VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYKRESLOVÁNÍ PLANET S VYUŽITÍM ADAPTIVNÍCH ÚROVNÍ DETAILŮ

ADAPTIVE LEVEL OF DETAIL FOR PLANET RENDERING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MATĚJ MORAVEC**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL MATÝŠEK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Moravec Matěj**
Program: Informační technologie
Název: **Vykreslování planet s využitím adaptivních úrovní detailů**
Adaptive Level of Detail for Planet Rendering
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte techniky pro vykreslování planet s adaptivními úrovněmi detailů a seznamte se s herním enginem Unity.
2. Vybrané techniky popište a navrhněte interaktivní aplikaci pro demonstraci zvolených metod.
3. Implementujte interaktivní aplikaci využívající vybrané techniky.
4. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte video prezentující projekt.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, rozpracování bodu 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Matýšek Michal, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato bakalářská práce se zabývá vykreslováním planety v reálném čase s adaptivní úrovní detailu terénu v enginu Unity. Pro adaptivní změnu úrovní detailů je použit a modifikován algoritmus Chunked LOD. Pro efektivní vykreslování dat je použit instancing. Využívá se grafické karty pro vytvoření geometrie planety pomocí výškových map a také je aplikována teselace pro postupné morfování při změně úrovně detailu.

Abstract

This bachelor thesis is focused on real time planet rendering with adaptive level of terain detail in Unity engine. Chunked LOD algorithm is modified and used for adaptive change of level of detail. Intancing is used for efective data rendering. Graphic card is used for creating planet geometry with heightmaps and there is also tessellation used for gradual morfing of level change of detail.

Klíčová slova

Unity, quadtree, Chunked LOD, frustum culling, teselace, instancing, výšková mapa, adaptivní úroveň detailů

Keywords

Unity, quadtree, Chunked LOD, frustum culling, tessellation, instancing, heightmap, adaptive level of detail

Citace

MORAVEC, Matěj. *Vykreslování planet s využitím adaptivních úrovní detailů*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Matýšek

Vykreslování planet s využitím adaptivních úrovní detailů

Prohlášení

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

.....
Matěj Moravec
16. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Michalu Matýškovi za odborné rady a ochotu být nápomocný v každé situaci.

Obsah

Kapitola 1

Úvod

V počítačových hrách se objevují mapy, které mají velikost i několik desítek kilometrů čtverečních, a hráč se v nich může libovolně pohybovat. Při přemisťování se hráč může dostat z jedné strany mapy na druhou bez jediného načtení nové scény nebo ztráty kvality terénu. To, co hráč nevidí, je postupná změna úrovně detailu terénu, která okolo něj probíhá, když mění svoji pozici po mapě.

Tématem této bakalářské práce je využití metody pro adaptivní změnu úrovně detailu terénu a její aplikace na planetu, která bude mít velikost v průměru i několik tisíců kilometrů. Existuje mnoho projektů nejen herního žánru, které se zabývají problematikou vykreslování planet v reálném čase. Mezi nejznámější z nich patří virtuální glóbus *Google Earth* zobrazující planetu Zemi nebo vesmírná hra *Kerbal Space Program*, která používá procedurální generování na vytvoření sluneční soustavy.

Práce je rozdělena do několika kapitol. V kapitole 2 jsou popsány různé techniky pro vytvoření terénu s adaptivní úrovní detailu, které dokáží v reálném čase vykreslit i terén s rozlohou stovek čtverečních kilometrů, zároveň představuje techniky pro zvýšení efektivity vykreslování terénu. Další problematikou, kterou se kapitola zabývá, je získávání dat terénu a způsob jejich reprezentace ve 3D scéně. Kapitola 3 popisuje jednotky grafické karty, které byly využity při vytváření terénu, a krátce představí využité vývojové prostředí Unity. V kapitolách 4 a 5 je představen návrh aplikace a celkový popis implementace jednotlivých technologií. Popis testování aplikace je popsán v kapitole 6. Závěrečná kapitola 7 shrnuje výsledky práce a udává možnosti budoucího vývoje aplikace.

Kapitola 2

Vykreslování terénu

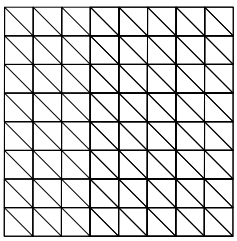
Začátek této kapitoly je zaměřen na způsob, jak reprezentovat a získávat data pro 3D terén. Vzhledem k objemu dat, které je zapotřebí pro detailní vykreslení všech míst na mapě, jsou ukázány základní algoritmy využívající adaptivní úrovně detailu terénu v reálném čase. Dalším bodem je popsána transformace krychle a plochy na kouli a dodání realistického efektu pomocí osvětlovacího modelu. Závěrem jsou popsány metody pro zvýšení efektivity vykreslování terénu.

2.1 Reprezentace terénu v počítačové grafice

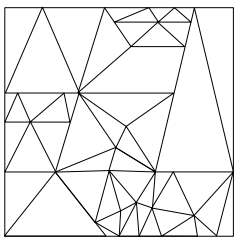
V počítačové grafice se pro popsání geometrie terénu používá jako základní primitivum trojúhelník. Sloučením primitiv vzniká takzvaná polygonální síť, kterou lze rozdělit na pravidelnou a nepravidelnou.

Pravidelná polygonální síť (viz obrázek 2.1a) je tvořena ze stejně velkých trojúhelníků, které jsou rovnoměrně rozloženy. Počet trojúhelníků v síti je roven $(l-1)(h-1)\cdot 2$, kde l je délka a h je výška sítě. Mezi hlavní výhody patří jednoduchá implementace, nízká paměťová náročnost, a díky pravidelnosti sítě lze snadněji přistupovat k vrcholům a manipulovat s nimi.

Nepravidelná polygonální síť (viz obrázek 2.1b) je tvořena obecnými trojúhelníky, které mohou mít různou velikost a libovolné rozmístění. Při popisování málo členitého terénu potřebuje méně trojúhelníku než pravidelná, a díky jejich rozmanitosti lze geometrii sítě upravit tak, aby místa členitého terénu byla popsána detailněji. Oproti pravidelné síti, která uchovává pouze jednu souřadnici, protože rozestupy mezi body mají konstantní velikost a lze je dopočítat, se však musí uchovávat všechny tři souřadnice bodů. Z tohoto důvodu vzniká vetší paměťová složitost, která omezuje využitelnost.



(a) Pravidelná polygonální síť.

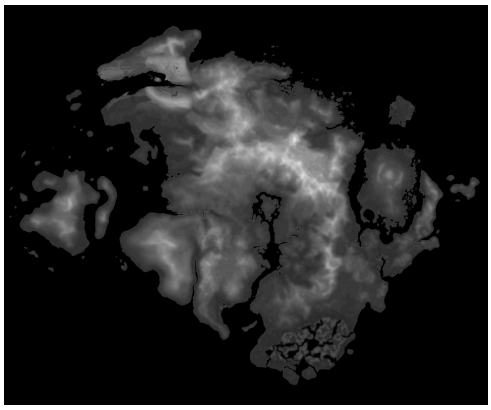


(b) Nepravidelná polygonální síť.

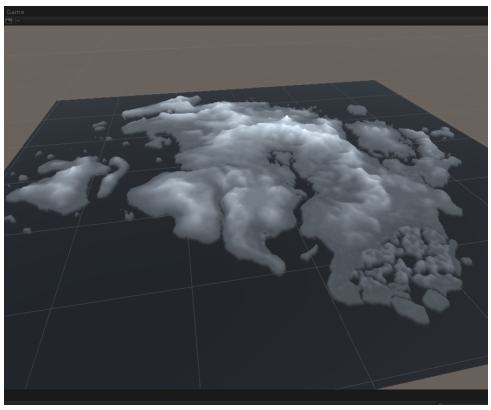
Obrázek 2.1: Ukázka reprezentace dat ve 3D scéně pomocí polygonálních sítí.

2.1.1 Výškové mapy

Pro samotné vykreslení terénu v pravidelné polygonální síti je zapotřebí získat výšku h jednotlivých vertexů. Výšku lze získat z výškových map, díky kterým lze vykreslit terén veliký i několik stovek kilometrů čtverečních. Výškové mapy představují způsob, jak jednoduše získávat a ukládat výškové hodnoty. Obvykle jsou reprezentovány 8bitovým obrázkem v odstínech šedi. Princip výškových map je založen na myšlence, že každý pixel obrázku reprezentuje výškovou hodnotu, která může být vynesena ve výsledném modelu. U 8bitového obrázku černý pixel, který má hodnotu 0, reprezentuje nejnižší možnou výšku. Bílý pixel, který má hodnotu 255, představuje nejvyšší možnou výšku. Na obrázku 2.2b je znázorněn vytvořený výsledný model terénu pomocí výškové mapy 2.2a. Mnohdy 256 výškových hodnot, které obsahuje 8bitový obrázek, nemusí stačit, proto se může přistoupit k použití 24bitového obrázku, který může obsahovat až 16 777 216 výškových hodnot. Nevýhodou výškových map je absence vykreslování jeskyní a převisů, protože na jednom pixelu dokáže být pouze jedna hodnota. Výškové mapy lze získat z předem připravených dat, nebo procedurálně vygenerovat.



(a) Výšková mapa. Převzato z [3].



(b) Terén vytvořený pomocí výškové mapy.

Obrázek 2.2: Znázornění převodu výškové mapy na 3D model.

2.2 Úrovně detailů povrchu terénu

Geometrie terénu, která je vytvořena pomocí výškové mapy, může mnohdy obsahovat až desítky miliónů bodů. Při vykreslování tak rozsáhlého terénu by bylo velice náročné vykreslit všechny body v reálném čase. Proto byla vyvinuta technika na změnu úrovně detailu modelu, která se nazývá LOD (*Level Of Detail*) [17]. Princip metody spočívá v nepotřebě vykreslovat vzdálené nebo rovinné objekty v tak velkém detailu, protože na obrazovce mohou modely zabírat pouze pár pixelů, a přitom se skládat i z tisíce trojúhelníků. Při použití algoritmů pro různou úroveň detailu terénu se může mnohonásobně urychlit jeho zobrazování. LOD se rozděluje na diskrétní a spojité.

