno výše, u paprsku se bude využívat geometry shader, který je popsán v [9]. Do tohoto shaderu vstupuje bod, který je náhodně vygenerovaný na hladině. Ještě ve vertex shaderu je pomocí ModelView matice transformován. Výstupem geometry shaderu jsou čtyři vrcholy tvořící jedno primitivum `triangle_strip`.

```
layout(points)in;
layout(triangle_strip, max_vertices=4)out;
```

Kód č. 3.6: *Specifikace formátu vstupních dat do geometry shaderu a výstupní formát dat z geometry shaderu.*

Výstupem bude tedy čtyřúhelník. Aby měly všechny paprsky směr od pozice Slunce, využívá se homogenní souřadnice pozice světla (taktéž v Model space). Tato souřadnice se odečtením od vstupního bodu a normalizací bude dále používat pro vytvoření vrcholů paprsku.

Výpočet pozice každého vrchu se provádí ještě před převedení do pohledu kamery. Samotný výpočet spočívá především ve vytvoření čtyř-složkového vektoru, který určuje odchylku od vstupního bodu v ose x a y. Následující konstrukce vygeneruje čtyři vrcholy a tím vznikne jedno primitivum:

```
vec2 k=vec2(-1,1)*(normalize(gl_in[0].gl_Position-l)).yx;
gl_Position = m * (vec4(k,0.0,0.0) + gl_in[0].gl_Position-l);
EmitVertex();
gl_Position = m * (vec4(-k,0.0,0.0) + gl_in[0].gl_Position-l);
EmitVertex();
gl_Position = m * (vec4(k,0.0,0.0) + gl_in[0].gl_Position);
EmitVertex();
gl_Position = m * (vec4(-k,0.0,0.0) + gl_in[0].gl_Position);
EmitVertex();
EndPrimitive();
```

Kód č. 3.7: *Postup generování vrcholů v geometry shaderu.*

kde vektor `k` slouží jako výše zmíněná odchylka od vstupního bodu, `gl_in[0].gl_Position` je pozice vstupního bodu v model space, `l` je homogenní pozice zdroje světla v model space a `m` je projekční matice.

3.6.2.2 Textura paprsků

Textura paprsku je tvořena světlou tyrkysovou modří. V základu je alfa kanál textury postupně klesající k horizontálním okrajům. Ve fragment shaderu se alfa kanál ještě modifikuje podle viditelnosti daného paprsku, který je do shaderu předáván spolu s vrcholy pomocí VAO. Textura je nastavena na `GL_CLAMP_TO_BORDER`, což zaručí, že textura bude přilepena k okrajům.

3.6.2.3 Vlastnosti paprsku

První vlastností každého paprsku je pozice. Vygeneruje se tedy náhodný bod na hladině. Pozice paprsku zůstává po dobu jeho životnosti. Další vlastností je viditelnost. Paprsek zvyšuje svoji viditelnost až do maximální stanovené hodnoty, poté viditelnost klesá až do úplné průhlednosti.

Doba života je náhodně vygenerovaná hodnota. Pomocí ní a maximální viditelnosti se určí krok. Ten slouží pro růst i klesání viditelnosti a vychází ze vztahu:

$$ray.step = \frac{maxVisibility}{ray.lifetime \cdot 2} \quad (21)$$

kde $step$ je krok pro daný paprsek, $maxVisibility$ je hodnota určující maximální velikost alfa kanálu, která je stejná pro všechny paprsky, a $lifetime$ je doba života daného paprsku. Násobeno dvěma je z toho důvodu, že paprsky postupně sílí a poté slábnou, což vyžaduje dvakrát menší krok.
Pro viditelnost paprsku platí vztah:

$$ray.visibility = ray.visibility \pm ray.step \quad (22)$$

O znaménku rozhoduje, zdali se paprsek objevil a sílí, nebo už slábne a zaniká. Jakmile doba života vyprší, dojde k vygenerování nové náhodné pozice, nové náhodné doby života a celý proces začíná znovu.

3.6.3 Kaustiky

Posledním světelným efektem realizovaným v této práci jsou kaustiky. Ty vznikají zakřivením světelného paprsku nějakou zakřivenou plochou, v tomto případě vodní hladinou. Kaustiky jsou vytvořeny jako objemová (3D) textura. Velmi autentická napodobenina kaustik je možná vytvořením 3D Voroného diagramu, jehož 2D implementace je popsána v kapitole 2.2.2. Trojrozměrný diagram se vytvoří pouze přidáním třetí dimenze do výpočtů.

