

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

PRŮLET NAD PLANETOU V REÁLNÉM ČASE

BAKALÁŘSKÁ PRÁCE

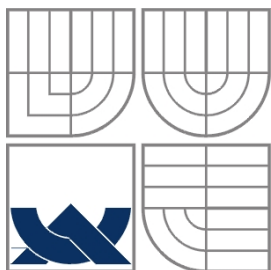
BACHELOR'S THESIS

AUTOR PRÁCE

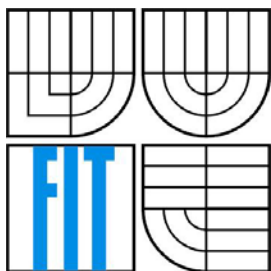
AUTHOR

PETR POSPÍŠIL

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PRŮLET NAD PLANETOU V REÁLNÉM ČASE

FLY OVER RUGGED PLANET

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR POSPÍŠIL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. RADOVAN JOŠTH

BRNO 2009

Abstrakt

Tato práce se zabývá 3D zobrazením povrchu planety Země v reálném čase. Podstatná část se věnuje přípravě rozsáhlých zdrojových dat (výškové mapy a textury Země) a jejich efektivnímu načítání z harddisku přímo do paměti grafické karty. Jednotlivé úrovně detailů povrchu planety jsou řešeny pomocí metody GeoMipMapping. Využity jsou algoritmy horizon a frustum culling, které ořezávají skryté části planety. Nanášení výškových dat je postaveno na speciálně navrženém algoritmu zpracování dat přímo v grafické kartě za asistence vertex a geometry shaderů.

Abstract

This thesis deals with a 3D visualization of the Earth's surface in real time. A significant part is dedicated to the buffering of the large amount of data (a height map and textures of Earth) and its effective reading from hard disk directly to the memory of a graphic card. Individual levels of detail of the Earth's surface are made by GeoMipMapping. The algorithms "horizon" and "frustum culling" are used, which cut off the hidden parts of the planet. The application of the height data is based on a custom algorithm calculating the data directly in the graphic card, in assistance with its vertex and geometry shaders.

Klíčová slova

planeta Země, úroveň detailu, GeoMipMapping, horizont culling, frustum culling, vlákna, výšková mapa planety Země, textura planety Země

Keywords

planet Earth, level of detail, GeoMipMapping, horizont culling, frustum culling, threads, planet Earth heightmap, planet Earth texture

Citace

Petr Pospíšil: Průlet nad planetou v reálném čase, bakalářská práce, Brno, FIT VUT v Brně, 2009

Průlet nad planetou v reálném čase

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radovana Joštha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Pospíšil
15. 5. 2009

Poděkování

Děkuji Ing. Radovanovi Jošthovi za odbornou pomoc při návrhu a vytváření této práce.

© Petr Pospíšil, 2009

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

Obsah	1
1 Úvod	2
2 Získávání povrchových dat planety	3
2.1 Výšková mapa a textura	3
2.1.1 Zdroj výškové mapy a textury	4
2.2 Generování dat za běhu programu	4
3 Zobrazování povrchových dat planety	5
3.1 Úroveň detailu	5
3.1.1 QuadTree	5
3.1.2 GeoMipMapping	6
3.2 Výběr vhodné úrovně detailu	7
3.3 Mapování dat planety	7
3.3.1 Z plochy na kouli	7
3.3.2 Z krychle na kouli	8
3.4 Ořezání zbytečně zobrazovaných dat	9
3.4.1 Frustum culling	9
3.4.2 Horizon culling	10
4 Využití grafických karet	12
4.1 Programovatelná linka	12
4.2 Shadery	13
4.2.1 Vertex shader	13
4.2.2 Pixel shader	13
4.2.3 Geometry shader	13
5 Návrh	15
5.1 Vlastní algoritmus	15
5.2 Aplikace	16
6 Implementace	17
6.1 Příprava zdrojových dat	17
6.1.1 Soubory výškové mapy	17
6.1.2 Soubory textury	18
6.1.3 Pomocný soubor	19
6.2 Načítání zdrojových dat	19
6.3 Vlastní algoritmus	20
6.3.1 Výpočet souřadnic bodů	20
6.3.2 Výpočet souřadnic textur	21
6.4 Výběr jednotlivých úrovní detailu	21
6.4.1 Pomocí chyby v pixelech na obrazovce	21
6.4.2 Pomocí vzdálenosti od středů políček	21
6.5 Atmosféra	22
6.6 Omezení	22
7 Testování aplikace	23
8 Závěr	27
8.1 Budoucí vývoj	27

1 Úvod

Zobrazování planet patří mezi složitější oblasti grafiky. Programátor se potýká s mnoha různými problémy, které v jiných oblastech často řešit nemusí.

Povrchy jednotlivých planet jsou obvykle popsány množstvím dat. Musí se s nimi efektivně nakládat, protože načtení všech dat najednou není možné. I se současně vysokým výkonem hardwaru nelze celý povrch zobrazovat vždy v nejvyšší kvalitě. Vykresluje se pouze část, které může uživatel skrze svou obrazovku v daném okamžiku vidět. Vykreslovaná část často není v nejvyšší možné úrovni detailu, protože to z hlediska vzhledu není ani vyžadováno. Aplikace podle vzdálenosti či jiných kritérií vybírá, jak v dané chvíli kvalitní část použít. Na zobrazování planet lze využít techniky, které jsou často používány pro zobrazení terénu v jiných grafických aplikacích. Tyto metody se však musí občas modifikovat, abychom získali požadované výsledky.

V *druhé* kapitole se budeme věnovat, jakými druhy uspořádání dat lze povrch planety reprezentovat a jaké tyto metody skýtají úskalí.

Ve *třetí* kapitole popíšeme jeden konkrétní algoritmus pro zobrazování několika úrovní detailu terénu, který je po menších úpravách vhodný i k použití pro planetu. Dále se budeme zabývat, jakými způsoby lze vytvořit ze vstupních dat planetu a jak ořezávat nezobrazené části planety.

Čtvrtá kapitola se věnuje způsobům využití grafických karet v dnešních aplikacích.

Pátá a *šestá* kapitola popisují návrh a implementaci zobrazení planety v reálném čase. Zde si řekneme více o některých problémech, které je nutné vyřešit. Zmíníme se také o omezení, které aplikaci částečně limituje.

V *sedmé* kapitole jsou provedené rychlostní testy. Je monitorována spotřeba paměti a počet obrázků za vteřinu. Zmíníme se o detailech, které musely být doladěny, aby aplikace běžela plynuleji. Budeme se také věnovat nastavení základních konstant pro úroveň detailu a dalších, které ovlivňují finální výsledek.

2 Získávání povrchových dat planety

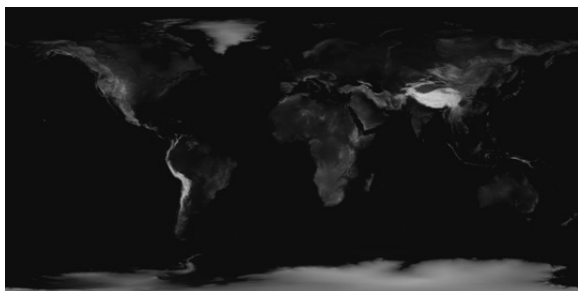
Abychom mohli zobrazit planetu, musíme nejprve získat data, která budou věrně popisovat její povrch. Obvykle jsou tato uložena ve formě volně stažitelných souborů. Je však možné zdrojová data i „ručně“ vytvořit, či generovat až za běhu programu.

V této kapitole si popíšeme dva základní způsoby reprezentace povrchových dat, které patří mezi nejpoužívanější. Prvním způsobem je využití předpřipravené výškové mapy a textury. Výšková mapa, jak již její název vypovídá, obsahuje hodnoty nadmořských výšek v metrech na povrchu planety. Textura naopak popisuje barevné rozložení povrchu viditelné lidským okem. Výšková mapa a textura tedy dohromady tvoří všechny potřebné informace pro zobrazení povrchu dané planety. Druhým způsobem je generování dat za běhu programu pomocí matematických funkcí, které generují pseudonáhodný povrch. Pokud nám tedy nezáleží na přesně definovaném vzhledu, lze zvolit tento způsob.

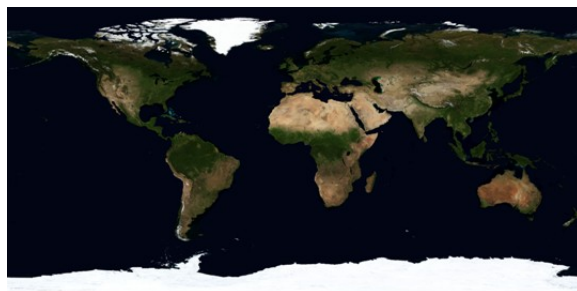
Tato práce se ale zaměřuje na využití výškové mapy a textury planety Země, proto se nebudeme příliš podrobně zabývat generovanými daty.

2.1 Výšková mapa a textura

Jak již bylo řečeno v úvodu kapitoly, výšková mapa obsahuje výšky jednotlivých bodů v metrech. Textura se obvykle kryje přesně s výškovou mapou, aby ji bylo možné jednoduše a hlavně přesně nanášet. Tento přístup umožňuje velmi dobře kontrolovat detaily zobrazené planety, jelikož aplikace pouze zobrazuje data předem uložená na harddisku, nikoliv až za běhu generovaná procesorem. Na druhou stranu tento způsob neumožňuje popsat celou planetu v příliš vysoké kvalitě. Navíc v současné době nejkvalitnější dostupná zdrojová data zabírají obrovské množství paměti, často i několik desítek GB (Giga Bajtů).

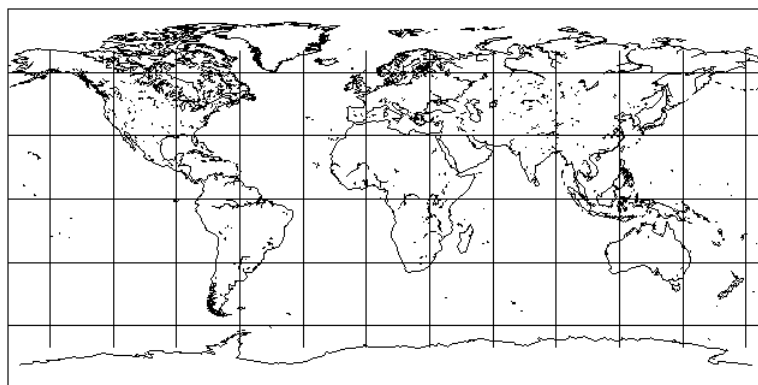


Obrázek 2.1: Výšková mapa Země



Obrázek 2.2: Textura Země

Výškové body jsou obvykle seřazeny podle geografických souřadnic. Jednotlivé řady mají počátek a konec na 180. poledníku. Řádky se nachází za sebou v pořadí od severního k jižnímu pólu. Aby však mohla být data takto seřazena, musí se obraz planety převést na plošné zobrazení. K tomuto účelu slouží velké množství různých projekcí. V oblasti výškových map patří k nejpoužívanějším projekce s názvem „plate carrée“ (viz obrázek 2.3). Jedná se o velmi jednoduchý převod, který byl objeven již okolo roku 100 našeho letopočtu. V této projekci mají rovnoběžky mezi sebou konstantní vzdálenost a tvoří přímky. To samé platí i o polednicích. Poledníky jsou kolmé na rovnoběžky. Dohromady tvoří „mřížku“. Oblasti blízké se k pólům jsou roztažené, proto se rozlišení oblastí pólů a rovníku liší. Tato projekce reprezentuje vztah jednotlivých pixelů s geografickou pozicí, proto se často upřednostňuje pro reprezentaci zdrojových dat v grafických aplikacích.



**Obrázek 2.3: Plate carrée projekce planety Země,
převzato z internetových stránek věnujících se Plate carrée projekci [1]**

Protože jsou data k dispozici v paměti již během inicializace aplikace, lze z nich získat užitečné informace, které se dále během programu využijí, např., jaké jsou souřadnice nejvyšší hory v oblasti a která oblast je pokryta výhradně vodní plochou, či rovinou.

2.1.1 Zdroj výškové mapy a textury

Předlohou výškové mapy a textury, jak již bylo výše naznačeno, je planeta Země. Je zvolena z důvodu, že se jedná o nejvíce zmapované těleso naší soustavy. Samotná data byla získána z [2]. Jedná se o databázi textur a výškových map pořízených agenturou NASA. Projekt nese název *Blue Marble next generation*. Data jsou získána pomocí satelitů. Předtím, než jsou volně k dispozici, musí být odstraněn šum. Ten je často způsoben například velkou oblačností, ale také i oblastmi jako jsou deštné pralesy, které se jeví pro satelit příliš tmavé. Pohlcují velké množství světla.

Maximální dostupná velikost dat je 86 400 na 43 200 bodů při rozlišení přibližně 500m na jeden pixel v oblasti rovníku. Soubor výškové mapy i textury jsou k dispozici v binárním formátu. Pro výškovou mapu se jedná o 16-ti bitová celá čísla se znaménkem v rozsahu od -32 768 do 32 767, což je pro povrch Země, kde nejvyšší hora *Mount Everest* dosahuje výšky 8 850m, dostatečný výškový rozsah. Textura je uložena v hodnotách po 24 bitech, které se skládají z jednotlivých barevných složek (barevné složky jsou červená, zelená a modrá). Každá z nich má velikost jeden bajt. V paměti uložené soubory zabírají přibližně 7GB (výšková mapa) a 11GB (textura).

2.2 Generování dat za běhu programu

Hlavní výhodou tohoto přístupu je fakt, že není nutné načítat data z harddisku, ale všechna data počítá procesor. O generování výškové mapy se starají matematické funkce, mezi které nejčastěji patří Perlinův šum (anglicky *Perlin noise*) a fractional Brownian Motion (zkráceně fBM). Tímto postupem se ale připravujeme o možnost zasahovat do detailů vzhledu planety. Jelikož jsou data náhodně vygenerována, neexistuje k nim přesná textura. Je možné texturu náhodně vygenerovat stejně jako výškovou mapu, vhodně zvolit zabarvení povrchu, či použít sadu textur menších rozměrů, které následně podle nadmořské výšky a geografických souřadnic nanese. Takto nanášená data však mohou vypadat plasticky. Lze však opět použít matematické funkce, které budou představovat při procesu nanášení „náhodný prvek“. Pokud jako zdroj hodnot využijeme tuto metodu, nejsme omezeni kvantitou dat. Je možné generovat planety, co se týče výškové mapy, i v rozlišení menším jak jeden metr na pixel. Avšak generování v takto malém rozlišení je poněkud náročné na výkon.