2.2.1 Diskrétní úrovně LOD

Diskrétní úroveň detailů byla poprvé představena v roce 1976 [5] a ve většině 3D aplikací se používá dodnes. V případě použití diskrétní úrovně detailů je nutné předem připravit několik detailů modelu. Jaká úroveň modelu bude vybrána pro vykreslení, závisí na zvolené

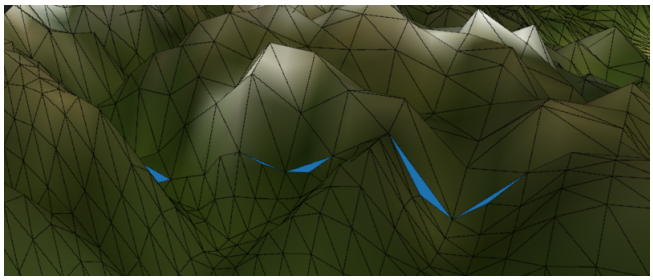
metrice. Obvykle je jako základní metrika zvolena vzdálenost pozorovatele od objektu. Každé úrovni modelu je přiřazena hodnota, při které dojde k přepnutí na jiný model. Hlavní výhodou této metody je, že modely mohou být vytvořeny před spuštěním aplikace a není nutno zatěžovat procesor nebo grafickou kartu výpočty za běhu. Nevýhodou je, že někdy může zobrazovat detailnější modely, než je potřeba, nebo naopak zobrazit méně detailní modely, když je potřeba zobrazit více detailní model. Při přepínání mezi různými úrovněmi modelu vzniká artefakt nazvaný *popping effect* [17], který se projevuje jako náhlý skok v geometrii modelu. Lze ho odstranit postupným morfováním mezi variantami modelu [16].

2.2.2 Spojité úrovně LOD

Spojitá úroveň se liší od diskrétní v tom, že nevytváří individuální modely pro úrovně, ale vytvoří datovou strukturu, ve které je uloženo kontinuální spektrum detailů [11]. Úrovně detailu modelu jsou transformovány v malých krocích z datové struktury v době běhu programu. Spojitá úroveň volí jednotlivé úrovně modelu podle potřeby. Při změně úrovně modelu nedochází k *popping* efektu, protože změna je prováděna po malých krocích. Nevýhodou je větší náročnost na výpočet.

2.2.3 T-spoje a trhliny

U algoritmů, které používají úrovně detailu terénu, mohou vznikat trhliny a T-spoje. Obojí způsobuje nedokonalosti v geometrii terénu (viz obrázek 2.3). Nejčastěji vznikají na společné hraně dvou různě detailních úrovní terénů, kde oblast s větší úrovní detailu má na společné hraně více vertexů než jeho sousední, a způsobuje tak nespojitost. Trhlina vzniká, když se vertex nenachází na společné hraně. U T-spojů se sice vertexy nacházejí na společné hraně, ale kvůli absenci normál z druhé strany hrany dochází k zaokrouhlovacím chybám a vzniká rozdíl v osvětlení modelu.



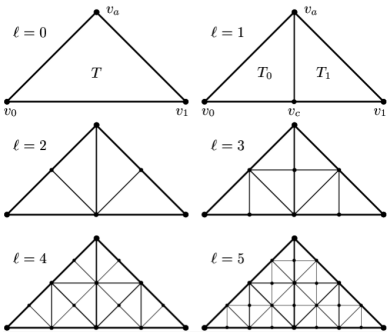
Obrázek 2.3: Vznik trhlina v terénu při rozdílné úrovni detailů dvou bloků.

2.3 Algoritmy pro tvorbu terénu

Tato sekce představuje algoritmy, které se používají pro vykreslení terénu. Díky těmto algoritmům může být efektivně vykreslen v reálném čase terén velký až několik stovek čtverečních kilometrů.

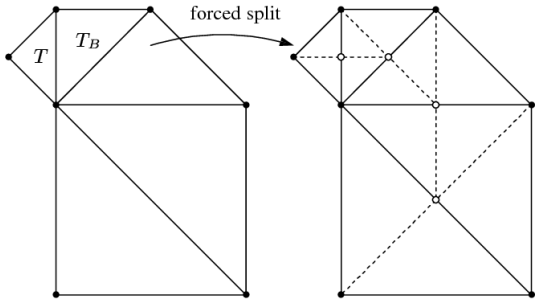
2.3.1 ROAM

V roce 1997 publikoval tým pod vedením Marka Duchaineau algoritmus s názvem Realtime Optimally Adapting Meshes (ROAM) [7]. V této metodě se rozděluje pomocí binárního trojúhelníkového stromu. Jako základ celé struktury slouží dva pravoúhlé rovnoramenné trojúhelníky, které dohromady tvoří čtverec. Nejvýše postavený uzel představuje nejnižší detail terénů. Nižší úrovně binárního stromu vznikají operací *split*, kdy každý uzel může být rekurzivně rozdělen na dva stejně velké trojúhelníky v polovině přepony. Opačná operace se nazývá *merge*, ta naopak trojúhelníky spojuje a tvoří terén, který má menší detail. Úroveň dělení může být závislá na geometrii terénu, na vzdálenosti od pozorovatele nebo na minimální a maximální hloubce binárního stromu. Princip je znázorněn na obrázku 2.4.



Obrázek 2.4: Rozdělení jednotlivých úrovní u algoritmu ROAM. Převzato z [7].

Jak bylo zmíněno, pomocí operace *split* je rozdělen trojúhelník na dva trojúhelníky o poloviční velikosti. Při této operaci je zapotřebí dodržet podmínku, že musí být společně se stejným uzlem rozdělen i sousední uzel, se kterým má společnou přeponu. Tento proces se opakuje, dokud má podmínka význam. Kdyby se podmínka nedodržela, vznikaly by trhliny a T-spoje na společné přeponě. Nucené rekurzivní rozdělení za účelem zabránit vzniku trhlin je znázorněno na obrázku 2.5.



Obrázek 2.5: Na levém obrázku je potřeba rozdělit trojúhelník T. Na pravém obrázku je výsledek po nuceném rozdělení. Převzato z [7].

Vlastnosti ROAM

- Využívá snímkové koherence – terén nemusí být generován pro každý snímek znovu, ale lze využít předchozí krok, který je pouze aktualizován.

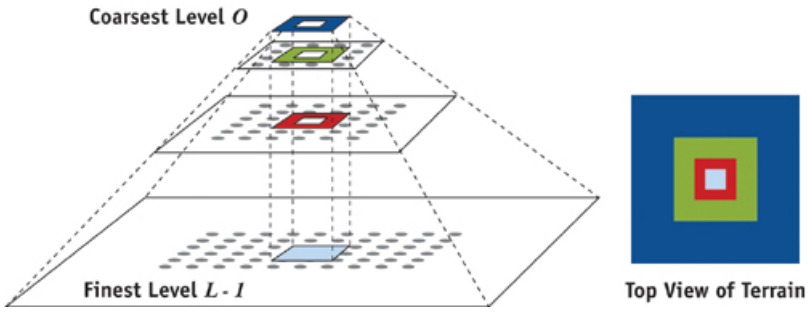
- Vznik trhlin řeší implicitně.
- Je možná regulace počtu trojúhelníku v terénu.

2.3.2 Geometry Clipmaps

Metodu Geometry Clipmaps vyvinuli v roce 2004 Frank Lossasso a Huegues Hoppe [10]. Základním principem je rozdělení terénu pomocí pyramidové hierarchie, která je centrovaná okolo pozorovatele (viz obrázek 2.6). Metoda používá pro reprezentaci terénu pravidelnou polygonální síť. Každá úroveň obsahuje stejný počet vertexů jako předešlá, ale rozestupy mezi nimi jsou dvojnásobné (případě i větší). Vykreslování terénu probíhá od nejmenšího detailu po největší. Úroveň detailu není určena geometrií terénu, ale určuje ji vzdálenost pozorovatele od terénu. Detaily jsou aktualizovány pokaždé, když dojde k pohybu pozorovatele. K tranformaci vrcholů na pravidelné síti dochází až ve vertex shaderu, který vzorkuje výškovou mapu a transformuje vertexy do požadované výšky.

Vlastnosti Geometry Clipmaps

- Lze plně implementovat na grafické kartě [2].
- Lze použít stejnou techniku LOD na textury.
- Při vykreslování rovinného terénu je geometrie příliš detailní, protože algoritmus nebere v potaz členitost terénu.
- Při rychlejším pohybu pozorovatele se nemusí stihnout vykreslit úrovně s větší detailností.



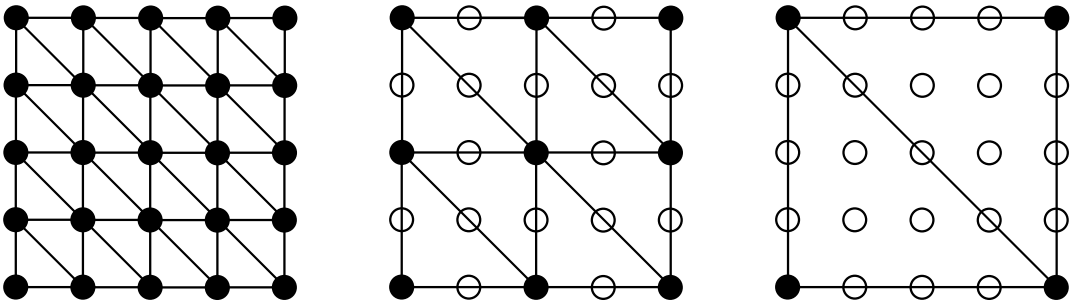
Obrázek 2.6: Pyramidová hierarchie u Geometry Clipmaps. Napravo je pohled ze shora. Převzato z [2].

2.3.3 GeoMipMapping

V roce 2000 představil Willem H. de Boer algoritmus GeomMipMapping [6]. Autor se ve své práci snaží využít potenciálu grafické karty, a co nejvíce zbavit zátěže procesor. GeoMipMapping posílá na grafickou kartu více dat, než je potřeba, za účelem ušetření procesoru zbytečných výpočtů.

Pro reprezentaci dat používá algoritmus pravidelnou polygonální síť. Hlavní důvody, proč Boer využil pravidelnou síť, jsou v její snadné implementaci a v nízkých paměťových

nárocích. Terén je rozdělen do stejně velkých čtvercových bloků, kde každý blok musí mít počet vertexů na vertikální a horizontální straně roven $2^n + 1$, kde $n > 0$ (viz obrázek 2.7a). Bloky obsahují několik úrovní, které jsou vytvořeny v době předzpracování. Při vytváření nové úrovně jsou použity pouze liché vertexy z předešlé úrovně. Na obrázcích 2.7a, 2.7b a 2.7c jsou znázorněny první tři úrovně detailu.

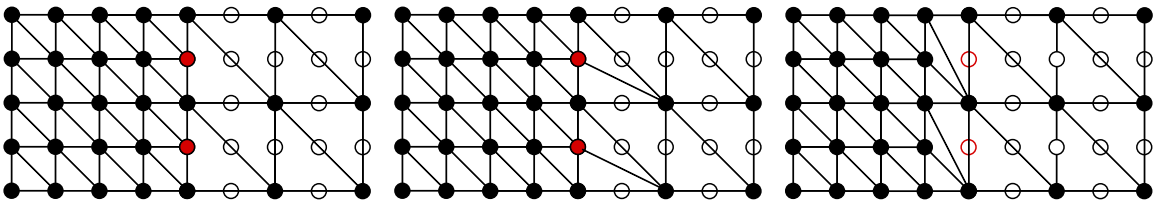


(a) GeoMipMapping úrovně 0. (b) GeoMipMapping úrovně 1. (c) GeoMipMapping úrovně 2.