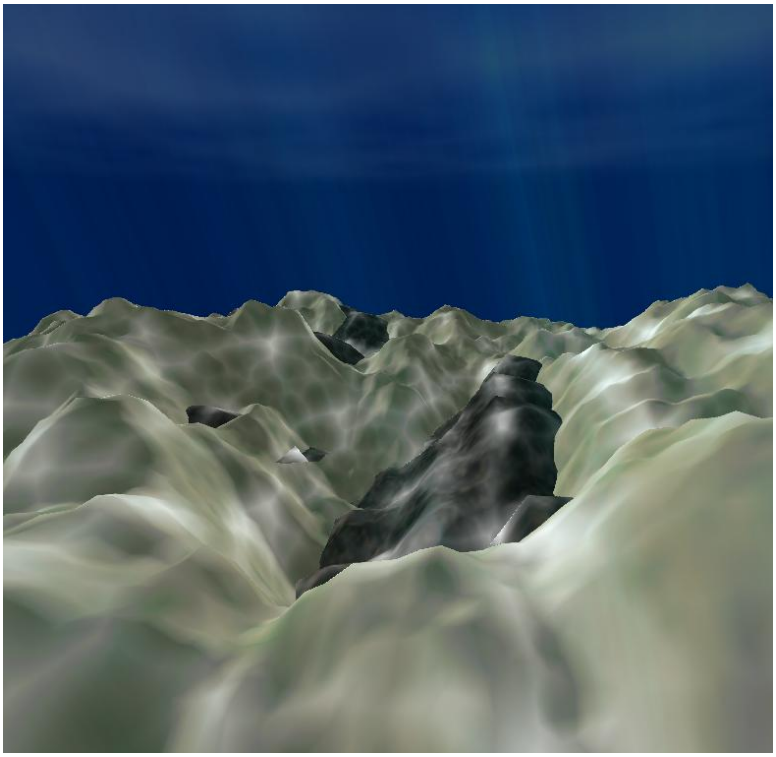
Na texturování objektu se využívá tzv. triplanárního texturování. Tento způsob nanáší textury podél souřadných os. Pro výpočet určující, kterou texturu použít, se využívá normály povrchu a souřadnice povrchu. Více textur se dá použít v případě, že normály nejsou rovnoběžné s některou ze souřadných os. Výpočet hodnoty kaustik v daném bodě se ve fragment shaderu provádí postupem uvedeným v kódu 3.7:

```
float CausticsValue =
       texture(caustics,(Position*stretch+vec3(T,0,0))).a*abs(N.x)+
       texture(caustics,(Position*stretch+vec3(0,T,0))).a*abs(N.y)+
       texture(caustics,(Position*stretch+vec3(0,0,T))).a*abs(N.z);
```

Kód č. 3.8: *Výpočet hodnoty kaustik v čase pro danou pozici*

kde `caustics` je handler pro přístup k textuře kaustik, `Position` je vektor určující pozici , `T` je čas, `N` je normalizovaná normála a `stretch` je koeficient roztažení textury kaustik v terénu.

Kaustik je také možno využít pro simulaci vodní hladiny. Pokud se bod nachází nad určitou mezí, je na průsečík polopřímky a roviny (zastupující hladinu) v nějaké výšce namapována kaustika.



Obrázek 3.5: *Aplikování kaustik, paprsků a vodní hladiny*

3.7 Bublinky

Částicový systém nalézá uplatnění i zde. Bublinky se objevují vždy v určitém místě, generátoru bublin. Nejprve jsou bublinky malé, s rostoucí výškou však roste jejich velikost. Jakmile dosáhnou hladiny, zmizí. Zároveň se zde využívá instanciování pro vytvoření více takovýchto generátorů.

3.7.1 Geometrie bubliny

Pro vytvoření bubliny se využívá geometry shaderu, kde podobně, jako v podkapitole 3.6.2.1, vstupuje do shaderu bod a výstupem budou čtyři vrcholy tvořící `triangle_strip`. Zde se jeden vrchol počítá následujícím způsobem:

```
gl_Position = m * (vec4(-Size,-Size,0.0,0.0) + gl_in[0].gl_Position);
```

Kód č. 3.9: *Ukázka výpočtu jednoho vrcholu*

`Size` určuje poloměr bubliny. Do geometry shaderu se posílá z vertex shaderu, jelikož každá bublina má jiný poloměr. Ten je generován náhodně, avšak před vstupem do vertex shaderu se mírně mění. Modifikace totiž simuluje platnost stavové rovnice ideálního plynu:

$$p \cdot V = n \cdot k \cdot T, \quad (23)$$

kde p je tlak plynu, V je objem plynu, n je látkové množství, k je Boltzmannova konstanta a T je termodynamická teplota. Tato rovnice není přesně naimplementována, ale simulace její vlastnosti - s rostoucí výškou se plyn rozpíná a s ním se mění poloměr bubliny - ano.

3.7.2 Textura bubliny

Bublinka má tvar koule, kde směrem ke středu roste průhlednost. Jelikož primitivum vytvořené v geometry shaderu je vždy natočeno směrem k pozorovateli, vytvoří se textura, která bude znázorňovat kružnici. Pro vytváření textury se vychází z obecné rovnice kružnice v kartézském souřadném systému se středem kružnice v bodě (0,0):

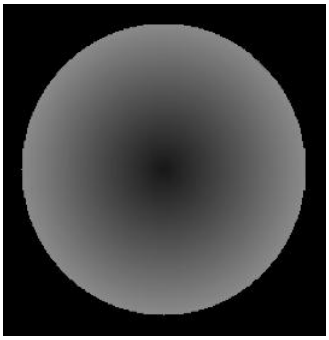
$$x^2 + y^2 = r^2 \quad (24)$$

kde x a y je množina bodů a r je poloměr kružnice.