3 Zobrazování povrchových dat planety

Z předchozí kapitoly máme základní představu, v jakém formátu budou k dispozici zdrojová data aplikace. Nyní se budeme věnovat, jak tato data efektivně zobrazit. Musíme si však uvědomit, že bezhlavé načtení a zobrazení všech informací najednou je z hlediska výkonu nemožné, a také zcela zbytečné. Proto se budeme zabývat způsoby, které se uplatňují pro efektivní a rychle zobrazování i velkého množství dat planety.

V této kapitole si popíšeme význam úrovně detailu (anglicky *level of detail*) pro zobrazování terénu. Podrobněji bude zmíněn algoritmus Geometrical MipMapping (dále jen GeoMipMapping), který je založen na metodě dělení dat, resp. oblastí, zvané QuadTree. Budeme se také zabývat způsoby, které jsou vhodné pro výběr úrovně detailu. Jelikož je GeoMipmapping navrhnut pro zobrazení rovných ploch, nikoliv planet, musíme zajistit převod dat na „kouli“. Na závěr kapitoly si povíme o možnostech ořezání nadbytečných částí planety, které nejsou pozorovatelné z hlediska uživatele a ještě více tak snížit množství zobrazovaných dat.

3.1 Úroveň detailu

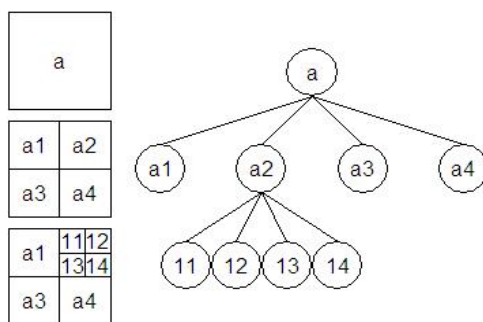
Planetu reprezentuje obrovské množství dat, které nelze zobrazit všechny najednou v plné kvalitě. Potýkali bychom se totiž jak s nedostatkem výkonu, tak také s nedostatkem paměti. Proto se velmi často v případech, kdy je třeba vykreslovat obrovské objekty, uplatňuje úroveň detailu. Je-li nějaký objekt příliš vzdálený od pozorovatele, není třeba jej vykreslovat v nejlepší kvalitě. Plně dostačuje méně kvalitní verze, která zabírá méně paměti a mnohem rychleji se vykresluje.

Existuje nepřeberné množství algoritmů implementujících úroveň detailu pro zobrazení terénu. Zohledňují vzdálenost pozorovatele a členitost terénu. Terén samotný je rozdělen na logické části, které se podle vhodné situace rozdělují na menší, či kvalitnější části. Nikoliv všechny algoritmy jsou vhodné pro zobrazování celých planet. Jeden z nejpoužívanějších algoritmů, který se používá pro zobrazování planet, se nazývá GeoMipMapping, jehož základem je zjednodušená metoda zvaná QuadTree.

3.1.1 QuadTree

Jako zdroj informací pro tuto kapitolu byla použita Wikipedie [3].

Tuto metodu definovali pánové R. Finkel a J. L. Bentley v roce 1974. Jedná se o stromovitou datovou strukturu. Obvykle se používá k rozdělování 2D prostoru. Uzel je reprezentován čtvercovou plochou, není tomu tak vždy. Každý uzel stromu má schopnost rozdělit se na čtyři další uzly (viz obrázek 3.1), které mají všechny vlastnosti svého předka. Mohou se tedy dále dělit, pokud je třeba.



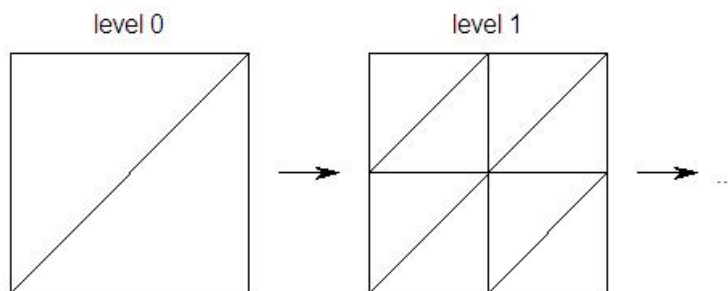
Obrázek 3.1: Ukázka dělení uzlů v QuadTree

Pokud se jedná o 2D prostor, mají často následnické uzly menší velikost, či obsahují větší množství informací. Tento jev lze využít jak pro optimalizace, tak i pro reprezentaci povrchu jako celku, kde následnické uzly představují kvalitnější úroveň detailu. Z metody QuadTree vychází velké množství pokročilých algoritmů, mimo jiné i GeoMipMapping, který jej využívá pro dělení trojúhelníků jednotlivých úrovní detailu.

3.1.2 GeoMipMapping

Jako zdroj informací pro tuto kapitolu byla použit vědecký článek popisující GeoMipMapping [4].

Jako zdroj dat využívá GeoMipMapping 2D pole, resp. výškovou mapu. Nevýhodou tohoto přístupu jsou chybějící „převisy“ a „jeskyně“. Terén je rozdělen na jednotlivá políčka (anglicky *patch*). Jedná se o logické celky, které mají vlastní úroveň detailu. Sousední políčka se mezi sebou vzájemně překrývají, aby nevznikaly mezery. Hraniční body mají společné. Tento systém má malou nevýhodu, že hraniční body se musí uchovávat a zpracovávat dvakrát. Každé políčko musí mít velikost $2^n + 1 \times 2^n + 1$, kde n je celkový počet úrovní detailu. V nejnižší úrovni se skládá pouze ze čtyř bodů, které tvoří 2 trojúhelníky. Při přepnutí na o stupeň lepší kvalitu se každý trojúhelník v políčku rozdělí na čtyři další, stejně jako v metodě QuadTree (viz obrázek 3.2).



Obrázek 3.2: Dělení obsahu políčka na lepší úroveň detailu

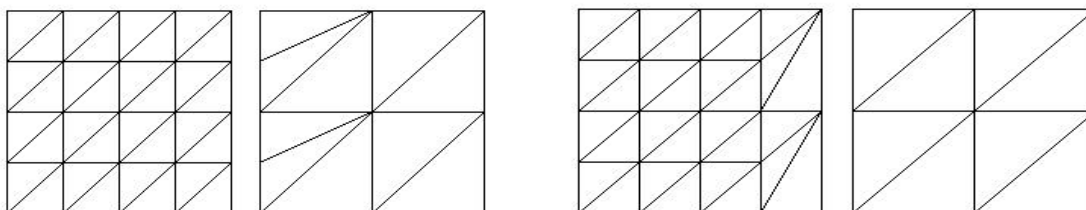
Ovšem v tomto případě se políčko na menší logické celky nedělí, dělí se pouze jeho obsah. Proces dělení se zastaví, jakmile je dosaženo maximálního počtu bodů, které dané políčko může obsahovat. Zdrojová data terénu musí dodržet velikost danou následující rovnicí, kde x je počet políček na ose x , y je počet políček na ose y a n je celkový počet úrovní detailu (viz rovnice 3.1).

$$x' = x \cdot 2^n + 1$$

$$y' = y \cdot 2^n + 1$$

Rovnice 3.1: Velikost výškové mapy
v metodě GeoMipMapping

Pokud sousední políčka mají rozdílné úrovně detailu, vznikají mezi nimi „trhliny“, které působí jako rušivý element pro pozorovatele. Je nutné tento problém řešit. Políčka musí mít přehled o počtu hraničních bodů svých sousedů. Jakmile má některý ze sousedů menší úroveň detailu, znamená to, že některé hraniční body ze sousedovy strany chybí. Musí se proto jedno z políček přizpůsobit druhému. Nastane-li situace, kdy máme dvě sousední políčka s rozdílnou úrovní detailu, máme dvě možnosti. Buď se políčko s menší úrovní detailu přizpůsobí zvýšením počtu svých hraničních bodů, nebo se naopak přizpůsobí políčko s lepší kvalitou (viz obrázek 3.3), kdy hraniční body lepší kvality přesunou na pozici nejbližších bodů nižší kvality. Druhý způsob má jednu velmi důležitou výhodu. Rozdíl mezi úrovněmi detailu sousedů nemusí být pouze o jednu úroveň, protože samotné posouvání bodů je mnohem jednodušší než jejich přidávání.



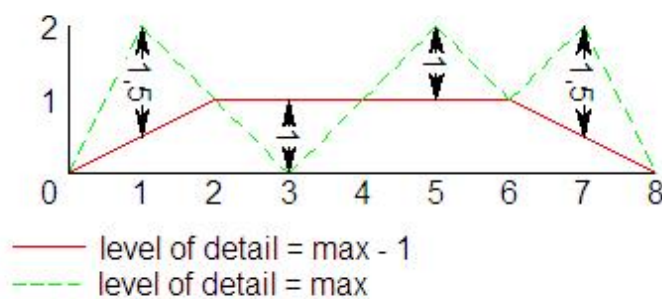
Obrázek 3.3: Vzájemné přizpůsobení úrovní detailu
(vlevo přizpůsobení menší úrovně větší, vpravo větší úrovně menší)

3.2 Výběr vhodné úrovně detailu

Pro rozhodnutí, jakou úroveň detailu použít, lze použít dvě metody, které jsou vhodné i pro GeoMipMapping.

Základní metodou je využití vzdálenosti pozorovatele od daného políčka. Jakmile uživatel překročí jistou vzdálenostní hranici, políčko se přizpůsobí.

V druhé metodě se využívá „chybovost“ jednotlivých úrovní detailu (viz obrázek 3.4). Každá nižší úroveň se liší od nejkvalitnější o danou průměrnou výšku (označíme δ). Ve většině případů výškový rozdíl roste se snižováním kvality jednotlivých políček. Lze jej tedy zohlednit pro určení, jaká úroveň LOD se použije pro dané políčko vzhledem k pozici a pohledu uživatele. Nejprve je nutné přetransformovat velikost rozdílu výšek terénu do pohledu uživatele, který je z perspektivy. Po transformaci δ reprezentuje chybu v pixelech na obrazovce (označíme ε). Pokud je ε menší než námi zvolený koeficient τ , je zobrazovaná chyba natolik velká, že je nutné přepnout zobrazení políčka o jeden stupeň kvality výše. Platí však dvě podmínky. Pokud je τ zvoleno příliš malé, k přepnutí na lepší kvalitu dojde podstatně dříve. Pokud je zvoleno příliš velké, políčko je obvykle reprezentováno nejnižší kvalitou a k jeho přepnutí nemusí nikdy dojít.



Obrázek 3.4: Výšky s úrovní detailu na maximum a o jednu úroveň nižší s vyznačenými rozdíly

3.3 Mapování dat planety

Planetu je možné popsat jako kouli, která má svůj střed a poloměr. Zdrojová data jsou obvykle reprezentována jako plocha, pokud pracujeme s výškovou mapou a texturou. Musíme zajistit transformaci bodů z plochy na kouli. Pokud však nepracujeme s plošnou výškovou mapou, ale s generovanými daty, kde nejsme vázáni jejich formátem, lze zvolit i způsob převodu krychle na kouli.

3.3.1 Mapování z plochy na kouli

Pro tento způsob mapování se používá rovnice pro převod sférických souřadnic na kartézské (viz rovnice 3.2), kde r je poloměr koule, θ je odklon od osy x a φ je odklon od osy z . Úhly volíme v rozsahu $0 \leq \theta \leq \pi$ a $0 \leq \varphi < 2\pi$.

Data výškové mapy jsou však popisována geografickými souřadnicemi (severní a jižní šířkou, východní a západní délkou), které se od sférických liší oborem hodnot. Geografická šířka se nachází v oboru hodnot $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$ a délka v oboru hodnot $-\pi \leq \varphi \leq \pi$. Aby nevznikaly při mapování komplikace, je vhodné geografické souřadnice nejdříve převést na sférické. K šířce se vždy nejprve přičte $\frac{\pi}{2}$, zatímco k délce π .

$$x = r \cdot \sin(\theta) \cdot \cos(\varphi)$$

$$y = r \cdot \sin(\theta) \cdot \sin(\varphi)$$

$$z = r \cdot \cos(\theta)$$

**Rovnice 3.2: Mapování sférických souřadnic
na kartézské (plocha na kouli)**

Tento způsob převodu souřadnic tvoří největší hustotu bodů na pólech a nejmenší okolo rovníku, což občas způsobuje nežádoucí efekty při vizualizaci a nevyrovnané požadavky na výkon. Problém snižujícího se výkonu na pólech lze však vyřešit poměrně snadno. Budeme brát v úvahu geografickou pozici políček, díky které budeme ovlivňovat množství vykreslených bodů. Nevýhodou algoritmu je jeho vyšší náročnost na výkon, jelikož se zde provádí operace sinus a cosinus. Na druhou stranu se nabízí jednoduchá implementace a snadná manipulace. Pomocí rovnice 3.3 lze získat zpět z kartézských souřadnic x , y a z původní hodnoty θ , φ a r .

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\varphi = \text{atan2}(y, x)$$

$$\theta = \text{atan2}(\sqrt{x^2 + y^2}, z)$$

**Rovnice 3.3: Mapování kartézských souřadnic
na sférické (koule na plochu)**

3.3.2 Z krychle na kouli

Pokud bychom měli data uspořádána jako krychli, museli bychom použít jiný postup. Jedná se o transformaci krychle na kouli pomocí rovnice 3.4. Výpočetně je tento způsob převodu méně náročný než rovnice 3.2. Je nepoužitelný pro práci s plochou, protože mapuje 3D objekty. Museli bychom výškovou mapu, která je reprezentovaná 2D plochou, převést na krychli. Převod je netriviální operace hlavně v oblasti pólů. Pokud jsou však data generována za běhu programu, je možné data reprezentovat jako krychli a nikoli jako plochu.

$$x' = x \cdot \sqrt{1 - \frac{y^2}{2} - \frac{z^2}{2} + \frac{y^2 \cdot z^2}{3}}$$

$$y' = y \cdot \sqrt{1 - \frac{z^2}{2} - \frac{x^2}{2} + \frac{z^2 \cdot x^2}{3}}$$

$$z' = z \cdot \sqrt{1 - \frac{x^2}{2} - \frac{y^2}{2} + \frac{x^2 \cdot y^2}{3}}$$

Rovnice 3.4: Mapování krychle na kouli

Metoda mapování krychle na kouli vytváří stejnou hustotu bodů na celém povrchu. Při užití tohoto převodu se neobjevují nežádoucí vzory v zobrazení planety. Jelikož rovnice 3.4 obsahuje odmocniny, je převod z koule na krychli komplikovanější než u rovnice 3.2. Je nutné pro každou z rovnic rozlišit, ze které oblasti původní krychle vstupní data pocházejí.