Obrázek 2.7: První tři úrovně algoritmu GeoMipMapping.

Jako rozhodovací kritérium, která úroveň detailu bude zvolena, slouží metrika chyby. Základní metrika chyby, která se používá, je vzdálenost pozorovatele od středu bloku. Tato metrika ale způsobuje *popping effect*. Aby se mohly jednotlivé bloky lehce oříznout pomocí metody *frustum cullin* (rozebrána v sekci 2.6.1), zvolil autor pro ukládání jednotlivých bloků terénu kvadratickou stromovou strukturu (*quadtree*). Každý uzel obsahuje osově orientovaný kvádr (tvz. *bounding box*), který slouží pouze pro obalení daného uzlu, a listy stromu obsahují informace o blocích. Metodou *frustum culling* se rekurzivně od kořene testují osově orientované kvádry jednotlivých uzlů, zda zasahují do zorného pole kamery.

Při přechodu mezi bloky, které mají různou úroveň detailu, vznikají trhliny a T-spoje. Algoritmus řeší vznik trhlin dvěma způsoby. První způsob je přidání vertexu do uzlu, který má menší úroveň detailů. Způsob přidání je znázorněn na obrázku 2.8b, kde vertex, který je označen červenou barvou, představuje bod, který je přidán na blok s menší úrovní detailu. Druhým způsobem je odebrání vertexu v uzlu, který má větší úroveň detailu. Na obrázku 2.8c je znázorněn způsob odebrání vertexu, kde je odebrán vertex v červeném kroužku. Oba způsoby zajistí, že počet vertexů na společné hraně bude totožný.



(a) Neošetřená trhlina. (b) Ošetřená trhlina přidáním vertexu na méně detailním bloku. (c) Ošetřená trhlina odebráním vertexu na detailnějším bloku.

Obrázek 2.8: Ukázka vzniku trhliny a T-spoje u bloků, které mají rozdílnou úroveň detailu zobrazení a dvě možnosti, jak je na sebe napojit dva bloky a zabránit vzniku trhlin a T-spojů.

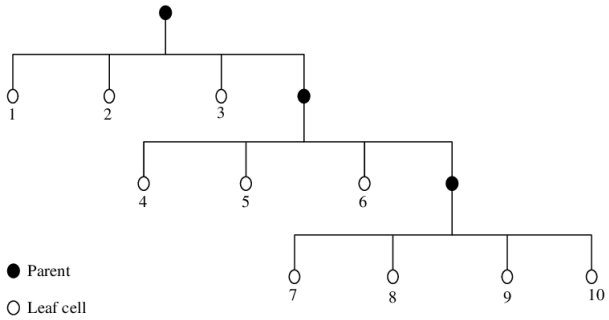
Vlastnosti GeoMipMappingu

- Sousedící bloky se mohou lišit pouze o jednu úroveň detailu.
- Větší náročnost při předzpracování terénu.
- Je nutné řešit problém s napojováním bloků, které mají různou úroveň. Algoritmus, na rozdíl od Chunked LOD, řeší mnohem lépe vznik trhlin.
- Díky kvadratické datové struktuře lze lehce ořezávat plochy, které se nenacházejí v zorném poli kamery.
- *Popping effect* může být eliminován pomocí morfingu.

2.3.4 Chunked LOD

Algoritmus Chunked LOD je velice podobný metodě GepMipMappingu. Poprvé byl představen v článku pod názvem *Rendering massive terrains using chunked level of detail control* [15]. Chunked LOD, narozdíl od GepMipMappingu, používá nepravidelnou polygonální síť.

Úrovně detailu nejsou aplikovány na jednotlivá primitiva, ale aplikují se na celé dlaždice terénu (odtud název *chunk*). Algoritmus je založen na kvadratické stromové datové struktuře, která uchovává informace o generované scéně. Nejvýše postavený uzel představuje nejmenší detail terénu. Uzel může mít čtyři poduzly, které mají velikost jednu čtvrtinu rodičovského uzlu, a představují jeho detailnější úroveň. Na obrázku 2.9 jsou vidět jednotlivé úrovně kvadratického stromu. Algoritmus před prvním vykreslením vygeneruje všechny uzly, proto je metoda statická a nelze měnit geometrii různých dlaždic za běhu programu.

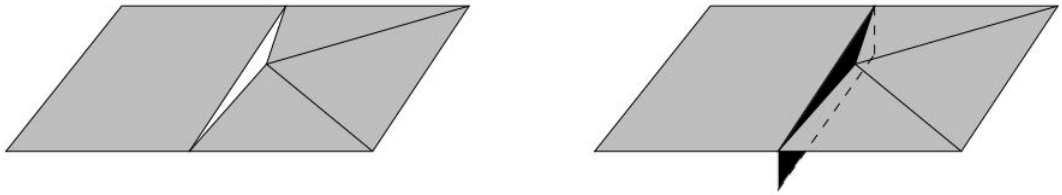


Obrázek 2.9: Jednotlivé úrovně rozdělení kvadratického stromu. Převzato z [4].

Jako u všech algoritmů, které používají úroveň detailu terénu, se musí řešit problém vzniku trhlin. Metoda Chunked LOD řeší problém vzniku trhlin přidáním pásu na každou hranu dlaždice (viz obrázek 2.10). Způsob zacelení není dokonalý, protože jednotlivé dlaždice budou stále nespojené a může docházet k vizuálním nesrovnalostem.

Vlastnosti Chunk LOD

- Díky nepravidelné polygonální síti dokáže lepé než metoda GeoMipMapping popsat terén při stejném počtu trojúhelníků.
- Větší náročnost při předzpracování terénu.
- Nedokonalý způsob napojování jednotlivých dlaždic.



Obrázek 2.10: Vznik trhliny a její ošetření u metody Chunked LOD. Obrázek vlevo znázorňuje, jak vypadá trhlina. Obrázek vpravo ilustruje zacelení trhliny pomocí pásu. Převzato z [15].

2.4 Metody mapování dat na planetu

Planetu, která je reprezentována geometrickým tvarem koule, lze vytvořit různými způsoby. Jeden ze způsobů je vytvoření krychle, která se následně transformuje do podoby koule. Jelikož jsou na planetu mapovány textury společně s výškovými mapami, je nutné použít i převod plochy na kouli.

2.4.1 Převod souřadnic krychle na kouli

Pro vytvoření podkladu pro planetu lze využít transformaci krychle na kouli pomocí vzorce (2.1), který vytvoří, na rozdíl od převodu plochy na kouli, rovnoměrně rozloženou hustotu bodů po celém objektu. Tato metoda umožňuje mnohem snadněji pracovat s jednotlivými zónami na povrchu koule.

$$v(o) = \begin{bmatrix} x \cdot \sqrt{1 - \frac{y^2}{2} - \frac{z^2}{2} + \frac{y^2 \cdot z^2}{3}} \\ y \cdot \sqrt{1 - \frac{z^2}{2} - \frac{x^2}{2} + \frac{z^2 \cdot x^2}{3}} \\ z \cdot \sqrt{1 - \frac{x^2}{2} - \frac{y^2}{2} + \frac{x^2 \cdot y^2}{3}} \end{bmatrix} \tag{2.1}$$

2.4.2 Převod souřadnic koule na souřadnice textury

Textury, které se mapují na povrch planety, jsou většinou reprezentovány jako plocha ve 2D souřadnicích. Zato planeta je popsána jako koule, která se nachází ve 3D souřadnicovém prostoru. Z toho důvodu se používá převod sférických souřadnic na kartézské podle rovnice:

$$x = r \cdot \sin(\theta) \cdot \cos(\lambda), \tag{2.2a}$$
$$y = r \cdot \sin(\theta) \cdot \sin(\lambda), \tag{2.2b}$$
$$z = r \cdot \cos(\theta), \tag{2.2c}$$

kde r udává poloměr planety, θ je úhel, který je odčítaný od směru osy z a λ je polární úhel v rovině xy měřený od osy x. Jelikož se textury nanášejí na existující kouli, je nutno

vypočítat tzv. uv souřadnice koule podle vzorce (2.3), které udávají pozici bodu koule na mapované textuře.

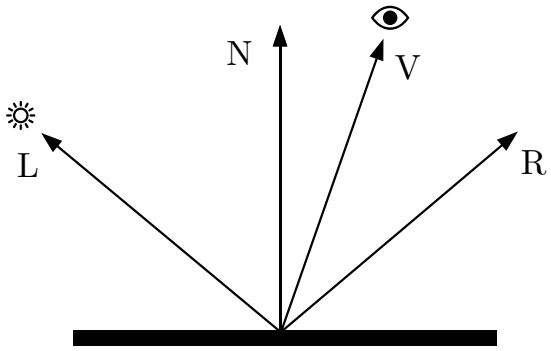
$$u = 0.5 + \frac{\arctan((d(z), d(x)))}{2\pi} \tag{2.3a}$$

$$v = 0.5 - \frac{\arcsin((d(y)))}{\pi} \tag{2.3b}$$

2.5 Phongův osvětlovací model

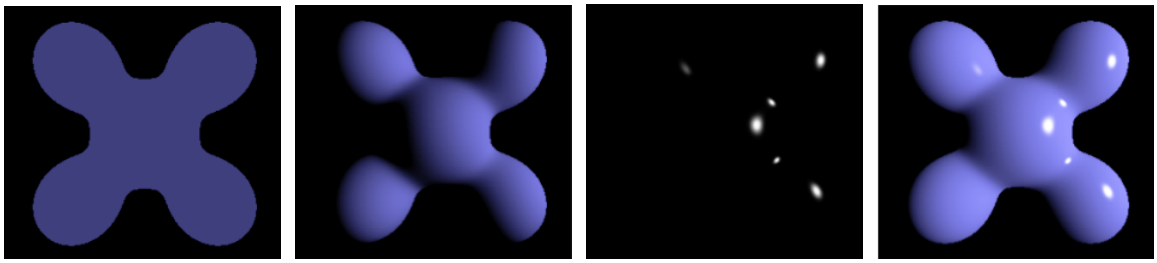
Osvětlení je nedílnou součástí při tvorbě 3D scény. Právě díky osvětlení je možné dosáhnout realistického vzhledu scény. Osvětlovací modely se dělí na fyzikální (realistické) a zjednodušené (empirické). Hlavní faktory u empirického modelu, které ovlivňují výsledné osvětlení, jsou směr a intenzita světla, vlastnosti materiálu a směr pohledu pozorovatele. Phongův osvětlovací model je empirický, byl navrhnut vietnamským vědcem Bui Tuong Phongem v jeho dizertační práci [14] z roku 1973. Phongův osvětlovací model pracuje se čtyřmi vektory (viz obrázek 2.11), kterými jsou:

- $\vec{R}$ – je vektor udávající odraz světla.
- $\vec{V}$ – je vektor pohledu pozorovatele.
- $\vec{N}$ – je normálový vektor.
- $\vec{L}$ – je vektor určující směr dopadajícího světla.