Body, pro které tato rovnice vyhovuje, jsou hraniční body. Pokud je součet větší, jedná se o nevyplněné místo s alfa kanálem rovným nule. Pokud je součet menší, zapisuje se do textury bílá barva s příslušnou hodnotou alfa kanálu. Ta se odvíjí od vzorce:

$$\alpha = \frac{\sqrt{x^2 + y^2}}{textureWidth} \quad (25)$$

kde α je počítaný alfa kanál a $textureWidth$ je šířka textury. Takto vytvořené textuře se nastaví mód `GL_CLAMP_TO_EDGE`. Takto vytvořená textura je na obrázku 3.6.



Obrázek č. 3.6: *Výsledná textura bublinky se snižujícím se alfa kanálem směrem ke středu kružnice.*

3.7.3 Vlastnosti bubliny

Každá bublinka v jednom generátoru má svou pozici, rychlost a hmotnost. Afektory bubliny jsou tíhová síla a vztlaková síla. Zároveň na bublinu působí prvek náhody, který simuluje Brownův pohyb.

Hmotnost bubliny se počítá podle rovnice měrné hustoty:

$$\rho = \frac{m}{V} \quad (26)$$

kde ρ je hustota plynu, m je hmotnost a V je objem bubliny. Objem se vypočítá ze vzorce pro výpočet objemu koule:

$$V = \frac{4}{3} \cdot \pi \cdot r^3 \quad (27)$$

kde π je Ludolfovo číslo a r poloměr bubliny.

Bublinka tedy postupně působením afektorů stoupá k hladině. Pokud pozice bubliny v ose y překročí určitou mez, bublinka zanikne a objeví se znovu na dně.

3.8 Řasy

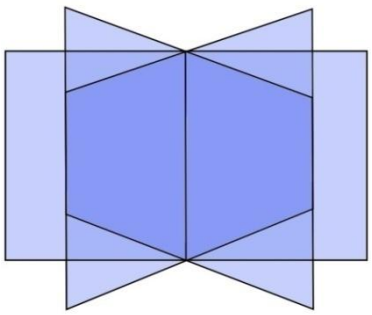
Aby podmořská scenérie nabyla živého dojmu, jsou vykreslovány řasy, které se mírně pohybují. Řasy se vyskytují ve shlucích a takovýchto shluků může být na mořském dně několik.

3.8.1 Geometrie řas

Vykreslovat řasy stéblo po stéblu nemá smysl, protože počet polygonů, který by se musel zpracovat, je obrovský. Scéna s mnoha stébly řas by nebyla vykreslovatelná v real time grafice. Proto musí být větší množství řas reprezentováno jen několika polygony.

Jeden ze způsobů řešení tohoto problému je vytvořit objekt, který se skládá ze tří polygonů, na které se nanese textura pro více řas. Aby byl zajištěn správný obraz nezávisle na aktuální linii pohledu, jsou tyto tři polygony překříženy. Výsledný objekt je vidět na obrázku 3.7.

Pohyb řas je implementován postupným posouváním vrcholů polygonů, které se nacházejí v objektu nahoře. Jelikož je pozice těchto vrcholů ve VAO známá, stačí v cyklu projít všechny požadované vrcholy a upravit jejich pozici. Každá pozice se mění o specifikovanou hodnotu, dokud nepřekročí jistou mez. Poté se směr pohybu obrací na druhou stranu.