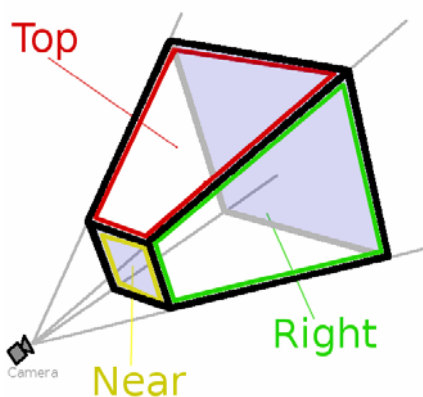
3.4 Ořezání zbytečně zobrazovaných dat

V oblasti zobrazování planet se často uplatňují optimalizační metody, které snižují množství zobrazovaných dat. Pokud bychom optimalizace nevyužili, zbytečně budeme pracovat s objekty, které momentálně nelze z pohledu uživatele vidět. Jsou buď mimo zobrazovaný kužel kamery v 3D prostoru, či jsou překryté jinými objekty, které jsou blíže k pozorovateli. Základní metodou je ořezání komolým kuželem pohledu (anglicky *frustum culling*). Avšak v aplikacích, které zobrazují planety, lze využít i metodu ořezání odvrácené strany povrchu planety (anglicky *horizon culling*).

Aby však bylo možno efektivně tyto metody použít, musíme testované objekty „zapouzdřit“ pomocí zjednodušené geometrie. Lze využít meznou kulovou plochu (anglicky *boundary sphere*), která pohlcuje celý objekt. Všechny testy viditelnosti se následně provádí pouze s kulovou plochou, nikoliv s jednotlivými částmi objektu popisovanými pomocí grafických primitiv. Takové testování by bylo velmi náročné z hlediska požadavků na výkon. Jelikož mezná kulová plocha obklopuje celý objekt a tato plocha není pomocí některé z optimalizačních metod viditelná, není již potřeba testovat objekt samotný, který se nachází uvnitř. Kulová plocha však nemusí pokrývat jen jediný objekt, může pokrývat celý shluk objektů. Tímto lze dosáhnout dalších optimalizací, kdy se nejprve testuje skupina objektů. Pokud je tato skupina viditelná, rozdělí se na podskupiny a ty se testují znovu. K tomuto účelu lze využít metodu založenou na algoritmu QuadTree (viz kapitola 3.1.1).

3.4.1 Frustum culling

V grafických aplikacích uživatel pozoruje 3D scénu pomocí perspektivního zobrazení. Pomocí tohoto zobrazení však uživatel obvykle vidí jen velmi malou část scény. Je proto vhodné objekty, které potenciálně nelze vidět, ořezat. K tomuto účelu právě slouží frustum culling. Perspektivní zobrazení ve 3D grafice je reprezentováno maticí „perspektivy“. Poloha a směr pohledu uživatele ve 3D scéně je v danou chvíli známá a je charakterizována maticí „viditelnosti“. Po vynásobení je možné z těchto matic určit polohu jednotlivých ploch, které udávají prostor viditelný z pohledu uživatele. Jedná se o část jehlanu, která leží mezi dvěma rovnoběžnými rovinami (anglicky *frustum*, viz obrázek 3.5). Na základě polohy jednotlivých ploch je možné testovat, zda je objekt z pohledu uživatele viditelný, či nikoliv.



Obrázek 3.5: Oblast viditelná uživatelem z pohledu kamery, převzato z Wikipedie [5]

Jedná se o základní optimalizační algoritmus. Díky jeho užití lze však i několikanásobně zvýšit výkon při zobrazení scény. Tento algoritmus je využitý při zobrazování v grafických kartách. V tomto případě se však neprovádí ořezání celých objektů, ale jsou testovány jednotlivé geometrické primitivy, ze kterých se objekt skládá.

Testování všech objektů ve scéně vzhledem k šesti plochám, které ohraničují oblast viditelnou uživatelem, je velmi neefektivní. Proto lze i v tomto případě použít jednoduchá vylepšení. Celou oblast lze obalit do mezní kulové plochy stejným způsobem jako objekty, u kterých testujeme viditelnost. Pomocí testu na průnik dvou koulí ořežeme první skupinu objektů, která je mimo

sestrojenou hraniční kouli. Dále se pro testování sestrojí kužel, který kryje podstatně menší prostor, než koule. Využitím testu průniku kužele a koule je množství viditelných objektů zredukováno na minimum. Až na závěr se použije původní ořezání vzhledem k šesti plochám, které již přesně reprezentují viditelný úsek uživatelem v 3D prostoru. Jako zdroj informací byl použit článek věnující se frustum culling [6].

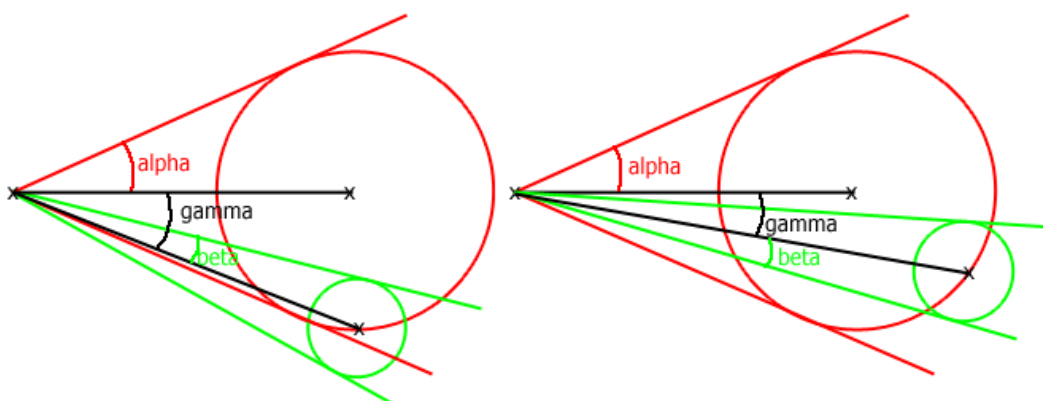
3.4.2 Horizon culling

Povrch planety je kulový, proto pozorovatel nemá možnost vidět celou planetu z jednoho místa. Část je k pozorovateli vždy odvrácená. Není ji tedy nutné zobrazovat, protože je vždy zakryta přivrácenou stranou. Existuje několik způsobů, jak docílit identifikace, které body se nachází na odvrácené straně planety.

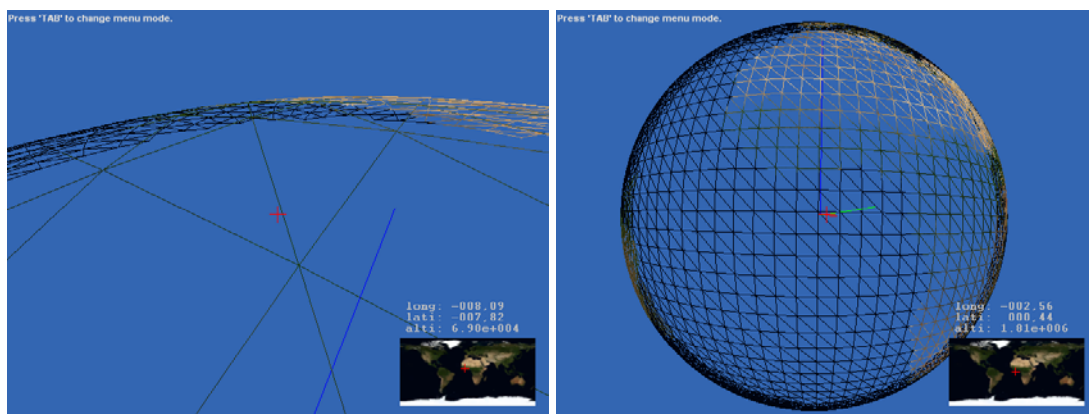
Základní, avšak nejméně přesný, je algoritmus využití ořezávací roviny. Polohu této roviny lze zvolit ve středu planety, nebo v bodech, které svírají pravý úhel mezi vektorem směřujícím k pozorovateli a středu planety. Normálový vektor plochy leží na spojnici pozorovatele a středu planety. Při použití ořezávací roviny však nebereme v úvahu výskyt vysokých pohoří, které budou ořezány, pokud se vyskytují za danou plochou, ale jsou z polohy pozorovatele stále viditelné.

Další možností je využít podobný princip jako ořezání odvrácených trojúhelníků, které se využívá v grafických kartách. Ovšem v tomto případě by se nejednalo o trojúhelníky, ale o celá políčka povrchu. Tato políčka by byla testována, zda nejsou odvrácena od pohledu uživatele. Tento způsob je velmi rychlý. Avšak opět nebere do úvahy vysoká pohoří. Částečně je možné tomuto jevu předejít, pokud snížíme práh pro rozhodování, zda bude políčko oříznuto, či nikoliv.

Mnohem spolehlivější způsob, jak brát v úvahu i vysoká pohoří, je využití již zmiňované mezní kulové plochy obklopující povrchová políčka planety. Vzájemně porovnáváme vždy dvě koule. Jsou jimi planeta a mezní kulová plocha konkrétního políčka. Dále je třeba znát polohu pozorovatele. Od pozorovatele vedeme ke každé z koulí kužel. Plášť kužele se dotýká koule v bodech, které z pohledu pozorovatele tvoří hraniční oblast koule. Z daného principu vyplývá, že středový vektor prochází středem koule. Algoritmus pracuje s využitím úhlu, který svírá plášť a středový vektor daných kuželů (označíme α pro kužel protínající planetu a β pro kužel protínající mezní kulovou plochu políčka). A také s úhlem γ , který svírají středové vektory kuželů. Platí vztah, že pokud $\gamma < \alpha - \beta$ je políčko ořezáno (viz obrázek 3.6 a obrázek 3.7). Tato metoda však neuvažuje se vzdáleností k pozorovateli. Ořízne i políčka, která jsou na přivrácené straně planety a splňují výše uvedenou nerovnici. Algoritmus je nutné doplnit o plochu, jejíž normálový vektor směřuje od středu planety k pozorovateli a prochází body, které z pohledu pozorovatele tvoří hraniční oblast koule. Tato plocha určuje, které z původně ořezaných políček patřily k odvrácené a které k přivrácené straně planety. Jako zdroj informací byl použit článek věnující se horizon culling [7].



Obrázek 3.6: Ořezání horizontu (vlevo není za horizontem, vpravo je za horizontem)



Obrázek 3.7: Ořezání horizontu v aplikaci

4 Využití grafických karet

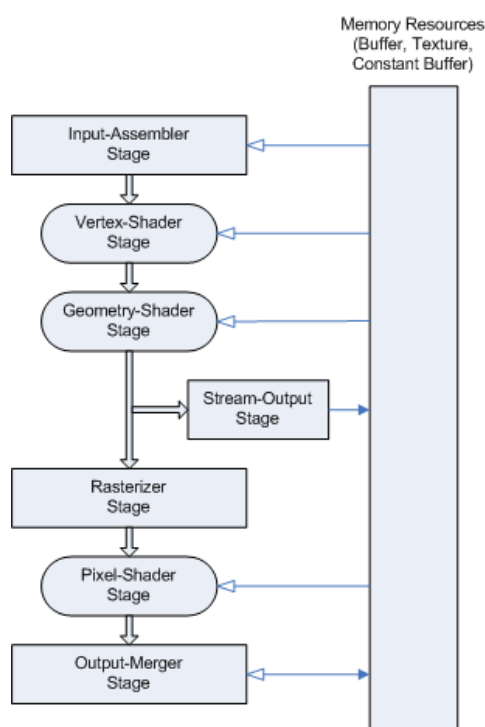
V současnosti jsou grafické adaptéry hojně využívány grafickými aplikacemi. Je však nutné mít k dispozici prvky, které lze naprogramovat, aby splnily požadované operace, jakmile data, které jsou zde reprezentována grafickými primitivami, vstoupí do grafické karty. Tyto prvky se nazývají shadery a jsou součástí programovatelné linky (anglicky *programmable pipeline*), kterou každá primitiva musí projít, než bude zobrazena, či s ní bude jinak naloženo.

Tato kapitola popisuje strukturu programovatelné linky. Podrobněji se věnuje shaderům, kde krátce charakterizuje funkce, které plní.

4.1 Programovatelná linka

Programovatelná linka je pojem, který se užívá v grafických API jako je DirectX, či OpenGL. V těchto grafických API se vertex, geometry a pixel shadery popisují konsistentní sadou instrukcí, i když jednotlivé shadery plní rozdílné funkce. Tento způsob se nazývá Sjedený Model Shaderů (anglicky *Unified Shader Model*). V DirectX 10 je Sjedený Model Shaderů známý pod pojmem Shader Model 4.0. Shadery jsou zde programované pomocí vysokoúrovňového jazyka, který se nazývá High Level Shader Language (zkráceno HLSL). Základním programovacím jazykem shaderů je assembler, který se skládá z jednotlivých instrukcí. Postupem času však byly vyvinuty vysokoúrovňové kompilované jazyky, které umožňují zjednodušit a zpřehlednit programování shaderů dostupných v programovatelné lince.

Programovatelná linka v DirectX 10 (viz obrázek 4.1) se dělí na několik stádií, kterými postupně procházejí data, než jsou vykreslena. Jako zdroj informací byl použit článek popisující programovatelnou linku na internetových stránkách Microsoft Developer Network [8].



Obrázek 4.1: Programovatelná linka DirectX 10, převzato z Microsoft Developer Network [8]

- Input-Assembler - Stará se o příjem geometrických dat.
- Vertex shader - viz kapitola 4.2.1
- Geometry shader - viz kapitola 4.2.3

- Stream-Output - Geometry shader přináší možnost přesměrovat geometrické primitivy z programovatelné linky na výstup. Nebudou se tedy zpracovávat dále grafickou kartou a je možné je využít pro další účely později. Pro tento účel slouží právě Stream-Output.
- Rasterizer - Provádí ořezání primitiv, které nebudou vykresleny. Připravuje data pro pixel shader.
- Pixel shader - viz kapitola 4.2.2
- Output-Merger - Generuje finální datový výstup z programovatelné linky.