Obrázek 2.11: Vektory, které používá Phongův osvětlovací model.

Výsledná barva pixelu u Phongova osvětlovacího modelu je odvozena ze tří základních složek světla, které se odrážejí od povrchu. Jsou to ambientní složka, difúzní složka a zrcadlová složka. Sečtením všech tří složek vznikne výsledný model (viz obrázek 2.12d).



(a) Ambientní složka. (b) Difúzní složka. (c) Zrcadlová složka. (d) Výsledný model.

Obrázek 2.12: Phongův osvětlovací model. Převzato z [1].

2.5.1 Ambietní složka

Ambientní složka (obrázek 2.12a) osvětluje rovnoměrně celý model. U světla dochází k mnohočetnému odrážení a lámání, proto se světlo i v malém množství dokáže dostat na místa, která nejsou přímo osvětlená. Právě ambientní složka reprezentuje všudypřítomné světlo, které nesouvisí s pozicí pozorovatele ani se světelným zdrojem. Vypočítá se podle vzorce:

$$I_a \quad = \quad I_A \cdot r_a, \tag{2.4}$$

kde I_A je intenzita okolního světla a r_a je odrazivý koeficient materiálu objektu.

2.5.2 Difúzní složka

Na rozdíl od ambientní složky bere difúzní složka v úvahu, pod jakým úhlem světlo dopadá na povrch. Difuzní složka (viz obrázek 2.12b) představuje část světla, které se po dopadu rovnoměrně rozloží. Je závislá na svítivosti zdroje a intenzita klesá s rostoucí vzdáleností. Pro výpočet difúzní složky je také důležitý sklon plochy vůči světlu. Difúzní složka se dá vyjádřit vztahem:

$$I_d = I_L \cdot r_d \cdot (\vec{L} \cdot \vec{N}), \tag{2.5}$$

kde I_L je intenzita dopadajícího světla, r_d je koeficient odrazu, $\vec{L}$ je vektor dopadajícího paprsku a $\vec{N}$ je normála povrchu.

2.5.3 Zrcadlová složka

Phongův osvětlovací model se od Lambertova osvětlovacího modelu [13] liší v zrcadlové složce (viz obrázek 2.12c). Lesklé světlo udává intenzitu té části světla, která se odráží od modelu k pozorovateli. Lze ji vyjádřit pomocí vzorce:

$$I_s = I_L \cdot r_s(\vec{V} \cdot \vec{R})^h, \tag{2.6}$$

kde I_L je intenzita dopadajícího světla, k_s je koeficient odrazivosti, který určuje míru zrcadlové složky zastoupenou v odražené intenzitě, $\vec{V}$ je vektor pohledu, $\vec{R}$ udává vektor odrazu světla a h je koeficient udávající ostrost odlesku.

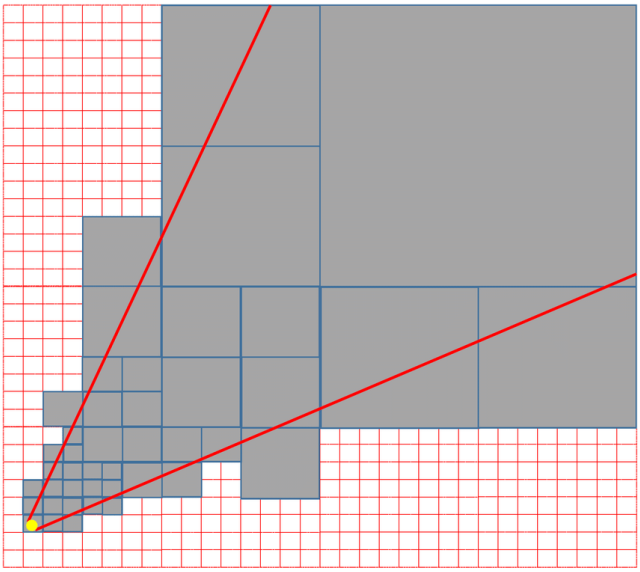
2.6 Optimalizační techniky

Algoritmy, které používají úrovně detailu terénu v reálném čase, lze zefektivnit implementováním optimalizačních technik. Tato sekce popisuje techniky *view frustum culling* pro snižování počtu vykreslovaných objektů ve scéně a metodu *GPU Instancing*, která dokáže vykreslit mnoho stejných objektů jedním příkazem.

2.6.1 View frustum culling

View frustum culling [17] je optimalizační metoda pro zmenšení počtu objektů, které jsou vykreslovány ve scéně. Zorné pole lze brát jako komolý jehlan tvořený šesti ořezovými rovinami. Princip této metody spočívá v tom, že se porovnají hranice všech objektů, které se nacházejí ve scéně s komolým jehlanem, který se získá z projekční matice. Tuto metodu má smysl používat pouze tehdy, když se ve scéně nachází více objektů.

Testování každého primitiva je výpočetně velice náročné, proto lze plochy rozdělit na větší objekty, které se ohraničí osově orientovaným kvádrem a následně se tyto kvádry testují, zda zasahují do zorného pole kamery (viz obrázek 2.13). Při rozdělení na větší plochy nelze dosáhnout takové přesnosti, jako při testování každého primitiva zvlášť, ale zato se zásadně ušetří výpočetní výkon. Díky této metodě lze vykreslovat jen potřebné objekty, které leží částečně nebo úplně v pohledu kamery. Tato metoda vykresluje také objekty, které jsou v zorném poli, ale kvůli jejich otočení či zastínění jiným objektem nemohou být vidět. Tento problém řeší například metoda *occlusion culling*, která testuje, zda je objekt kompletně zastíněn jiným objektem, a nemusí být tedy vykreslen.



Obrázek 2.13: View frustum culling aplikovaný ve stromové struktuře. Šedé plochy znázorňují oblast, která je vykreslena. Převzato z [12].

2.6.2 GPU instancing

U mnoha počítačových her se vyskytují modely, které se mohou ve scéně několikanásobně opakovat, a přitom se mohou lišit pouze v poloze nebo ve velikosti modelu. Jako příklad lze uvést pás asteroidů, který může obsahovat jen několik různých druhů kamenů, ale každý

kámen může mít až statisíce kopií, které se momentálně mohou lišit pouze v pozici a směru otáčení. Vykreslovat každý takový objekt individuálně (tzn. nastavovat grafický řetězec a volat příkaz *draw*) je však velmi neefektivní.

Tento problém lze vyřešit pomocí metody zvané instancing, která dokáže vykreslit všechny objekty se stejným modelem jedním příkazem. Procesor vygeneruje nebo načte model, který chce opakovaně vykreslit. Pro každý model, který chce vykreslit, vytvoří buffery, do kterých se uloží data všech instancí. Mezi data, která mohou být vložena, patří pozice, barva, rotace a mnoho dalších.

Kapitola 3

Použité technologie

Kapitola popisuje technologie, které byly použity při vytváření planety s adaptivní úrovní detailu terénu. Začátek kapitoly se stručně zabývá vývojovým prostředí Unity, následně je popsán vykreslovací řetězec společně s jednotlivými úrovněmi, které jsou využívány v aplikaci.

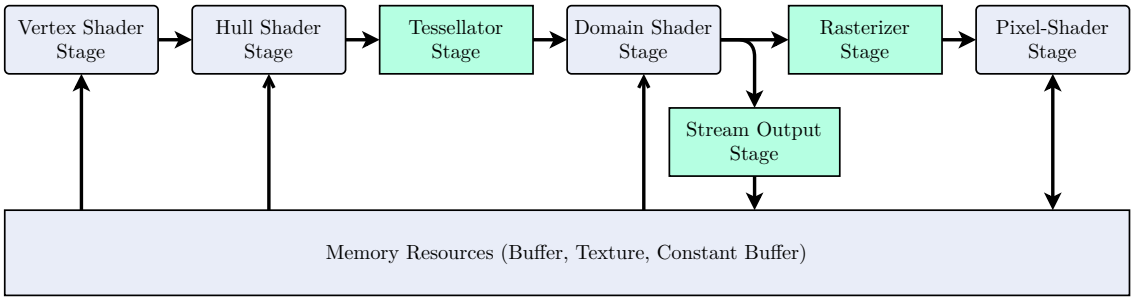
3.1 Vývojové prostředí

Jako vývojové prostředí je zvolen multiplatformní engine Unity 3D. Umožňuje tvorbu 2D a 3D her nejen pro Windows, ale i pro konzole nebo chytré telefony. Unity umožňuje vytvářet skripty pomocí jazyků C# nebo JavaScript. Samotné Unity obsahuje komponentu s názvem *terrain*, která dokáže vygenerovat plochu a na ní vytvářet terén pomocí nástrojů nebo výškové mapy. Komponenta obsahuje základní principy změny úrovně detailu podle vzdálenosti pozorovatele a členitosti terénu. Pro menší rozměry terénu je tato komponenta dostačující, ale pro větší terén nebo pro planetu s adaptivní úrovní detailu komponenta není použitelná. Cílem této bakalářské práce je vytvoření komponenty, která předčí *terrain* v tom, že bude schopna vykreslit planetu s adaptivní úrovní detailu terénu s průměrem do několika tisíc kilometrů.

3.2 Vykreslovací řetězec

Vykreslovací řetězec představuje abstrakci toho, jak funguje grafická karta. Obsahuje několik úrovní ve stanoveném pořadí. Jednotlivé úrovně vykreslovacího řetězce jsou znázorněny na obrázku 3.1. Programovatelná část řetězce obsahuje program, který se nazývá shader. Shadery jsou krátké programy, které kompletně běží na grafické kartě a umožňují manipulaci s primitivy a práci s pixely. K tvorbě těchto menších programů se používají specializované jazyky. OpenGL používá k tvorbě shaderů jazyk GLSL[1], DirectX používá HLSL[2]. Unity používá pro tvorbu shaderů modifikovaný HLSL jazyk. Syntaxe všech tří jazyků vychází z programovacího jazyka C. Jednotlivé úrovně vykreslovacího řetězce, které používá aplikace, jsou podrobněji popsány v následujících podsekcích.

[1] GLSL – OpenGL Shading Language
[2] HLSL – High Level Shader Language



Obrázek 3.1: Vykreslovací řetězec na grafické kartě. Neprogramovatelné úrovně jsou označeny barevně a programovatelné jsou označeny šedivě. Převzato a upraveno z [9].

3.2.1 Vertex Shader

Vertex shader je programovatelná jednotka na grafické kartě. Paralelně se provede nad každým vrcholem, který se nachází ve scéně. U vrcholů může provádět libovolné transformace, například s pozicí, normálou nebo výpočet texturových souřadnic. Mezi jednotlivými invokacemi, které se provádí nad každým vrcholem, nelze sdílet žádná data.