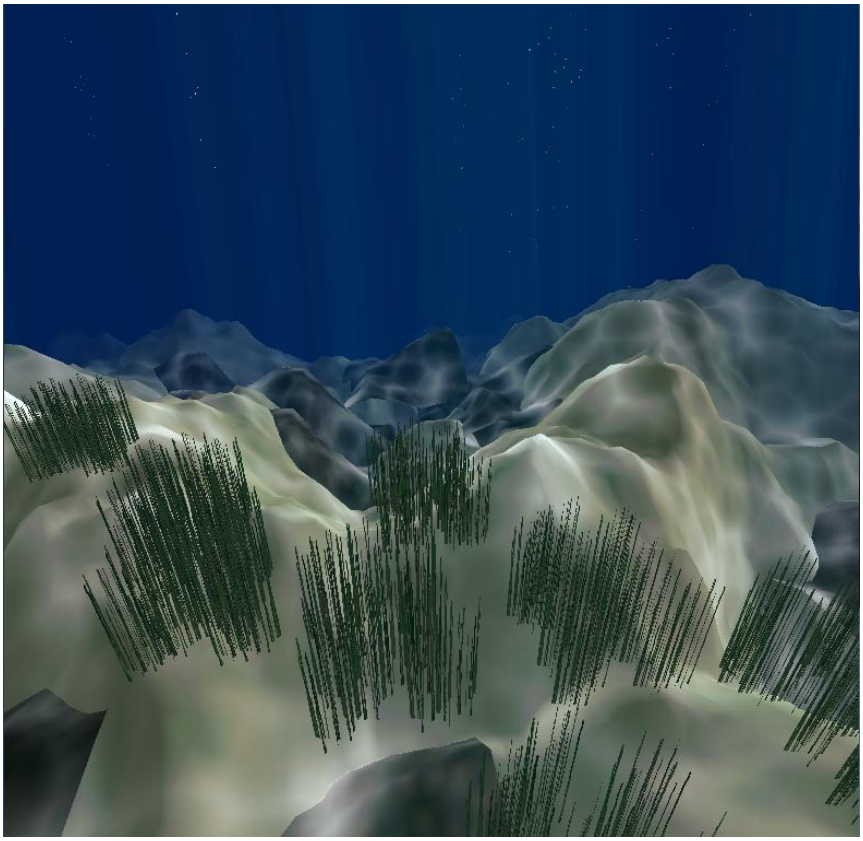


Obrázek 3.7: *Překřížené polygony (obrázek převzat z [11])*

3.8.2 Textura řas

Pro jeden polygon je nutné mít texturu obsahující více stébel řas. Jedním ze způsobů vytvoření této textury je vystřelovat částici a její cestu zapsat do textury. Tím se získá několik křivek připomínajících řasy. Aby řasy působily realističtějším dojmem, je pro obarvování tras částice použito několika barev. Tato barva je náhodně vybrána a přiřazena stéblu řasy.

Textura je inicializovaná s alfa kanálem rovným nule. Barvy řas jdou nejrůznější odstíny zelené. Při vytváření textury se tedy pro každou křivku zvlášť určí náhodně jeden z předdefinovaných odstínů zelené. Aby se zobrazovaly všechny strany polygonů a nejen ty, které jsou hloubkou blíž, povolí se blenging pomocí `glEnable (GL_BLEND)` a nastavení blendingu pomocí `glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`. Výsledná textura s tímto nastavením je patrná z obrázku 3.8, kde jsou řasy vykresleny.



Obrázek 3.8: *Aplikování řas*

3.9 Kamera

Pohyb ve skyboxu je realizován pomocí myši. Implementováno je otáčení kolem osy y. Algoritmus otáčení je řešen pomocí zjištění rozdílu pozice kurzoru od středu vykreslovacího okna. Pro korekci rychlosti otáčení se tento rozdíl vynásobí citlivostí. Poté se výsledek přičte k úhlu dosavadního otočení v ose x a v ose y.

Pomocí těchto úhlů se připravuje ModelView matice. Princip výpočtu spočívá v otáčení kolem souřadných os o úhel, který odpovídá dosavadnímu otočení. Výpočet matice rotované kolem osy x a y se provádí zvlášť. Výsledná ModelView matice se získá vynásobením těchto matic.

```
glm::mat4 A = glm::rotate(glm::mat4(1.0f), angleX[i], glm::vec3(0.0, 1.0, 0.0));
glm::mat4 B = glm::rotate(glm::mat4(1.0f), angleY[i], glm::vec3(1.0, 0.0, 0.0));
glm::mat4 ModelView = A*B;
```

Kód č. 3.10: *Výpočet ModelView matice pomocí rotování o daný úhel v ose x a y*

3.10 Rozdělení projektu

Projekt je psán v jazyce C++. Vyvíjen byl ve vývojovém prostředí Microsoft Visual Studio 2012. Zdrojové kódy projektu mají přes 3400 řádků. Zdrojové soubory jsou v projektu členěny do tří logických části - zdrojové soubory, hlavičkové soubory a shadery.

mainFunc

- Zde jsou funkce pro práci s hlavním oknem, jeho inicializaci a obsluhu
- Funkce pro vytvoření implementace metody deferred shading
- Obsluha vykreslování
- Zpracování konfiguračního souboru
- Implementace časovače pro výpočet hodnoty kaustik
- Zpracování finálního vertex a fragment shaderu, jejich slinkování do programu

openGLDriver

- Součástí je inicializace vykreslování, nastavení view portu
- Obsahuje funkce pro práci s projekční maticí (vytvoření projekční matice)

camera

- V tomto souboru je implementována práce s kurzorem
- Výpočet úhlů, o které se scéna natáčí

perlinNoise

- Obsahuje implementaci 2D a 3D Perlinova šumu

land

- Funkce pro vytvoření geometrie terénu písku, výpočet normál a texturovacích koordinát
- Vytvoření textur pro písek
- Zpracování shaderů a jejich slinkování do programu

skybox

- Vytvoření geometrie vodní plochy
- Zpracování všech shaderů a výsledného shader programu

reef

- Zpracování geometrie kamene, výpočet normál a texturovacích koordinát
- Generování pozic pro posun instancí, vytvoření textur pro kameny
- Zpracování shaderů a jejich slinkování

bubbles

- Vygenerování bodů reprezentujících bubliny
- Vytvoření textury
- Generování pozic pro další generátory
- Výpočet diferenciálních rovnic a specifikace vlastností jednotlivých částic
- Zpracování vertex, geometry a fragment shaderu a jejich slinkování do programu

rays

- Předpřipravení geometrie paprsků (vygenerování bodů na hladině)
- Vytvoření textury paprsků, výpočet průhlednosti, aktualizace vlastností částic
- Zpracování shaderů a slinkování

seaweed

- Vygenerování geometrie objektu řas
- Vytvoření textury
- Implementace pohybu
- Generování nových pozic v rámci jednoho shluku, generování posunu pro instance
- Zpracování shaderů a jejich slinkování do shader programu

3.11 Systémové a hardwarové požadavky

Projekt byl vyvíjen pro operační systém Windows. Pro práci s okny je využito WINAPI. Pro práci s shadery se využívá GLSL verze 400. Aplikace byla vyvíjena na grafické kartě s podporou OpenGL 4.3.0.