4.2 Shadery

Jedná se o sadu instrukcí, které flexibilně provádějí operace za pomoci hardwaru na grafických kartách. V současné době jsou jejich nedílnou součástí. Programuje se s nimi programovatelná linka, kde zajišťují matematické výpočty, transformace bodů, světelné efekty, nanášení textur a další. V současné době existují tři druhy shaderů. Jsou jimi vertex, pixel a geometry shader. Ovšem dříve tomu tak nebylo. Shadery se vyvíjely společně s grafickými kartami. Jako první byly programovatelné vertex shadery, které pracovaly s jednotlivými body geometrických primitiv. V další generaci přibýly pixel shadery pracující s jednotlivými pixely. A v poslední generaci karet se objevily geometry shadery, které se konceptuálně od předchozích dvou shaderů výrazně liší. Nepracují totiž s jednotlivými body či pixely, ale pracují s grafickými primitivami jako celkem.

4.2.1 Vertex shader

Vertex shadery pracují s jednotlivými body grafických primitiv. Jejich úkolem je transformace bodů z 3D pozice v prostoru na 2D pozici na obrazovce. Mohou měnit jejich pozici, dále také barvu, souřadnice textur a další. V novějších verzích je možné i přistupovat do vlastních textur, což bylo dříve pouze možné v pixel shaderech. Nelze body přidávat, ani odebírat. Jelikož jejich zpracování probíhá v grafické kartě paralelně, lze je využít i pro náročnější matematické výpočty, které budou probíhat mnohem rychleji, než kdyby je prováděl procesor.

Vertex shadery se nejčastěji využívají pro deformace, animace a vytváření realistických povrchů. Existuje mnoho možností jak využít vertex shadery.

4.2.2 Pixel shader

Pixel shadery se hlavně zaměřují na práci se světly, barvami a texturami. Nepracují s jednotlivými body, ale s pixely. O texturování se starají texturovací jednotky, které nanášejí textury s velmi vysokým výkonem. Jsou spouštěny ve větším měřítku než vertex shadery, pracují totiž s pixely, nikoliv s body. Je proto vhodnější přenechat složitější výpočty vertex shaderům, které jsou k těmto účelům přizpůsobeny.

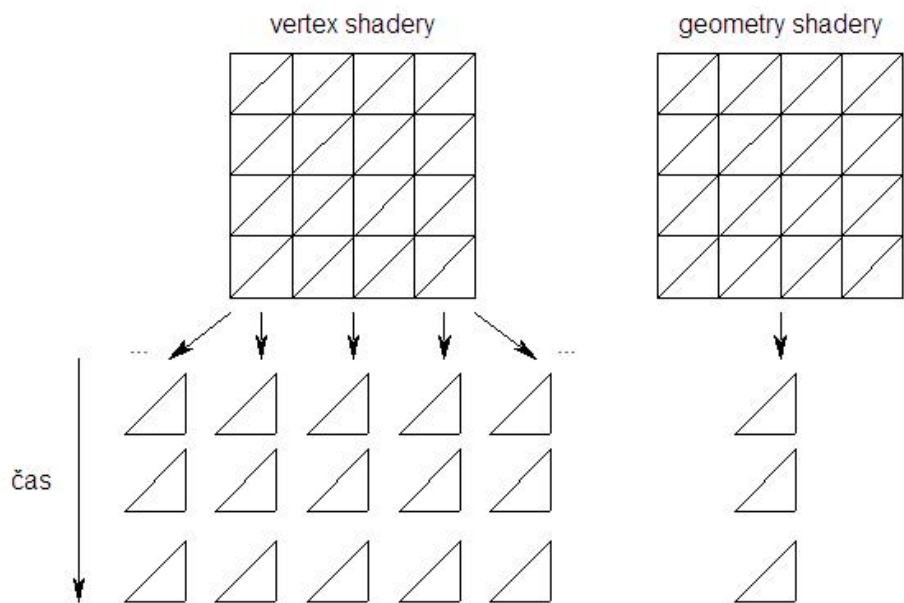
Používají se při stínování, osvětlování, bump-mappingu a dalších efektech.

4.2.3 Geometry shader

Nejnovější součástí programovatelné linky grafických karet jsou geometry shadery. Pracují přímo s grafickými primitivami. Nejčastěji jimi jsou trojúhelníky, dále také mohou pracovat s úsečkami, či jednotlivými body. Geometry shadery mohou upravovat počet vykreslených primitiv. Mohou množství redukovat, nebo naopak přidávat.

Platí jistá omezení, ať již ze strany výkonu, tak i implementace. Pokud chceme použít geometry shader na zvyšování detailů přidáváním nových primitiv, jsme silně omezeni maximálním počtem bodů, které může geometry shader emitovat. Limit je nastaven na 1024 bodů. Pro větší počet nebyl bohužel navržen. Jedinou možností je několikanásobný průchod primitiv programovatelnou linkou. Tato možnost je ale silně neefektivní. Další nevýhodou je sériové zpracování primitiv vytvořených geometry shaderem (viz obrázek 4.2). Nebude tedy možné plně využít výkon grafických karet založený na paralelním zpracování.

I přes svá omezení se geometry shadery dají použít pro stíny, zobrazování do krychle, či vytváření bodových spritů.



Obrázek 4.2: Průchod dat shadery (vlevo procházejí paralelně skrze vertex shader, vpravo jsou emitované pomocí geometry shaderu, vychází z něj pouze sériově pouze sériově)

5 Návrh

Než budeme aplikaci navrhovat, je vhodné si ujasnit, co je jejím cílem. Aplikace musí být schopna v reálném čase zobrazit planetu, jejíž data máme uložena v souborech na harddisku. Uživatel by měl být schopen ovlivňovat základní parametry zobrazení. Zejména možnost nastavovat koeficienty pro velikost planety, výšku pohoří, úroveň detailu, zdroje vstupních dat.

Nejprve bude vhodné navrhnout vlastní systém zobrazování úrovní detailu, který vychází z metody GeoMipMapping. Při návrhu tohoto algoritmu využijeme programovatelnou linku. Z tohoto důvodu je nutné vyřešit načítání dat z harddisku a jejich nahrání do paměti grafické karty.

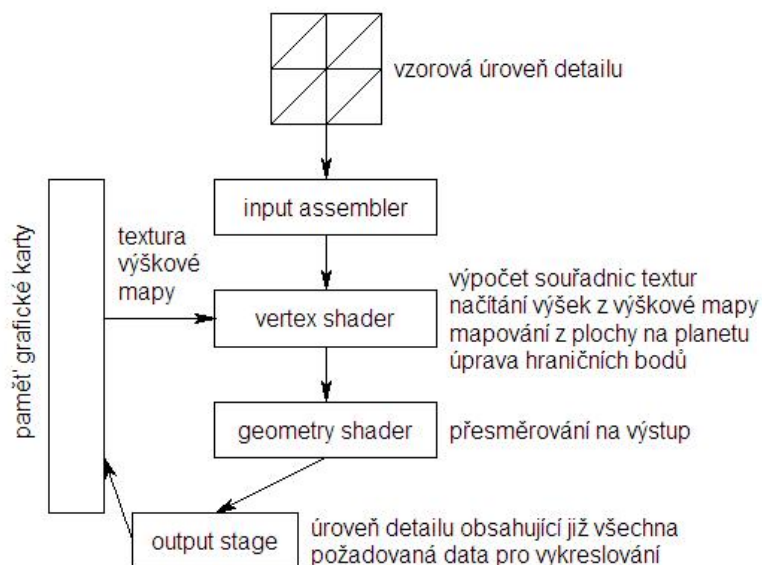
Tato kapitola se nejprve zabývá návrhem vlastního algoritmu pro zobrazování úrovní detailu. Poté se zaměří na návrh řešení ostatních částí programu. Mezi ně patří efektivní načítání dat a využití ořezávacích metod horizon a frustum culling.

5.1 Vlastní algoritmus

Algoritmus vychází z metody GeoMipMapping (viz kapitola 3.1.2). Celou planetu bude reprezentovat pole jednotlivých políček. Jak bude dané pole velké, záleží na množství vstupních dat. Každé políčko bude mít vlastní úroveň detailu.

Pro tvorbu dat, které budou zobrazovány jako úsek povrchu planety, využijeme grafickou kartu. Abychom maximalizovali výpočetní rychlost, použijeme vertex shadery. Vertex shader bude při výpočtech přistupovat do textury, která bude obsahovat hodnoty z výškové mapy pro konkrétní úroveň detailu. Je vhodné po ukončení výpočtu úrovně detailu políčka data uložit pro následné několikanásobné zobrazování, abychom je nemuseli pokaždé znovu zbytečně přepočítávat. K tomuto účelu lze využít geometry shader, který má schopnost přesměrovat data z programovatelné linky do paměti grafické karty. Zde budou v neměnné podobě uloženy pro další využití. Výstupní data mají stejnou strukturu jako vstupní.

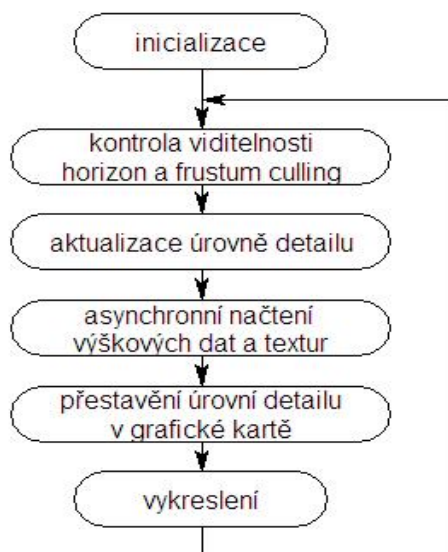
Vertex shader bohužel na rozdíl od geometry shaderu neumí amplifikovat množství geometrických primitiv v grafické kartě jejich dělením. Proto je nutné, aby uspořádání dat již bylo v požadované úrovni detailu při vstupu do programovatelné linky. Tento problém lze vyřešit vygenerováním všech „vzorových“ úrovní detailu během startu aplikace. Do těchto vzorových políček se pak všechna potřebná data nanesou až při průchodu přes vertex shader (viz obrázek 5.1). Políčka je poté obdrží z grafické karty již s osazenými výškovými daty.



Obrázek 5.1: Algoritmus generování nové úrovně detailu pro políčko

5.2 Aplikace

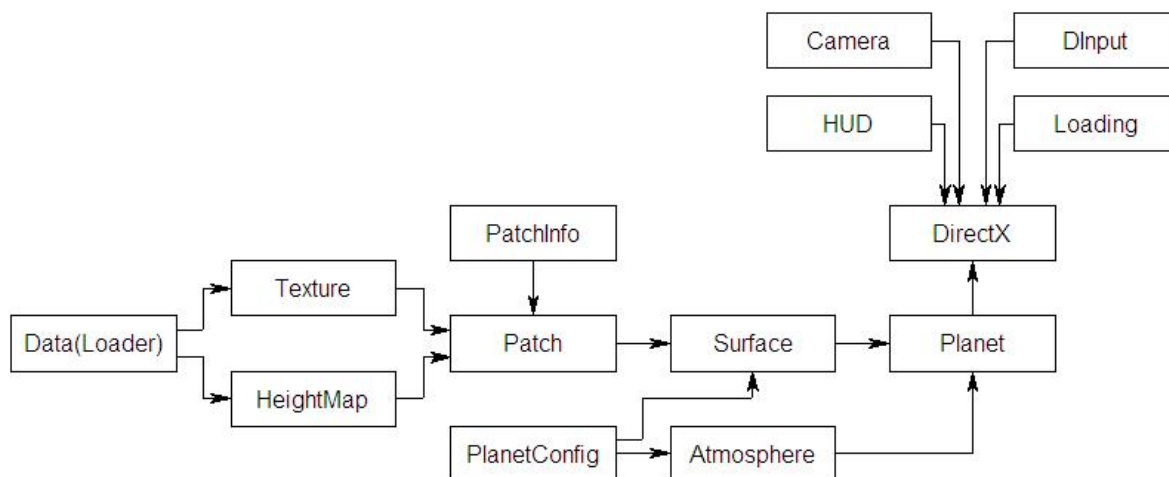
Zjednodušené schéma hlavního programu (viz obrázek 5.2) zobrazuje její hlavní logiku. Po spuštění se zinicilizují všechny potřebné informace a datové struktury, včetně již zmíněných „vzorových“ políček. Jakmile program vstoupí do hlavní smyčky, musí se každé políčko zkontrolovat, zda se nachází na odvrácené straně planety, či je mimo uživatelem pozorovaný prostor. Pokud není, musí se aktualizovat jeho současná úroveň kvality. Nastala-li změna, je nutné, aby jeho tvar byl změněn. Zdrojová data planety budou načítána asynchronně. Jakmile jsou data připravena, dojde k přestavění úrovně detailu políčka. Přestavování se provádí s využitím algoritmu popsaného v kapitole 5.1. Aby však nedocházelo k viditelným mezerám mezi políčkem, jehož úroveň kvality se změnila, a jeho sousedy, musí se také změnit hranice sousedů. Na závěr jsou všechna viditelná políčka vykreslena.



Obrázek 5.2: Zjednodušené schéma hlavního programu

6 Implementace

Jako programovací jazyk je zvolen C++. Pro práci s grafickou kartou je použita knihovna Direct3D 10, která je součástí balíku DirectX od společnosti Microsoft. Z hlediska použití Direct3D 10 je aplikace určena pro platformu Windows Vista a novější. Knihovna DirectX 10 byla zvolena z důvodu podpory geometry shaderů, které se objevily až v poslední řadě grafických adaptérů.



Obrázek 6.1: Schéma uskupení tříd v aplikaci

6.1 Příprava zdrojových dat

Zdrojová data z hlediska využití v aplikaci musí splňovat předem dané požadavky na velikost a formát. Proto je nutné data nejprve zpracovat. Výšková mapa a textura se zpracovávají každá zvlášť, protože jejich příprava vyžaduje odlišné způsoby řešení. Protože popisujeme planetu, musíme velikost vstupní výškové mapy i textury upravit. Implementace počítá s výškovou mapou a texturou o jiné velikosti než GeoMipMapping (viz rovnice 3.1). Aby nevznikala mezera mezi polokoullemi planety na 180. poledníku, musíme spojit koncové body. Proto je šířka výškové mapy a textury požadována o jeden sloupec menší (viz rovnice 6.1). Při zpracování chybějící poslední sloupec čerpá hodnoty z prvního a tím je zajištěna shoda hodnot. Výška zůstává nezměněna.

$$x' = x \cdot 2^n$$

$$y' = y \cdot 2^n + 1$$

Rovnice 6.1: Upravená rovnice pro velikost zdrojových dat.