3.2.2 Teselace

Vertex shader se stará o transformaci vrcholů, ale neumožňuje rozdělování primitiv a přidávání nových. O tuto činnost se stará teselace, která se nachází za úrovní vertex shader. Teselace je relativně nový prvek, který je tvořen z úrovní hull shader, tesselator a domain shader. Hull shader a domain shader jsou narozdíl od tesselatoru plně programovatelné.

Teselace je proces rozdělení primitiv na více spojených primitiv tím, že se do modelu přidají nové vertexy. Výhodou teselace je, že nové vertexy vznikají přímo na grafické kartě, a tím se nezatěžuje procesor. Díky tomu lze použít model s nízkým počtem primitiv a následně na grafické kartě vygenerovat více primitiv, které vytvoří mnohem hladší povrch tělesa. Velké uplatnění teselace je například při vykreslení terénu, kde umožňuje implementovat postupné morfování při přechodu z jedné úrovně detailu do druhé.

3.2.3 Pixel shader

Pixel shader je poslední programovatelná úroveň grafického řetězce. Pixel shader se provede nad každým zpracovaným fragmentem, který vznikne při rasterizaci. Vstupní hodnoty jsou získány interpolací mezi předanými vertexy. Pro fragment může být například z interpolovaných souřadnic vypočítána jeho barva a hloubka.

Kapitola 4

Návrh

Cílem této bakalářské práce je vytvořit planetu s adaptivní úrovní detailu. Vykreslování terénu probíhá v reálném čase. Aplikace využívající adaptivní úroveň detailu terénu planety jsou takové, které dokáží vykreslit terén detailně při pohledu přímo z povrchu planety, i celou planetu při pohledu z vesmíru. Aplikace využívá pro různou úroveň detailu terénu modifikovaný algoritmus Chunked LOD 2.3.4, který umí zobrazit terén v několika různých detailech. Uživatel má možnost v reálném čase měnit velikost planety a maximální výšku hor, případně měnit výškovou mapu i s texturami.

4.1 Adaptivní úroveň detailu terénu

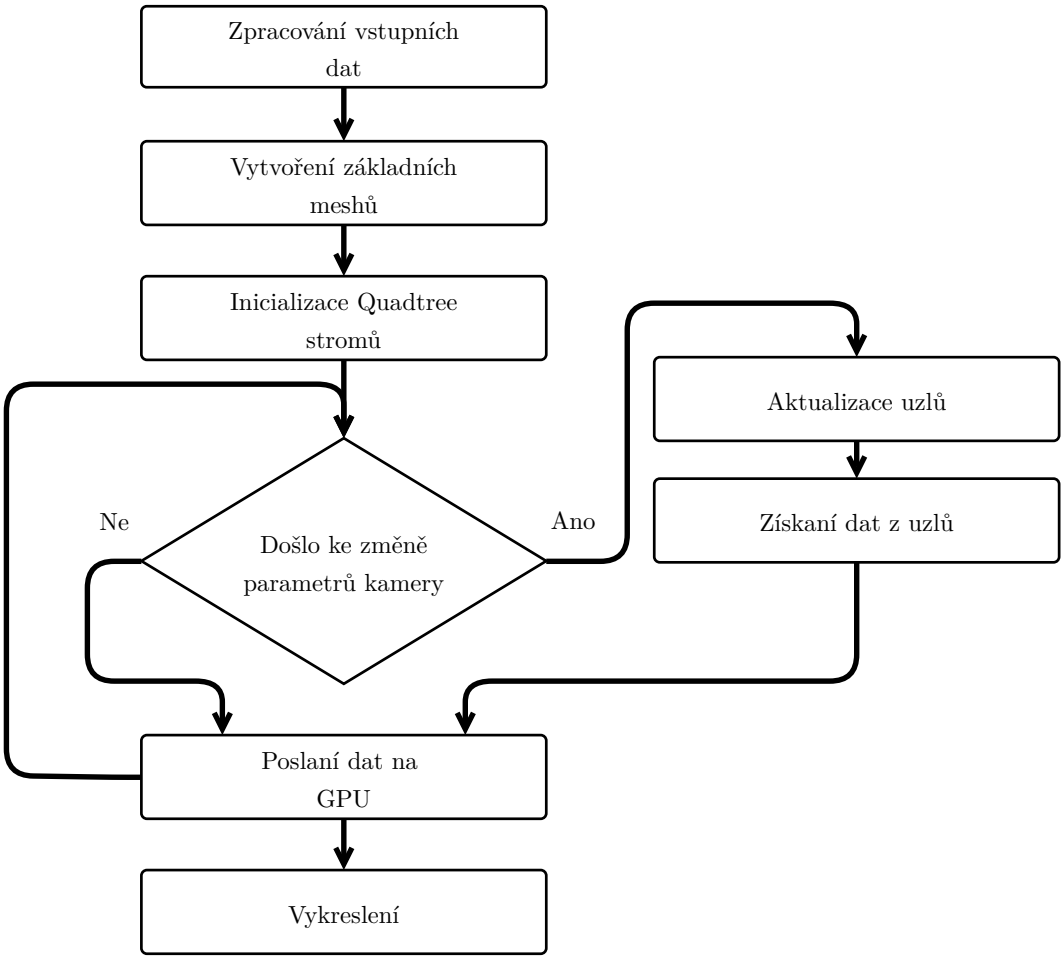
Pro adaptivní úroveň detailu terénu planety byl zvolen a upraven algoritmus Chunked LOD (viz 2.3.4). První důležitou modifikací algoritmu je použití pravidelné polygonální sítě místo nepravidelné. Druhou modifikací je jiný způsob řešení trhlin v geometrii terénu, kde není použit na zacelení trhliny pásek, který se přidává na každou hranu listu, ale je použit podobný způsob zacelení trhlin, jako u metody GeoMimMaping 2.3.3. Další z menších modifikací spočívá v tom, že terén není vytvořen v době předzpracování, ale je vytvářen přímo za běhu programu. To zaručuje možnost dynamické změny terénu. Při zpracování uzlů ve stromové struktuře neprobíhá vytváření geometrie terénu, ale pouze se počítá možná pozice a velikost dané dlaždice. Podoba terénu je modifikována do výsledné podoby na grafické kartě. Pro zvýšení efektivity vykreslování lze snadno aplikovat ořezávání dlaždic pomocí *frustum cullingu*, díky uložení dat v kvadratické stromové struktuře. Všechny uzly jsou testovány do té doby, dokud platí podmínka, že leží v zorném poli kamery.

4.2 Struktura aplikace

Na obrázku 4.1 jsou vidět postupná stádia aplikace. V prvním kroku aplikace zpracuje data, jako jsou výšková mapa, textury, průměr planety, maximální výška hor atd. Mohou být zadána implicitně nebo od uživatele. Dále vytvoří několik typů pravidelné polygonální sítě (dále jen „mesh“). Jakmile jsou vytvořeny meshe, aplikace inicializuje šest kvadratických stromů, kde každý z nich představuje jednu stranu krychle, která bude transformována na kouli. Každý z kořenů stromu může rekurzivně vytvořit další listy.

V této fázi se aplikace dostává do smyčky, která začíná dotazem, zda došlo od poslední iterace ke změně parametrů kamery. Pokud ano, aplikace začne aktualizovat uzly, které se nacházejí v zorném poli kamery. Z aktualizovaných uzlů jsou následně získána data, která

jsou poslána na grafickou kartu na další zpracování a vykreslení. Pokud kamera nezměnila svoji rotaci nebo pozici, přeskočí se krok, kdy se mají aktualizovat uzly, a přejde se rovnou k poslání dat na grafickou kartu. Následně se celá akce od dotazu na kameru provádí znovu.



Obrázek 4.1: Zjednodušené schéma aplikace.

4.3 Využití grafické karty

Jak bylo zmíněno výše, terén je transformován až na grafické kartě. Je tvořen z dlaždic, které využívají pro efektivní vykreslení metodu instancing (viz sekce 2.6.2). Následně se všechny vertexy vzniklých instancí začnou paralelně zpracovávat ve vertex shaderu. Vertex shader vypočítá pro každý vertex jeho pozici na planetě. Dále pomocí teselace může dojít k postupnému morfování mezi úrovněmi detailu terénu tak, aby nedošlo ke skokové změně. Následně pixel shader dojde k výběru textur a výpočtu osvětlovacího modelu.

Kapitola 5

Implementace

Tato kapitola popisuje implementační detaily. První část kapitoly popisuje implementaci metody Chunked LOD a její modifikace. Druhá část je zaměřena na popis jednotlivých kroku ve vykreslovacím řetězci.

5.1 Implementace modifikovaného Chunked LOD

Jak bylo zmíněno v sekci 4.1, pro adaptivní vykreslování detailu terénu byl zvolen modifikovaný algoritmus Chunked LOD. Jelikož pro vytvoření planety je použit převod z krychle na kouli, algoritmus inicializuje šest kvadratických stromů (pro každou stranu krychle jeden strom). Geometrie jednotlivých dlaždic není tvořena na procesoru, ale vytváří se na grafické kartě pomocí výškových map. Díky tomu jednotlivé uzly uchovávají pouze pozici, rotaci a velikost dané dlaždice. Všechny vykreslené dlaždice mají stejný počet vertexů, liší se pouze v měřítku vykreslení.

5.1.1 Aktualizace uzlů

Aktualizace uzlů probíhá pouze tehdy, když kamera změní rotaci, pozici nebo dojde ke změně dat ze strany uživatele. Při aktualizaci se začne procházet postupně všech šest stromů. Jako první krok probíhá kontrola pomocí metody *frustum culling*, zda se plocha dlaždice nachází alespoň částí v zorném poli kamery. Když je odpověď záporná, uzel i s jeho potomky se nenacházejí v zorném poli a algoritmus nemusí pokračovat v aktualizaci této větve. Při kladné odpovědi uzel vypočítá svoji vzdálenost od kamery a na základě výsledku rozhodne, zda bude vykreslen, nebo se provede jedna z operací *merge* nebo *split* (viz sekce 5.1.3).

Každý uzel je reprezentován jedním z několika typů meshů, kde typ meshe závisí na úrovni sousedních uzlů. Protože typy meshů jsou odvozeny od svého okolí, které při prvním průchodu ještě nemusí být plně aktualizované nebo vytvořené, je zapotřebí znovu projít všechny viditelné uzly ve stromech za účelem zjištění typu meshe daných uzlů a získaní potřebných dat pro vykreslení.