3.12 Konfigurační soubor

Nedílnou součástí této práce je konfigurační soubor. Pomocí něj je možné nastavit scénu podvodního světa nejrůznějšími kombinacemi hodnot. Je možné upravovat členitost terénu, počet vykreslovaných objektů a dalších. Podrobněji budou jednotlivá nastavení popsána v podkapitole 3.12.2.

3.12.1 Syntaxe

Komentáře jsou rozlišeny pomocí znaku '#'. Struktura jednotlivých nastavení je klíčové slovo, rovnítko a hodnota bez mezer. Každé klíčové slovo musí být odděleno znakem nového řádku. Očekávaný datový typ je vždy uveden v komentáři.

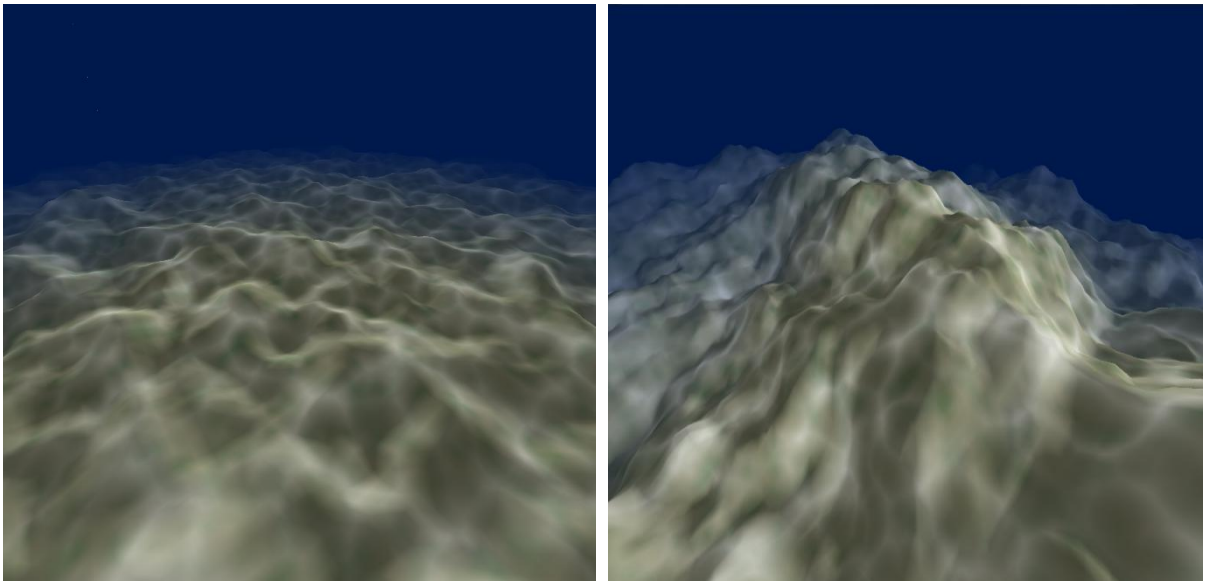
Klíčová slova rozeznávaná v konfiguračním souboru jsou popsána v následující kapitole. Jejich krátký popis je obsažen přímo v konfiguračním souboru.

3.12.2 Popis jednotlivých nastavení

U terénu je variace možných nastavení. Jednak se specifikují parametry pro Perlinův šum, dále v jaké hloubce je dno posazeno a nastavení mlhy.

Keyword	Hodnota	Popis
spOctaves	unsigned int	Počet oktáv Perlinova šumu pro písek.
spPersist	float	Persistence pro Perlinův šum pro písek
spFreq	float	Frekvence jako další parametr.
spAmpl	float	Poslední parametr Perlinova šumu, určuje amplitudu.
spHeighConst	float	Konstanta pro zvýšení nebo snížení y hodnot terénu písku.
sDepth	float	Volba hloubky, ve které se písečný terén bude nacházet.
srFog	float	Pozice mlhy pro písek a kameny.

Tabulka č. 3.1: *Specifikace nastavení parametrů pro terén písku.*



Obrázek č. 3.9: *Vlevo: terén s nastavením Perlinova šumu - počet oktáv 4, perzistence 1.4, frekvence 1.0, amplituda 0.8 a použitá výšková konstanta 1.0, Vpravo: nastavení šumu - počet oktáv 4, perzistence 2.9, frekvence 1.8, amplituda 0.8 a použitá výšková konstanta 1.5*

Pro kamínky je nastavení obdobné, jako pro terén písku. Jsou zde parametry pro Perlinův šum, výšková konstanta a hloubka kamenů. Navíc je zde volba počtu kamenů na dně.