6.1.1 Soubory výškové mapy

Výškovou mapu je potřeba nejprve upravit, aby splňovala požadavky na velikost (viz rovnice 6.1). Výšková mapa dosahuje velikosti i několik GB, proto je vhodné navrhnout efektivní způsob zpracování. Pokud velikost souboru přesahuje maximální hodnotu 32 bitového celého čísla bez znaménka (přibližně 4GB), nelze pro jeho načtení použít standardní funkce. Je nutné použít jiný postup.

V rané fázi implementace aplikace se pracovalo s výškovou mapou jako celkem. K tomuto účelu byly využity funkce, které umožňují mapovat část obsahu souboru bez nutnosti, aby byla načtena z harddisku do operační paměti. Pracujeme tedy s údaji na harddisku, nikoliv s údaji uloženými v operační paměti, jak je to u ostatních aplikací obvyklé. Výhodou tohoto přístupu je práce s výškovou mapou jako celkem, není nutné soubor nijak rozdělovat. Chceme-li získat data pro políčko, které se nachází na řádce „Y“ a jeho pořadí je „X“, musíme nejprve načíst celý řádek „Y“.

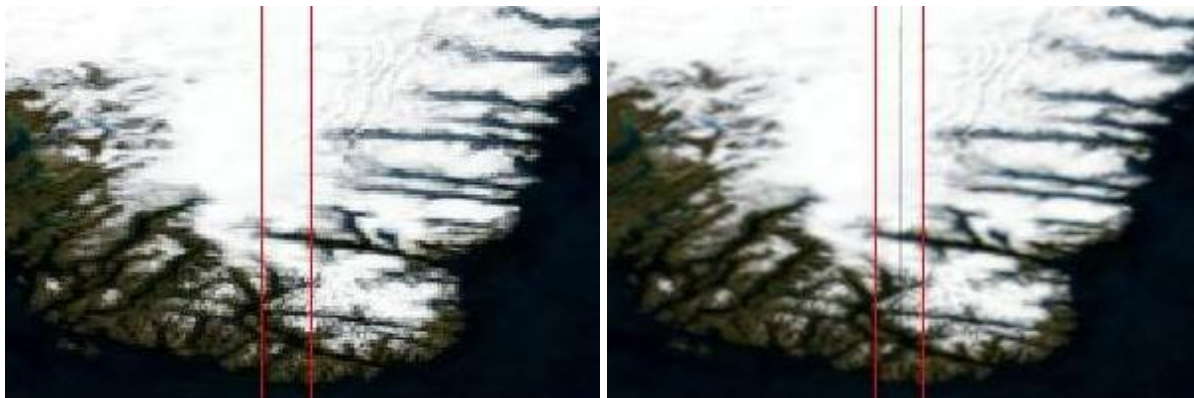
Následně je nutné vyextrahovat jen data, která jsou potřeba jen pro dané políčko „X“. Extrakce dat však není sekvenční, musí se přeskakovat data jiných políček, proto dochází ke ztrátě výkonu. Zbytek řádkových informací načtených z výškové mapy lze z hlediska optimalizací využít pro ostatní políčka, ovšem většina dat byla načtena zbytečně. Při použití této metody jsou komplikací políčka, která jsou poslední na každém řádku. Jelikož při použití metody GeoMipMapping každé políčko sdílí hraniční body se sousedy, je nutné, aby poslední políčka měla k dispozici informace o hraničních bodech políček, která jsou na začátku stejného řádku.

Z hlediska načítání celých řádků a nesequenčního přístupu k datům výškové mapy však způsob zmíněný v předchozím odstavci nevyhovoval požadavkům na efektivitu načítání dat. Proto byl navržen poněkud odlišný koncept. Výšková mapa je rozdělena na jednotlivá políčka. Každý soubor obsahuje data pro jedno políčko včetně hraničních bodů se svými sousedy. Pokud tedy načítáme data z jednotlivých souborů, jsou vždy načítána sekvenčně. Dojde tak ke zlepšení výkonu. Jistou nevýhodou je fakt, že jsou některá data uložena několikrát. Avšak jejich počet je z hlediska velikosti celé výškové mapy zanedbatelný. Jelikož velikost souborů políček zcela jistě nepřesáhne maximální hodnotu 4GB, lze použít standardní funkce pro načítání souborů. Není potřeba ošetřovat speciální případy v koncových oblastech řádků, které nastávají v metodě popsané v předchozím odstavci, protože každé políčko má všechna potřebná data ve svém vlastním souboru.

6.1.2 Soubory textury

Velikost textury v bodech vždy nemusí odpovídat velikosti výškové mapy, i když ve většině případů tomu tak je. Textura pro danou výškovou mapu může mít mnohem kvalitnější rozlišení. Je však vhodné ji také rozdělit na logické celky. Při tvorbě logických celků je však potřeba vzít v úvahu i následnou komprimaci po jejich načtení do aplikace. Komprimační algoritmy dostupné v Direct3D vyžadují velikost textur $2^n \times 2^n$, proto bude vhodné z hlediska ušetření paměti tento požadavek respektovat. Textura se na jednotlivá políčka nanáší pomocí souřadnic, které mají rozsah $<0,1>$ v **R**. Není nutné, aby velikosti políček výškové mapy a textury byly shodné.

Jako nejlepší se jeví možnost rozdělení textury na logické části, které se kryjí políčky výškové mapy. Textury sdílí hranice se sousedními texturami obdobně, jak to bylo u políček výškové mapy. Daný způsob poskytuje jednoduchou správu dat pro aplikaci, protože logické celky odpovídají jedna ku jedné. Problém nastává, pokud chceme využít kvalitnější filtrování textur pro dosažení lepších vizuálních výsledků. Využívají okolní body, aby byly dosaženy mnohem kvalitnější výsledky, než při užití základního filtrování. Avšak hraniční body jednotlivých políček nemají své okolní body kompletní. Využijí se body z druhé strany políčka. Při užití kvalitnějších filtrování tedy vznikají na hranicích mezi políčky viditelné hranice, jelikož textury políček samozřejmě nejsou symetrické (viz obrázek 6.2).

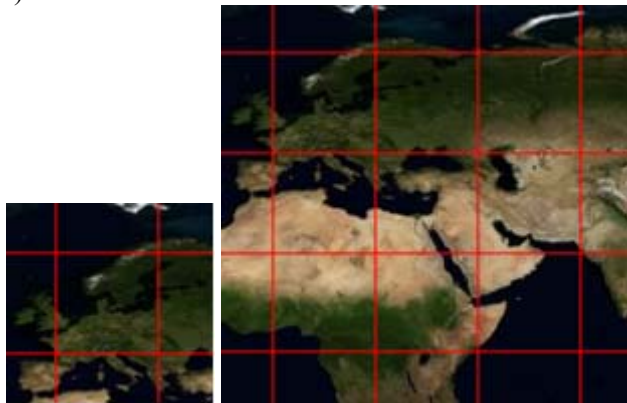


Obrázek 6.2: Viditelná hranice při použití kvalitnějšího filtrování (vlevo základní, vpravo anizotropní)

Tento problém lze řešit rozšířením přesahu mezi jednotlivými políčky. Je nutné zachovat velikost, kterou požadují komprimační algoritmy. Pokud použijeme metodu popsanou v předchozí kapitole společně s přesahem, velikost políček textury se zvýší přibližně 4krát (viz obrázek 6.3).

Z tohoto důvodu nelze metodu, kde si celky výškové mapy a textury odpovídají jedna ku jedné, použít.

V implementaci programu si logické celky vzájemně neodpovídají. Každé políčko textury obsahuje několik políček výškové mapy. I tento způsob však musí zajistit přesahy mezi políčky, aby nevznikaly viditelné hranice. Máme-li například texturu o velikosti 2048x2048 bodů a jednotlivá políčka výškové mapy mají velikost 513x513, jedna textura obsahuje 3x3, tedy 9 políček. Zbytek tvoří okraje, které pomáhají předcházet vizuálním problémům s hranicemi mezi sousedními texturami. Vezmeme-li v úvahu dané rozložení, nárůst velikosti jednotlivých textur je přibližně 1,77krát (viz obrázek 6.3).



Obrázek 6.3: Jednotlivé logické celky textur planety (vlevo pokrývající jedno políčko (nárůst 4x), vpravo 9 políček (nárůst 1,77x))

6.1.3 Pomocný soubor

Aby aplikace nemusela při každém spuštění procházet všechny soubory výškových map za účelem získání základních dat, je použitý pomocný binární soubor. Jeho účelem je uchovávat pouze nejdůležitější, popř. předem vypočítaná data, a zrychlit tak start aplikace. Všechny soubory výškové mapy mají informace v pomocném souboru. Uchovávají se výšky čtyř rohů políčka, výška středního bodu, maximální výška a také předem vypočítaná data pro metodu výběru úrovně detailu (viz kapitola 3.2).

6.2 Načítání zdrojových dat

Systém uložení dat navržený v kapitole 6.1 zjednodušuje práci v rámci jejich následného načítání. Není nutné řešit nesequenční načítání v jednom souboru. Každé políčko má vlastní soubor, který mu poskytuje veškerá potřebná výšková data. Pokud je třeba, aplikace tedy načte data, která se vztahují k pouze danému políčku, včetně hranic s jeho sousedy. Aplikace se nezabývá velikostí větší než 4GB u jednotlivých souborů políček. Průměrná velikost jednotlivých políček se totiž pohybuje maximálně v desítkách MB.

Jelikož aplikace načítá kvalitnější data za běhu programu, nikoliv ihned po spuštění, je potřeba řešit nízkou rychlost odezvy harddisku. Pokud bychom data načítali sekvenčně v hlavní smyčce programu, aplikace by neběžela plynule. Z tohoto důvodu se využívá paralelismus pomocí vláken (anglicky *threads*). Vlákna umožňují aplikaci načítat velké množství data, aniž by byla na rychlosti jejich načítání přímo závislá.

Výšková data jsou uložena v čisté binární podobě (ve formátu RAW). Nelze je zpracovat za pomoci funkcí, kterými disponuje knihovna DirectX. Pokud políčko mění úroveň detailu, vznesou požadavek potřeby nových dat. Daný požadavek je uložen do fronty, kde se nachází i požadavky ostatních políček a čeká na vyřízení. Hlavní aplikace nečeká do doby, než je požadavek vybaven, ale pokračuje v běhu. Z toho plyne, že políčko neobdrží ihned svá data. Musí se přizpůsobit čekací době. Obvykle se použije poslední úroveň detailu, která byla zpracována. Pokud se však políčko předtím nacházelo na odvrácené straně planety, nemá data předchozí úrovně detailu k dispozici. Pro tento účel se použije základní, který se skládá pouze ze čtyř bodů. Tyto body jsou k dispozici již od startu aplikace, jelikož jsou načtena pomocného souboru (viz kapitola 6.1.3). Mezitím co běží hlavní

aplikace, vlákno načte výšková data. Jakmile při dalším průchodu opět políčko vznesе požadavek, data jsou již k dispozici. Pokud ne, opakuje se stejná situace znovu. Načtená data se uvolní, jakmile se políčko opět nachází za horizontem planety.

Textury se načítají odlišným způsobem než výškové mapy. Soubory textur jsou uloženy v obrázkovém formátu (například formát PNG). Dané formáty se na rozdíl od výškových map již dají načítat pomocí knihovních funkcí poskytovaných DirectX. K načítání textur z obrázků slouží funkce *D3DX10CreateTextureFromFile*. Ovšem tato funkce pracuje sekvenčně. Knihovna DirectX 10 na rozdíl od svých předchůdců je však navržena pro nativní podporu vícevláknových aplikací. K tomuto účelu slouží rozhraní *ID3DX10ThreadPump*, které zprostředkovává funkci vláken. Pokud použijeme objekt tohoto rozhraní jako parametr funkce *D3DX10CreateTextureFromFile* její chování se změní na asynchronní.

Samotný postup je podobný načítání výškových dat. Políčko požádá o data k textuře. Zavolá se funkce *D3DX10CreateTextureFromFile*, která začne paralelně načítat data ze souboru. Políčko se v každém průchodu smyčkou v hlavním programu dotazuje, zda jsou data ze souboru textury načtená a připravená v paměti. Velikost textur nesouhlasí s velikostí jednotlivých políček, proto je nutné zajistit, že se daná textura bude načítat pouze jednou pro všechna políčka, která o ni současně žádají. Jakmile se všechna políčka, které daná textura pokrývá, nachází za horizontem, jsou její data uvolněna z paměti.

Jelikož jsou textury načítány paralelně během programu, nejsou ihned k dispozici. Musíme chybějící texturu na omezenou dobu nahradit. Pokud tak neučiníme, na daném místě, kde se má textura nacházet, bude chybět. Proto je vhodné použít texturu mnohem menšího rozlišení, kterou lze použít jako dočasnou náhradu, než jsou potřebná data k dispozici. Snížíme tak rušivý vliv chybějící kvalitní textury na minimum.

6.3 Vlastní algoritmus

Již při inicializaci aplikace je znám počet úrovní detailů, které jsou použity pro zobrazení políček. Jednotlivé úrovně detailů jsou značeny pomocí posloupnosti čísel. Předpokládejme velikost políčka například 513x513 bodů. V tomto případě je podle rovnice (viz rovnice 3.1) celkový počet úrovní detailů (a tedy i „vzorových“ políček) devět. Označíme je od nejkvalitnější, která je označena hodnotou „nula“, až po nejméně kvalitní, která má hodnotu „devět“. Při tvorbě políček se používá číslo 2^n , kde n je hodnota úrovně detailu. Určuje, s jak velkým přírůstkem generujeme jednotlivé body na ose x i na ose y . Pro nejkvalitnější úroveň detailu je tedy přírůstek 2^0 , zatímco pro nejméně kvalitní 2^9 .

Při inicializaci se vygenerují všechny vzorové úrovně detailů, které se poté uloží pro další zpracování. Tato vzorová políčka obsahují pouze jednotlivé body, jejichž počet odpovídá požadavkům jednotlivých úrovní detailu. Samostatné body kromě indexů „X“ a „Y“, na kterém řádku a sloupci se v políčku nachází, nenesou žádné další informace.