5.1.2 Řešení trhlin a T-spojů

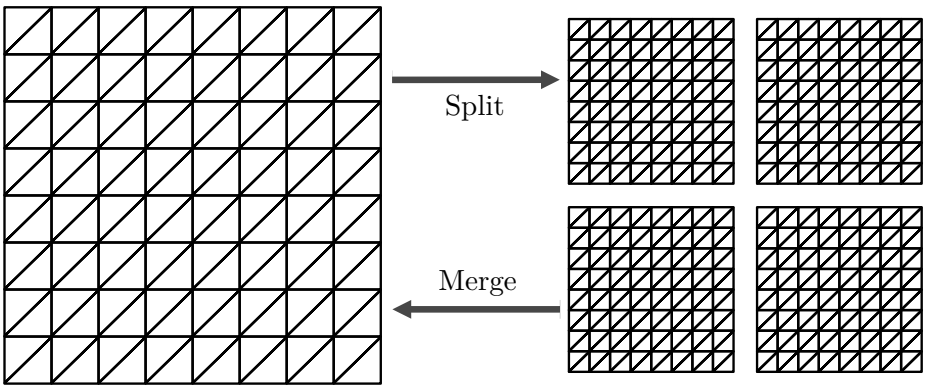
Aplikace řeší vznik trhlin a T-spojů (viz sekce 2.2.3) obdobně jako metoda GeoMipMapping. Dlaždice s menší úrovní detailu se přizpůsobí na společné hraně dlaždici s vyšší úrovní detailu. Přizpůsobení dlaždice je znázorněno na obrázku 2.8b. Proces zamezení vzniku trh-

lin a T-spojů, je počítán na procesoru. Každý uzel může obsahovat až čtyři ukazatele na sousední uzly o stejné velikosti. Při zjišťování, jaké hrany zkoumaného uzlu sousedí s uzly o vyšší úrovni detailu, zkoumaný uzel prochází ukazatele na sousední uzly, zda neobsahují potomky pro vykreslení. V případě, že potomky pro vykreslení obsahují, znamená to, že sousední uzly mají vyšší úroveň detailu a je nutné společnou hranu označit.

Podle počtu označených hran uzel vybere požadovaný mesh, který ho bude reprezentovat. Všechny možné varianty meshe, které mohou nastat, jsou znázorněny na obrázku 5.2. Aby existovalo co nejmenší množství variant, je použita statická třída *CalculateRotation*, která vypočítá odpovídající stupeň rotace meshe. K samotné rotaci dochází ve vertex shaderu.

5.1.3 Operace split a merge

Operací *split* dochází k vytvoření čtyř nových uzlů (viz obrázek 5.1), které mají čtvrtinovou velikost oproti rodičovskému uzlu. Při inicializaci nového uzlu se vypočítá jeho pozice, která odpovídá umístění na krychli. Protože pozice uzlu na krychli neodpovídá jeho skutečné pozici na planetě a během výpočtů pomocí metody *frustum culling* by vznikala nekorektní data, vypočítá se jeho skutečná pozice podle vzorce převodu krychle na kouli 2.1. Následně se okolo vzniklé pozice vytvoří osově orientovaný kvádr, který se používá při zjišťování vzdálenosti dlaždice od kamery. Operací *merge* dochází ke sloučení uzlů (viz obrázek 5.1), při které se rekurzivně odstraňují všichni potomci. Tento způsob má jednu negativní vlastnost v tom, že musí v případě opětovného přiblížení kamery znovu vytvořit stejné uzly, které už byly jednou vytvořeny.



Obrázek 5.1: Operace split a merge.

5.1.4 Chybová metrika

Jako chybová metrika je zvolena vzdálenost pozorovatele A od osově orientovaného kvádru uzlu. Nejprve se na osově orientovaném kvádru najde nejbližší bod k pozorovateli B a následně se vypočítá vzdálenost bodů $|AB|$ v prostoru. Aby nedocházelo k lineární změně úrovně detailu terénu, je změna detailu závislá na velikosti dané dlaždice. Rozdělení uzlu proběhne tehdy, když $(scale \cdot 2 \cdot \tau) > |AB|$, kde $scale$ je velikost dané dlaždice a τ je koeficient, který může nastavit uživatel pro změnu úrovně detailu.

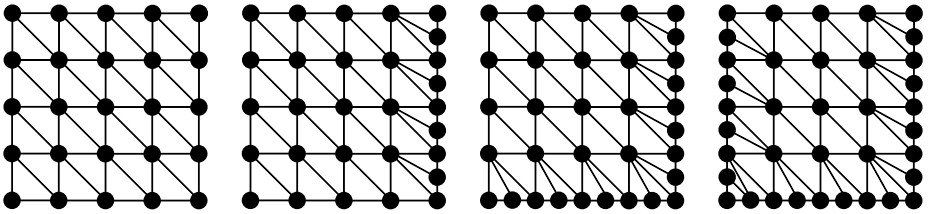
5.2 Ořezávání odvrácených stran

Aby byla metoda *frustum culling* více efektivní, je při každém pohybu kamery modifikován dosah vzdálené ořezové roviny. Vzdálená ořezová rovina je upravována tak, aby se vždy rovnala vzdálenosti pozorovatele od středu planety. Tím se zaručí, že odvrácená strana planety od pozorovatele nikdy nebude vykreslena. Algoritmus je dále rozšířen o ořezávání ploch, které se nacházejí za horizontem. Při zjišťování, zda se dlaždice nachází za horizontem, je vypočítán úhel mezi její normálou, směřující od středu planety, a normálou, která je vedena od pozice kamery do středu dlaždice. Pokud je výsledný úhel větší než devadesát stupňů, dlaždice se nachází za horizontem a nebude vykreslena.

5.3 Generování meshů

Jak už bylo zmíněno v návrhu, modifikovaný Chunked LOD používá pro reprezentaci terénu pravidelnou polygonální síť. Jelikož všechny uzly používají jen několik typů meshů, u kterých se vertexy transformují do podoby terénu až na grafické kartě, je zbytečné, aby každý uzel posílal svůj mesh zvlášť na grafickou kartu. Proto aplikace používá instaning 2.6.2, který je implementovaný ve třídě *DrawMeshInstanced*. Třída *MeshGenerator* při inicializaci vytvoří pouze čtyři druhy meshů znázorněných na obrázku 5.2.

Původně aplikace generovala pouze jeden druh meshe, který používaly všechny uzly, ale pro zamezení vzniku trhlin mezi jednotlivými dlaždicemi bylo potřeba vygenerovat další tři možnosti, které mohou nastat.



Obrázek 5.2: Všechny varianty vygenerovaných meshů, kterou mohou být použity ve výsledném terénu.

5.4 Výpočty na grafické kartě

Program využívá instancing pro efektivní vykreslení velkého počtu dlaždic. Na straně procesoru se vytvoří buffery, do kterých jsou vložena data všech dlaždic, která byla při kroku aktualizace označena k vykreslení. Samotné zpracování ve vykreslovacím řetězci probíhá ve fázích vertex shader, teselace a pixel shader.

5.4.1 Vertex Shader

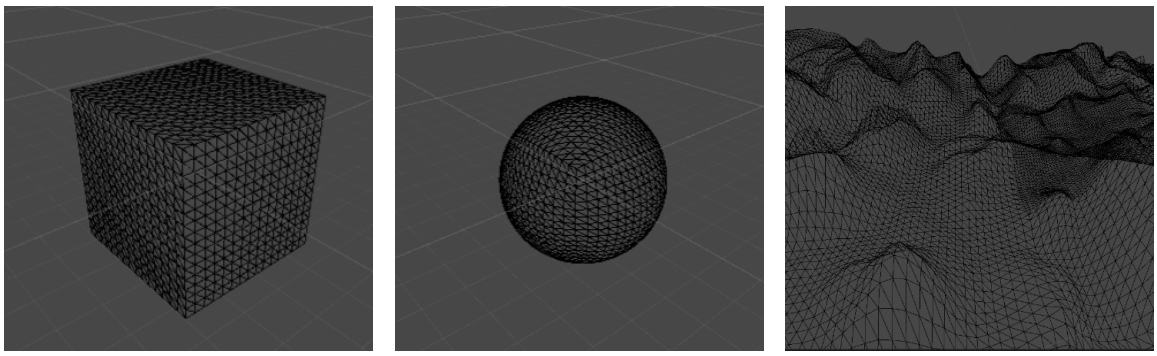
Vertex shader se stará o transformaci vertexů do podoby planety. Každý vertex si nese informaci o jeho indexu, který udává, do jaké instance patří. Díky tomu vertex získá správná data, která jsou přiřazena dané instanci. Data jsou uložena ve dvou bufferech. Konkrétně se jedná o buffery *positionBuffer* a *directionsBuffer*. První ze zmíněných obsahuje data, která určují pozici středu dlaždice a její velikost. Druhý buffer obsahuje úhel rotace a normálu, která definuje, na jaké straně krychle se daná dlaždice nachází.

Jak bylo zmíněno v sekci 5.3, bylo nutné vytvořit čtyři meshe pro zamezení vzniku trhlin. Aby existovalo co nejméně meshů, které jsou posílány na grafickou kartu, jsou rotovány tak, aby pokryly všechny možné varianty pro spojování dlaždic s různou úrovní detailu. Ve vertex shaderu se vypočítá správná transformace pro jednotlivé vrcholy podle zadané rotace tak, aby správně spojila dlaždice s různou úrovní detailu a nedošlo k trhlinám.

Když jsou vertexy správně transformovány o rotaci, dalším krokem vertex shader využije normálu, kterou získal z *directionsBuffer*, aby transformoval vrcholy do podoby krychle. Následně použije *positionBuffer*, podle kterého zvětší rozestupy mezi vertexy daného meshe a přičte k jejich pozici střed meshe získaný z bufferu. Nově vzniklá pozice odpovídá umístění na krychli. Když je krychle z vertexů vytvořena, jednotlivé vrcholy se transformují z krychle na kouli podle vzorce (2.1). Dalším krokem vertex shader vypočítá texturové souřadnice uv, viz vzorec (2.3), díky kterým dokáže získat výšku daného pixelu z výškové mapy. Výslednou výšku přičte k vertexu pomocí vzorce:

$$vertex.xyz = vertex.xyz + n \cdot (h \cdot h_m), \tag{5.1}$$

kde $vertex.xyz$ je pozice daného vrcholu, n je normála ve vrcholu, h je výška získaná z výškové mapy a h_m je maximální výška, které může terén nabývat. Celý proces transformace meshů do podoby planety je znázorněn na obrázku 5.4.



Obrázek 5.3: Postupná transformace dlaždic, která probíhá ve vertex shaderu, do podoby planety s terénem.

5.4.2 Teselace

Při přechodu dlaždice z jedné úrovně detailu do druhé dochází ke skokovým změnám (*popping effect*), kdy je náhle přidána nebo odebrána polovina vertexů dané dlaždice, a dochází tak k viditelné změně geometrie terénu. V této práci je (*popping effect*) téměř eliminován pomocí teselace.