Keyword	**Hodnota**	**Popis**
rocksNum	unsigned int	Počet instancí kamínků na mořském dně.
rpOctaves	unsigned int	Počet oktáv Perlinova šumu pro kameny.
rpPersist	float	Persistence pro Perlinův šum pro kameny.
rpFreq	float	Frekvence jako další parametr.
rpAmpl	float	Poslední parametr Perlinova šumu, určuje amplitudu.
rpHeighConst	float	Konstanta pro zvýšení nebo snížení y hodnot kamenů.
rDepth	float	Volba hloubky, ve které se kameny budou nacházet.

Tabulka č. 3.2: *Specifikace nastavení parametrů pro kameny.*

Další velkou částí konfiguračního souboru je nastavení řas. Je možné specifikovat velikost shluků, hustotu objektů ve shluku, hloubku řas a také pozici mlhy. Textura řas je také podrobně specifikovatelná. Nastavuje se šíře stébla, maximální výška a počet řas na jednu texturu.

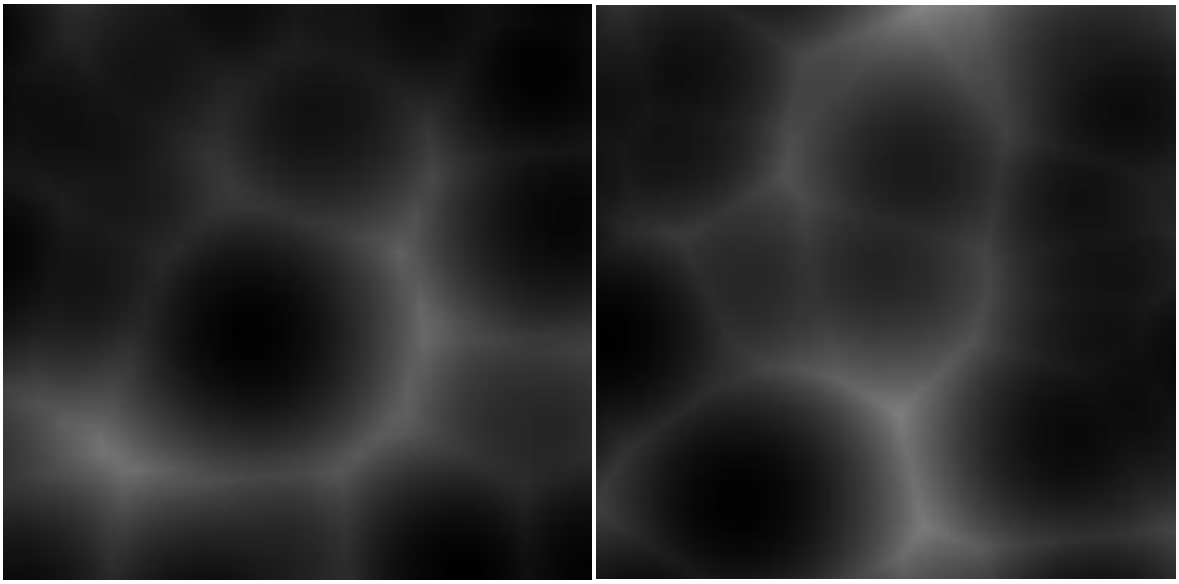
Keyword	**Hodnota**	**Popis**
sClustersNum	unsigned int	Hodnota určující počet shluků řas.
swNum	unsigned int	Počet objektů (složených polygonů) v jednom shluku řas.
sWidth	float	Tímto lze určit, jak široká stébla řas v textuře mají být.
sHeight	float	Možnost nastavit, jaká bude maximální výška řas v textuře.
swDepth	float	Hodnota, která určujíce, v jaké hloubce budou řasy zasazeny.
swFogPos	float	Maximální pozice mlhy pro řasy.

Tabulka č. 3.3: *Specifikace nastavení parametrů pro řasy.*

Kvalita kaustik se dá určit pomocí několika parametrů. Mezi ně patří velikost textury, počet bodů ve Voroného diagramu a také roztažení textury na terénu.

Keyword	Hodnota	Popis
textureWHD	unsigned int	Volba velikosti (a tedy i kvality) textury kaustik. Tato objemová textura má stejnou výšku, šířku i hloubku, proto stačí jediná hodnota. S větší velikostí textury souvisí i delší doba zpracování.
pointsNum	unsigned int	Počet generátorů ve Voroného diagramu pro texturu kaustik.
stretch	float	Roztažení kaustik na mořském dně.

Tabulka č. 3.4: *Parametry pro generování textury kaustik.*



Obrázek č. 3.10: *Vlevo velikost textury kaustik 32x32x32, počet generátorů 70, Vpravo velikost textury 128x128x128, počet generátorů 70.*

Parametry pro počet generátorů bublinek a jejich hloubku, počet slunečních paprsků, výšku mořské hladiny a velikost okna je také možné v konfiguračním souboru nastavit.

Keyword	Hodnota	Popis
winSize	unsigned int	Nastavení velikosti okna pro vykreslování scény. Okno aplikace má čtvercový tvar, takže jedno číslo značí jak šířku, tak výšku.
raysNum	unsigned int	Počet slunečních paprsků, které se budou ve vodě objevovat a mizet.
bGenNum	unsigned int	Volba počtu generátorů bublinek na mořském dně.
bDepth	float	Určuje, v jaké hloubce se mají generátory bublinek nacházet.
wLvlHeight	float	Výška, ve které se bude nacházet mořská hladina.