Při požádání políčka o změnu úrovně detailu na jinou se výšková mapa načte do paměti. Data vzorového políčka, které reprezentuje shodnou úroveň detailu, jsou poslána ke zpracování do programovatelné linky grafické karty.

6.3.1 Výpočet souřadnic bodů

Pomocí indexů, které identifikují jednotlivé body, lze ve vertex shaderech dospět až k jejich umístění. Pokud jsou známy indexy levého horního bodu v rámci celé planety, přičtením indexů jednotlivých bodů získáme celkový index „X“ a „Y“ u všech bodů políčka v rámci planety. Indexy se převedou na geografické souřadnice, ze kterých získáme jejich polohu na planetě (viz kapitola 3.3). K poloměru planety se musí přičíst aktuální výška, která je získána z hodnoty výškové mapy na daných souřadnicích „X“ a „Y“. O výslednou hodnotu se nakonec souřadnice posunou od středu planety.

Před výpočtem pozic bodů je však nutné brát v úvahu možnost rozdílné úrovně detailu mezi sousedními políčky. Pokud souseď právě zpracovávaného políčka je zobrazen v nižší úrovni detailu, je nutné změnit indexy hraničních bodů (viz kapitola 3.1.2). K identifikaci lze použít jejich indexy. Každý hraniční bod je vždy jeden z indexů nulový, nebo o maximální hodnotě. Indexy všech

hraničních bodů s daným sousedem jsou upraveny tak, aby došlo k jejich posunutí k nejbližšímu indexu, který se objevuje v sousedním políčku s méně kvalitní úrovní detailu.

6.3.2 Výpočet souřadnic textur

Je třeba zajistit správné nanášení textur. K výpočtu texturovacích souřadnic dochází ve vertex shaderech ihned po výpočtu jejich umístění na povrchu. Z důvodů, které jsou uvedeny v kapitole 6.1.3, je nutno pracovat se dvěma rozdílnými texturovacími souřadnicemi. Jedna slouží pro dočasnou nekvalitní texturu a druhá pro texturu, která dosahuje již dostatečného rozlišení. Při výpočtu souřadnic lze opět vycházet z indexů jednotlivých bodů v rámci planety. Dočasná textura pokrývá celou planetu, proto pro výpočet souřadnic stačí mít k dispozici index bodu na planetě a celkový počet bodů výškové mapy planety. Kvalitnější textury však nepokrývají celou planetu, ale vždy pouze její část. K identifikaci texturovacích souřadnic, kde se levý horní bod políčka v textuře nachází, je použita rovnice 6.2.

$$\text{souřadnice_X-Y} = (\text{pozice_políčka_X-Y} \% \text{počet_políček_na_řádku_v_textuře}) * \\ ((\text{velikost_políčka} - 1) / \text{velikost_textury}) + \\ ((\text{velikost_políčka} - 1) / 2) / \text{velikost_textury}$$

Rovnice 6.2: Rovnice k identifikaci, kde se nachází v textuře levý horní bod.

Určí se souřadnice políčka v textuře. Následuje výpočet pozice prvního bodu v textuře. Protože však textura má okolo obsažených políček hranici, která má přesně velikost jedné poloviny políčka, musíme vypočtenou pozici prvního políčka zvýšit o tuto hranici. Uvedená rovnice platí jak pro souřadnici „X“, tak i pro „Y“.

6.4 Výběr jednotlivých úrovní detailu

Jelikož aplikace využívá zobrazení planety pomocí několika úrovní detailu, musíme mít možnost určení, v jaké kvalitě se všechna políčka v daný okamžik zobrazí. Součástí implementace jsou dva způsoby výběru úrovně detailu políček.

6.4.1 Pomocí chyby v pixelech na obrazovce

Tento způsob vychází z algoritmu popsaného v kapitole 3.2. Nejprve je nutné předem vypočítat všechny výškové chyby pro každou úroveň kvality jednotlivých políček. Jelikož se jedná výpočetně o velmi náročnou operaci, proto ji není vhodné provádět při každém spuštění aplikace. Proto jsou tyto údaje předem vypočítané a uloženy v pomocném souboru (viz kapitola 6.1.3).

Po načtení je každá z chybových výšek využita jako referenční výška středního bodu políčka pro danou úroveň detailu. Referenční výšky se budou srovnávat s reálnou výškou středního bodu. Všechny střední body budou převedeny na kartézské souřadnice planety. Před každým výběrem vhodné úrovně detailu se musí všechny body převést pomocí transformačních matic do pohledu uživatele. Zde vypočteme chyby středních bodů jednotlivých úrovní detailů od středního bodu s reálnou výškou. Tyto vzdálenosti poté slouží jako jednotlivé chyby v pixelech z pohledu uživatele. Poté procházíme od nejnižší úrovně detailu k nejvyšší, než nalezneme úroveň, kde je chyba menší než předem zvolená konstanta. Tato úroveň detailu se následně využije pro zobrazení políčka na obrazovce. Konstanta je nastavitelná uživatelem v aplikaci.

6.4.2 Pomocí vzdálenosti od středů políček

Tato metoda využívá vzdálenosti dvou bodů v kartézské soustavě souřadnic (viz rovnice 6.3). Pokud je vzdálenost nižší, než zvolená hodnota pro danou úroveň detailu, využije se tato úroveň pro zobrazení políčka.

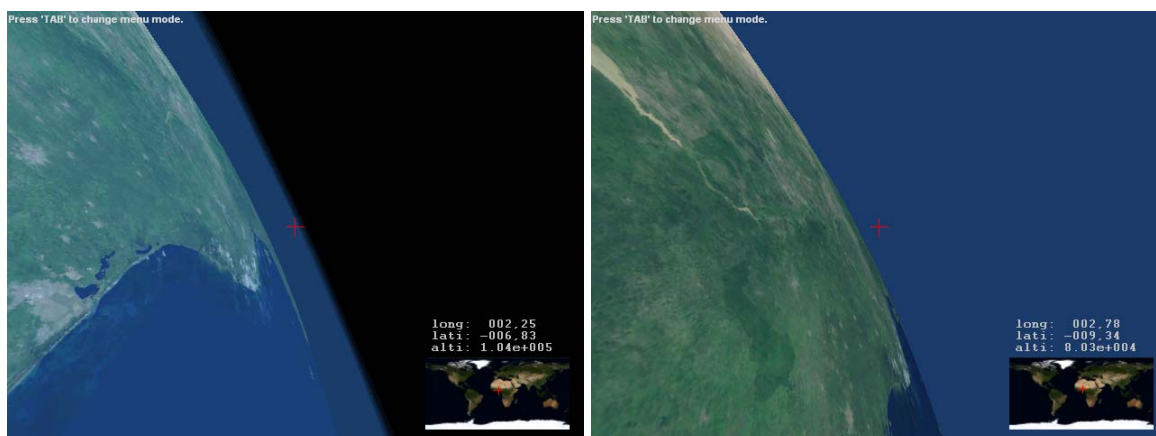
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Rovnice 6.3: Vzdálenost dvou bodů.

Aplikace bere do úvahy poloměr planety. Pokud je vzdálenost mezi políčkem a uživatelem větší než je poloměr planety, použije se nejnižší úroveň detailu. Úroveň detailu se zvyšuje lineárně se snižující se vzdáleností. Uživateli je umožněno nastavit koeficient, který ovlivňuje, v jaké vzdálenosti se budou úrovně přepínat.

6.5 Atmosféra

Součástí aplikace je i zjednodušená atmosféra. Jedná se o několik za sebou poskládaných koulí. Každá koule je průhledná. Dohromady tak tvoří dojem atmosféry o několika vrstvách. Problém však nastává, pokud bychom vletěli pod tyto vrstvy. Atmosféra by zmizela. Aplikace tento problém řeší jednoduchým způsobem. Pokud se uživatel objeví uvnitř atmosféry, pozadí se vybarví barevným odstínem atmosféry, čímž viditelně omezíme „mizející“ atmosféru (viz obrázek 6.4).



Obrázek 6.4: Atmosféra (vlevo se nacházíme nad atmosférou, vpravo uvnitř)

6.6 Omezení

Souřadnice ve scéně se u grafických karet popisují pomocí typu *float*. Hlavním omezením tohoto typu je nízká přesnost, protože se jedná o 32bitový typ s plovoucí desetinnou čárkou. Tento datový typ dokáže bez ztráty přesnosti uchovat číslo o maximálním rozsahu šesti desetinných míst. Pokud bychom chtěli zobrazit Zemi v rozlišení jednoho metru v reálné velikosti s poloměrem 6378 kilometrů, dostali bychom se na hranici přesnosti. Aplikace proto nezobrazuje planetu Zemi v plné velikosti. Velikost planety je nastavitelná uživatelem. Jelikož při menším poloměru jsou zmenšené i všechny výšky načítané z výškové mapy, aplikace umožňuje výšky několikanásobně zvýšit.

Řešením tohoto problému je použití 64bitových čísel *double* pro reprezentaci souřadnic. Bohužel je však grafické adaptéry ve velké míře nepodporují. Lze to však řešit přepočítáváním souřadnic ve scéně na souřadnice v prostoru viditelné uživatelem. O přepočítávání se stará procesor, který umí pracovat i s 64bitovými datovými typy. Tato aplikace však počítá souřadnice pomocí grafické karty. Z tohoto důvodu je čtení dat procesorem značně komplikované, protože výsledná data jsou přítomna v paměti grafické karty a nikoli v operační paměti.

7 Testování aplikace

Aplikace byla testována na sestavě o konfiguraci Intel Core2 Duo CPU E8500 3.16GHz, 4GB RAM, NVIDIA GeForce GTX 260 s operačním systémem Windows Vista Ultimate Edition. Zdrojové soubory jsou přeložené pomocí překladače dodávaného společně s vývojovým prostředím Visual Studio 2008.

Pro testování jsou použita vstupní data o různé kvalitě. Jsou označeny *nízká*, *střední* a *vysoká* kvalita (souhrnné údaje viz tabulka 7.1). Data vychází z původní výškové mapy a textury o velikosti 86400x43200 bodů. Soubory jsou zmenšeny na požadovanou velikost a rozděleny na požadované logické části aplikací za použití pomocných nástrojů.

Tabulka 7.1: Detaily zdrojových dat aplikace

Kvalita	Typ	Celková velikost [v bodech]	Počet políček	Velikost políčka [v bodech]	Velikost ¹ [v GB]
nízká	výšková mapa	16384x8193	32x16	513x513	0,26
	textura	16384x8193	11x6	2048x2048	0,13
střední	výšková mapa	32768x16385	64x32	513x513	1,00
	textura	32768x16385	22x11	2048x2048	0,51
vysoká	výšková mapa	64536x32769	128x64	513x513	4,01
	textura	64536x32769	43x22	2048x2048	1,66

Všechny testy jsou provedeny s planetou o poloměru 4 000 000m. Nejprve otestujeme výkon zobrazení zdrojových dat o různých kvalitách. Pro úroveň detailu je použita metoda založená na vzdálenosti (dále jen metoda „vzdálenosti“) s nastavenou konstantou na hodnotu 0,6 jednotek. Z tabulky (viz tabulka 7.2) jsou jasně patrné velké rozdíly mezi jednotlivými kvalitami zdrojových dat, což je z hlediska množství dat v jednotlivých úrovních detailu pochopitelné. Z hodnot získaných z *vysoké* kvality je viditelné, že nejvyšší výkon dosahuje aplikace přibližně 500km nad povrchem. Při delší vzdálenosti horizon culling propouští větší množství zobrazovaných políček. U kratší vzdálenosti je sice políček znatelně méně, avšak jejich úroveň detailu je vysoká.

Tabulka 7.2: Test rozdílu ve výkonu mezi jednotlivými úrovněmi kvalit (poloměr 4 000 000m, metoda „vzdálenosti“ s konstantou nastavenou na 0,6).

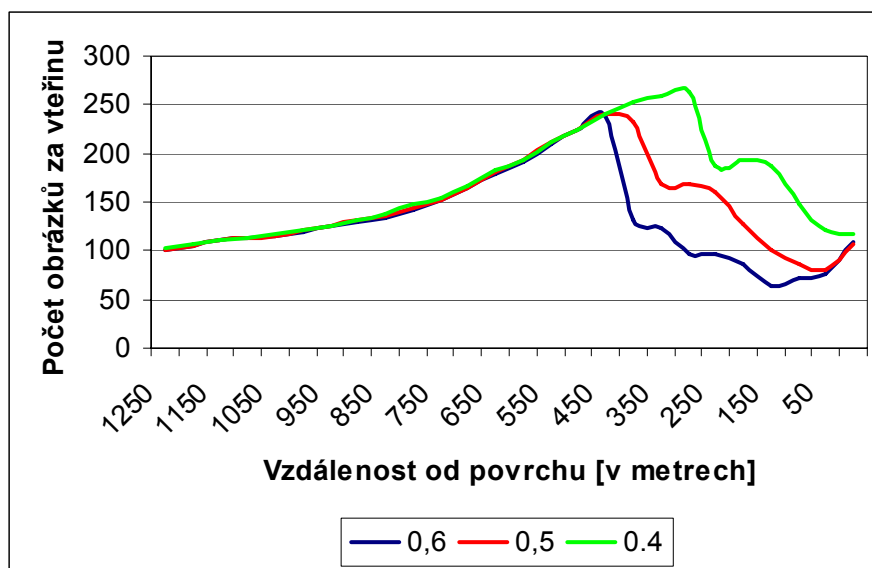
Kvalita/Vzdálenost	2000 Km	1000 Km	500 Km	250 Km	150 Km	50 Km	0 Km
nízká [FPS]	310	340	380	400	410	410	420
střední [FPS]	190	250	300	200	220	220	190
vysoká [FPS]	55	120	210	100	75	65	70

Následující testy se provádí pouze při zobrazování *vysoké* kvality.