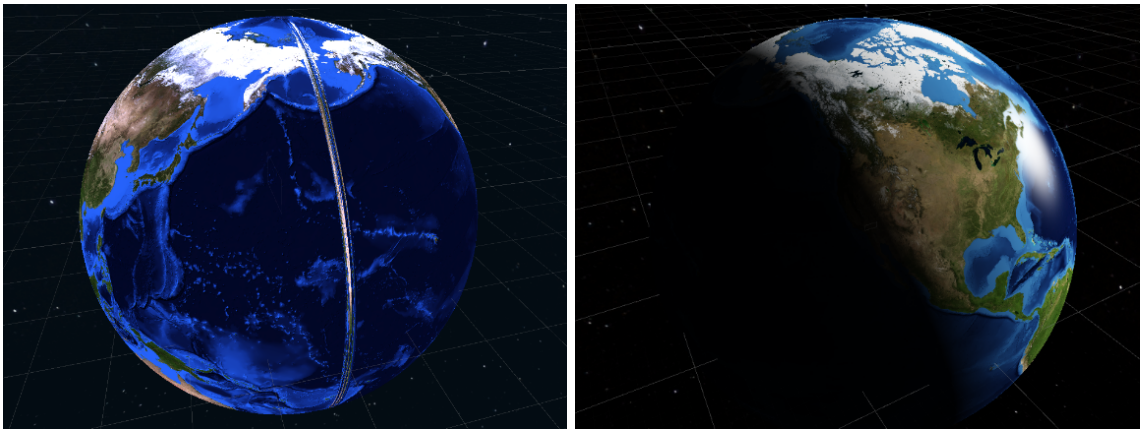
Jako první jednotka u tesselace je hull shader, kterému je na vstupu předán polygon. Hull shader určí pro každou hranu polygonu teselační faktor, který je závislý na velikosti dané hrany a vzdálenosti jejího středu od pozorovatele. Při přibližování kamery k hraně se úroveň teselace lineárně zvyšuje až do své maximální hodnoty, kdy dojde k rozdělení dlaždice pomocí operace *split*. Získané teselační faktory následně hull shader předá do jednotky tesselator. Tesselator automaticky na základě přijatých teselačních faktorů rozdělí polygon na více primitiv vytvořením nových vertexů.

Poslední jednotkou u teselace je domain shader, který vypočítá pozice nově vzniklých vertexů. Je invokován pro každý nově vzniklý vrchol. Jako vstupní parametry jsou mu předány všechny informace o původním polygonu a barycentrické souřadnice, které jsou získány z tesselatoru. Interpolací původních hodnot podle barycentrických souřadnic vzniknou data pro vertex, který byl vytvořen v tesselatoru. Nově vzniklému vertexu je vypočítána nová výška, která je získána z výškové mapy.

5.4.3 Pixel shader

Pixel shader vypočítává finální barvu daného fragmentu. Z interpolovaných hodnot určí texturovací souřadnice *uv*. K výpočtu používá, stejně jako vertex shader, vzorec (2.3). Následně využije získané *uv* souřadnice pro zjištění barvy pixelu z textury. Původně se, pro snížení výpočtů na grafické kartě, *uv* souřadnice počítaly pouze ve vertex shaderu, který je dále předával strukturou do pixel shaderu. Tento přístup se ale ukázal jako nedostačující, protože v místě napojování začátku a konce textury docházelo k interpolaci *uv* souřadnic blízké jedničky do nuly, a to vytváří v místě napojování textury svislý pruh přes celou planetu (viz obrázek 5.4a).

Po získaní barvy se vypočítá difúzní a lesklá složka, podle Phongova osvětlovacího modelu 2.5. Díky tomu lze získat na odvrácené straně planety od Slunce tmu a simulovat den a noc (viz obrázek 5.4b). Aby pozorovatel mohl vidět v reálném čase úroveň a uspořádání dlaždic okolo něj, je implementována možnost přepínání mezi normálním modelem a modelem, který různými barvami zobrazuje dlaždice s rozdílnou úrovní detailu.



(a) Vznik svislého pruhu zaokrouhlovací chybou v pixel shaderu.

(b) Osvětlení planety pomocí Phongova osvětlovacího modelu.

Obrázek 5.4: Mapování textur na planetu s osvětlovacím modelem.

5.5 SkyBox

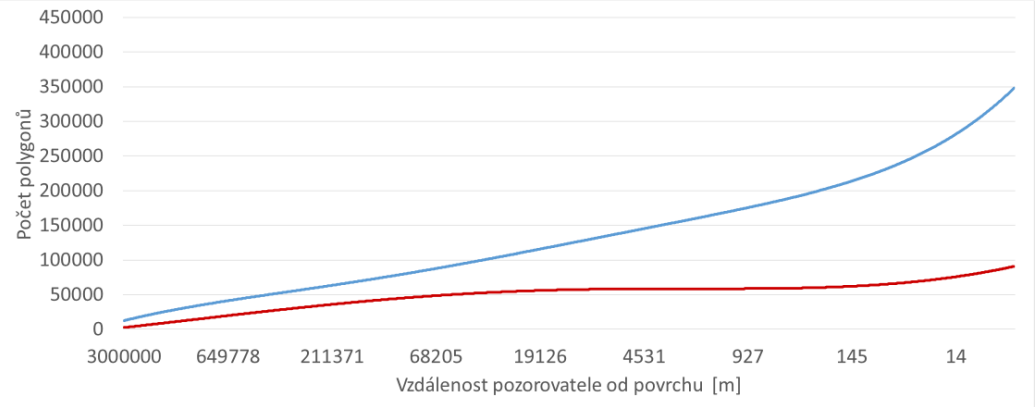
V aplikaci je implementován jednoduchý SkyBox, který ztvárňuje okolní vesmír. Pro lepší vizuální efekt je implementována postupná změna SkyBoxu, když se pozorovatel začne přibližovat k povrchu planety. Protože na část planety odvrácenou od slunce nedopadá světlo, musí se postupně změnit i SkyBox zpět na hvězdnou oblohu. Kdyby se tak nestalo, scéna by vypadala nepřirozeně. Tento problém je vyřešen pomocí výpočtu difúzní složky světla pomocí vzorce 5.4.1, kde $\vec{N}$ je normála kamery, která směřuje od středu planety.

Kapitola 6

Testování a výsledek

Aplikace byla vyvíjena a testována na notebooku Lenovo Z570 s operačním systémem Microsoft Windows 10. Počítač má operační paměť 12 GB, grafickou kartu GeForce GT 540M s 2 GB paměti a procesor Intel Core i5-2430M. Při vývoji a testování byla použita data, která byla získána z [8].

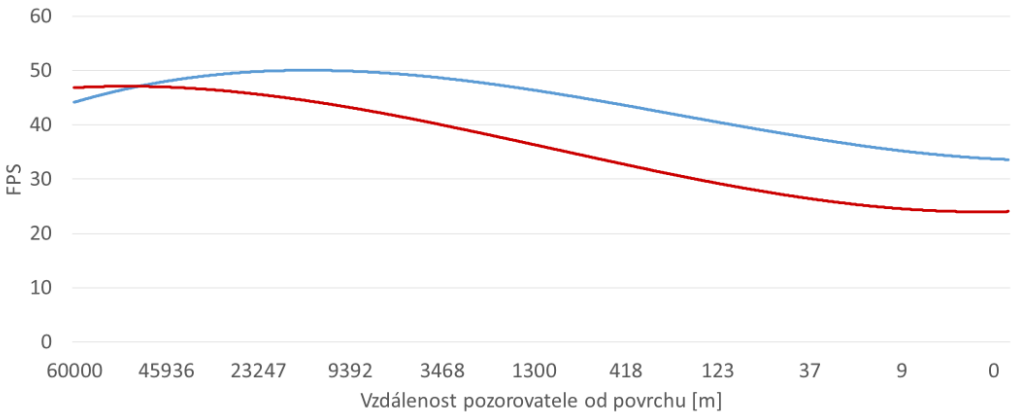
První test proběhl nad planetou s poloměrem 3000 km. Bylo testováno, kolik polygonů se vykreslí při použití základní metody *frustum cullin* a kolik po implementaci rozšíření, které ořezává dlaždice za horizontem. Rozšíření metody ukázalo veliký posun v efektivitě zobrazování terénu, kdy se při stejné vzdálenosti pozorovatele vykresluje mnohem méně polygonů. Výsledek testování je ukázán na grafu 6.1.



Obrázek 6.1: Graf porovnává základní metodu *frustum culling* (modrá barva) s jejím rozšířením o ořezávání dlaždic, které jsou za horizontem (červená barva).

Druhé testování proběhlo nad planetou s poloměrem 60 km. Pro testování byla zvolena teselace, která vytváří nové polygony přímo na grafické kartě, a tím umožňuje postupné morfování dlaždic, při přechodu z jedné úrovně do druhé. Z výsledků testování 6.2 lze vidět, že použitím teselace dochází ke snížení zobrazovacích snímků za vteřinu.

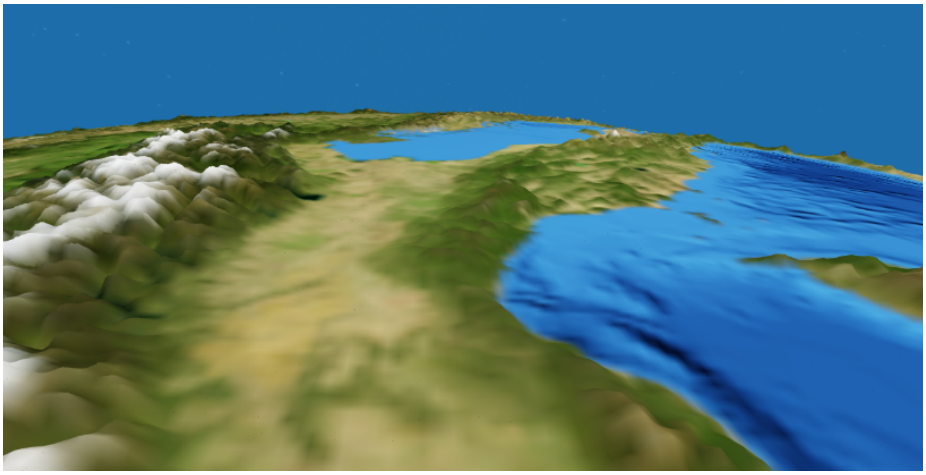
Třetí testování proběhlo nad planetou s průměrem 4000 km. Pozice pozorovatel byla statická 6500 m nad povrchem. Na obrázku 6.3 je zobrazen pohled z jeho pozice. Cílem testování bylo porovnat vykreslení na různých rozlišeních oken. Z naměřených hodnot, které jsou v tabulce 6.1, je vidět, že při rozlišení o velikostech 1920x1080 a 1366x768 je počet snímků za sekundu se zapnutou teselací velmi nízký.



Obrázek 6.2: Graf porovnávání hodnoty FPS při přibližování pozorovatele k povrchu planety s (červena barva) a bez použití teselace (modrá barva).