Tabulka č. 3.5: *Nastavení bublinek, paprsků, hladiny a velikosti okna.*

Obrázek č. 3.11: *Vlevo vykreslení 40 paprsků ,Vlevo vykreslení 200 paprsků. Díky většímu množství se více překrývají a jsou více viditelné.*

4 Závěr

Úkolem této bakalářské práce bylo seznámit se s problematikou a vytvořit podvodní animovaný skybox v OpenGL. Během tvorby jsem vyzkoušela několik různých alternativních řešení problémů. Například zprvu jsem vytvářela geometrii každého stébla řasy zvlášť, avšak v rámci optimalizace pro animování řas jsem zvolila metodu popsanou v kapitole 3.8.

Tato práce má velké množství možných rozšíření. Například vytvořit více druhů mořského dna, útesy, více druhů kamenů a mořského porostu. Dalším rozšířením je implementovat proudění mořské vody, podle kterého by se odvíjel pohyb porostu a bublin. Přidání stínování. Dále vytvořit rybičky a jiné mořské živočichy. Rybičky by mohly být řešeny pomocí billboardingu a mapování několika textur pro simulaci pohybu ploutví. Ve vodě vytvořit plovoucí nečistoty. Je zde také velká řada možných optimalizací. Například implementace šumu do shaderu nebo optimalizace vytváření Voroného diagramu.

Vytvořila jsem podmořský svět obsahující písek, který je tvořen sítí trojúhelníků a členitostí nastavitelnou Perlinovým šumem. Kameny byly taktéž řešeny jako síť trojúhelníků, na kterou se nanese textura. Dále jsem vytvořila řasy, které jsou tvořeny objektem složeným ze tří protínajících se ploch, na který se nanese textura s několika stébly. Generují se bublinky, které jsou reprezentovány bodem, avšak díky geometry shaderu vzniká primitivum `triangle_strip`, který je vždy natočen ke kameře. Také jsem implementovala různé světelné efekty. Kupříkladu kaustiky, které jsou vytvářeny pomocí Voroného diagramu, nebo světelné paprsky, které jsou také vytvářeny v geometry shaderu z bodu a různě se objevují a mizí.

Tento svět lze jednoduchými úpravami konfiguračního souboru měnit dle vlastních představ. Je možné nastavit členitost terénu, množství a tvar kamenů a vzhled shluků řas. Nastavitelná je i hloubka objektů a výška mořské hladiny. Také je možné nakonfigurovat kaustiky - kvalitou textury či počtem generátorů ve Voroného diagramu.

S programováním grafických aplikací pomocí OpenGL jsem se setkala poprvé až v této práci. Díky vývoji této práce jsem získala mnoho zkušeností a znalostí, co se do programování v OpenGL a procedurálního generování týče. Vývoj aplikace mě velice bavil, byl pro mě velkým přínosem a těším se na získávání nových znalostí při vytváření rozšíření této práce v budoucnu.

Literatura

[1] Above The Sea. In: *Braynzar soft* [online]. 2012 [cit. 2013-05-13]. Dostupné z: http://www.braynzarsoft.net/vision/index.php?p=VT&texture=25

[2] SEGAL SEGAL, Mark a Kurt AKELEY. *The OpenGL Graphics System: A Specification (Version 4.3 (Core Profile))*. 2012.

[3] SHREINER, Dave. OpenGL: průvodce programátora. Vyd. 1. Překlad Jiří Fadrný. Brno: Computer Press, 2006, 679 s., [16] barev. obr. příl. DTP. ISBN 80-251-1275-6

[4] Texturing: a procedural approach. 3rd ed. Amsterdam: Morgan Kaufmann Publishers, 2003, xxiii, 687 s. ISBN 15-586-0848-6.

[5] PERLIN, Ken. Making noise. [online]. 1999 [cit. 2013-05-12]. Dostupné z: http://www.noisemachine.com/talk1/index.html

[6] Perlin noise. In: *Farao* [online]. 2004 [cit. 2013-05-13]. Dostupné z: http://farao.czweb.org/perlin.htm

[7] Aurenhammer, F.: *Voronoi Diagrams - A survey of a Fundamental Geometric Data Structure.* ACM Computing Surveys, 1991.

[8] JIŘÍ, Žára a Jiří ŽÁRA. Moderní počítačová grafika. Vyd 1. Brno: Computer Press, 2004, 609 s. ISBN 80-251-0454-0.

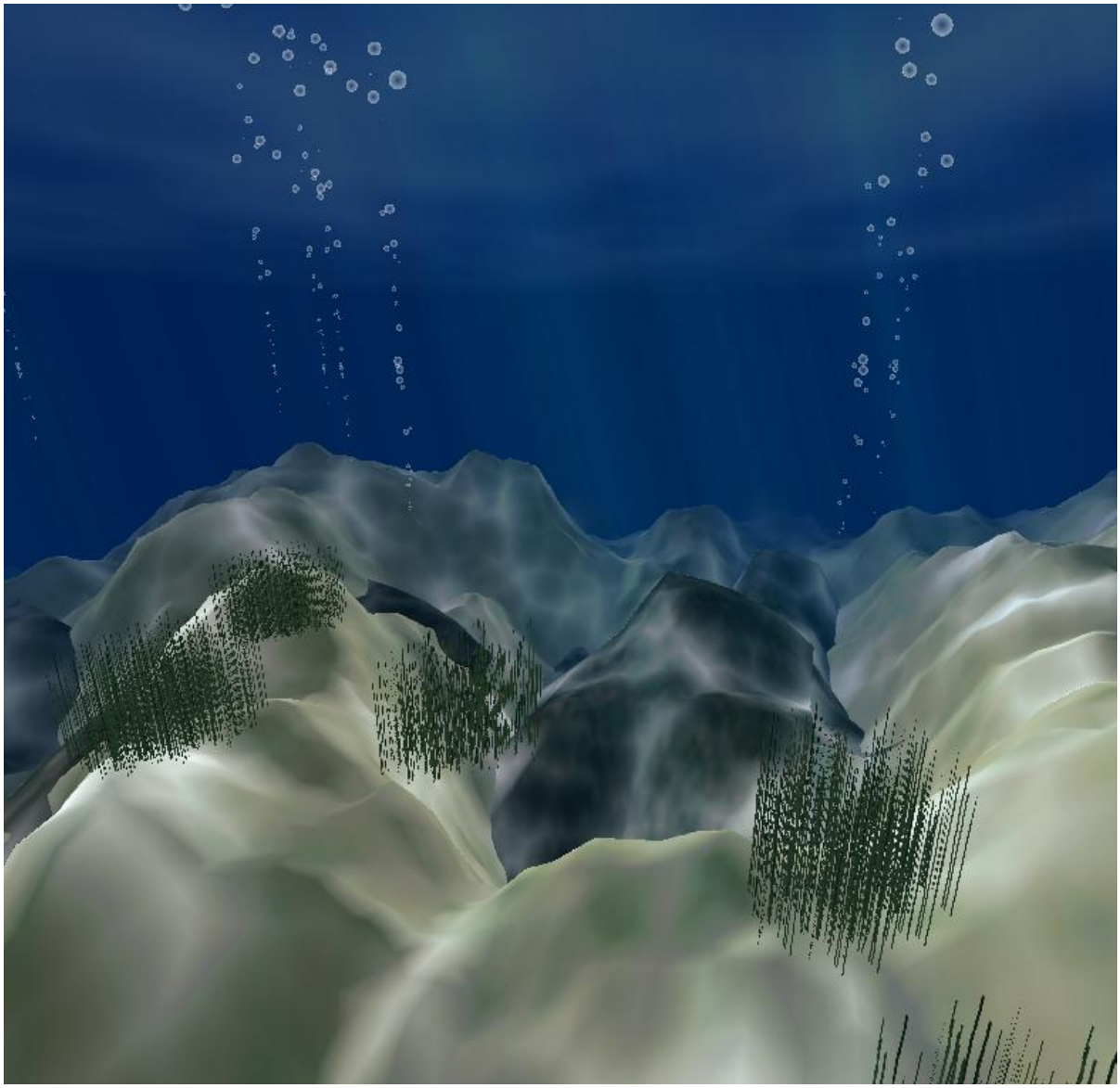
[9] Communications of the ACM: a monthly publication of the ACM Publications Office. Illumination for computer generated pictures. 1975, č. 18. DOI: 0001-0782. Dostupné z: http://delivery.acm.org/10.1145/370000/360839/p311-phong.pdf

[10] WOLFF, David. OpenGL 4.0 shading language cookbook: over 60 highly focused, practical recipes to maximize your use of the OpenGL shading language. 1st pub. Birmingham: Packt Publishing, 2011, iii, 323 s. ISBN 978-184-9514-767.

[11] FERNANDO, R. GPU gems. Vyd. 1. Boston: Addison-Wesley, 2004. ISBN 03-212-2832-4.

Seznam příloh

Příloha A - Obrázek



Obrázek A.1: *Výsledná scenérie odpovídající nastavení konfiguračního souboru v příloze B*

Příloha B - Ukázkový konfigurační soubor

```
########################   Aplikace   ##############################
winSize=800

########################   Kaustiky   ##############################
textureWHD=32
pointsNum=70
stretch=0.1

########################   Paprsky    ##############################
raysNum=500

########################   Bublinky   ##############################
bGenNum=25
bDepth=20

########################   Kameny     ##############################
rocksNum=30
rpOctaves=4
rpPersist=2.0
rpFreq=1.0
rpAmpl=0.7
rpHeighConst=10.0
rDepth=42.0

########################   Rasy       ##############################
sClustersNum=20
sWidth=8
swDepth=25
sHeigh=120
swNum=18
swObj=12
swFogPos=140

########################   Pisek      ##############################
spOctaves=4
spPersist=2.9
spFreq=1.8
spAmpl=0.8
spHeighConst=1.5
sDepth=20.0
srFog=85

########################   Hladina    ##############################
wLvlHeight=100
```

Příloha C - CD

- Zdrojové soubory
- Spustitelný soubor
- Plakát
- Readme
- Video