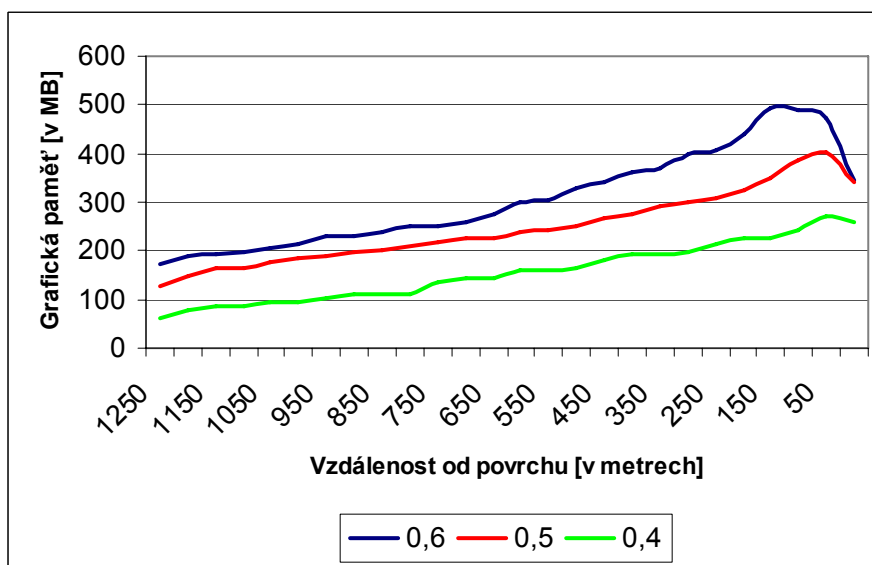
Další test se zaměřuje na nastavení úrovně detailu. Grafy (viz obrázek 7.1 a obrázek 7.2) ukazují rozdíly v obrázcích za vteřinu a ve spotřebě grafické paměti při odlišném nastavení algoritmu pro výběr zobrazované úrovně detailu políček založeného na vzdálenosti. Výsledky ukazují, že pokud chceme omezit množství spotřebované grafické paměti, stačí dostatečně snížit úroveň detailu. Využití grafické paměti pochopitelně narůstá s přibližováním k planetě, protože se zobrazují jednotlivá

¹ Výšková mapa je ve formátu .RAW, textura .PNG.

políčka ve vyšší kvalitě. Načítání textur je vylepšeno tak, že se načítají pouze pro políčka, která nejsou v nejnižší úrovni. Dojde tak k další viditelné úspoře paměti a celkovému zrychlení načítání.

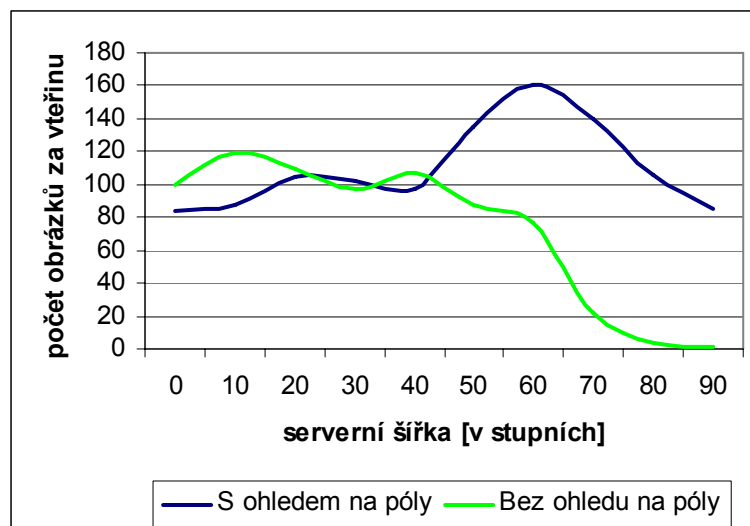


Obrázek 7.1: Vztah vzdálenosti a počtu obrázků na vteřinu při několika úrovních detailu s metodou „vzdálenosti“.



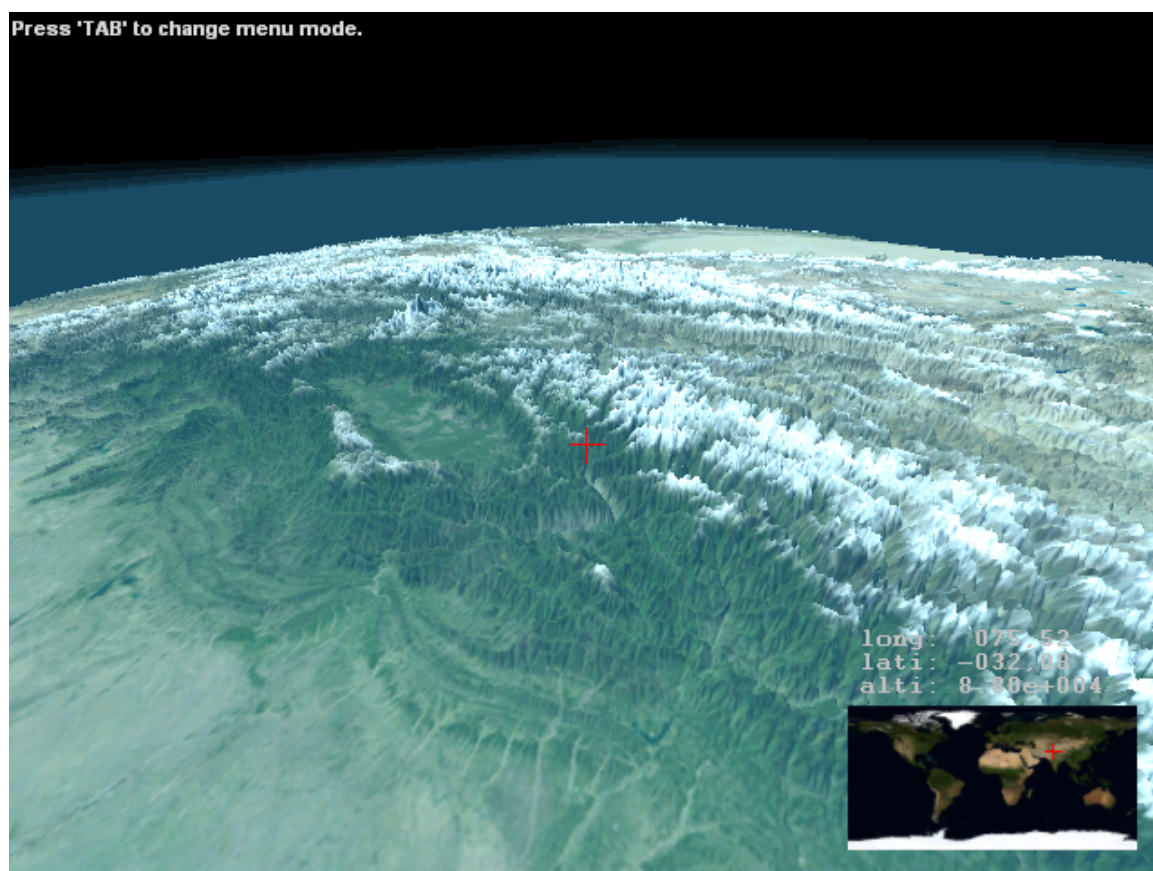
Obrázek 7.2: Vztah vzdálenosti a spotřeby grafické paměti při několika úrovních detailu s metodou „vzdálenosti“.

Jelikož jsme k mapování výškové mapy použili rovnici popsanou v kapitole 3.3.1, tvoří se na pólech mnohem vyšší hustota políček než v oblasti rovníku. Abychom předcházeli problémům, aplikace bere v úvahu při výběru úrovně detailu i pozici políčka na planetě. Na pólu se tedy zobrazuje mnohem více políček, než v oblasti rovníku, ale mají nižší úroveň detailu. Graf (viz obrázek 7.3) vyjadřuje rychlé snižování počtu obrázků za vteřinu při pohybu od rovníku k pólu (zelená křivka). Pokud bychom chtěli v budoucnu problémům tohoto typu předejít, bylo by vhodné při dalším vývoji pracovat raději s krychlí, nežli s plochou. Tímto bychom dosáhli konstantního výkonu ve všech oblastech planety, jelikož by políčka měla všude konstantní velikost.



Obrázek 7.3: Úbytek obrázků za vteřinu v oblasti pólů

Poslední test se podrobněji zabývá nastavením úrovně detailu. Kamera uživatele se nachází ve fixní pozici nad Himalájemi (viz obrázek 7.4). Postupně se mění parametry algoritmů pro výběr úrovně detailu (viz obrázek 7.5), výsledné údaje jsou zaznamenány v tabulce (viz tabulka 7.3). Tabulka obsahuje počet zobrazených trojúhelníků, spotřebu paměti, počet obrázků za vteřinu a v jakých úrovních detailu se nachází zobrazená políčka.



Obrázek 7.4: Himaláje

Tabulka 7.3: Podrobné údaje o stavu aplikace při změnách konstant metod pro výběr úrovní detailu

Zvolená metoda	Hodnota konstanty	Zobrazené trojúhelníky [v tisících]	Množství paměti ² [v MB]		FPS ³	Počet zobrazených políček v jednotlivých úrovních detailu ⁴									
			vm	tex		0	1	2	3	4	5	6	7	8	9
Vzdálenost „Distance“	0,6 -	1672	151	232	117	2	3	5	6	6	8	6	10	7	92
	0,5 -	1069	104	176	168	1	3	3	5	4	6	7	5	5	106
	0,4 -	592	72	112	232	0	4	1	3	4	4	3	3	8	115
chyba v pixelech „pixel error“	- 0,0001	15398	619	216	12	25	14	10	12	12	8	7	11	10	36
	- 0,001	3476	171	160	66	6	2	1	3	3	7	5	13	11	94
vzdálenost + chyba v pixelech	0,6 0,0001	1671	143	232	73	2	3	5	6	6	7	6	9	9	92
	0,6 0,001	1606	139	232	126	2	3	4	3	3	4	5	9	20	92
	0,5 0,0001	1069	105	176	171	1	3	3	5	4	6	7	5	5	106
	0,5 0,001	1051	102	176	157	1	3	3	3	4	3	4	7	11	106
	0,4 0,0001	592	79	112	227	0	4	1	3	4	4	3	3	8	115
	0,4 0,001	584	79	122	228	0	4	1	2	4	3	3	5	8	115



Obrázek 7.5: Způsoby výběru úrovní detailu
(vlevo pomocí vzdálenosti, uprostřed pomocí pixel error, vpravo kombinace)

Z testů vyplývá, že „chybu v pixelech“ není vhodné nechat rozhodovat o úrovni detailu bez omezení pomocí metody „vzdálenosti“. Planeta je rozdělena na políčka, ovšem vzhled povrchu planety je dosti nerovnoměrný, například Himaláje se zobrazí v nejvyšší úrovni detailu mnohem rychleji, než rovinná pouštní oblast, i když se pouštní oblast nachází mnohem blíže k uživateli.

² vm - výšková mapa, tex - textura

³ FPS - frames per second (obrázky za vteřinu)

⁴ (nejvyšší úroveň detailu) 0 <= číslo <= 9 (nejnižší úroveň detailu)

8 Závěr

Cílem této práce bylo 3D zobrazení povrchu planety ve vysoké kvalitě s pomocí grafické karty, a to v reálném čase. Aplikace je postavena na speciálně navrženém způsobu využití vertex a geometry shaderů pro tvorbu různých úrovní detailů políček povrchu planety přímo v grafické kartě.

Zadání bakalářské práce se podařilo splnit. Vytvořená aplikace je plně funkční. Občas se však projevuje přetíženost komunikační linky (PCI-e) grafické karty velkým množstvím dat, což způsobuje pokles množství obrázků za vteřinu. Tento problém lze v aplikaci vyřešit změnou nastavení parametrů metod pro výběr úrovní detailů.

Zdrojová data (výšková mapa i textura), převzatá od NASA [2], jsou použita v rozlišení přibližně 500m na jeden bod. Část práce se zabývá efektivním rozdělením těchto dat na menší soubory. Jakmile jsou data rozdělena a připravena, řeší se jejich načítání z harddisku přímo do paměti grafické karty, kde se zpracovávají. Data jsou paralelně načítána za běhu aplikace pomocí vláken. O zobrazování povrchu planety se stará metoda GeoMipMapping. Jedná se o algoritmus pro zobrazení terénu v několika úrovních detailu najednou ve vztahu k pozorovateli. Povrch planety mimo pozorovací oblast uživatele je ořezán pomocí algoritmu frustum culling. Povrch na odvrácené straně planety je ořezán pomocí algoritmu horizon culling.

Pro zlepšení vizuálního efektu je součástí aplikace i zjednodušená atmosféra.

Aplikace obsahuje i jednoduché uživatelské rozhraní, které dovoluje uživateli měnit parametry metod pro výběr úrovní detailů přímo za běhu aplikace.

8.1 Budoucí vývoj

V aplikaci je možné vylepšit mapování zdrojových dat, kdy by se použilo uspořádání pomocí krychle místo plochy, protože testy ukázaly, že mapování z plochy na kouli není velmi vhodné z hlediska měnící se hustoty políček na planetě směrem k pólům. V aplikaci je tento problém vyřešen způsobem, kdy při výběru úrovně detailu políčka bereme do úvahy i jeho vzdálenost od pólu. Migrace na mapování z krychle na kouli by tento problém zcela vyřešilo. Bylo by však nejprve nutné vyřešit převod plošných dat.

Dále by bylo vhodné se věnovat podrobněji atmosféře, která by velkým dílem vylepšila vzhled planety. Možností je implementovat metodu atmosférického rozptylu světla, která se postará i o stín na odvrácené straně planety a také o východ a západ Slunce. Pokud bychom přidali oblačnost, aplikace by vypadala velmi realisticky.

Pokud se týče grafického zevnějšku povrchu, lze dodělat zobrazení stínů, zobrazení paprsků slunce, odlesky od vodní hladiny, animovanou vodu pomocí vertex shaderů a mnoho dalšího. Vše, co se v dnešní době využívá pro zobrazování krajin, lze použít i v rámci celé planety.

Pokud by se aplikace v budoucnu nezaměřovala pouze na zobrazení planety Země, lze zobrazovat celý solární systém včetně Slunce, či generovat pseudonáhodné planety pomocí matematických funkcí.