Rozlišení obrazovky	**FPS bez teselace**	**FPS s teselací**
1920x1080	24	17
1366x768	30	20
800x600	34	28

Tabulka 6.1: Porovnání průměrné hodnoty FPS u různých velikostí rozlišení obrazu při průměru planety 4000 km s pozorovatelem, který je 6500 m nad povrchem země.



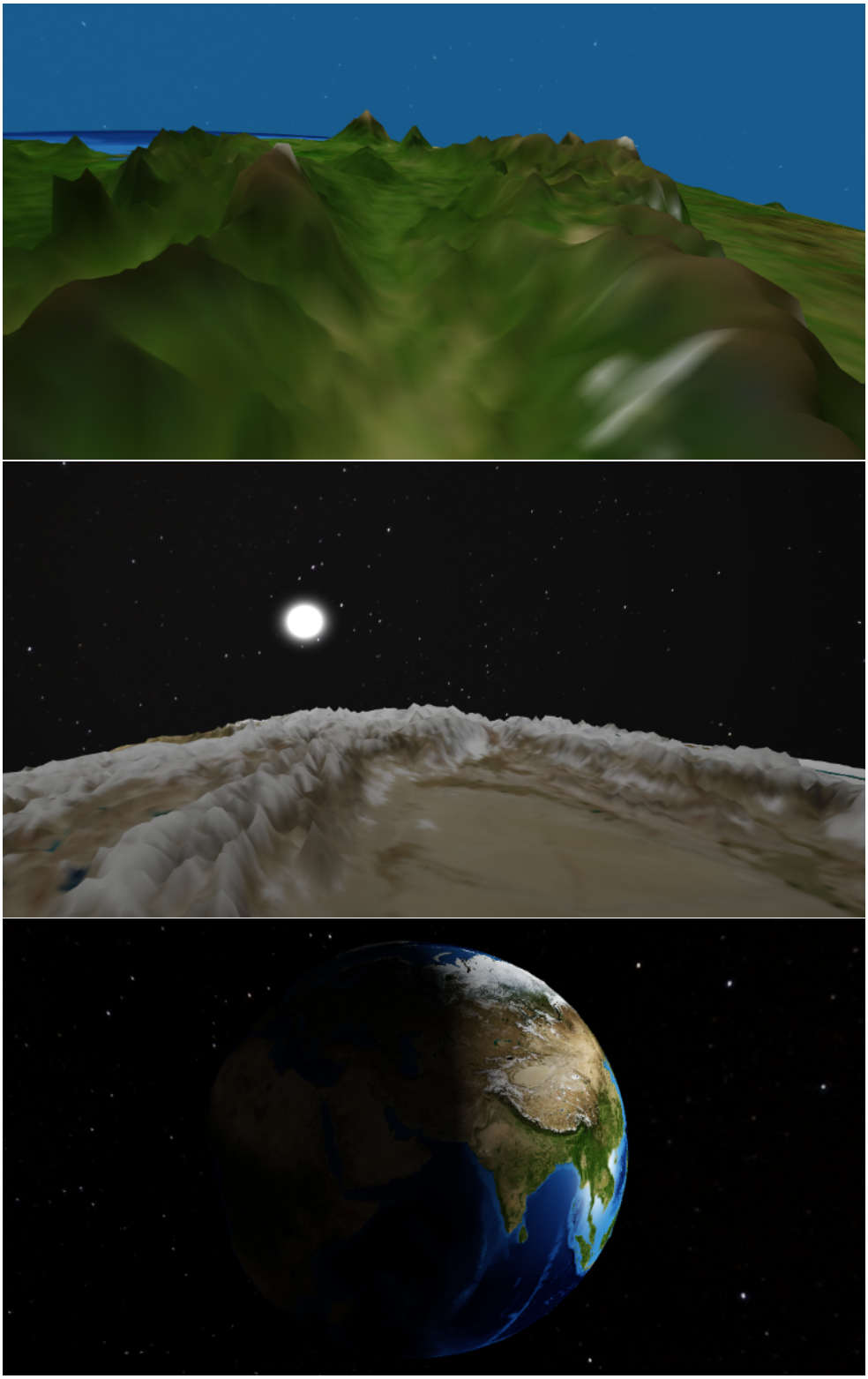
Obrázek 6.3: Pohled pozorovatele při třetím testování. Výška je 6500 m nad povrchem.

Poslední testování proběhlo nad planetou s průměrem 4000 km. Pozice pozorovatele byla statická 2235 km nad povrchem. Na obrázku 6.4 je zobrazen pohled z jeho pozice. Cílem testování bylo porovnat vykreslení na různých rozlišeních oken. Výsledky měření jsou v tabulce 6.2.

Rozlišení obrazovky	FPS bez teselace	FPS s teselací
1920x1080	59	35
1366x768	58	44
800x600	58	46

Tabulka 6.2: Porovnání průměrné hodnoty FPS u různých velikostí rozlišení obrazu při průměru planety 4000 km s pozorovatelem, který je 2235 km nad povrchem.

Obrázek 6.4: Pohled pozorovatele při čtvrtém testování. Výška je 2235 km nad povrchem.



Obrázek 6.5: Ukázky výsledné scény.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vykreslení planety s adaptivní úrovní detailu terénu v reálném čase, která by využívala pro vytváření terénu grafickou kartu. Nejdříve bylo zapotřebí nastudovat různé algoritmy pro tvorbu rozměrného terénu a seznámit se s problematikou jeho převodu na planetu.

Na základě nastudované literatury byla vytvořena plně funkční aplikace, která dokáže v reálném čase vykreslit planetu s adaptivní úrovní detailu terénu, mající až několik tisíc kilometru v průměru a měnit její základní vlastnosti za běhu programu. V aplikaci je implementován modifikovaný Chunked LOD, který využívá instancing pro efektivní vykreslování velkého množství dlaždic. Pro efektivní zobrazování je geometrie terénu modifikovaná přímo na grafické kartě pomocí vertex shaderu. Aby nevznikaly náhlé skoky při změně úrovně detailu dlaždice, je implementována teselace, která zajistí plynulý přechod z jedné úrovně do druhé. Pro lepší vizuální efekt byl implementován Phongův osvětlovací model, pomocí kterého je na planetě vytvořena iluze dne a noci.

Díky rozmanitosti aplikace se nabízí hned několik možností budoucího vývoje. Stávající řešení by se mohlo vylepšit zejména v kvalitě zobrazení textur, kdy by při přibližování pozorovatele k povrchu planety docházelo k postupné změně textur celé planety za detailnější na základě směru normály a původní barvy. Pro další vylepšení vzhledu planety by bylo možné vytvořit atmosféru, která by byla založena na aproximaci rozptylu světla v atmosféře. Dále by bylo možné úplně přesunout všechny výpočty algoritmu pro adaptivní úroveň detail terénu přímo na grafickou kartu a přenechat procesor jiným výpočtům. Pro zvýšení detailnosti pohoří na planetě by byla možnost generování výškových map, které by se následně použili pro dosažení vyšších detailů povrchu terénu.

Literatura

[1] Phong shading. *Phong shading* [online]. [cit. 18.04.2019].
URL https://en.wikipedia.org/wiki/Phong_shading

[2] Asirvatham, A.; Hoppe, H.: Terrain rendering using GPU-based geometry clipmaps. *GPU gems*, ročník 2, č. 2, 2005: s. 27–46.

[3] Azgaar: Image to Heightmap converter. Feb 2018.
URL https://azgaar.wordpress.com/2018/02/26/image-converter/

[4] Borthwick, A.; Cruz Leon, S.; Józsa, J.: Adaptive quadtree model of shallow-flow hydrodynamics. *Journal of Hydraulic Research - J HYDRAUL RES*, ročník 39, 10 2001: s. 413–424, doi:10.1080/00221680109499845.

[5] Clark, J. H.: Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, ročník 19, č. 10, 1976: s. 547–554.

[6] De Boer, W. H.: Fast terrain rendering using geometrical mipmapping. *Unpublished paper, available at http://www. flipcode. com/articles/article geomipmaps. pdf*, 2000.

[7] Duchaineau, M.; Wolinsky, M.; Sigeti, D. E.; aj.: ROAMing terrain: real-time optimally adapting meshes. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, IEEE, 1997, s. 81–88.

[8] Earth, N. V.: Home. [online]. [cit. 17.04.2019].
URL https://visibleearth.nasa.gov/

[9] GrantMeStrength: Graphics Pipeline - Windows applications.
URL https://docs.microsoft.com/en-us/windows/desktop/direct3d11/overviews-direct3d-11-graphics-pipeline

[10] Losasso, F.; Hoppe, H.: Geometry clipmaps: terrain rendering using nested regular grids. In *ACM Transactions on Graphics (TOG)*, ročník 23, ACM, 2004, s. 769–776.

[11] Luebke, D.; Reddy, M.; Cohen, J. D.; aj.: *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[12] Martinez Rubi, O.; Verhoeven, S.; van Meersbergen, M.; aj.: Taming the beast: Free and open-source massive point cloud web visualization. 11 2015, doi:10.13140/RG.2.1.1731.4326/1.

[13] Oren, M.; Nayar, S. K.: Generalization of Lambert's Reflectance Model. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive*

Techniques, SIGGRAPH '94, New York, NY, USA: ACM, 1994, ISBN 0-89791-667-0, s. 239–246, doi:10.1145/192161.192213.
URL http://doi.acm.org/10.1145/192161.192213

[14] Phong, B. T.: Illumination for computer generated pictures. *Communications of the ACM*, ročník 18, č. 6, 1975: s. 311–317.

[15] Ulrich, T.: Rendering massive terrains using chunked level of detail control. In *Proc. ACM SIGGRAPH 2002*, 2002.

[16] Wagner, D.: Terrain geomorphing in the vertex shader. *ShaderX2, Shader Programming Tips and Tricks with DirectX 9*, 2004.

[17] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer press, 2004.

Příloha A

Návod

Při spuštění aplikace se zobrazí základní menu, ve kterém si uživatel může nastavit základní parametry, kterými jsou průměr planety, maximální výška hor, detailnost dlaždic a má možnost vypínání a zapínání teselace, která částečně zamezuje *popping effectu*.

Ovládání

- W, A, S, D – Pohyb kamery.
- Q, E – Rotace kamery.
- R, F – Rotace Slunce.
- SHIFT – Zrychlení pohybu.
- T – Přepínáním mezi texturou a módem, který barevně zvýrazní úrovně dlaždic.
- ESC – Přístup k ovládání uživatelského rozhraní.

Uživatelské rozhraní

Uživatel se po stisku klávesy *ESC* přepne do uživatelského rozhraní (viz obrázek A.1), ve kterém má přístup ke změnám základních parametrů stejně jako v menu, bez možnosti měnit detailnost dlaždic.

Obrázek A.1: Uživatelské rozhraní za běhu aplikace.