Literatura

- [1] John Savard: The Plate Carrée [online].
[cit 2009-19-04].
Dostupné na URL: <http://www.quadibloc.com/maps/mcy0104.htm>
- [2] NASA: Blue Marble, Next Generation [online].
Aktualizováno 2008-25-02 [cit 2009-13-04].
Dostupné na URL: http://visibleearth.nasa.gov/view_set.php?categoryID=2355
- [3] Wikimedia Foundation, Inc.: QuadTree [online].
Aktualizováno 2009-30-04 [cit 2009-02-05].
Dostupné na URL: <http://en.wikipedia.org/wiki/Quadtree>
- [4] Willem H. de Boer: Fast Terrain Rendering Using Geometrical MipMapping [online].
Aktualizováno 2000 [cit 2009-15-04].
Dostupné na URL: http://www.flipcode.com/archives/article_geomipmaps.pdf
- [5] Wikimedia Foundation, Inc.: Viewing frustum [online].
Aktualizováno 2008-15-11 [cit 2009-19-04].
Dostupné na URL: http://en.wikipedia.org/wiki/Viewing_frustum
- [6] FLIPCODE.COM: Frustum Culling [online].
Aktualizováno 2003-01-04 [cit 2009-20-04].
Dostupné na URL: http://www.flipcode.com/archives/Frustum_Culling.shtml
- [7] Infinity project: Horizon culling [online].
Aktualizováno 2008-04-03 [cit 2009-23-04].
Dostupné na URL: http://www.gamedev.net/community/forums/mod//journal/journal.asp?jn=263350&reply_id=3173799
- [8] Microsoft Corporation: Pipeline Stages (Direct3D 10) [online].
Aktualizováno 2008 [cit 2009-24-04].
Dostupné na URL: <http://msdn.microsoft.com/en-us/library/bb205123.aspx>

Seznam příloh

Příloha 1. Ovládání programu

Příloha 2. Nastavení konfiguračního souboru

Příloha 3. Příprava vlastních zdrojových dat

Příloha 4. Metrika kódu

Příloha 5. Obrázková dokumentace

Příloha 6. DVD

Příloha 7. Plakát

Příloha 1:

Ovládání programu

Použité klávesy:

Pohyb:

W - dopředu
A - doleva
S - dozadu
D - doprava
Q, E - rotace pohledu
SHIFT - rychleji

Ostatní:

Levé tlačítko myši - aktivuje a deaktivuje ovládání myši
Mezerník - přepínání mezi drátovým (WIREFRAME) a plným (SOLID) zobrazením
H - přepíná viditelnost HUDu
Tabulátor - přepínání mezi jednotlivými menu
F1, F2, F3, F4 - ovládací klávesy v menu
ESC - konec programu

Popis uživatelského rozhraní:

Press 'TAB' to change menu mode.

Engine info:

153.401 FPS

Rendered patches (total: 450) [lod]count

[0=max]0, [1]0, [2]0, [3]0, [4]0, [5]11, [6]15, [7]14, [8]16, [9=min]394

Rendered triangles (total: 0 008 916)

Rendered textures (total: 22) [status]count

[loaded]22, [loading]0

Estimated memory usage (total: 179MB) [type(containers)]used

[Planet texture(Tex)]176MB

[Planet heightmap(Tex+Buf)]1+2=3MB

Engine info zobrazuje základní informace o stavu, ve kterém se aplikace nachází. Součástí jsou údaje o počtu právě zobrazených políček, v jaké úrovni detailu se nachází, počet zobrazených trojúhelníků, počet načtených textur v aplikaci a odhadované množství spotřebované paměti grafické karty pro textury a výškové mapy jednotlivých políček.

Press 'TAB' to change menu mode.

LOD method settings (F1,F2 = up,down)(F3 = activate):

=> Distance.

=> Pixel Error.

=> Pixel Error + Distance.

=> Disabled.

LOD variable settings (F1,F2 = up,down)(F3,F4 = decrease,increase):

=> Distance: 0.6

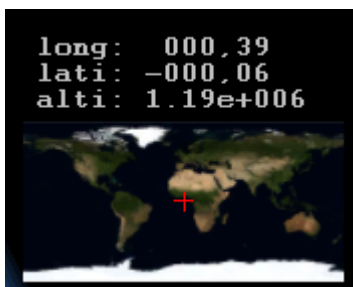
=> Pixel error: 5e-005

=> Disabled LOD: 9

Pomocí *LOD Method settings* lze vybírat jakou metodu pro výběr úrovně detailu použít. Právě aktivní je označena zeleně, pozice kde se nachází uživatelův výběr je označen červenou šipkou.

Pomocí *LOD variable settings* lze nastavovat hodnoty pro výběr úrovně detailu.

Pohybujeme se v menu pomocí F1 a F2, operace provádíme s F3 a F4.



Tato část je pouze informativní. Sděluje uživateli, na jakých geografických souřadnicích se právě nachází. Údaje nabývají hodnot od -180 do 180 pro zeměpisnou délku a -90 do 90 pro zeměpisnou šířku. Současně také udává, v jaké nadmořské výšce se pohybuje. Pro lepší orientaci, kde se uživatel nachází je zobrazena zmenšená mapa planety s vyznačeným umístěním kamery uživatele. Doporučuje se využívat tuto mapu pro navigaci po planetě.

Příloha 2:

Nastavení konfiguračního souboru

Aplikace hledá v aktuální složce konfigurační soubor, jehož jméno je předáno v příkazové řádce. Pokud není jméno zadáno, hledá se konfigurační soubor výchozí se jménem „config.cfg“.

config.cfg:

```
_global_
<
    /* rozliseni obrazovky */
    /* (640x480 || 800x600 || 1024x768 || 1280x1024) */
    screen_width "800"
    screen_height "600"

    /* obraci osu y mysi */
    /* INT(0 || 1) */
    invert_mouse "0"
>
earth
<
    _global_
    {
        /* realny polomer */
        /* INT(1 - 10 000 000) */
        real_radius "6378000"

        /* zobrazeny polomer */
        /* INT(1 - min(real_radius, 4 000 000)) */
        planet_radius "4000000"
    }

    _surface_
    {
        /* jak moc budou zvyšeny hodnoty z vyskove mapy */
        /* [prava_vyska = planet_radius/real_radius*heights_extrusion] */
        /* INT(1 - 200) */
        heights_extrusion "4"

        /* pocet policek vyskove mapy */

        /* delka */
        /* INT(1 - MAX_INT) */
        patch_x_count "128"

        /* vyska */
        /* INT(1 - MAX_INT) */
        patch_y_count "64"

        /* velikost policka vyskove mapy */
        /* INT((2^x)+1) */
        patch_size "513"

        /* velikost policka textury */
        /* INT((2^n) kvuli mip-mapingu) */
        texture_size "2048"
```

```

/* kolik celych ctvercu je v jedne texture x*x pro 3 je to celkem 9 a
jsou radky 0.5 1 1 1 0.5 sloupce co same (2 jsou tam jen z pulky
kvuli filtrovani..) */
/* INT(1 - INT_MAX) */
texture_patch_count "3"

/* konstanty pro vyber LOD */

/* konstanta pouzita v metode vzdalenosti */
/* FLOAT(0.01 - 1.5) */
distance_constant "0.5"

/* konstanta pouzita v metode pixel error */
/* FLOAT(0.000005 - 0.1) */
pixel_error_constant "0.001"

/* jmena souboru zdrojovych dat */

/* soubory policek vyskove mapy */
patch_file_name_pattern "65536x32769\patch_%dx%d_heightmap_513.raw"

/* poradi usporadani 16-ti bitovych hodnot ve
vyskove mape IBM = little, MAC = big*/
/* (IBM || MAC)*/
patch_file_byte_order "IBM"

/* soubory policek textury */
texture_file_name_pattern
"65536x32769\patch_%dx%d_textureRGB_2048.png"

/* pomocny soubor vyskove mapy pro policka */
info_file_name "65536x32769\patch_basic_info.bin"

/* soubor pro mene kvalitni nahradni texturu */
coarse_texture_file_name "texture3x4096x2048.png"
}

_atmosphere_
{
/* kvalita jedne strany krychle */
/* INT(2^n) */
sphere_cube_patch_size "32"

/* pocet atmosferickych kouli za sebou */
/* INT(1 - INT_MAX) */
sphere_count "5"

/* v jake vzdalenosti od povrchu bude prvni atmosfericka koule */
/* [sphere_radius = planet_radius * init_scale] */
/* FLOAT(1.001 - 1.2) */
init_scale "1.023"

/* s jakym prirustkem polomeru se budou pridava
zbyte atmosfericke koule */
/* [sphere_radius = init_scale + scale_increment *
altitude/planet_radius] */
/* FLOAT(0.001 - 1.0) */
scale_increment "0.02"

/* barva atmosfery */
/* [atmosphere_color_current_xxx = atmosphere_color_xxx *

```

```
atmosphere_color_alpha * sphere_count] */
/* FLOAT(0.0 - 1.0)*/
atmosphere_color_red "0.172"
atmosphere_color_green "0.379"
atmosphere_color_blue "0.684"
atmosphere_color_alpha "0.12"

/* konstanta kolik se přičte z barvy atmosféry povrchu při vstupu
do atmosféry při 0.0 bude mít povrch původní barvu textury */
/* FLOAT(0.0 - 1.0)*/
surface_inside_atmosphere_light_konstant "0.5"
}
>
```

Příloha 3:

Příprava vlastních zdrojových dat

Proces přípravy vlastních dat je poněkud náročnější z hlediska času i ovládání. Nejprve musíme stáhnout výškovou mapu a texturu. Lze použít i soubory přiložené na DVD.

Jakmile máme data k dispozici, musíme je zpracovat, aby je mohla aplikace využívat. Ke zpracování je na DVD přiloženo několik pomocných nástrojů. Práci s nimi postupně popíšeme na příkladu tvorby dat. Před použitím nástrojů je však nejprve soubory zapotřebí zmenšit na požadovanou velikost. Lze toho docílit například pomocí Photoshopu. Soubory musí mít velikost $x' = x \cdot 2^n$ a $y' = y \cdot 2^n + 1$, x-ová souřadnice je o 1 menší, jelikož poslední políčka berou poslední sloupec souřadnic z druhé strany. Pokud jde o velikosti hodnot v bytech, výšková mapa musí mít hodnoty o dvou bytech (short int), textura 3x jeden byte (RGB). U výškové mapy lze dvou bytová data mít uložena i ve formátech little endian i big endian.

Pomocné nástroje k přípravě dat:

`clamp.exe` - slouží k oseknutí hodnot výškové mapy, použito hlavně na záporné výšky
 <[path/]filename> - cesta k výškové mapě
 <width> - šířka výškové mapy
 <height> - výška výškové mapy
 <min_clamp> - minimální povolená výška (nula a výše)
 <max_clamp> - maximální povolená výška (do 32768)
 <[IBM] || [MAC]> - pořadí dat little endian nebo big endian (podle výškové mapy)

`height_map_splitter.exe` - slouží k rozsekání výškové mapy na jednotlivá políčka
 <[path\]heightMapFileName> - cesta k výškové mapě
 <[path\]patchFileNamePattern.raw>
 - cesta s vzorem pro pojmenování vzniklých souborů
 <width> - šířka výškové mapy
 <height> - výška výškové mapy
 <hm_patch_size> - velikost políčka na výškové mapě

`texture_splitter.exe` - slouží k rozsekání textury na jednotlivá políčka
 <[path\]textureFileName> - cesta k textuře
 <[path\]texturePatchFileNamePattern.bmp>
 - cesta s vzorem pro pojmenování vzniklých souborů
 <width> - šířka textury
 <height> - výška textury
 <hm_patch_size> - velikost políčka výškové mapy
 <hm_patch_per_line_in_texture_patch>
 - počet políček výškové mapy v jednom řádku políčka textury

`height_map_info.exe` - slouží k extrakci a přepočítání základních údajů o políčkách výškové mapy
 <[path\]infoFileName.bin> - jméno a umístění vzniklého souboru
 <[path\]patchFileNamePattern.raw> - cesta s vzorem pro pojmenované soubory políček
 <countX> - počet políček X (sloupce)
 <countY> - počet políček Y (řádky)

<hm_patch_size> - velikost políčka
<[IBM] || [MAC]> - pořadí dat little endian nebo big endian (podle výškové mapy)

conv.bat - slouží ke komprimaci textur z BMP na PNG

<countX-1> - počet políček X (sloupce) - 1

<countY-1> - počet políček Y (řádky) - 1

<[path\]filenamepattern.bmp>

- cesta s vzorem pro pojmenované soubory textur (pozor: liší se od zbylých, viz dále)

<outputpath> - cesta, kam se mají ukládat výsledné soubory

Konkrétní návod přípravy dat:

V tomto návodu budeme pro ukázkou pracovat se soubory výškové mapy a texturou o velikostech 32768x16385. Na tuto velikost byly zmenšeny pomocí Photoshopu. Formát dat uložených ve výškové mapě je little endian (IBM). Soubory se jmenují „hm32768x16385.raw“ a „tex32768x16385.raw“. Jsou uloženy ve stejném adresáři jako nástroje.

Zbavíme je záporných výškových údajů, se kterými si aplikace neporadí.

```
clamp.exe heightmap32768x16385.raw 32768 16385 0 9000 IBM
```

Rozdělíme výškovou mapu na jednotlivá políčka, budou uložena ve složce „hm“.

```
height_map_splitter.exe hm32768x16385.raw hm\patch_%dx%d_hm_513.raw 32768 16385 513
```

Vytvoříme pomocný soubor se základními údaji o všech políčkách výškové mapy.

```
height_map_info.exe hm\hm_info.bin hm\patch_%dx%d_hm_513.raw 64 32 513 IBM
```

Rozdělíme texturu na jednotlivá políčka, budou uložena ve složce „tex“. Poslední hodnota „3“ znamená, že budou 3 políčka výškové mapy na řádku v jednom políčku textury, tzn., že jedno políčko textury obsahuje celkem 9 celých políček výškové mapy a má velikost 2048x2048 bodů.

```
texture_splitter.exe tex32768x16385.raw tex\patch_%dx%d_tex_513.bmp 32768 16385 513 3
```

Abychom ušetřili množství dat, které musí načítat program, zkonvertujeme textury z formátu BMP do PNG. Ke konverzi je využit nástroj „conv.exe“, který je součástí balíku DirectX. Aby byl použitelný pro všechny políčka textur, je volán pomocí dávkového souboru „conv.bat“.

```
conv.bat 63 31 tex\patch_%Xx%Y_tex_513.bmp tex2\
```

Příloha 4:

Metrika kódu

Název aplikace	Pouze kód [řádků]	Pouze komentáře [řádků]	Celkový počet [řádků]
planet	7200	2500	9700
clamp	200	50	250
height_map_splitter	500	200	700
height_map_info	600	200	800
texture_splitter	700	300	1000
celkem	9200	3250	12450

Příloha 5:

Obrázková dokumentace:

